Try before You Buy: Privacy-preserving Data Evaluation on Cloud-based Machine Learning Data Marketplace

[1]Qiyang Song, [2,3]Jiahao Cao, [1]Kun Sun, [2,3]Qi Li, and [2,3]Ke Xu
[1]Department of Information Sciences and Technology, George Mason University
[2]Department of Computer Science and Technology, Tsinghua University
[3]Institute for Network Sciences Cyberspace and BNRist, Tsinghua University
qsong4@gmu.edu,caojh15@gmail.com,ksun3@gmu.edu,{qli01,xuke}@tsinghua.edu.cn

ABSTRACT

A cloud-based data marketplace provides a service to match data shoppers with appropriate data sellers, so that data shoppers can augment their internal data sets with external data to improve their machine learning (ML) models. Since data may contain diverse values, it is critical for a shopper to evaluate the most valuable data before making the final trade. However, evaluating ML data typically requires the cloud to access a shopper's ML model and sellers' data, which are both sensitive. None of the existing cloud-based data marketplaces enable ML data evaluation while preserving both model privacy and data privacy. In this paper, we develop a privacy-preserving ML data evaluation framework on a cloud-based data marketplace to protect shoppers' ML models and sellers' data. First, we provide a privacy-preserving framework that allows shoppers and sellers to encrypt their models and data, respectively, while preserving data functionality and model functionality in the cloud. We then develop a privacy-preserving data selection protocol that enables the cloud to help shoppers select the most valuable ML data. Also, we develop a privacy-preserving data validation protocol that allows shoppers to further check the quality of the selected data. Compared to random data selection, the experimental results show that our solution can reduce 60% prediction errors.

CCS CONCEPTS

• **Security and privacy** → *Cryptography*; **Privacy-preserving protocols**.

KEYWORDS

Data Market, Neural Networks, Privacy-preserving

ACM Reference Format:
[1]Qiyang Song, [2,3]Jiahao Cao, [1]Kun Sun, [2,3]Qi Li, and [2,3]Ke Xu. 2021. Try before You Buy: Privacy-preserving Data Evaluation on Cloud-based Machine Learning Data Marketplace. In *Annual Computer Security Applications Conference (ACSAC '21), December 6–10, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3485832.3485921

ACSAC '21, December 6–10, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8579-4/21/12…$15.00
https://doi.org/10.1145/3485832.3485921

1 INTRODUCTION

Nowadays, the advent of deep learning techniques significantly improves the performance of traditional machine learning (ML) in a number of fields, such as image recognition [36], speech recognition [15], and natural language processing [29]. The data-driven deep learning algorithms heavily rely on a large amount of good-quality data to train a well-performing learning model, and there is a need for customers to augment or enrich their internal data sets with external data. To enable large-scale data acquisition, the industry gradually develops a number of data marketplaces [3, 9, 18], which use the Data-as-a-Service mode [41] to build a data exchanging platform for both enterprises and individuals.

A data marketplace usually offers data of different values for training a ML model, where the valuable data may significantly improve the model performance and the less valuable data have minor contribution. Therefore, it is crucial for a cloud-based data marketplace to assist data shoppers in evaluating and purchasing the most valuable data. Typically, evaluating ML data for a data shopper may require the cloud to access the shopper's model and sellers' data. However, sellers are not willing to expose their data before the final payment is settled, in fear of their data being leaked and thus losing values. Meanwhile, shoppers are reluctant to disclose their proprietary models since the models are their digital assets. Therefore, it is necessary to design a privacy-preserving ML data evaluation framework on a cloud-based data marketplace.

Existing cloud-based data marketplaces [17, 21, 23] provide flexible data search and evaluation. However, they do not support data evaluation for ML models. Some semi-supervised machine learning techniques [1, 2] can be used to evaluate valuable training data from a data pool according to model functionality. However, these techniques do not provide privacy protection on models and data in the cloud. Existing ML encryption frameworks [7, 14, 16, 19, 27, 31, 32] can be applied to preserve model privacy and data privacy in the cloud; however, since they are built on either homomorphic encryption (HE) [7, 14, 16, 19] or secure multi-party computation (MPC) schemes [27, 31, 32], they incur prohibitively expensive computation or communication overhead when executing model functions. Furthermore, as these frameworks are not specially designed to evaluate data, we cannot rely on them to provide privacy-preserving data evaluation.

In this paper, we propose Primal, a privacy-preserving and efficient machine learning data evaluation framework on a cloud-based data marketplace. It not only enables data sellers to sell ML data in the cloud without exposing data, but also allows data shoppers to evaluate and purchase valuable ML data without leaking their models. Primal is powered by three designs, namely, an efficient

ML encryption protocol, a privacy-preserving data selection protocol, and a privacy-preserving data validation protocol. Specifically, the ML encryption protocol enables shoppers and sellers to encrypt their models and data in the cloud. The privacy-preserving data selection and data validation protocol allow the cloud to help shoppers screen out valuable data from sellers' encrypted data.

Our ML encryption protocol allows shoppers and sellers to encrypt their models and data without depriving the functionality of models and data in the cloud. Particularly, the functionality means that the cloud can perform training and prediction operations with encrypted models and data. For a neural network model, the first step in training and prediction operations is matrix or convolution computation. As the two types of computation can be transformed to inner product operations, we utilize inner product functional encryption (IFE) [1, 2] to encrypt input data and the parameters of the first hidden layer. For the parameters of the remaining layers, we apply matrix transformation to convert these parameters with random numbers. Since IFE and matrix transformation are lightweight, we achieve higher efficiency than HE-based and MPC-based ML encryption.

Note that the cloud cannot directly evaluate valuable data for a data shopper since sellers' data are encrypted in the cloud. Therefore, we design a privacy-preserving data selection protocol that allows the cloud and a shopper jointly to select valuable data. For a shopper's model, valuable data refer to some informative data that can significantly improve its model performance. As active learning [5, 13, 20] can estimate data informativeness with prediction values, the cloud and shopper can utilize this technique to select valuable data without seeing the original data. To provide prediction values for evaluation, our data selection protocol offers a prediction approach that allows the cloud to use a shopper's encrypted model to predict sellers' encrypted data. Then, the shopper can retrieve some prediction values from the cloud to evaluate sellers' data.

Although the data selection protocol can select informative data for a shopper, the selected data may contain irrelevant or falsely labeled data, which may poison the shopper's model. Therefore, we design a privacy-preserving data validation protocol that allows the shopper to further examine the quality of the selected data. Note that the shopper cannot access the original data before the final payment. Thus, it cannot directly estimate data quality. Fortunately, another possible approach is to use the selected data to retrain the shopper's model and observe the change of model performance to estimate data quality. More specifically, the shopper first requires the cloud to retrain its encrypted model with the selected data, and it then collects some prediction values to estimate data quality.

We conduct a security analysis of Primal and prove its security in the untrusted cloud. Additionally, we carry out experiments to demonstrate the effectiveness and efficiency of Primal. Experimental results show Primal can effectively select valuable ML data that significantly improves model performance. Compared to random data selection, Primal can reduce 60% prediction errors. Even if the selected data contain irrelevant and falsely labeled data, Primal can detect such data with more than 99% accuracy. Furthermore, the results show that Primal achieves high efficiency on encryption and machine learning operations. For example, Primal can provide 17× faster prediction operations compared to a popular HE-based ML encryption framework.

In summary, we make the following contributions:

- We propose a privacy-preserving and efficient ML data evaluation framework on a cloud-based data marketplace.
- We provide an efficient ML encryption protocol that can preserve both the privacy and functionality of sellers' models and shoppers' data in the cloud.
- We design a privacy-preserving data selection protocol and data validation protocol that allow the cloud to help a shopper select and refine valuable data.
- We conduct a security analysis and experiments to demonstrate the security, effectiveness, and efficiency of Primal.

2 BACKGROUND

In this section, we provide the necessary background on neural networks, active learning, and inner product functional encryption.

Neural Networks. Typical neural networks, e.g., CNNs and MLPs, consist of an input layer, hidden layers, and an output layer. Particularly, the input layer receives high dimensional input data, the output layer outputs the corresponding prediction values[1], and the hidden layers learn the relationship between input data and prediction values. In this section, we briefly introduce two fundamental types of hidden layers, i.e., fully-connected layers and convolution layers. A fully-connected layer receives the input data from the previous layers and perform computation as $f(\mathbf{W}\vec{x} + b)$, where $\mathbf{W}$ is a parameter matrix, $\vec{x}$ is the input data, b is a bias, and f is an activation function. Note that b can be embedded into the matrix $\mathbf{W}$. A convolution layer performs convolution computation between convolution kernels and input data (a two-dimensional matrix), runs activation functions, and then outputs a feature map (activation values) to the next layer. Here, we provide a brief review of some technical terms as follows.

- Convolution computation: maps a two-dimension matrix to a new matrix with convolution kernels. It consists of multiple convolution strides. In each stride, it executes inner product operations between a subarea of the input matrix and kernels.
- Activation function: is a non-linear function. Prior work has proposed a variety of activation functions, such as Relu, Sigmoid, and Squared Function [10, 26].
- Cost function: computes the prediction errors between prediction values and labels, which can reveal prediction performance.
- Feed forward: is a process that data flows from the input layer to the output layer. It finally outputs prediction values.
- Back propagation: is a part of model training. It computes parameter gradients $\nabla\mathbf{W}$ according to a cost function and then updates parameters $\mathbf{W}$. To be precise, the parameters are updated as $\mathbf{W} - \alpha * \nabla\mathbf{W}$, where α is a learning rate.

Active Learning. Active learning is a semi-supervised machine learning approach, enabling a trainer to select informative training data from an unlabeled data pool. Compared with random data selection, active learning can select informative data to significantly change the decision boundary of models. In the field of active learning, uncertainty selection [5, 13, 20] is a popular data selection algorithm, which can select informative data with prediction values instead of accessing original data.

[1]Prediction values refer to the confidence values on multiple classes.

Inner Product Functional Encryption (IFE). As a new generation of public-key encryption, inner-product functional encryption allows an untrusted party to efficiently compute inner products over ciphertexts with restricted decryption keys. It works as follows:

- Setup(1^k): takes a security parameter 1^k as input, and outputs a master key pair (msk, mpk).
- KeyGen($\vec{x}, msk$): takes a vector $\vec{x}$ and a master secret key msk as input, and outputs a secret functional key sk_x.
- Encrypt($\vec{y}, mpk$): takes a vector $\vec{y}$ and a master secret key mpk as input, and outputs a vector of ciphertexts $\vec{C}_y$.
- Decrypt($\vec{C}_y, sk_x$): takes a ciphertext $\vec{C}_y$ and a secret functional key sk_x as input, and outputs the original inner product $\vec{x} \cdot \vec{y}$.
- MDec($\vec{C}_y, msk$): takes a ciphertext $\vec{C}_y$ and a master secret key msk as input, and outputs the original vector $\vec{y}$.

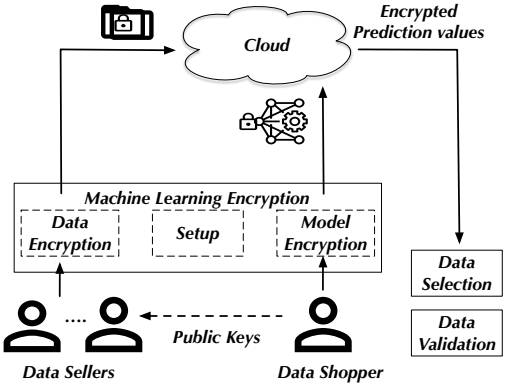


Figure 1: Primal Architecture

3 SYSTEM OVERVIEW

In this section, we first describe our threat model and assumptions. We then present the Primal architecture.

3.1 Threat Model and Assumptions

In this paper, we consider a typical cloud-based data marketplace consisting of a data shopper, a group of data sellers, and an untrusted cloud. To gain benefits from their data, sellers tend to upload their data to the cloud for sale. The shopper tries to purchase valuable sellers' data from the cloud to improve its ML model. Rather than access the original sellers' data, the shopper retrieves auxiliary information (i.e., prediction values) from the cloud to evaluate sellers' data. We assume the cloud is *honest-but-curious* [8, 28, 33]. In other words, the cloud follows the predefined protocol faithfully, but it may exploit the shopper's model and sellers' data to make profits or derive some personal information.

We do not particularly consider the adversarial relationship between the shopper and sellers. Although model inversion attacks [12, 39] may facilitate the shopper to derive original data from prediction values, these attacks are difficult to be launched since they require enormous data to model the relationship between prediction values and the original data; however, the shopper who intend to purchase data often have a small amount of data. Additionally, we assume sellers desire to profit fully from their data. Therefore, they do not launch data poisoning attacks [35] since most of them often degrade the shopper's model performance, and thus they can be easily detected. In this paper, we only consider that sellers may unintentionally hold a small set of irrelevant or mislabeled data.

3.2 System Architecture

Figure 1 illustrates the architecture of Primal, a privacy-preserving and efficient ML data evaluation framework on a cloud-based data marketplace. Overall, Primal contains three components: (i) *a group of data sellers*, who encrypt their data and upload them to the cloud for sale; (ii) *a data shopper*, who encrypts its model parameters and uploads them to the cloud for evaluation; (iii) *a cloud server*, which can help the shopper evaluate sellers' encrypted data.

Primal is powered by three designs: an efficient machine learning encryption protocol, a data selection protocol, and a data validation protocol. Our encryption protocol applies lightweight inner product functional encryption (IFE) and matrix transformation to encrypt ML models and data. Additionally, it also allows the cloud to perform prediction or training operations with the encrypted model and data. By collecting prediction values from the cloud, the data selection protocol utilizes active learning to select potentially valuable data. As the selected data may contain irrelevant or falsely labeled data, the data validation protocol further examines data quality to screen out high-quality data.

Overall, Primal consists of five phases: *Setup*, *Data encryption*, *Model enryption*, *Data selection*, and *Data validation*, where the first three phases are included in our ML encryption protocol. Here, we go through the five phases to provide a brief view of our framework.

Setup. The data shopper chooses a security parameter and initializes a set of random numbers, a master public key, and a master secret key. Then, the master public key is broadcast to data sellers for data encryption, and the master secret key and random numbers are used to encrypt model parameters.

Data Encryption. Sellers utilize IFE to preserve both data privacy and data functionality in the cloud. Particularly, the functionality means that the cloud can perform training and prediction operations with the shopper's encrypted model and sellers' encrypted data. In training and prediction operations, there are two types of computation related to sellers' data: matrix and convolution computation. As the two types of computation can be transformed into inner product operations, it is natural for sellers to transform their data into vectors and encrypt them using IFE.

Model Encryption. There are mainly two types of computation in the first hidden layer of a seller's model, i.e., matrix or convolution computation. Note that the two types of computation can be converted to inner product operations. Therefore, the shopper can transform the parameters of the first hidden layer to vectors and utilize IFE to encrypt them. As a result, the privacy and functionality of the first hidden layer can be preserved. Nevertheless, IFE cannot be applied to encrypt an entire multi-layer model since it only supports simple inner product operations. Therefore, the shopper also utilizes matrix transformation to encrypt the parameters of the remaining layers.

Data Selection. For an ML model, valuable data contain much informativeness and can significantly improve model performance. Particularly, data informativeness can be revealed by the prediction

values of data. Therefore, it seems possible for the shopper to perform some prediction operations and then collect prediction values to estimate data informativeness. However, the shopper cannot access sellers' data before the final payment. Therefore, it relies on the cloud to perform prediction operations with its encrypted model and sellers' encrypted data. Then, it decrypts prediction values and adopts active learning to select informative data.

Data Validation. Although informative data may significantly improve model performance, they could contain irrelevant data or falsely labeled data, which may mislead the classification of models. Therefore, the shopper needs to examine the quality of the selected data. However, the shopper cannot directly access sellers' data to estimate data quality before the final payment. Fortunately, for a specific ML model, the prediction values of some data can reveal the quality of previously training data. Therefore, the shopper can inform the cloud to retrain its model with the selected data and then use the retrained model to output some encrypted prediction values. Next, the shopper retrieves these prediction values and decrypts them to estimate data quality.

4 EFFICIENT MACHINE LEARNING ENCRYPTION PROTOCOL

In this section, we first utilize inner product functional encryption (IFE) and matrix transformation to propose an efficient ML encryption protocol, providing encryption approaches for input data and the parameters of neural networks. Then, we illustrate how to encrypt a typical CNN model and input data using our ML encryption protocol. Although we only display the encryption approach for a CNN model, our ML encryption protocol can also be generalized to encrypt other neural network models.

4.1 Setup

Our ML encryption protocol provides a setup approach for a data shopper. It runs IFE's setup algorithm to initialize a master key pair (mpk, msk) and chooses a set of random numbers R from a finite field $\mathbb{Z}_q$. After initialization, the shopper sends mpk to data sellers for data encryption, and stores msk and R to encrypt model parameters. Particularly, msk is used to encrypt the parameters of the first hidden layer with IFE, and R is used to transform parameters. To be precise, $\vec{R_i} \in R$ is an n_i-dimension vector of random numbers used to transform the parameters of the i-th fully-connected hidden layer, and $\mathbf{R_i} \in R$ is an $m_i \times n_i$ matrix of random numbers used to transform the parameters of the i-th convolution hidden layer.

Algorithm 1: Data Encryption for Matrix Computation

Input: data $\vec{x}$, a master public key mpk;
Output: encrypted data $\vec{C_x}$;
$\vec{C_x} \rightarrow$IFE.encrypt($\vec{x}$, mpk);
return $\vec{C_x}$;

4.2 Data Encryption

Our ML encryption protocol allows data sellers to protect data and also retain data functionality in the cloud. Particularly, the functionality means that data can be used for training and prediction. In training and prediction, there are two types of computation related to input data, i.e., matrix and convolution computation. Therefore, we provide two data encryption approaches for matrix and convolution computation, respectively.

Algorithm 2: Data Encryption for Conv. Computation

Input: data $\mathbf{X}$ (a two-dimension matrix), a convolution kernel width K_w, a stride length s, a master public key mpk;
Output: encrypted data $\mathbf{C_X}$;
$t \leftarrow (\mathbf{X}.width - K_w + 1)/L_{str}$;
initiate a $(t \times t \times (K_w)^2)$ matrix $\mathbf{Ct}_X$;
for $i \in \{0, \cdots, t\}$ **do**
  for $j \in \{0, \cdots, t\}$ **do**
    $x_s \leftarrow i * s$, $y_s \leftarrow j * s$;
    convert $\mathbf{X}[x_s : x_s + K_w][y_s : y_s + K_w]$ to $\vec{X'}$;
    $\mathbf{C}_X[i][j][:] \leftarrow$ IFE.encrypt($\vec{X'}$, mpk);

Algorithm 1 shows the data encryption approach for matrix computation. As Figure 2 shows, the computation between input data $\vec{x}$ and a parameter matrix can be naturally divided to inner product operations between $\vec{x}$ and multiple vectors. Thus, we can directly use IFE to encrypt $\vec{x}$, which preserves both data privacy and its functionality of matrix computation. Algorithm 2 shows the data encryption approach for convolution computation. As Figure 4 shows, the convolution computation between two-dimension input data $\mathbf{X}$ and a kernel can be decomposed to inner product operations between a subarea of $\mathbf{X}$ and a (transformed) kernel. Therefore, we first split $\mathbf{X}$ to multiple subareas and transform them to a set of vectors $\{\vec{X'_{11}}, \vec{X'_{22}}, \cdots\}$. Then, we utilize IFE to encrypt $\{\vec{X'_{11}}, \vec{X'_{22}}, \cdots\}$. As a result, both data privacy and the functionality of convolution computation can be preserved.

4.3 Model Encryption

Our ML encryption protocol provides model encryption approaches that allow a data shopper to preserve the privacy and functionality of its model in the cloud. Note that neural network models mainly consist of fully-connected and convolution layers, and there are two different types of computation, i.e., matrix and convolution computation, in a fully-connected and a convolution layer. As the two types of computation can be decomposed into inner product operations, and IFE enables inner product operations over ciphertexts, we can utilize IFE to encrypt the parameters of a hidden layer.

Although applying IFE can protect the parameters of a hidden layer, it is not sufficiently secure since it reveals computation results in plaintexts. Additionally, since it only supports simple inner product operations, it can only be applied to encrypt the first hidden layers. Therefore, we also apply matrix transformation to encrypt entire model parameters. To support matrix transformation across layers, we set the activation function to the squared function. Prior work [10, 26] has proved squared activation functions are as expressive as common activation functions, e.g. Relu and Sigmoid.

Encrypting a Fully-connected Layer. Algorithm 3 shows the encryption approach for a fully-connected layer. If this fully-connected

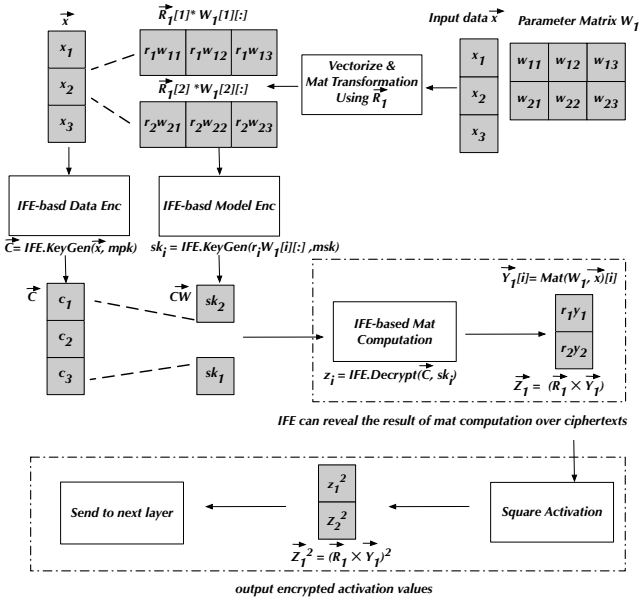


Figure 2: IFE-based Data Encryption, Model Encryption, and Matrix Computation in a Fully-connected Layer (First Hidden Layer).

Algorithm 3: Encrypting a Fully-connected Layer (the i-th Hidden Layer)

Input: a parameter matrix $\mathbf{W}_i$, a master secret key msk, a vector of random numbers $\vec{R}_i$ chosen for the i-th layer, and a vector of random numbers $\vec{R}_{i-1}$ chosen for the $(i-1)$-th layer (if $i > 0$) ;

Output: encrypted parameters $\mathbf{CW}_i$;

$\emptyset \rightarrow \mathbf{CW}_i$ or $\vec{CW}_i$;
if $i == 1$ **then**
  for $j \in \{0, \cdots, \mathbf{W}_i.row\_num\}$ **do**
    $\vec{CW}_i[j] \leftarrow$IFE.KeyGen($\vec{R}_1[j] * \mathbf{W}[j][:], msk$);
  return $\vec{CW}_i$;
else
  for $j \in \{0, \cdots, \mathbf{W}_i.row\_num\}$ **do**
    for $k \in \{0, \cdots, \mathbf{W}_i.col\_num\}$ **do**
      $\mathbf{CW}_i[j][k] \leftarrow \mathbf{W}_i[j][k] * \vec{R}_i[j]/(\vec{R}_{i-1}[k])^2$;
  return $\mathbf{CW}_i$;

layer is the first hidden layer, it is natural to convert a parameter matrix $\mathbf{W_1}$ to row vectors and apply IFE to encrypt them. However, only applying IFE to encrypt parameters is not sufficiently secure since IFE can reveal computation results in plaintexts. The original parameters and data may be derived from the plaintexts through matrix decomposition. Therefore, we also apply matrix transformation to convert $\mathbf{W_1}$.

As Figure 2 shows, we first decompose $\mathbf{W_1}$ to row vectors and apply matrix transformation to convert them. To be precise, we choose a vector of random numbers $\vec{R}_1 = \{r_1, r_2, \cdots\}$ and transform each row vector $\mathbf{W_1}[j][:]$ to a vector $\vec{R}_1[i]\mathbf{W_1}[j][:]$. Then, we use IFE to encrypt the transformed vectors. By combining IFE with matrix transformation, the result $\vec{Z}$ of matrix computation can be fully protected. Particularly, the result revealed by IFE is a random vector $\vec{Z}_1 = \vec{R}_1 \times \vec{Y}_1$. Therefore, attackers cannot learn the original computation result $\vec{Y}_1$ without knowing $\vec{R}_1$. By executing the square activation function with the result $\vec{Z}_1$, this layer will output random activation values $\vec{Z}^2 = \vec{R}_1^{\,2} \times \vec{Y}_1^{\,2}$ to the next layer.

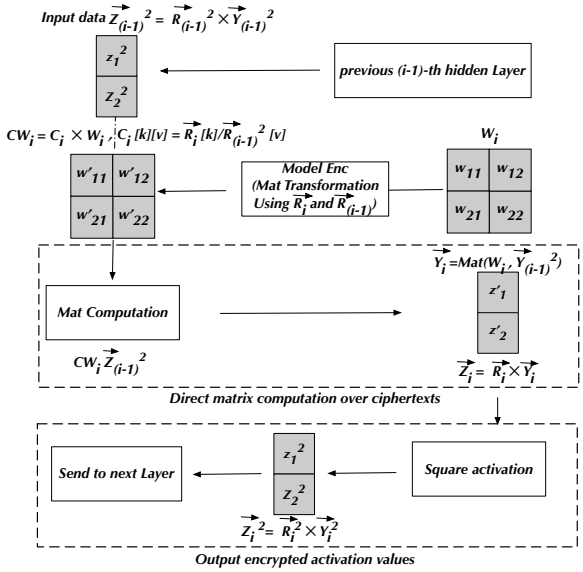


Figure 3: Model Encryption and Matrix Computation in a Fully-connected Layer (i-th Hidden Layer).

If a fully-connected layer is the i-th hidden layer ($i > 1$), we choose a vector of random numbers and utilize matrix transformation to protect the parameters of this layer. As Figure 3 shows, this layer receives the output (activation values) from the previous layer as input data. Particularly, the input data from the previous layer can be presented as $\vec{Z}^2_{i-1} = \vec{R}^2_{i-1} \times \vec{Y}^2_{i-1}$. For instance, the input from the first fully-connected hidden layer are $\vec{Z}^2_1 = \vec{R}^2_1 \times \vec{Y}^2_1$. As the original input $\vec{Y}^2_{i-1}$ is encrypted by a vector of random numbers $\vec{R}^2_{i-1}$, we cannot perform correct matrix computation between $\vec{Y}^2_{i-1}$ and the parameters of this layer. To guarantee correct matrix computation, we also need to eliminate the effect of random numbers $\vec{R}^2_{i-1}$ in matrix transformation.

As Figure 3 shows, we choose a vector of random numbers $\vec{R}_i$ and transforms the parameters $\mathbf{W_i}$ to $\mathbf{W'_i} = \mathbf{C_i} \times \mathbf{W_i}$, where $\mathbf{C_i}$ is a matrix whose element is $\vec{R}_i[k]/\vec{R}^2_{i-1}[v]$. As a result, each element $\mathbf{W_i}[k][v]$ of the parameters is randomized by a different random number $\vec{R}_i[k]/\vec{R}^2_{i-1}[v]$. Therefore, the privacy of parameters are fully preserved. Additionally, we multiply each row vector $\mathbf{W_i}[k][:]$ of parameters with a vector $1/\vec{R}^2_{i-1}$. This eliminates the effect of random numbers $\vec{R}^2_{i-1}$ from the input data in matrix computation, which guarantees the correct execution of matrix computation. In

this way, this layer can perform correct matrix computation between the transformed parameters and input data, and then output the result as $\vec{Z}_i = \vec{R}_i \times \vec{Y}_i$. As $\vec{R}_i$ is a random vector, the original result $\vec{Y}_i$ cannot be derived from $\vec{Z}_i$. By executing the square activation function with $\vec{Z}_i$, this layer will output random activation values $\vec{Z}_i^2 = \vec{R}_i^2 \times \vec{Y}_i^2$ to the next layer.

Encrypting a Convolution Layer. Algorithm 4 shows the encryption approach for a convolution layer. For simplicity, we only consider a convolution kernel in this layer. This algorithm can be generalized to support multiple kernels. If a convolution layer is the first hidden layer, we convert a kernel into vectors and apply IFE to encrypt kernel parameters. Particularly, as convolution computation in this layer can be decomposed to inner product operations, we can utilize IFE to retain the convolution functionality of this layer. However, only applying IFE to encrypt the kernel is not secure since IFE can reveal computation results in plaintexts. Therefore, we also apply matrix transformation to convert the kernel.

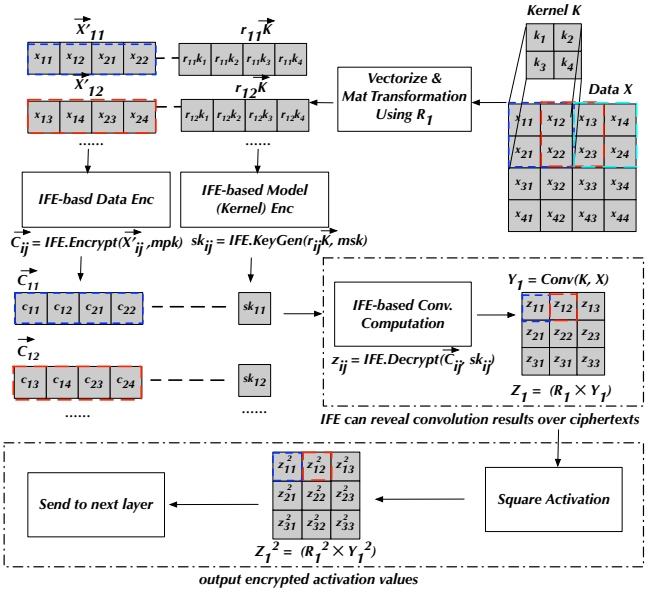


Figure 4: IFE-based Data Encryption, Model Encryption, and Convolution Computation in a Convolution Layer (First Hidden Layer). Here, the convolution stride is 1.

As Figure 4 shows, we first convert the parameters of a kernel to a vector $\vec{K}$ and apply matrix transformation to convert it. To be precise, we first choose a matrix of random numbers $\mathbf{R}_1$, whose each element is r_{ij}. Then, we transform $\vec{K}$ to a set of random vectors $\{r_{11}\vec{K}, r_{12}\vec{K}, \cdots\}$ and utilize IFE to encrypt these vectors. By combining IFE with matrix computation, the result $\mathbf{Z}$ of convolution computation can be fully protected. As Figure 4 shows, the result $\mathbf{Z_1}$ revealed by IFE is a random matrix $\mathbf{Z_1} = \mathbf{R_1} \times \mathbf{Y_1}$. Therefore, the original result $\mathbf{Y_1}$ cannot be derived without $\mathbf{R_1}$. By executing the square activation function with $\mathbf{Z_1}$, this layer will output random activation values $\mathbf{Z_1}^2 = \mathbf{R_1}^2 \times \mathbf{Y_1}^2$ to the next layer.

If a convolution layer is the i-th hidden layer ($i > 1$), we choose a matrix of random numbers and utilize matrix transformation

Algorithm 4: Encrypting a Conv. Layer (the i-th Hidden Layer)

Input: the width of input I_w, a kernel $\mathbf{K}_i$, a stride length s, a master secret key msk, a matrix of random numbers $\mathbf{R}_{i-1}$ chosen for the $(i-1)$-th layer (if $i > 1$), and $\mathbf{R}_i$ chosen for the $(i-1)$-th layer;

Output: encrypted parameters $\mathbf{CK}_i$;

1 $t \leftarrow (I_w - |\mathbf{K_i}| + 1)/s$;
2 **if** $i == 1$ **then**
3 initiate a $t \times t$ matrix $\mathbf{CK}_i$;
4 **for** $k \in \{0, \cdots, t\}$ **do**
5 **for** $v \in \{0, \cdots, t\}$ **do**
6 $\mathbf{CK}_i[k][v] \leftarrow$ IFE.KeyGen($\mathbf{R_i}[k][v] * \mathbf{K}_i, msk$);
7 **else**
8 initiate a $(t \times t \times |\mathbf{K}_i| \times |\mathbf{K}_i|)$matrix $\mathbf{CK}_i$;
9 **for** $k \in \{0, \cdots, t\}$ **do**
10 **for** $v \in \{0, \cdots, t\}$ **do**
11 $\mathbf{CK}_i[k][v][:][:] = \mathbf{R_i}[k][v]/\mathbf{R}^2_{i-1}[k*s : k*s+|\mathbf{K_i}|][v*s : v*s+|\mathbf{K_i}|] \times \mathbf{K_i}$;
12 **return** $\mathbf{CK}_i$;

to protect the kernel of this layer. As Figure 5 shows, this layer receives the output (activation values) from the previous layer as input data. Particularly, the input data from the previous layer can be presented as $\mathbf{Z_i}^2 = \mathbf{R_{i-1}}^2 \times \mathbf{Y_{i-1}}^2$. For instance, the input from the first convolution hidden layer is $\mathbf{Z_1}^2 = \mathbf{R_1}^2 \times \mathbf{Y_1}^2$. As $\mathbf{Z_{i-1}}^2$ is encrypted by a matrix of random numbers $\mathbf{R_{i-1}}^2$, we cannot directly perform convolution computation between $\mathbf{Z_{i-1}}^2$ and the (transformed) kernel. To guarantee correct convolution functionality, we also need to eliminate the effect of random numbers $\mathbf{R_{i-1}}$ in matrix transformation.

As Figure 5 shows, we choose a matrix of random number $\mathbf{R_i}$ and transform the kernel $\mathbf{K_i}$ to a set of random kernel matrices $\{\mathbf{CK_{00}}, \mathbf{CK_{01}}, \cdots\}$, according to the convolution stride length s and kernel width $|\mathbf{K_i}|$. As line 11 of Algorithm 4 shows, each matrix $\mathbf{CK_{kv}}$ is randomized by different random matrices. Thus, the privacy of the kernel is fully preserved. Furthermore, matrix transformation eliminates the effect of random numbers in the subarea and guarantees the convolution functionality of this layer. Specifically, the input data $\mathbf{Z_{i-1}}^2$ used in the $(k \times v)$-th stride of convolution is multiplied by a random matrix $\mathbf{R}^2_{i-1}[k*s : k*s+|\mathbf{K_i}|][v*s : v*s+|\mathbf{K_i}|]$. As the corresponding kernel matrix $\mathbf{CK_{kv}}$ is divided by $\mathbf{R}^2_{i-1}[k*s : k*s+|\mathbf{K_i}|][v*s : v*s+|\mathbf{K_i}|]$, the effect of random numbers are eliminated and thus the inner product operation between the subarea and $\mathbf{CK_{kv}}$ can be executed correctly.

In this way, after performing a series of inner product operations between multiple subareas and $\{\mathbf{CK_{00}}, \mathbf{CK_{01}}, \cdots\}$, we can obtain the result of entire convolution computation $\mathbf{Z_i} = \mathbf{R_i} \times \mathbf{Y_i}$. As $\mathbf{R_i}$ is a random matrix, the original convolution result $\mathbf{Y_i}$ cannot be derived. After running the square activation function, this layer will output random activation values $\mathbf{Z_i}^2 = \mathbf{R}_i^2 \times \mathbf{Y}_i^2$.

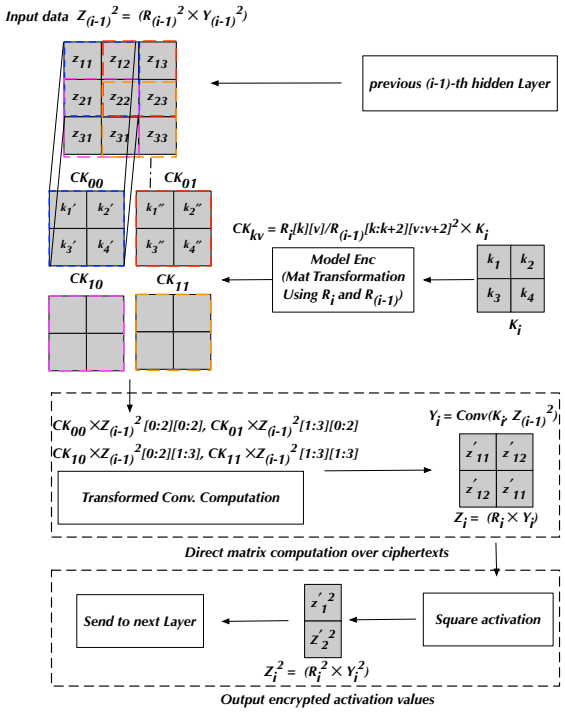


Figure 5: Model Encryption and Convolution Computation in a Convolution Layer (i-th Hidden Layer). Here, the convolution stride is 1.

4.4 Case Study: Encrypting a CNN Model and Data

A typical CNN model, e.g., LeNet-5 [22], consists of convolution layers, pooling layers, and followed by fully-connected layers. We do not provide a special encryption approach for pooling layers in our ML encryption protocol since they are essentially convolution layers. Here, we show how to encrypt a CNN model and input data using our ML encryption protocol.

Model and Data Encryption. In CNN tasks, data are first input into a convolution layer. Therefore, we utilize Algorithm 2 to encrypt data, which retains the functionality of convolution in ciphertexts. For a typical CNN model, we firstly utilize Algorithm 4 to encrypt the parameters of convolution layers and then utilize Algorithm 3 to encrypt the parameters of fully-connected layers. We notice that only the parameters of the first hidden layer are encrypted by the IFE cryptographic scheme. Therefore, the encryption time and execution time of a privacy-preserving CNN model are determined by the first hidden layer.

Packing Optimization. When using a typical CNN model to predict images, adjacent image subareas are often mapped to an element of feature map (intermediate output) by convolution and pooling layers. Therefore, we can pack these subareas into a ciphertext to reduce the encryption time and execution time. Here, we consider a typical pooling layer, i.e., mean pooling layer. Let $\mathbf{F}_{poo}$ be the output feature map of this pooling layer, $\mathbf{Z}$ be the output feature map of the previous convolution layer, w_p be the pooling size. The feature map $\mathbf{F}_{poo}$ output by convolution and pooling layers can be represented as follows:

$$\mathbf{F}_{poo}[i][j] = \sum_{x=i*w_p, y=j*w_p}^{x=i*w_p+w_p, y=j*w_p+w_p} \mathbf{Z}[x][y]/w_p^2, \tag{1}$$

Let $\mathbf{K}$ be the convolution kernel and $\mathbf{X}$ be the convoluted data. Considering $\mathbf{Z}[x][y] = \mathbf{K} \times \mathbf{X}_{x,y}$, where $\mathbf{X}_{x,y} = [x*s : x*s+w_k][y*s : y*s+w_k]$, we can extend Equation 1 as follows:

$$\begin{aligned} \mathbf{F}_{poo}[i][j] &= \sum_{x=i*w_p, y=j*w_p}^{x=i*w_p+w_p, y=j*w_p+w_p} (\mathbf{K} \times \mathbf{X}_{x,y})/w_p^2 \\ &= (\sum_{x=i*w_p, y=j*w_p}^{x=i*w_p+w_p, y=j*w_p+w_p} \mathbf{X}_{x,y}/w_p^2) \times \mathbf{K} \\ &= AVG(X_{x,y}) \times \mathbf{K} \end{aligned} \tag{2}$$

According to Equation 2, we can pack multiple image subareas $X_{x,y}$ using $AVG(X_{x,y})$. As a result, the convolution and pooling computation can be executed in one step. Therefore, the image size, the execution time, and the encryption time of the CNN model can be reduced w_p^2 times.

5 PRIVACY-PRESERVING DATA SELECTION

We first present a privacy-preserving data prediction approach that allows the cloud to perform prediction operations with a shopper's encrypted model and sellers' encrypted data. Based on this prediction approach, we offer a privacy-preserving data selection protocol.

5.1 Privacy-preserving Data Prediction

Algorithm 5: Feed Forward in a Fully-connected Layer (the i-th Hidden Layer)

Input: encrypted parameters $\mathbf{CW}_i$ or $\vec{CW}_i$, the encrypted input from the $(i-1)$-th layer (if $i > 1$, the input is $\vec{Z}_{i-1}^2$; else if $i == 1$, the input is $\vec{C}_x$);

Output: the output $\vec{Z}_i$ of the i-th layer;

```
if i == 1 then
    for j ∈ {0, ··· , |CW_i|} do
        Z_i ← IFE.Decryt(Z_{i-1}^2, CW_i[j]);
else
    Z_i ← CW_i Z_{i-1}^2;
return (Z_i)^2;
```

Fully-connected Layers. Algorithm 5 shows the feed forward process in fully-connected layers. If a fully-connected layer is the first hidden layer, it applies IFE's functional decryption to perform matrix computation and output $\vec{Z}_1 = \vec{R}_1 \times \vec{Y}_1$, where $\vec{Y}_1$ is the original result and $\vec{R}_1$ is a random number vector. If a fully-connected layer is the i-th hidden layer ($i > 1$), it directly performs matrix computation as follows.

Algorithm 6: Feed forward in a Convolution Layer (the i-th Hidden Layer)

Input: encrypted kernel parameters $\mathbf{CK}_i$, the encrypted input from the $(i-1)$-th layer (if $i > 1$, the input is $\mathbf{Z}^2_{i-1}$; else if $i == 1$, the input is $\mathbf{C_X}$), the kernel width K_w and stride length s;
Output: the output $\mathbf{Z}^2_i$ of the i-th layer;

```
if i == 1 then
    for k, v ∈ {0, ··· , |CK_i|} do
        Z_i[k][v] ← IFE.Decryt(C_X[k][v][:], CK_i[k][v]);
else
    for k, v ∈ {0, ··· , |CK_i|} do
        Z_i = Z^2_{i-1}[k * s : k * s + K_w][v * s :
          v * s + K_w] × CK_i[k][v][:][:];
return (Z_i)^2;
```

$$\begin{aligned}\vec{Z}_i &= \mathbf{CW}_i\vec{Z}^2_{i-1} = (\mathbf{C}_i \times \mathbf{W}_i)(\vec{R}^2_{i-1} \times \vec{Z}^2_{i-1}) \\ &= (\mathbf{C_i}\vec{R}^2_{i-1}) \times (\mathbf{W_i}\vec{Z}^2_{i-1}) \\ &= \vec{R}_i \times \vec{Y}_i\end{aligned} \tag{3}$$

where $\vec{Y}_i$ is the original result and $\vec{R}_i$ is a random vector. After running the square activation function, this layer outputs random activation values $\vec{Z}^2_i = \vec{R}^2_i \times \vec{Y}^2_i$, where $\vec{Y}^2_i$ is the original output of the i-th layer.

Convolution Layers. Algorithm 6 shows the feed forward process in convolution layers. If a convolution layer is the first hidden layer, it transforms convolution computation to inner product operations between the encrypted kernel parameters and subareas of input data, and then utilizes IFE's functional decryption to reveal the convolution result $\mathbf{Z_1} = \mathbf{R_1} \times \mathbf{Y_1}$, where $\mathbf{R_1}$ is a random number matrix, and $\mathbf{Y_1}$ is the original result. If a convolution layer is the i-th hidden layer ($i > 1$), it also converts convolution computation to inner product operations between the subareas $\mathbf{Z}^2_{i-1,kv}$ of input data and random kernel matrices $\mathbf{CK}_i = \{\mathbf{CK}_{11}, \cdots\}$. Let K_w be the kernel width and s be the stride length. The $(k*v)$-th inner product operation is as follows.

$$\begin{aligned}\mathbf{Z}_i[k][v] =& \mathbf{CK}_i[k][v][:][:] \cdot \mathbf{Z}^2_{i-1,kv} \\ =& \mathbf{R}_i[k][v](\mathbf{K} \cdot \mathbf{Z}^2_{i-1,kv}) \\ =& \mathbf{R}_i[k][v]\mathbf{Y}_i[k][v],\end{aligned} \tag{4}$$

where $\mathbf{Z}^2_{i-1,kv} = \mathbf{Z}^2_{i-1,kv}[s*k : s*k+K_w][s*v : s*v+K_w]$. By performing multiple inner product productions, this layer outputs the convolution result $\mathbf{Z}_i = \mathbf{R}_i \times \mathbf{Y}_i$, where $\mathbf{R}_i$ is a random matrix and $\mathbf{Y}_i$ is the original result. Finally, this layer runs the square activation function to output random activation values $\mathbf{Z}^2_i = \mathbf{R}^2_i \times \mathbf{Y}^2_i$.

5.2 Data Selection

As valuable data contain much informativeness and can significantly improve model performance, a shopper can evaluate data informativeness to screen out potentially valuable data. Note that data informativeness for a specific model can be revealed by prediction performance. Therefore, the shopper can collect the prediction values of sellers' data to estimate data informativeness. As the shopper cannot access sellers' data before the final payment, it relies on the cloud to output the prediction values of sellers' data. To protect model privacy and data privacy as well as enabling prediction operations in the cloud, we leverage privacy-preserving data prediction to propose a data selection protocol. It works as follows.

Step 1: prediction operation. First, this protocol requires the shopper and sellers to upload their encrypted model and data into the cloud. Second, the cloud uses the shopper's encrypted model to predict sellers' encrypted data (see Section 5.1). It then sends encrypted prediction values back to the shopper.

Step 2: data section based on prediction values. In a prediction operation, as aforementioned, each fully-connected layer outputs a random vector $\vec{Z}_i{}^2 = \vec{R}_i{}^2 \times \vec{Y}_i{}^2$, and each convolution layer outputs a random matrix $\mathbf{Z}^2_i = \mathbf{R_i}^2 \times \mathbf{Y_i}^2$. Therefore, if the prediction values $\vec{Z}_n{}^2$ are from a fully-connected layer, the shopper decrypts them by multiplying with $1/\vec{R}_i{}^2$. If the prediction values $\mathbf{Z_n}^2$ are from a convolution layer, the shopper decrypts them by multiplying with $1/\mathbf{R_i}^2$. After obtaining the original prediction values, the shopper then applies active learning to evaluate data informativeness.

We choose an uncertainty sampling algorithm as our active learning algorithm. Most existing uncertainty sampling algorithms [37] use the entropy score of prediction values to select informative data for binary classification; however, the entropy score cannot accurately reflect the uncertainty degree for multi-class learning since the score is always affected by unimportant classes. Hence, we adopt Best-vs-Second-Best (BvSB) selection algorithm [20] that considers the difference between the best and the second guess.

Avoiding selecting duplicates or near-duplicates that seem to be valuable, our data selection protocol does not choose a large dataset at one time. Instead, it selects valuable data in multiple steps. In each step, the shopper selects and purchases a small set of samples to train their ML models. As a result, the features of these samples will be learned by the shopper's model, and thus the informativeness of the corrsponding duplicates and near-duplicates will decrease. Subsequently, the duplicates and near-duplicates are not likely to be evaluated as valuable.

6 PRIVACY-PRESERVING DATA VALIDATION

We first present a privacy-preserving data training approach that allows the cloud to perform training operations with a shopper's encrypted model and sellers' encrypted data. Based on this training approach, we then offer a privacy-preserving data validation protocol.

6.1 Privacy-preserving Model Training

There are multiple epochs in model training, each of which consists of a feed forward process and a back propagation process. We only present back propagation here since we have described feed forward in Section 5.1. Considering the feature of our Primal framework, back propagation is designed to the multi-party computation involving an untrusted cloud, a data shopper (model owner), and multiple data sellers (data owners).

In a training epoch, the cloud first performs a feed forward process and sends encrypted prediction values to the shopper. Then, the shopper decrypts them and executes a cost function to output the prediction cost between the original prediction values $\vec{Z}_n^2$ and labels $\vec{L}$. Here, for simplicity but without loss of generality, we assume the cost function is the mean square error function. Therefore, the gradients of the output layer can be presented as $\delta = \vec{Z}_n^2 - \vec{L}$. For back propagation, the shopper needs to send the gradients to the cloud. As the cloud may infer $\vec{Z_n}^2$ from the gradients, the shopper chooses a random vector $\vec{R}_c = \{r_1, r_2, \cdots\}$ to randomize the gradients as $\delta' = (\vec{R}_c \times (\vec{Z}_n^2 - \vec{L}))$ and then sends them to the cloud. Subsequently, the cloud can compute the gradients of each hidden layer and update all parameters. Since the parameters of the first hidden layer and back layers are encrypted by two methods, respectively, the cloud updates parameters in two forms.

Back Hidden layers. For the i-th hidden layer ($i > 1$), the cloud can compute the following gradients according to the chain rule.

$$\frac{\partial \delta'}{\partial \mathbf{CW_i}} = \mathbf{T}_G^i \times \frac{\partial \delta}{\partial \mathbf{W_i}}, \tag{5}$$

where $\mathbf{T_G^i}$ is a random matrix, which can be presented as follows.

$$\mathbf{T}_G^i[k][v] = (\vec{R}_c \times \vec{R}_n) * \vec{R}_{i-1}^2[v]/\vec{R}_i[k], \tag{6}$$

where $\vec{R}_i$ is a random vector chosen for the i-th hidden layer, and $\vec{R}_n$ is a random vector chosen for the last hidden layer.

Recall that the parameters of the i-th hidden layer ($i > 1$) are $\mathbf{CW}_i = \mathbf{C}_i \times \mathbf{W}_i$, where $\mathbf{C}_i[k][v] = \vec{R}_i[k]/\vec{R}_{i-1}^2[v]$. Accordingly, the shopper can generate an update matrix $\mathbf{UP_i} = \mathbf{C_i}/\mathbf{T_G^i}$ and send it to the cloud. Then, the cloud can update parameters as follows.

$$\begin{aligned} \mathbf{CW_i}' &= \mathbf{CW_i} - \alpha * \mathbf{UP_i} \times \frac{\partial \delta'}{\partial \mathbf{CW_i}} \\ &= \mathbf{C_i} \times (\mathbf{W_i} - \alpha * \frac{\partial \delta}{\partial \mathbf{W_i}}), \end{aligned} \tag{7}$$

where α is the learning rate. In this way, the encrypted parameters are homomorphically updated to the new parameters.

First Hidden Layer. As input data and the parameters of the first hidden layer are encrypted by IFE, the cloud cannot directly compute the corresponding gradients. Instead, the cloud sends the gradients $\frac{\partial \delta'}{\partial \vec{Z}_1}$ to the seller C_x who owns the training data $\vec{x}$. Then, C_x can compute the following gradients.

$$\frac{\partial \delta'}{\partial \mathbf{CW_1}} = (\vec{x}\frac{\partial \delta'}{\partial \vec{Z}_1})^T = \mathbf{T_G^1} \times \frac{\partial \delta}{\partial \mathbf{W_1}}. \tag{8}$$

As the original parameters $\mathbf{W_1}$ of the first hidden layer are transformed to $\mathbf{CW}_1 = \mathbf{C}_1 \times \mathbf{W}_1$, the shopper should generate an update parameter matrix $\mathbf{UP_1} = \mathbf{C}_1/\mathbf{T}_G^1$ to homomorphically update the parameters. To be precise, the shopper send it to C_x, and C_x can generate the following gradients to update $\mathbf{CW}_1$.

$$\nabla \mathbf{CW_1} = \mathbf{UP_1} \times \frac{\partial \delta'}{\partial \mathbf{CW_1}} = \mathbf{C}_1 \times \frac{\partial \delta}{\partial \mathbf{W_1}}. \tag{9}$$

Note that the parameters $\mathbf{CW_1}$ are encrypted by IFE. It seems that the gradients $\nabla \mathbf{CW_1}$ cannot be directly used to update $\mathbf{CW_1}$. However, we notice that training operations do not need to modify $\mathbf{CW_1}$, provided that the output $\vec{Z}_1^2$ of the first hidden layer is correctly updated. Therefore, C_x shares $\nabla \mathbf{CW_1}$ with other sellers. Then, all sellers can compute the following output gradients with data $\vec{x}'$.

$$\nabla \vec{Z}_1^2 = (\alpha * \nabla \mathbf{CW}_1 \vec{x'})^2, \tag{10}$$

where α is the learning rate. With $\nabla \vec{Z}_1^2$, the cloud can update the output $\vec{Z}_1^2$ and start a new feed forward process.

6.2 Data Validation

As informative data may contain irrelevant or falsely labeled data that degrade a shopper's model performance, we design a data validation protocol to help the shopper validate the quality of the selected data. The key observation behind our data validation protocol is that the prediction performance of a model reveals the quality of previously training data. Therefore, the shopper requires the cloud to retrain its model with the selected data. Then, it can observe the prediction performance of the retrained model to estimate the quality of the selected data.

The prediction performance of a specific model can be revealed by prediction values. Thus, the shopper first requires the cloud to choose some sellers' data and use its retrained model to output the prediction values of these data. To not bias prediction values, the sellers' data should be chosen uniformly in this process. Then, the shopper collects and decrypts prediction values to estimate data quality. Particularly, the shopper can set a variable threshold T_q to screen out the data of different qualities. If the prediction errors are lower than T_q, the shopper will send the corresponding data IDs to the cloud and make a payment. To gain high validation accuracy, the shopper can perform fine-grained validation operations. Namely, the shopper can split validation data into smaller subsets and validate the subsets one by one.

7 SECURITY ANALYSIS

As our ML encryption protocol underlies the security of our framework Primal, we conduct a security analysis towards our ML encryption protocol to demonstrate the security of Primal. Here, we utilize the universal composition (UC) security framework [6] to define the security of our ML encryption protocol. In the security framework, there exists an environment $\mathcal{Z}$, which generates the input to all parties, reads all outputs, and in addition interacts with an adversary in an arbitrary way throughout the computation. The UC security is captured by the real world versus ideal world game. In the real world, we consider an adversary $\mathcal{A}$ who interacts with the real protocol Π_P. In the ideal world, we consider an adversary S (called a simulator) who runs the dummy protocol in the presence of the ideal functionality $\mathcal{F}_P$ (see Figure 6).

Definition 1 (ML Encryption Security). *We can say that our ML encryption protocol securely realizes a given functionality F_P if for any real-world $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that*

$$|Pr(Real_k(\mathcal{Z}, \mathcal{A}, \Pi_P)) - Pr(Ideal_k(\mathcal{Z}, \mathcal{S}, \mathcal{F}_P))| \leq negl(k) \tag{11}$$

Now, we give the following security theorem. (security proof can be found in Appendix A)

Theorem 1. *Our ML encryption protocol securely realizes a given functionality F_P if the random numbers chosen for each layer are pseudo-random, and the inner-product functional encryption scheme IFE is secure.*

Parameters: a cloud P, a shopper S, and sellers $C_1, \cdots, C_m$.
Setup: On input a security parameter from S, outputs mpk to $C_1, \cdots, C_m$, and store msk and random numbers internally.
Data encryption: On input data $\vec{x}$ or $\mathbf{X}$ from C_i, outputs $\vec{Cx}$ or $\mathbf{C_X}$ to P.
Model Encryption for a Layer: On input a parameter matrix $\mathbf{W}_i$ or a kernel $\mathbf{K}_i$ from S, outputs $\mathbf{CW}_i$ or $\mathbf{CK}_i$ to P.

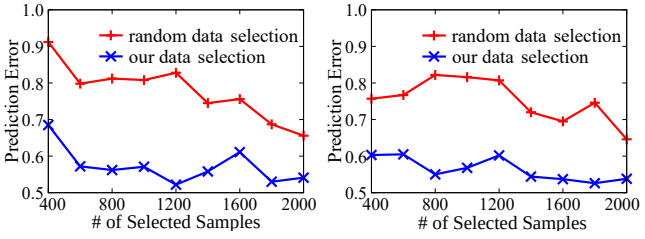
Figure 6: Ideal Function $\mathcal{F}_P$

8 SYSTEM EVALUATION

We implement the prototype of Primal, where IFE is implemented by a simple IFE scheme [1], and the matrix transformation mechanism is implemented by GMP [24], PBC [25], and Tensorflow [11]. We conduct experiments to evaluate the benefits of our data selection protocol, the accuracy of our data validation protocol, and the system overhead. Our experiments are performed on a PC containing four Intel Core-i5 2.3 GHz processors. We consider 100 sellers and one shopper to simulate a real-world data marketplace. Particularly, we randomly divide the training samples of MNIST [30] into 100 subsets and distribute them to each seller.

8.1 Benefits of Data Selection

To demonstrate Primal can provide valuable data to significantly improve model performance, we compare our data selection with random data selection. Particularly, we utilize the two methods to select data and then use the data to retrain two models of different scales. To be precise, the two models are trained with 5500 samples 55000 samples, respectively. Figure 7 shows the performance of the two models after retraining. We can see our data selection can reduce about 60% prediction errors compared to random selection when the number of selected samples is between 400 and 2000.



(a) Model 1 (b) Model 2

Figure 7: The benefit of data selection. Here, Model 1 and 2 are trained with 5500 and 55000 samples, respectively.

8.2 Accuracy of Data Validation

In this experiment, we use our data validation protocol to detect some low-quality samples. To better demonstrate the effectiveness of this protocol, we simulate specific low-quality samples that are most likely to evade this protocol. Recall that our validation protocol can easily detect the low-quality samples that deviate from the original distribution since these samples degrade the shopper's model performance. Therefore, our simulation policy should replace each sample's original label l_i with another label that does not deviate far away from the shopper's model distribution. Thus, the replacement label should be the label l_h or the label l_s, on which the shopper's model has the highest or second-highest prediction confidence. Specifically, if the original label l_i is l_h, we replace it with l_s. Otherwise, we replace it with l_h.

To gain high detection accuracy, we do not perform our validation protocol on whole low-quality samples at one time. Instead, we split these low-quality samples into subsets and then apply the validation protocol to detect each subset one by one. Figure 8 shows the accuracy of two different validation strategies, which are conducted at 10 granularity and 50 granularity. Here, granularity refers to the size of split subsets. Overall, the accuracy of the two validation strategies exceeds 80%. Particularly, the validation strategy of 10 granularity achieves much higher accuracy than the validation strategy of 50 granularity. We also notice that the strategy of smaller granularity consumes more computation sources since it needs to perform more validation operations. Therefore, the shopper should make a trade-off between security and efficiency.

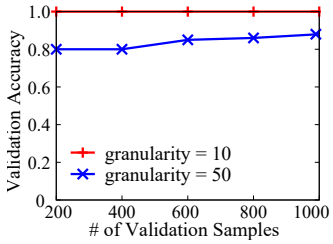
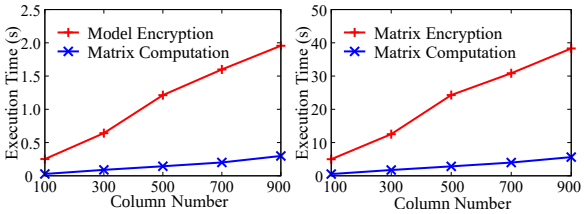


Figure 8: Accuracy of Data Validation

8.3 System Overhead

Here, we mainly measure the overhead of our ML encryption protocol since it determines system overhead, and extra overhead can be ignored. First, we show the overhead of IFE computation, which is the key component of our ML encryption protocol. Second, we apply this protocol to encrypt a specific CNN model and measure the execution time of relevant operations to demonstrate the efficiency of our system.



(a) Row Number=1 (b) Row Number=20

Figure 9: IFE-based Model Encryption and Matrix Computation

IFE-based Encryption and Computation. Figure 9 shows the execution time of IFE-based model encryption and matrix computation. Since convolution computation can be transformed to matrix computation, we only measure the time of matrix computation. We can see the execution time of model encryption and matrix computation is proportional to row and column numbers. Particularly, the execution time can be significantly reduced by GPU computing since a matrix can be divided into parallel vectors.

A Privacy-preserving CNN model. To demonstrate the efficiency of our ML encryption protocol, we compare it with E2DM [19], one of the most efficient ML homomorphic encryption approaches. We utilize our ML encryption protocol and E2DM to encrypt two CNN models with the same typology, respectively. The model typology consists of six layers: (i) input layer: a 28×28 input matrix; (ii) convolution layer: four 5×5 convolution kernels with the stride of 2; (iii) pooling layer: four 2×2 convolution kernels with the stride of 2; (iv) convolution layer: four 2×2 convolution kernels with the stride of 1; (v) fully-connected layer (FC): a 100×10 parameter matrix; (vi) output layer: a 10×1 prediction vector. Table 1 shows the execution time of the operations related to the two models. Our ML encryption protocol provides 17× faster feed forward than E2DM. As E2DM does not support back propagation, we only measure the back propagation for our ML encryption protocol. We notice that back propagation is much faster than feed forward since it does not involve any cryptographic computation (see Sec 6.1).

Table 1: Execution Time of CNN models

Operations	Execution Time (second)	
	E2DM	Ours
Data Encryption	0.40	0.48
Model Encryption	0.14	0.20
Feed Forward	35.88	2.59
Back Propagation	N/A	0.05

Specifically, the overheads of model and data encryption are the shopper-side and seller-side overhead, respectively. The overhead of feed forward is essentially the cloud-side prediction overhead in the process of data selection. Additionally, the total overhead of feed forward and back propagation represents the overhead of a cloud-side training epoch in the process of data validation.

9 RELATED WORK

Data Marketplaces. A cloud-based data marketplace [3, 9, 18, 21, 23] offers a platform where enterprises or individuals can sell their digital asset or purchase a dataset. Researchers have proposed a number of cloud-based data marketplace frameworks that provide flexible data search and evaluation. For instance, Koutris et al. [21] propose a back-end for data market, providing various query operations. Li et al. [23] present a correlation-driven data marketplace where shoppers can find the most correlated data for correlation analysis. Although these data marketplaces support shoppers to purchase expected data for a variety of interests, they fail to consider data privacy in the cloud. Hynes et al. [17] envision the first privacy-preserving data marketplace that leverages the Trusted Execution Environment (TEE) to protect data. However, some sensitive information may still be inferred from TEE [4, 34, 40], and TEE has some memory limits. Additionally, none of the existing data marketplaces support valuable data evaluation for ML tasks.

Machine Learning Encryption. Prior work has proposed a number of ML encryption frameworks [7, 14, 16, 19, 27], which are built on either homomorphic encryption (HE) or secure multiple-party computation (MPC). Although SIMD [38] can be applied to optimize HE-based ML frameworks, they still suffer from computational resource limitations, and the size of ciphertexts explosively grows with calculation numbers. Moreover, existing HE-based ML frameworks do not support flexible training operations. Therefore, data shoppers cannot utilize training operations to examine data quality. Compared to HE-based ML frameworks, MPC-based ML frameworks support training operations, and they are more efficient. However, they incur expensive communication overhead.

10 DISCUSSION

Generalizability. Our machine learning encryption protocol provides the encryption approaches for fully-connected/convolution layers and the corresponding input data. As pooling is essentially convolution, this encryption protocol can also protect pooling layers. Therefore, our Primal framework can protect various shopper's models in the process of data selection and validation, provided that the models consist of convolution, fully-connected, and pooling layers. Amongst these models, the representative models are CNN models, which have been widely adopted in various areas, especially computer vision.

11 CONCLUSION

In this paper, we propose a privacy-preserving and efficient ML data evaluation framework on a data marketplace, called Primal. It allows enterprises and individuals to sell their data and purchase valuable ML data without leaking their data and ML models. To preserve data privacy and model privacy in the cloud, we present a privacy-preserving ML protocol based on inner product functional encryption and matrix transformation. With this protocol, we design a privacy-preserving data selection protocol and data validation protocol that can provide valuable data for shoppers. Our security analysis and experiments demonstrate the security, efficiency, and effectiveness of Primal.

ACKNOWLEDGEMENT

This work was supported in part by the Office of Naval Research grants N00014-16-1-3214 and N00014-18-2893; in party by the National Science Foundation grant CNS-1815650; in part by NSFC under Grant 62132011; in part by BNRist under Grant BNR2020RC01013; in part by NSFC under grant 61825204; in part by Beijing Outstanding Young Scientist Program under grant BJJWZYJH012019100030011; and in part by the Shuimu Tsinghua Scholar Program. Qi Li and Jiahao Cao are the corresponding authors.

REFERENCES

[1] Abdalla, Michel et al. 2015. Simple functional encryption schemes for inner products. In *IACR International Workshop on Public Key Cryptography*. Springer, 733–751.
[2] Agrawal, Shashank et al. 2015. On the practical security of inner product functional encryption. In *IACR International Workshop on Public Key Cryptography*. Springer, 777–798.
[3] Bdex. 2018. First-ever Data Exchange Platform. https://www.bdex.com/
[4] Brasser, Ferdinand et al. 2017. Software Grand Exposure:SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies*.
[5] Campbell, Colin et al. 2000. Query learning with large margin classifiers. In *ICML*, Vol. 20. 0.
[6] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*. IEEE, 136–145.
[7] Chabanne, Hervé et al. 2017. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive* 2017 (2017), 35.

[8] David Cash et al. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 668–679.

[9] Dawex. 2019. Orchestrate data circulation with Data Exchange technology. https://www.dawex.com/

[10] Du, Simon S et al. 2018. On the power of over-parametrization in neural networks with quadratic activation. *arXiv preprint arXiv:1803.01206* (2018).

[11] An end-to-end open source machine learning platform. 2019. https://www.tensorflow.org/

[12] Fredrikson, Matt et al. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1322–1333.

[13] Freund, Yoav et al. 1997. Selective sampling using the query by committee algorithm. *Machine learning* 28, 2-3 (1997), 133–168.

[14] Gilad-Bachrach, Ran et al. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.

[15] Yifan Gong. 1995. Speech recognition in noisy environments: A survey. *Speech communication* 16, 3 (1995), 261–291.

[16] Graepel, Thore et al. 2012. ML confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*. Springer.

[17] Hynes, Nick et al. 2018. A demonstration of sterling: a privacy-preserving data marketplace. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2086–2089.

[18] IOTA. 2018. making it possible to securely store and sell. https://data.iota.org/

[19] Jiang, Xiaoqian et al. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1209–1222.

[20] Joshi, Ajay J et al. 2009. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2372–2379.

[21] Koutris, Paraschos et al. 2013. Toward practical query pricing with QueryMarket. In *proceedings of the 2013 ACM SIGMOD international conference on management of data*. ACM, 613–624.

[22] LeCun, Yann et al. 1998. Gradient-based learning applied to document recognition. 86, 11 (1998), 2278–2324.

[23] Li, Yanying et al. 2018. Cost-efficient data acquisition on online data marketplaces for correlation analysis. *Proceedings of the VLDB Endowment* 12, 4 (2018), 362–375.

[24] The GNU Multiple Precision Arithmetic Library. 2018. https://gmplib.org/

[25] The Pairing-Based Cryptography Library. 2018. https://crypto.stanford.edu/pbc/

[26] Livni, Roi et al. 2014. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*. 855–863.

[27] Mohassel, Payman et al. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.

[28] Naveed, Muhammad et al. 2014. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 639–654.

[29] Nenkova, Ani et al. 2012. A survey of text summarization techniques. In *Mining text data*. Springer, 43–76.

[30] THE MNIST DATABASE of handwritten digits. 2013. http://yann.lecun.com/exdb/mnist/

[31] Riazi, M Sadegh and Samragh, Mohammad et al. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. *IACR Cryptology ePrint Archive* 2019 (2019), 171.

[32] Rouhani, Bita Darvish et al. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*. ACM.

[33] SABRINA DE CAPITANI DI VIMERCATI et al. 2010. Encryption Policies for Regulating Access to Outsourced Data. *ACM Transactions on Database Systems* 35, 2 (2010), P.12.1–12.46.

[34] Schwarz, Michael et al. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[35] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom Goldstein. 2021. Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks. In *International Conference on Machine Learning*. PMLR, 9389–9398.

[36] Sebe, Nicu et al. 2005. *Machine learning in computer vision*. Vol. 29. Springer Science & Business Media.

[37] Burr Settles. 2009. *Active learning literature survey.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

[38] Smart, Nigel P et al. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* 71, 1 (2014), 57–81.

[39] Xi Wu, Matthew Fredrikson, Somesh Jha, and Jeffrey F Naughton. 2016. A methodology for formalizing model-inversion attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, 355–370.

[40] Xu, Yuanzhong et al. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.

[41] Zheng, Zibin et al. 2013. Service-generated big data and big data-as-a-service: an overview. In *2013 IEEE international congress on Big Data*. IEEE, 403–410.

A SECURITY PROOF

We repeat the security theorem for our efficient machine learning encryption protocol here and provide a proof sketch.

Theorem 1. *Our ML encryption protocol securely realizes a given functionality F_P if the random numbers chosen for each layer are pseudo-random, and the inner-product functional encryption scheme IFE is secure.*

Proof 1. According to Definition 1, the protocol Π_P is said to securely realize a given functionality F_P if for any real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with F_P in the ideal world. In our framework, the real-world adversary $\mathcal{A}$ can corrupt the cloud. To prove the security, we describe an ideal-world adversary $\mathcal{S}$ that simulates $\mathcal{A}$. To be precise, $\mathcal{S}$ runs $\mathcal{A}$ internally by playing the role of a seller and a shopper as follows.

Playing the role of a shopper in $\mathcal{Z}$. Note that the functions of Setup and Model Encryption involve a shopper. Here, the ideal-world adversary $\mathcal{S}$ simulates the output of these functions. First, $\mathcal{S}$ chooses a random number mpk^* and sends it to sellers as a master public key. In the real world, the master public key mpk is also randomly chosen, the environment $\mathcal{Z}$ cannot distinguish the simulated mpk^* from the real mpk.

Second, $\mathcal{S}$ simulates the encrypted parameters of the i-th hidden layer as follows. If $i = 1$ and the layer is a fully-connected layer, $\mathcal{S}$ fills random numbers in a vector $\vec{CW}_1^*$ as encrypted parameters. If $i > 1$ and the layer is a fully-connected layer, or $i = 1$ and the layer is a convolution layer, $\mathcal{S}$ fills random numbers in a two-dimension matrix $\mathbf{CW}_\mathbf{i}^*$ (or $\mathbf{CK}_\mathbf{i}^*$) as encrypted parameters. If $i > 1$ and the layer is a convolution layer, $\mathcal{S}$ fills random numbers in a four-dimension matrix $\mathbf{CK}_\mathbf{i}^*$ as encrypted parameters. Then, $\mathcal{S}$ sends the encrypted parameters to $\mathcal{A}$. Here, recall that the real encrypted parameters $\vec{CW}_1$ (or $\mathbf{CK_1}$) of the first hidden layer are encrypted by IFE. Therefore, we can say the environment $\mathcal{Z}$ cannot distinguish the simulated $\vec{CW}_1^*$ (or $\mathbf{CK}_\mathbf{1}^*$) from the real $\vec{CW}_1$ (or $\mathbf{CK_1}$) if IFE is secure. Additionally, recall that the real encrypted parameters of the i-th ($i > 1$) hidden layer are transformed by random numbers. If this layer is a fully-connected layer, and its encrypted parameters $\mathbf{CW}_i$ can be presented as follows.

$$\mathbf{CW}_i[x][y] = \mathbf{W_i}[x][y] * \vec{R}_i[x]/(\vec{R}_{i-1}[y])^2, \tag{12}$$

where $\vec{R}_i$ is the random numbers chosen for the i-th hidden layer, and $\mathbf{W_i}$ is the original parameters. Note that each parameter is randomized by a random number. Therefore, we can say the environment $\mathcal{Z}$ cannot distinguish the simulated $\mathbf{CW}_i^*$ from the real $\mathbf{CW}_i$ if $\vec{R}_i$ and $\vec{R}_{i-1}$ are pseudo-random in a finite field. If this layer is a convolution layer, and its encrypted parameters $\mathbf{CK}_i$ can be presented as follows.

$$\begin{aligned}\mathbf{CK}_i[k][v][:][:] = \mathbf{R_i}[k][v]/\\ \mathbf{R}_{i-1}^2[k*s : k*s+|\mathbf{K_i}|][v*s : v*s+|\mathbf{K_i}|] \times \mathbf{K_i},\end{aligned} \tag{13}$$

where $\mathbf{R}_i$ is the random numbers chosen for the i-th hidden layer, $\mathbf{K_i}$ is the original kernel, s is the convolution stride. Note that each parameter is randomized by a random number. Therefore, we can

say the environment $\mathcal{Z}$ cannot distinguish the simulated $\mathbf{CK}_i^*$ from the real $\mathbf{CK}_i$ if $\mathbf{R}_i$ and $\mathbf{R}_{i-1}$ are pseudo-random in a finite field.

Playing the role of a seller in $\mathcal{Z}$. Note that only the function of Data Encryption involves a seller. Therefore, $\mathcal{S}$ only simulates the output of Data Encryption. It works as follows. If data $\vec{x}$ is input in a fully-connected layer, then $\mathcal{S}$ fills random numbers in a vector $\vec{C}_x^*$ and sends it to $\mathcal{A}$ as the encrypted data. If data $\mathbf{X}$ is input in a convolution layer, then $\mathcal{S}$ fills random numbers in a vector $\mathbf{C}_\mathbf{X}^*$ and sends it to $\mathcal{A}$ as the encrypted data. Recall that both the real encrypted data $\vec{C}_x$ and $\mathbf{C}_\mathbf{X}$ are encrypted by IFE. Therefore, we can say that $\mathcal{Z}$ cannot distinguish $\vec{C}_x^*$ or $\mathbf{C}_\mathbf{X}^*$ from the real $\vec{C}_x$ or $\mathbf{C}_\mathbf{X}^*$ if IFE is secure. □