12-2020

# Secure and verifiable inference in deep neural networks

Guowen XU
*University of Electronic Science and Technology of China*

Hongwei LI

Hao REN

Jianfei SUN

Shengmin XU

*See next page for additional authors*



## Citation

XU, Guowen; LI, Hongwei; REN, Hao; SUN, Jianfei; XU, Shengmin; NING, Jianting; YANG, Haoming; YANG, Kan; and DENG, Robert H.. Secure and verifiable inference in deep neural networks. (2020). *ACSAC '20: Proceedings of the 36th Annual Computer Security Applications Conference, Virtual, December 7-11*. 784-797.

**Available at:** https://ink.library.smu.edu.sg/sis_research/5910

# Secure and Verifiable Inference in Deep Neural Networks

Guowen Xu
guowen.xu@foxmail.com
University of Electronic Science and Technology of China

Hongwei Li*
hongweili@uestc.edu.cn
University of Electronic Science and Technology of China

Hao Ren
renhao.uestc@gmail.com
University of Electronic Science and Technology of China

Jianfei Sun
sjf215.uestc@gmail.com
University of Electronic Science and Technology of China

Shengmin Xu
shengmin_xu@stud.edu.sg
Singapore University of Technology and Design

Jianting Ning
jtning88@gmail.com
Fujian Normal University & Singapore Management University

Haomiao Yang
haomyang@uestc.edu.cn
University of Electronic Science and Technology of China

Kan Yang
Kan.Yang@memphis.edu
The University of Memphis

Robert H. Deng
robertdeng@smu.edu.sg
Singapore Management University

## ABSTRACT

Outsourced inference service has enormously promoted the popularity of deep learning, and helped users to customize a range of personalized applications. However, it also entails a variety of security and privacy issues brought by untrusted service providers. Particularly, a malicious adversary may violate user privacy during the inference process, or worse, return incorrect results to the client through compromising the integrity of the outsourced model. To address these problems, we propose SecureDL to protect the model's integrity and user's privacy in Deep Neural Networks (DNNs) inference process. In SecureDL, we first transform complicated non-linear activation functions of DNNs to low-degree polynomials. Then, we give a novel method to generate `sensitive-samples`, which can verify the integrity of a model's parameters outsourced to the server with high accuracy. Finally, We exploit Leveled Homomorphic Encryption (LHE) to achieve the privacy-preserving inference. We shown that our `sensitive-samples` are indeed very sensitive to model changes, such that even a small change in parameters can be reflected in the model outputs. Based on the experiments conducted on real data and different types of attacks, we demonstrate the superior performance of SecureDL in terms of detection accuracy, inference accuracy, computation, and communication overheads.

## KEYWORDS

Privacy Protection, Deep Learning, Verifiable Inference

*Corresponding author


*ACSAC 2020, December 7–11, 2020, Austin, USA*

ACM ISBN 978-1-4503-8858-0/20/12…$15.00
https://doi.org/10.1145/3427228.3427232

**ACM Reference Format:**
Guowen Xu, Hongwei Li, Hao Ren, Jianfei Sun, Shengmin Xu, Jianting Ning, Haomiao Yang, Kan Yang, and Robert H. Deng. 2020. Secure and Verifiable Inference in Deep Neural Networks. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3427228.3427232

## 1 INTRODUCTION

Deep learning(DL), as one of the promising emerging technologies, has penetrated all aspects of social life, such as face recognition [4, 29], autopilot [19, 63], and medical diagnosis [22, 35, 59]. To support automated services, many tech companies (such as Google, Microsoft, and Amazon) provide outsourced deep learning services, usually dubbed as Machine Learning as a Service (MLaaS) [24, 60, 63]. MLaaS can provide a series of customized training and inference services, along with requiring users to provide local data. A typical example is Azure ML Studio [4], which is developed by Microsoft and enables customers to easily build, deploy, and share advanced deep leaning-based algorithms in the cloud. Other platforms, such as TensorFlow, Caffe2 and MXNet, offer similar services for a fee. Despite such advantages, outsourcing deep learning to the cloud also brings about various security and privacy concerns [2, 29]. Particularly in inference services, customers are very concerned about their model's integrity and data privacy once outsourcing them to the cloud. An adversary, such as an untrusted cloud server, may return incorrect results to users by making some imperceptible modifications to the outsourced model. Such attacks have appeared in various applications including face recognition and image classification [30, 31]. On the other hand, privacy breaches in outsourcing inference services have been frequently reported in the media [4, 29]. Intuitively, once a user outsources its model to a cloud server, it is possible that the server will steal the intellectual property (i.e., parameters) of the outsourcing model, or collect the user's query history through the released API.

To address the above problems, several approaches have been proposed to mitigate privacy and security threats in DNNs [19, 49, 63]. For example, *Ghodsi et al.* propose SafetyNets [63], the first approach for verifiable execution of DNNs on an untrusted cloud.

Recent works [10, 27, 49] also achieve similar goals utilizing diverse technologies, such as trusted hardware SGX [49] and interactive proof systems [27]. However, these approaches mainly focus on the integrity (or correctness) of DNNs computations, which can hardly detect subtle attacks on the model's integrity. For example, in neural network trojan attacks [19], an adversary can slightly modify the model's parameters to make DNNs behave correctly for normal inputs, while misclassifying inputs containing a trigger predefined by the adversary. Moreover, most existing verifiable solutions [19, 27, 49] do not consider privacy-protection in the outsourcing inference process. That is, the user's private data, such as the model's parameters, query requests, and inference results are all disclosed to the server. This inevitably provides a large attacking surface for adversaries to breach the user's privacy. Therefore, it is urgent to design a generic verifiable protocol over the outsourced inference model, which is sensitive to model changes while protecting user's privacy.

*Challenges*: It is challenging to design a secure and privacy-preserving protocol that meets the above requirements. First, the heterogeneous cloud environments may bring a number of vulnerabilities (such as buffer overflow [62], network hijacking [46], etc) for adversaries to launch attacks. It is difficult to guarantee the model's integrity under different cloud operations. Second, once the user submits its model to the server, it will lose control over the use, access, and publishing of the model. Traditional integrity verification strategies (e.g., computing the hash values of protected data) can hardly work since the server can easily provide plausible verification results to users. Third, existing approaches [10, 27, 63] always verify the model's integrity by analyzing the model's outputs due to the black-box access to the server. However, in some model integrity attacks [31, 46], by slightly modifying the model's parameters, the adversary can make the classifier misclassify for specific attacker-chosen inputs, while processing correctly for other inputs. Therefore, it is very difficult to verify the model's integrity by only checking the outputs. Fourth, it is also challenging to propose a light-weight approach that is highly supportive of both model verification and data privacy protection. Existing privacy-preserving methods for neural networks are mostly evolved from three underlying techniques: *Secure Multi-Party Computation (SMC)* [4, 35], *Differential Privacy* [45, 62] and *Homomorphic Encryption*(HE) [2, 55]. However, technologies based on SMC and differential privacy may not be proper for the scenarios considered in this paper (see Section 2 for more details). Fully Homomorphic Encryption (FHE) is a potential solution. However, it leads to huge computation overhead. LHE [6, 65] (also called Somewhat Homomorphic Encryption), are faster than FHE, but only support limited addition and multiplication in ciphertext. Moreover, complicated non-linear activations such as ReLU, Sigmoid, and Tanh in DNNs, are not directly supported by LHE. Recent works [10, 29] exploit function approximation to convert non-linear activations to polynomials, however, these approaches are fragmented and generally cannot be applied to all activation functions.

*Our Contributions*: To address the above challenges, we propose SecureDL, a secure and verifiable inference protocol to protect the model's integrity and user's privacy in DNNs. In our SecureDL, we first transform non-linear activation functions to low-degree polynomials. Then, sensitive-samples are exploited to verify the correctness of the model's parameters. In the end, LHE is used to provide the privacy-preserving DNNs inference. In summary, our contributions can be summarized as follows:

- We first convert complicated non-linear activation functions such as ReLU, Sigmoid, and Tanh into polynomials, to facilitate the implementation of LHE in general DNNs, and the generation of sensitive-samples. We prove that given an error bound, it is possible to approximate any function with a low-degree polynomial.
- We design a novel sensitive-samples generation method to protect the model's integrity. We show that our `sensitive-samples` are very sensitive to changes in the model parameters, such that even a small parameter change can be reflected in the model outputs. In addition, our `sensitive-samples` can be applied to general neural networks, with no assumptions on DNNs architecture, hyper-parameters, and training methods.
- We conduct extensive experiments on different datasets to demonstrate the high performance of SecureDL in terms of inference accuracy, detection accuracy, computation, and communication overheads.

The remainder of this paper is organized as follows. In Section 2 and Section 3, we introduce the related works, outline the background and problem statement. In Section 4, we give the details of our SecureDL. Next, performance evaluation is presented in Section 5. Finally, Section 6 concludes the paper.

## 2 COMPARISON WITH EXISTING WORKS

In this section, we introduce the latest related works about privacy-preserving and verifiable deep learning, and compare them with our proposal.

### 2.1 Privacy-preserving Deep Learning

As briefly mentioned, most of the existing research results evolve from three underlying techniques: i.e., differential privacy [1, 23, 37, 45, 52, 62, 64], secure multi-party computing [4, 29, 34, 41] and homomorphic encryption [2, 5, 14, 21, 53].

**Differential privacy-based framework**: Differential privacy technology is mainly used in distributed or centralized DNNs training processes, where each data owner or the server disturbs the sensitive data by adding disturbances to the original data, weights, or loss functions. The propose of differential privacy-based DNNs training is to reduce the negative impact of the addition of noise on training as much as possible, and ensure data security under the pre-privacy budget. For example, *Shokri et al.* [45] design the first privacy-preserving deep learning model with differential privacy. It ensures that the user's data privacy is not compromised by selectively sharing local parameters to the server. *Agarwal et al.*[1] proposed cpSGD, an efficient and differentially-private distributed stochastic gradient descent(SGD) in training process. By adding noise that satisfies a specific distribution (such as Laplace distribution) to the original gradient, it can achieve the unrecoverability to the original data and the high accuracy of training. Other works such as [38], [62], and [50], also propose diverse strategies to make a trade-off between training accuracy and data privacy, like add noise to weights [38], set privacy budget dynamically [62], etc.

**Comparison**: To the best of our knowledge, differential privacy is generally only applicable to training for DNNs due to its inherent properties. That is, it can only guarantee that the disturbed dataset is roughly consistent with the original dataset in statistical properties, does not retain any attributes between the single disturbed data and its corresponding original data. However, our focus is to solve the data privacy in the inference phase. That is, we require users to send a single encrypted query request to the server, then the server performs inference services in the ciphertext environment and returns the ciphertext results. Hence, if we exploit differential privacy technology to disturb a single query request, or the parameters of the outsourced model, this will inevitably lead to noise that is difficult to offset, thereby weakening the accuracy of model classification.

**SMC based framework**: SMC enables two or more parties to evaluate a function on their inputs without disclosing the inputs to each other, that is, all inputs are kept private by the respective owners. In general, the existing works with SMC as the underlying architecture can be categorized into three approaches: i.e., Garbled Circuit(GC)-based [3, 40, 42], Secret Sharing-based [4, 9, 61], and Mixed Protocol-based [26, 34, 41], where GC-based approach is mainly applied to the secure computing of 2 or 3 parties, while Secret Sharing based approach is more suitable for distributed secure computing. For example, *Riazi et al.*[40] and *Ball et al.*[3] both design a new garbled circuit techniques for neural networks, and introduced new optimizations for modern neural network activation functions. In terms of Secret Sharing-based works, the most representative result is [4], which design a practical data aggregation protocol in federated training by using Shamir's $t$-out-of-$n$ secret sharing protocol [15]. In recent years, studies [35] have shown that it is difficult to achieve practical in communication overhead or computational overhead by using secret sharing or garbled circuits alone. To combat that, Mixed Protocol-based works [26, 34, 41], i.e., hybrid secure computation frameworks exploiting mixed use of secret sharing, homomorphic encryption, and garbled circuits, have been proposed and applied in various fields. These mixed protocols usually use additive secret sharing or homomorphic encryption to perform linear operations in the deep learning process, while non-linear calculations are delivered to garbled circuits for implementation. The experimental results [34, 41] show that such a hybrid approach tends to show better performance.

**Comparison**: SMC-based technology is a good way to provide secure training and inference services, but in general, this requires each party involved in secure computing to honestly execute a predetermined protocol. In our scenario, the server is considered to be an active, malicious adversary with an incentive to destroy established procedures (including compression calculations, modification of parameters, etc) to violate the integrity of the original protocol. It is difficult to construct an efficient SMC protocol under such threat model. On the hand, the design of SMC is to ensure that the participants evaluate a targeted function without knowing each other's input secrets. However, in our scenario, the parameters of the outsourcing model and the user's inputs are held by the user itself, and we only require the server's computing and storage resources without sharing secrets with each other.

**HE based framework**: Homomorphic encryption can perform specific mathematical operations in ciphertext without knowing the unencrypted data. Such characteristics make it perform training and inference services in ciphertext gracefully. Based on the differences in the mathematical operations supported, HE can be classified into partial (additive or multiplicative) HE [2, 39], FHE and LHE [6, 21, 65] (also called Somewhat Homomorphic Encryption). Partial HE is usually used in distributed (collaborative or federated) DNNs training due to the simplicity of the mathematical operation supported in the ciphertext. For example, *Phong et al.* [2] exploited the Partial HE to encrypt users' local gradients before uploading to the cloud, which provides secure data aggregation for multi-users. FHE offers an elegant way to provide secure DNNs training and inference without even any interaction. For example, *Dowlin et al.* [14] and *Bourse et al.* [5] respectively propose FHE-based training and inference methods, where the user only needs to send the encrypted input to the server and receive the returned ciphertext result. However, due to the inefficiency of existing FHE schemes, most applications prefer to use LHE [6, 21, 65] which is faster than FHE, but only supports a limited depth of encryption and multiplication operations in ciphertext, because it removes the reduction process of noise introduced by multiple computations, such as bootstrapping [65]. *Hesamifard et al.* [21] propose CryptoDL, a Privacy-preserving Machine Learning as a Service (MLaaS) with LHE. In CryptoDL, complex nonlinear activation functions such as Sigmoid and tanh are replaced by polynomials. Then, CryptoDL relies on the HElib library [17] to complete the training and prediction of DNNs in the ciphertext.

**Comparison**: As discussed above, FHE is not practical in terms of efficiency [1]. Therefore, in this paper, we use LHE to achieve privacy-preserving inference. Similar to work CryptoDL [21], we also convert complex nonlinear activation functions into polynomials, and use the HElib library to achieve LHE implementation. However, compared with CryptoDL, we give a formal proof that it is possible to approximate any continuous function with a polynomial whose error from the objective function is within a given bound, while CryptoDL only provides a scratch. Moreover, as admitted in CryptoDL, it only considers the approximation of continuous functions, and cannot provide conversion to non-continuous functions such as Rectified Linear Unit(ReLU,$y = \max(0, x)$). Nevertheless, RelU has become a highly regarded activation function in the field of image recognition. Compared with CryptoDL, our work can transform any activation functions into polynomials, where we use discrete least squares [8] to give a heuristic conversion algorithm for non-continuous functions. Experiments (See Section 5) show that we can still find satisfactory low-degree polynomials.

## 2.2 Verifiable Deep Learning

In this paper, we focus on the verifiability of the results returned by the server during the inference phase. In summary, the existing results can be roughly divided into two directions: i.e, Trusted Execution Environments(TEE)-based [18, 48, 49] and Verifiable Computing (VC)-based [10, 27, 63]. TEE provides a secure enclave to run a deep learning model, where the model/data owner can use hardware and software protections to isolate sensitive computations from the untrusted software stack. In this way, data privacy

[1]We also confirmed this argument in the experimental part. For more details, please see Section 5.3

and the integrity of the calculation process can be hard protected. *Florian et al.* [49] propose Slalom, which enables all linear layers in DNN from TEE (such as Intel SGX or Sanctum) to be executed by a faster but untrusted co-located processor. Verifiable computing (VC) can provide proofs of computational integrity without any assumptions on hardware. *Ghodsi et al.* [63] proposed the first verifiable approach SafetyNets. In SafetyNets, a specific type of DNNs framework will be converted into an arithmetic circuit, under which the server interacts with the user multiple times to verify the correctness of the returned results. Recently, *Keuffer et al.* [27] also design an efficient proof composition for verifiable computation, which proposes a method for constructing several dedicated and efficient VC schemes using the universal VC protocols.

**Comparison**: TEE-based works rely on hardware to fulfill the requirements for privacy and integrity while we aim to to construct a verifiable solution without any hardware assumptions. VC-based approaches focus on the integrity (or correctness) of DNNs computations performed by the cloud provider, which can hardly detect subtle attacks on model's integrity, such as the neural network trojan attacks [32] and targeted poisoning attack. In this paper, we propose a general method for generating sensitive samples and use them to detect the completeness of the server's calculation results. We note that recent work [19], also designed an efficient way to generate sensitive samples against subtle attacks on the model's integrity. However, the way to generate sensitive samples in [19] is only for DL models that exclusively contain continuous activation functions. Moreover in [19], to ensure that an untrusted server is difficult to distinguish between real samples and sensitive samples, the feasible domain for selecting sensitive samples is limited to a small domain, which weakens the sensitivity of sensitive-samples to model's changes. Compared with [19], our scheme for generating sensitive samples is applicable to all neural network frameworks, because all complex activation functions (including non-continuous) have been transformed into polynomials (continuous). Moreover, Applying LHE guarantees that all the user's encrypted input is indistinguishable to the server. As a result, in the process of generating sensitive samples, we can choose samples with the highest sensitivity to model modification as the optimal samples.

# 3 BACKGROUND AND PROBLEM STATEMENT

In this section, we first review the concept of DNNs, and then describe the scenario, threat model and security and privacy requirements considered in this paper.

## 3.1 Deep Neural Networks

As shown in Figure 1, a DNN usually consists of one input layer, one or more hidden layers and one output layer. Each two adjacent layers are connected by weights $\boldsymbol{\omega}$ (i.e., model's parameters), where each circle represents a neuron associated with an element-wise nonlinear activation function $\varphi$ (i.e., sigmoid, ReLU, softmax, etc). Here we use DNN training process to describe how it works. Specifically, given a training sample $(\boldsymbol{x}, \boldsymbol{y})$, the input $\boldsymbol{x}$ will be iteratively propagated to the next layer with linear transformations and nonlinear activation functions. Then, the neural network outputs the inference result $\bar{\boldsymbol{y}}$ in the last layer. This process is usually called

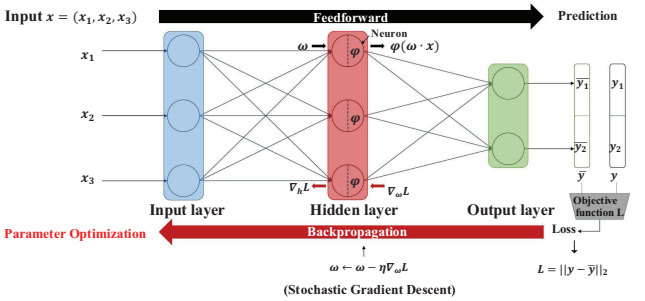


**Figure 1: General DNNs training process**

`feedforward`. To find the optimal parameters (i.e., $\boldsymbol{\omega}$) for accurately reflecting the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, a loss function $\boldsymbol{L}$ is adopted to measure the distance between $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$. Usually, $\boldsymbol{L}$ is given as $\boldsymbol{L}$=$||\boldsymbol{y} - \bar{\boldsymbol{y}}||_2$, where $||\cdot||_2$ is the $l_2$ norm of a vector. Then, to minimize the loss function $\boldsymbol{L}$, the Stochastic Gradient Descent (SGD) algorithm [45, 62] is used to find the optimal parameters $\boldsymbol{\omega}$. We call this process as `backpropagation`. After the DNN converges to pre-set accuracy, it can be used for subsequent inference. In this paper, we focus on the model's integrity and user's privacy in the inference process.

## 3.2 Scenario

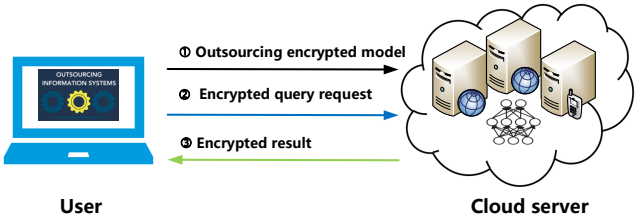


**Figure 2: Our Scenario**

As shown in Figure 2, our SecureDL has two generic entities, a user and a cloud server. To receive inference services, the user (also called the client) first encrypts its well-trained DL model and outsources it to the cloud[2]. Then, the server allocates resources for this model for a fee, such as assigning computing and storage resources, and releasing APIs for the inference service. In the inference process, once the user submits its encrypted query request to the cloud, the server performs the preset operations of the outsourced model, and returns the corresponding encrypted inference result (such as classification and regression) to the user. The above scenario has been widely used in the field of outsourcing computing [10, 19, 20]. In this way, model owner can not only save resources required for local storage and execution of the DL model, but also enjoy real-time inference services without geographical and hardware (partial) restrictions.

[2]We do not consider the details of the model training, that is, the user can train the model locally, or fine-tune a model obtained from the public model zoo. Please note that the user needs to convert all activation functions not supported by LHE to polynomials before the model is trained.

### 3.3 Threat Model and Security and Privacy Requirements

In our SecureDL, the cloud server is considered as the main adversary. On the one hand, it may infer users' data privacy by utilizing the mastered prior knowledge [54, 56–58], such as the encrypted dataset, query records, and ciphertext results. On the other hand, it may also try to compromise the model's integrity. Specifically, we consider, but not limited to, the following attacks to compromise the model's integrity hosted in the cloud: ①The cloud server can exploit the potential vulnerabilities in the network and the service interface to implement the attack. ②The cloud server can use a simpler or compressed model to replace the original model, thereby breaking the integrity of the model. ③The cloud server has full access to the encrypted outsourced model, user's inputs, and inference results. It can launch attacks based on this information.

Under the above threat model, we formulate the security and privacy requirements as follows.

- *Protect the model's integrity*: In order to obtain certain benefits, a malicious server is fully capable of modifying the model's parameters and architecture to deceive customers. Our goal is to detect any subtle model changes in an efficient manner.
- *Guarantee the confidentiality of the model's parameters*: The model's parameters are valuable intellectual property, which may be generated with a lot of resources, and even contain some user's proprietary information. A secure outsourcing inference service should protect this information from being leaked to the server.
- *Privacy protection of user's requests and inference results*: In the inference process, a user will submit its query requests to the server for inference services (such as image classification and numerical prediction, etc). Sometimes, these data are sensitive and may contain user's personal information (such as avatar, health status, and psychological behavior). In addition, the inference results always imply some relationships with the user's inputs. Therefore, the privacy of user's requests and inference results should be protected from being leaked to the server.

## 4 PROPOSED SCHEME

The goal of SecureDL is to realize the privacy-preserving outsourced inference services while guaranteeing the model's integrity. To achieve this, we first design a general function approximation algorithm to transform non-linear activation functions to low-degree polynomials. This will facilitate the generation of `sensitive-samples` and the and the application of LHE in general DNNs. Then, we generate generic `sensitive-samples` to verify the correctness of model parameters in the inference process. In the end, to protect the user's privacy, LHE is used to provide privacy-preserving DNNs inference.

### 4.1 Function Approximation

As discussed before, the most notable shortcoming of LHE is that it only supports limited number of addition and multiplication operations in the encrypted domain. Also, complicated non-linear activation functions in DNNs, such as Sigmoid ($y = \frac{1}{1+e^{-x}}$) and ReLU ($y = \max(0, x)$), are not directed supported by LHE. For the smooth execution of LHE, these complicated non-linear activation functions need to be approximated by functions (such as polynomials) that only contain addition and multiplication.

On the other hand, given a bound on the error between the original activation function and its transformed polynomial, the degree of the transformed polynomial should be minimal to boost efficiency [53]. Therefore, to reduce the overhead of the LHE during DNN inference process, a priority task of our SecureDL is to design such an algorithm, which can find the low-degree polynomial within a given error.

Theorem 1. *Given an error bound $\epsilon$, let $M(x)$ be a continuous function on the closed interval $[a, b]$, there exists a polynomial $p(x)$ that satisfies $|M(x)–p(x)| < \epsilon$ for all $x$ belong to the interval $[a, b]$.*

*Proof*: This theorem is based on the Weierstrass Approximation theorem [12]. Briefly, we first construct **Bernstein** polynomial [11, 12] based on the Weierstrass Approximation theorem. Then, we prove that any continuous function can be approximated by the **Bernstein** polynomial with any given error bound $\epsilon$. For detailed proof, please refer to the APPENDIX1.1.

Based on Theorem 1, given an objective continuous activation function $M(x)$ and an error bound $\epsilon$, we can use the **Bernstein** function to find a satisfactory polynomial. However, it does not give a method of how to find a low-degree polynomial. Moreover, **Bernstein** polynomial has been proven to be inefficient in approximating any continuous function [11, 12]. In general, the polynomial function classes used to approximate the known function $M(x)$ are diverse. Even if the function class is selected, the function $p(x)$ used as an approximate representation of $M(x)$ is still determined in various ways. For example, work [21] suggests exploiting Legendre polynomials [44] and Chebyshev polynomials [7] to approximate the original function, while other works, such as [29], propose to use piecewise interpolation [51] to achieve the transformation of the objective function into polynomials. In this paper, we exploit the well-known least square approximation algorithm [8] (see APPENDIX 1.2) to generate the low-order polynomial, since it is very efficient compared to the above methods, and can be easily realized by standard programming software such as Matlab, CurvFit, and Prism. As shown in **Algorithm 1**, given an error bound $\epsilon$ and a target non-linear activation function $M(x)$, we first initialize the degree of approximated polynomial to 1 (i.e., $N = 1$). Then, we get the approximated polynomial $p_N(x)$ based on the least square approximation algorithm[3]. Next, we verify whether $\{|p_N(x) - M(x)|_2^2 \leq \epsilon\}$ (see Eqn.(12) in APPENDIX 1.2 for the definition), if so, the current polynomial is returned as the final low-degree polynomial; otherwise, let $N = N + 1$, continue to generate $p_N(x)$ and iteratively verify the above operation until we find a degree polynomial(i.e., $p^*(x)$ ) that meets the above constraint. Based on Theorem 1, for any continuous activation functions, the iterative process in **Algorithm 1** (lines 4-5) is finite, thus we can certainly find a low-degree polynomial $p^*(x)$ satisfying the constraint $\{|p^*(x) - M(x)|_2^2 \leq \epsilon\}$. For the discontinuous activation functions such as ReLU, we use discrete least squares algorithm [8] to approximate them. Experiments

[3]Please note that for discontinuous activation functions such as ReLU, Theorem 1 is not true. Hence, for these discontinuous activation functions, we use discrete least squares algorithm [8] to approximate them. Experiments (see Section 5) show that we can still find satisfactory low-degree polynomials.

(See Section 5) show that we can still find satisfactory low-degree polynomials.

**Algorithm 1** Generating the low-degree polynomial

**Input:** A given error bound $\epsilon$ and the target non-linear activation functions $M(x)$.
**Output:** The low-degree polynomial $p^*(x)$.

1: $N = 1$. /* Initialize the degree of approximated polynomial to 1. */
2: $tmp = p_N(x)$ based on the least square approximation algorithm [8]. /* $p_N(x)$ denotes the approximated polynomial with degree $N$. */
3: **while** $\{|tmp - M(x)|_2^2 \geq \epsilon\}$ **do**
4: $N++$.
5: $tmp = p_N(x)$.
6: **end while**
7: $p^*(x) = p_N(x)$.
8: Return $p^*(x)$.

*Remark*: We notice that some works [10, 21, 29] have proposed methods to convert complex functions into polynomials. However, compared with them, we give a formal proof that given an arbitrary objective continuous function, it is feasible to approximate to a polynomial whose error from the objective function is within a given range, while existing works are fragmented or only provide a scratch. Moreover, Compared with existing works, our work can transform any activation functions into polynomials, where we use discrete least squares to give a heuristic conversion algorithm for non-continuous functions. For model detail, please refer to Section 2.1.

## 4.2 Sensitive-samples Generation and Inference with LHE

We assume that the cloud server may modify the outsourced model $f_\theta(x)$ to $f'_\theta(x)$, where $\theta$ is the set of all model's parameters. However, as discussed before, the cloud provider only provides a black-box way for users to verify the model's integrity. To address this challenge, similar to work [19], our main idea is to generate a small set of test samples $\{(\boldsymbol{c_i}, f_\theta(\boldsymbol{c_i}))|i = 1, 2, \cdots, v\}$ (called `sensitive-samples`), where $f_\theta(\boldsymbol{c_i})$ is the correct output with input $\boldsymbol{c_i}$. Then, we use these `sensitive-samples` to verify the model's integrity.

As show in Figure 3, we first generate a small set of `sensitive-samples` $\{(\boldsymbol{c_i}, f_\theta(\boldsymbol{c_i}))|i = 1, 2, \cdots, v\}$, and then send $\boldsymbol{c_i}, (i = 1, 2, \cdots v)$ to the cloud for classification. Once obtaining the classification results $f'_\theta(\boldsymbol{c_i})$,the user can check if the model is intact by only comparing the equality between $f_\theta(\boldsymbol{c_i})$ and $f'_\theta(\boldsymbol{c_i})$.

*4.2.1 The Goal of Sensitive-samples.* To achieve the above requirements, we need to find such `sensitive-samples` that are very sensitive to model changes. Moreover, the generated `sensitive-samples` should be hardly spotted by the adversary. In this section, we design a novel sample generation algorithm to generate the `sensitive-samples`. Specifically, the DNN model can be defined as $\boldsymbol{y} = f_\theta(\boldsymbol{x})$. We rewrite the DNN model as

$$\boldsymbol{y} = f(\boldsymbol{\omega}, \boldsymbol{x}) = [y_1, y_2, \cdots, y_m] = [f_1(\boldsymbol{\omega}, \boldsymbol{x}), \cdots, f_m(\boldsymbol{\omega}, \boldsymbol{x})]$$

where $\boldsymbol{\omega} = [\omega_1, \omega_2, \cdots, \omega_v]$ is a subset of $\theta$ in our consideration. It contains the weights and biases. $f_i(\boldsymbol{\omega}, \boldsymbol{x}), i = (1, \cdots m)$ represents the analytic expression of $y_i$ with input $\boldsymbol{x}$. Without loss of generality, we assume that the adversary compromises the outsourced model by modifying the correct parameters $\boldsymbol{\omega}$ to $\boldsymbol{\omega'} = \boldsymbol{\omega} + \Delta\omega$. Hence,

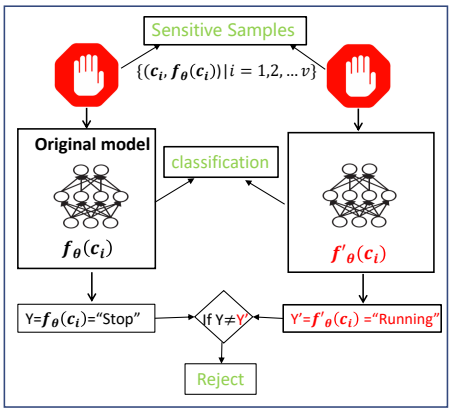


**Figure 3: Using sensitive samples to verify the integrity of the outsourced model**

the modified model can be denoted as $\boldsymbol{y'} = f(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x})$. In order to detect model anomalies sensitively, a good `sensitive-sample` $\boldsymbol{c}$ should maximize the difference between $\boldsymbol{y}$ and $\boldsymbol{y'}$. Formally, a `sensitive-sample` $\boldsymbol{c}$ should be the optimal value such that

$$\begin{aligned} \boldsymbol{c} &= \arg\max_{\boldsymbol{x}} ||f(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x}) - f(\boldsymbol{\omega}, \boldsymbol{x})||_2^2 \\ &= \arg\max_{\boldsymbol{x}} \sum_{i=1}^{i=m} ||f_i(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x}) - f_i(\boldsymbol{\omega}, \boldsymbol{x})||_2^2 \end{aligned} \tag{1}$$

To solve the above optimization problem, we first perform Taylor expansion[4] on $f_i(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x})$ as follows,

$$f_i(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x}) = f_i(\boldsymbol{\omega}, \boldsymbol{x}) + \frac{\partial f_i(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega + O(||\Delta\omega||_2^2) \tag{2}$$

Based on the Taylor expansion, the differences between $f_i(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x})$ and $f_i(\boldsymbol{\omega}, \boldsymbol{x})$ can be approximated as below,

$$||f_i(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x}) - f_i(\boldsymbol{\omega}, \boldsymbol{x})||_2^2 \approx ||\frac{\partial f_i(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega||_2^2 \tag{3}$$

Further, we have

$$\begin{aligned} ||f(\boldsymbol{\omega} + \Delta\omega, \boldsymbol{x}) - f(\boldsymbol{\omega}, \boldsymbol{x})||_2^2 &\approx \sum_{i=1}^{i=m} ||\frac{\partial f_i(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega||_2^2 \\ &= ||\frac{\partial f(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega||_F^2 \end{aligned} \tag{4}$$

where $||\cdot||_F$ denotes the Frobenius norm [33] of a matrix. By comparing with Eqn.(1) and Eqn.(4), it is obvious that we can find the optimal `sensitive-samples` by solving the following optimization problem,

$$\boldsymbol{c} = \arg\max_{\boldsymbol{x}} ||\frac{\partial f(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega||_F^2 \tag{5}$$

In some cases, the model's inputs are limited to a certain range (denoted as $[\mathcal{B}, Q]$). For instance, the intensities of all pixels are limited to $[0, 255]$ for valid image inputs. Therefore, the above optimization problem is modified as below.

$$\boldsymbol{c} = \arg\max_{\boldsymbol{x}} ||\frac{\partial f(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} \Delta\omega||_F^2, s.t. \boldsymbol{x} \in [\mathcal{B}, Q]^\eta \tag{6}$$

[4]since all complex activation functions (including non-continuous) have been transformed into polynomials (continuous), Taylor expansion is applicable to all neural network frameworks.

where $[\mathcal{B}, Q]^\eta$ is a convex set, and $\eta$ denotes the dimension of samples.

*4.2.2 Sensitive-samples Generation Algorithm.* Based on Eqn.(6), we give a sensitive-samples generation algorithm shown in **Algorithm 2**. Lines 1-4 initialize the algorithm (i.e., define the total number of iterations, and assign a random original sample to $\boldsymbol{c}$). Lines 5-7 set up the optimization function $||\frac{\partial f_j(\boldsymbol{\omega}, \boldsymbol{c})}{\partial \boldsymbol{\omega}}||_2^2$. Lines 8-14 utilize the gradient ascent technology [66] to find the optimal `sensitive-sample` under the constraints. Line 16 returns the `sensitive -sample` and its inference result.

**Algorithm 2** Sensitive-samples generation algorithm

**Input:** $\{f, \boldsymbol{\omega}, \iota\}$, where $f$ is the original model. $\boldsymbol{\omega}$ is the set of parameters in our consideration, and $\iota$ is the learning rate.
**Output:** $\{\boldsymbol{c}, f(\boldsymbol{\omega}, \boldsymbol{c})\}$, denotes the `sensitive-sample` and its inference result.

1: $\boldsymbol{x_0} = Init\_Sample()$. /* Randomly take a sample in the original sample set. */
2: $It\_MAX = \mathcal{G}$. /* Initialize the total number of iterations. */
3: $\boldsymbol{c} = \boldsymbol{x_0}$. /* Initialize the `sensitive-sample`. */
4: $i = 0$.
5: **for** $j = 1$ to $j = m$ **do**
6: $\quad \sigma_j = ||\frac{\partial f_j(\boldsymbol{\omega}, \boldsymbol{c})}{\partial \boldsymbol{\omega}}||_2^2$.
7: **end for**
8: **while** $\{(\boldsymbol{c} \in [\mathcal{B}, Q]^\eta) \&\& (i < It\_MAX)\}$ **do**
9: $\quad \Delta = 0$.
10: $\quad$ **for** $j = 1$ to $j = m$ **do**
11: $\quad\quad \Delta += \partial \sigma_j / \partial c$.
12: $\quad$ **end for**
13: $\quad \boldsymbol{c} = \boldsymbol{c} + \iota * \Delta$.
14: $\quad$ i++.
15: **end while**
16: Return $\{\boldsymbol{c}, f(\boldsymbol{\omega}, \boldsymbol{c})\}$.

*Remark*: Since user inputs and model parameters will be encrypted by LHE before outsourcing to the server, we claim that the server cannot distinguish sensitive samples from original samples and cannot generate sensitive samples by itself. However, our sensitive samples are not well protected against random output tampering from the server. For example, for a few victim inputs, the cloud server may manipulate the result in any way it wishes, while for all of the other inputs, it honestly follows the protocol to compute the result using the right model. For such untargeted random attacks, a potential solution is to increase the proportion of sensitive samples in the query process. On the other hand, we believe that such untargeted attacks are also risky for the server. Due to the indistinguishability of real and sensitive samples, once incorrectly returning incorrect output for sensitive samples, the server will face punishment or loss of reputation.

We note that recent work [19], also designed a way to generate sensitive samples to verify the model's integrity. However, it only works for DL models that exclusively contain continuous activation functions, and the feasible domain for selecting sensitive samples is limited to a small domain, which weakens the sensitivity of sensitive-samples to model's changes. Compared with work [19], our scheme is applicable to all neural network frameworks. Moreover, LHE's security guarantees that all the user's encrypted input is indistinguishable to the server. As a result, in the process of generating sensitive samples, we can choose samples with the highest sensitivity to model modification as the optimal samples. For more detail, please refer to section 2.2.

*4.2.3 Privacy-preserving Inference with LHE.* We have transformed the nonlinear activation functions into polynomials, and prepared the `sensitive-samples` for model verification. To protect the user's privacy, we adopt Leveled Homomorphic Encryption (LHE) to encrypt all user-related data, such as the model's parameters, user's query requests, and inference results. In this paper, we adopt HELib library [17] to implement all ciphertext inference. The library implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [13], as well as optimizations such as Smart-Vercauteren ciphertext packing techniques.

## 5 PERFORMANCE EVALUATION

In this section, we evaluate the performance of SecureDL in terms of inference accuracy, detection accuracy, and overhead. Specifically, the " Cloud " is simulated with a virtual machine with 48GB RAM, 12 CPU cores and Ubuntu 18.04. For generating encryption schemes in the HELib[4], we set the security parameter with 80 ( security level is equivalent to AES-128), the number of slots in the ciphertext with 0 (this allows HELib to automatically select the optimal number of slots), and $L = 20$ (control the maximum number of operations allowed in ciphertext without decryption). To reduce the increasing noise in the ciphertext computation, we allow the server to check the number of calculations after each ciphertext calculation. If the number of calculations is about to exceed a threshold, the server returns the ciphertext to the user for decryption, and then the user re-encrypts it in a fresh ciphertext and sends it to the server.

### 5.1 Inference Accuracy

We test the inference accuracy of SecureDL under a custom CNN network, which consists of two convolutional layers ( containing 20 feature maps and 50 feature maps, respectively), one average pooling layer and two fully connected layers (256 and 10 neurons, respectively). As discussed before, we transformed the non-linear activation functions into polynomials. Intuitively, this will affect the accuracy of the DNNs output. To quantitatively estimate the impact of this change on inference accuracy, we first approximate three activation functions (i.e., Sigmoid:$f_1(x) = \frac{1}{1+e^{-x}}$, ReLU:$f_2(x) = \max(0, x)$ and tanh:$f_3(x) = \frac{e^{2x}-1}{e^{2x}+1}$) according to the **Algorithm 1**. Here we use $P_{f_1(x)}, P_{f_2(x)}$ and $P_{f_3(x)}$ to represent the corresponding approximation polynomials, and the degrees of $P_{f_1(x)}, P_{f_2(x)}$ and $P_{f_3(x)}$ are 2, 2 and 3, respectively. Then, to comprehensively analyze the inference accuracy under different data sets, we select 6 datasets (i.e.,*Breast tissues*, *Crab*, *Ovarian*, *Wine*,*Climate*, and *Fertility*, shown in Table 5 of APPENDIX) from UC Irvine Machine Learning Repository [25], and compare the classification accuracy with those using the original activation functions.

As described in Table 1, compared with the existing activation functions, the approximated polynomials with two or three degrees are sufficient to achieve the expected accuracy. We can see that the

[4]Please note that HELib cannot directly support the operation of some pooling layers (such as Max pooling) in DNNs, because of the lack of the max operation over encrypted data. To address this, we use a scaled up version of average pooling (proposed in [14]) to replace these types of layers.

**Table 1: Inference Accuracy of Our SecureDL with Different Datasets**

| Acuracy / Dataset | $f_1(x)$ | $f_2(x)$ | $f_3(x)$ | $P_{f_1(x)}$ | $d_{P_{f_1(x)}}$ | $P_{f_2(x)}$ | $d_{P_{f_2(x)}}$ | $P_{f_3(x)}$ | $d_{P_{f_3(x)}}$ |
|---|---|---|---|---|---|---|---|---|---|
| Breast tissues | 87.00% | 93.00% | 86.40% | 86.10% | 2 | 92.47% | 2 | 85.17% | 3 |
| Crab | 92.23% | 96.32% | 89.30% | 91.26% | 2 | 96.17% | 2 | 89.12% | 3 |
| Ovarian | 93.75% | 97.21% | 89.25% | 92.77% | 2 | 97.01% | 2 | 89.05% | 3 |
| Wine | 96.29% | 97.21% | 90.23% | 96.19% | 2 | 97.18% | 2 | 89.73% | 3 |
| Climate | 95.37% | 96.54% | 89.21% | 95.32% | 2 | 96.25% | 2 | 89.04% | 3 |
| Fertility | 86.67% | 92.32% | 84.25% | 85.12% | 2 | 92.27% | 2 | 83.07% | 3 |

model's classification accuracy under approximation polynomials is very close to that under the original functions. For example, in terms of classification of wine images, the original function (i.e., ReLU:$f_2(x) = \max(0, x)$) has a classification accuracy of 97.21%, and our approximate function can still reach 97.18%. The reason for this is that in function approximation process, **Algorithm 1** has limited the maximum error bound between the original function and its approximated polynomial, which ensures that the neural network with these polynomial activation functions still shows good prediction accuracy.

## 5.2 Detection Accuracy

In this section, we estimate the detection accuracy of the proposed scheme. As mentioned earlier, the server cannot learn the model parameters since they have been encrypted by the LHE. However, to "purely" analyze the sensitivity of the generated sensitive samples to model changes, we considered a stronger adversary (i.e., the server), which is allowed to access the plaintext parameters, and even use some samples to retrain the model[5]. In theory, our `sensitive-samples` are generic and able to detect various integrity attacks against DNN models. To evaluate the detection accuracy, we consider four very subtle integrity attacks in our experiments.

- *Neural Network Trojan Attack* (**NNTA**[31]): The attack goal is to inject some trojans in the outsourced model to make DNNs behave correctly for normal inputs, while misclassifying the inputs containing triggers predefined by the adversary. The adversary can achieve the goal by modifying the selected parameters with triggers.
- *Targeted Poisoning Attack* (**TPA** [43]): To make DNNs misclassify the inputs to targeted outputs, the adversary can modify the parameters by retraining the model with customized data.
- *Model Compression Attack* (**MCA**[30]): For reducing cloud storage and computation cost, a malicious cloud provider may compress the original model into a simple model without visibly affecting inference accuracy.
- *Arbitrary Weights Modification* (**AWM**[36]): It is common that an adversary (such as the cloud server) can change any parameter of the outsourced model.

The specific datasets and DNNs models used in our experiments are described in Table 4 (see APPENDIX 1.3). We know that the form of model's outputs significantly affects the detection accuracy because we access the outsourcing model in only a black box way. Therefore, we consider the case of the server returning Top-k (k=1,3,5) classification labels to users, where the less information including in outputs (from Top-5 (most) to Top-1 (least)) means the harder to detect tampering using `sensitive-samples`. Next, we discuss the detection accuracy of our `sensitive-samples` under these four attacks. Moreover, the state-of-the-art approaches SSFDNN [19], is also adopted in our experiments to compare with our method.

*5.2.1 Neural Network Trojan Attack.* We first evaluate the detection accuracy of SecureDL under the neural network trojan attack, where the attack goal of NNTA is to make the model incorrectly classification by injecting trojan into the DNNs. In this experiment, we assume that the adversary launches attacks through modifying some selected neurons parameters with triggers on the VGG-16 [47] model, which consists of 14 convolution layers and four fully connected layers. Here the VGG-Face is a standard DNNs model used for face classification.

Specifically, by comparing with the state-of-the-art approaches SSFDNN [19] and the original samples, we first test the sensitivity of the `sensitive-samples` generated in SecureDL. As shown in Figure 4(a), the blue histogram shows the detection accuracy of randomly selected samples from the original samples, and the orange and yellow histogram represent the detection accuracy of samples generated by [19] and SecureDL, respectively. Here we require the server to return the Top-1 result with the different number of samples. Clearly, our `sensitive- samples` are very sensitive to model changes compared with randomly selected samples and sensitive-samples generated by SSFDNN [19]. This is because in an NNTA attack, the adversary can carefully modify certain parameters to make the model correctly classify in most cases, which is difficult to be detected by using randomly selected original samples. In addition, although [19] designs an efficient way to generate sensitive-samples. However, this method limits the range of samples to ensure the indistinguishability between sensitive-samples and original samples, which significantly weakens the sensitivity of `sensitive-samples` to model's changes.

Table 6(see APPENDIX) further shows the detection accuracy with under different values of $k$, where the symbol # denotes the number of query samples. We can find that SecureDL can achieve at least 90% accuracy even if returning the top-1 result. This is mainly due to the superiority of our `sensitive-samples` generation algorithm. Since all generated samples will be encrypted before being uploaded to the server, compared to SSFDNN [19], we do not limit the range of `sensitive-samples` to ensure the indistinguishability between `sensitive-samples` and original samples. As a result, the feasible domain of **Algorithm 2** will be greatly increased and we can get better `sensitive-samples`.

[5]Please note that such a strong adversary does not exist in SecureDL. Here we just demonstrate the sensitivity of sensitive samples to model changes under this fictitious assumption. Obviously, if SecureDL shows good detection accuracy under the strong adversary model, it will perform better in the weak adversary model.

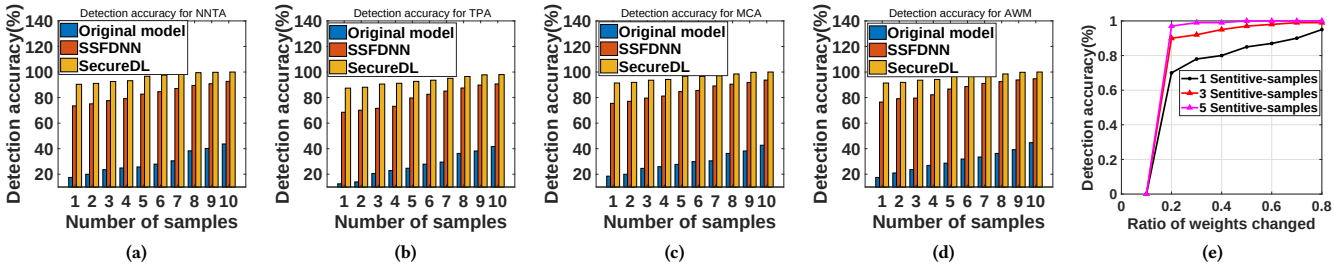


**Figure 4: Detection accuracy for different attacks.**

*5.2.2 Targeted Poisoning Attack.* Here we consider the error-specific attack, which means that the goal of the adversary is to make the model misclassify the inputs from the target class to a specific class that he desires. Specifically, the adversary modifies the model parameters by exploiting data poisoning method to retaining the model, and compromises the model to incorrectly classify "STOP" sign into "speed limit 50km/h" but behave normally in other cases. We conduct our experiment in a CNN model with GTSRB [31] dataset (contain more than 40 classes of traffic signs). The detailed experiment configuration is shown in Table 4(see APPENDIX).

Figure 4(b) shows the detection accuracy for the targeted poisoning attack with the different number of samples, where we also require the server to return the Top-1 result to the client. It is clear that the detection accuracy of our `sensitive-samples` is significantly better than SSFDNN [19] and randomly selected samples. Moreover, we also record the detection accuracy under different values of $k$ (shown in Table 7 of APPENDIX). Since the targeted poisoning attack considered in our experiment is less noticeable than the Trojan attack, the detection accuracy of the three types of samples is slightly reduced. However, compared to the other two samples, our `sensitive-samples` also show excellent detection performance.

*5.2.3 Model Compression Attack.* In this type of attack, the adversary tries to reduce the size of the outsourced model as much as possible without significantly affecting the model's inference accuracy, thus effectively reducing the storage and computation overhead of the cloud. In our experiments, we simulate the adversary to compress the model to a quarter of the original model, while the inference accuracy is only reduced by 3.7%. all the experiments are conducted over a CNN model with CIFAR [46] dataset. The CIFAR dataset consists of 6000 $32 \times 32$ images categorized into 10 classes.

As shown in Figure 4(c), we can see that the detection accuracy of our `sensitive-samples` is also significantly better than SSFDNN [19] and randomly selected samples. Moreover, we also record the detection accuracy under the different number of returned results (shown in Table 8 of APPENDIX). Obviously, compared to the other two samples, our `sensitive-samples` also show excellent detection performance.

*5.2.4 Arbitrary Weights Modification.* In this type of attack, the adversary (such as the cloud server) can launch attacks by arbitrarily modifying a subset of the weights. To simulate this, we assume the adversary modifies the weights with ratio $r$ (from 0.1% to 80%), and then we record the detection accuracy under the different ratios of changed weights. All the targeted weights are modified by adding standard Gaussian noise.

Figure 4(d) shows that our `sensitive- samples` are very sensitive to model changes compared with SSFDNN [19] and randomly selected samples. In order to show the relationship between weight changes and detection accuracy, we test the detection accuracy of `sensitive-samples` under different proportional parameter changes. As shown in Figure 4(e), the greater ratio of model parameters changed, the easier it is to be detected. Also, we record the detection accuracy under the different number of returned results (shown in Table 9 of APPENDIX). Compared to the other two samples, our `sensitive-samples` also show excellent detection performance.

## 5.3 Overhead Evaluation

we select datasets from UC Irvine Machine Learning Repository, and data in batch from MNIST [10] dataset to test the performance of SecureDL. We test the overhead of the proposed scheme under a custom CNN network, which consists of two convolutional layers ( containing 20 feature maps and 50 feature maps, respectively), two average pooling layer and two fully connected layers (256 and 10 neurons, respectively). Since HELib supports Single-Instruction-Multiple-Data (SIMD) [16, 28] techniques, which can achieve the time to run a batch (such as 282, 576, 1420, 3668, 6144, 8912, etc) of instances is equivalent to the time to run a single instance, we also use it to improve the classification efficiency (For more detail, please see Figure 5 in APPENDIX).

**Table 2: Running time of classification over encrypted data**

| datasets | Classification(s) | #C* | Noise Reduction(s) |
|---|---|---|---|
| Breast tissues | 181.23 | 14 | 115.84 |
| Crab | 174.56 | 14 | 112.07 |
| Ovarian | 214.33 | 14 | 138.06 |
| Wine | 182.57 | 14 | 116.26 |
| Climate | 193.64 | 14 | 124.71 |
| Fertility | 183.26 | 14 | 117.26 |

As shown in Table 2 (where#C denotes the number of communication between the user and the server for noise reduction), we require SecureDL to classify over encrypted data (i.e., *Breast tissues*, *Crab*, *Ovarian*, *Wine*,*Climate*, and *Fertility*, shown in Table 5

of APPENDIX), where the batch size is set 576. We can observe that SecureDL only needs at most 0.33 seconds to process the classification of an instance. This is mainly due to the following two reasons. First, we convert the complex activation function into a low-degree polynomial, which is advantageous for LHE to perform fewer calculations in the ciphertext. The other is the use of SIMD (see Figure 5 in APPENDIX for more details), which can process multiple ciphertexts in parallel, thereby accelerating the efficiency of ciphertext calculations. For the comprehensiveness of the experiment, we further increase the complexity of the selected CNN network and record the running time of SecureDL in different data sets. For more details, please refer to Table 10 to Table 12 in APPENDIX (where #CL* denotes the number of convolutional layers added to the benchmark model).

**Table 3: Performance compared with existing approaches**

| Dataset | Performance | SecureDL | Cryptonets | SecureML |
|---|---|---|---|---|
| MNIST | Accuracy | 96.23% | 95.93% | 91.67% |
| | Running time | 319(s) | 803(s) | 10648(s) |
| | Data transfer | 330.7(MB) | 776(MB) | 2.46(GB) |

Next, we test the cost of SecureDL and compare it with the state-of-the-art approaches Cryptonets [14] and SecureML [35], where we use the same experimental configuration (including hardware, data set and CNN model) to implement these three schemes. Cryptonets [14] is very similar to SecureDL. It utilizes FHE to encrypt all of the user's data and exploits square function to approximate the activation function. As shown in Table 3, where we set the batch of ciphertext with size 8192 (the same as works [14]), since Cryptonets does not consider how to convert the activation function into a low-degree polynomial and uses bootstrapping[65] to execute noise reduction, compared to SecureDL, it needs to perform more computations and generate larger size ciphertext to complete the same classification task. As a result, its computation and communication overhead is more than twice that of SecureDL.

SecureML [35] is the first work based on SMC. In their proposed approach, the data owner shares the data with the two servers which run a deep learning model using two-party computation (2PC) technique. As shown in Table 3, our SecureDL is significantly better than SecureML [35] in terms of computation and communication overhead. For example, to complete the same classification task over the MNIST dataset, our solution only needs to run 319(s) and transfer 330.7(MB) of data, whereas SecureML's cost is 10648(s) and 2.46(GB), respectively.

## 6 CONCLUSION

In this paper, we have proposed SecureDL, which can verify the integrity of DNNs model outsourced to the cloud, while protecting user's privacy in the inference process. we prove that our SecureDL can be applied to general neural networks, with no assumptions on DNNs architecture, hyper-parameters and training methods. Extensive experiments also demonstrated the superior performance of SecureDL in terms of inference accuracy, detection accuracy and overhead. In the further works, we intend to further improve the inference accuracy, and find ways to reduce the computation and communication overhead of SecureDL.

## ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YFB0802000, the National Natural Science Foundation of China under Grants 62020106013, 61972454, 61802051, 61772121, and 61728102, Sichuan Science and Technology Program under Grants 2020JDTD0007 and 2020YFG0298, the Peng Cheng Laboratory Project of Guangdong Province PCL2018KP004.

## REFERENCES

[1] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. 2018. cpSGD: Communication-efficient and differentially-private distributed SGD. In *Advances in Neural Information Processing Systems*. 7564–7575.

[2] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.

[3] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. 2019. Garbled Neural Networks are Practical. *IACR Cryptology ePrint Archive* 2019 (2019), 338.

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

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

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* 6, 3 (2014), 13–24.

[7] Yuhan Cai and Raymond Ng. 2004. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proceedings of the ACM SIGMOD*. 599–610.

[8] Jean-Paul Calvi and Norman Levenberg. 2008. Uniform approximation by discrete least squares polynomials. *Journal of Approximation Theory* 152, 1 (2008), 82–100.

[9] Chaochao Chen, Liang Li, Wenjing Fang, Jun Zhou, Li Wang, Lei Wang, Shuang Yang, Alex Liu, and Hao Wang. 2020. Secret Sharing based Secure Regressions with Applications. *arXiv preprint arXiv:2004.04898* (2020).

[10] Xuhui Chen, Jinlong Ji, Lixing Yu, Changqing Luo, and Pan Li. 2018. SecureNets: Secure Inference of Deep Neural Networks on an Untrusted Cloud. In *Asian Conference on Machine Learning*. 646–661.

[11] Rida T Farouki. 2012. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design* 29, 6 (2012), 379–419.

[12] AD Gadjiev and C Orhan. 2002. Some approximation theorems via statistical convergence. *The Rocky Mountain Journal of Mathematics* (2002), 129–138.

[13] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P Smart. 2012. Ring switching in BGV-style homomorphic encryption. In *International Conference on Security and Cryptography for Networks*. Springer, 19–37.

[14] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the ICML*. 201–210.

[15] S Dov Gordon and Jonathan Katz. 2006. Rational secret sharing, revisited. In *International Conference on Security and Cryptography for Networks*. Springer, 229–241.

[16] Robert J Gove, Keith Balmer, Nicholas K Ing-Simmons, and Karl M Guttag. 1993. Multi-processor reconfigurable in single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) modes and method of operation. US Patent 5,212,777.

[17] Shai Halevi and Victor Shoup. 2014. Algorithms in helib. In *Annual Cryptology Conference*. Springer, 554–571.

[18] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. 2018. Mlcapsule: Guarded offline deployment of machine learning as a service. *arXiv preprint arXiv:1808.00590* (2018).

[19] Zecheng He, Tianwei Zhang, and Ruby Lee. 2019. Sensitive-Sample Fingerprinting of Deep Neural Networks. In *Proceedings of the IEEE CVPR*. 4729–4737.

[20] Zecheng He, Tianwei Zhang, and Ruby B Lee. 2018. VerIDeep: Verifying Integrity of Deep Neural Networks through Sensitive-Sample Fingerprinting. *arXiv preprint arXiv:1808.03277* (2018).

[21] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. 2018. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018), 123–142.

[22] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *proceedings of the ACM CCS*. 603–618.

[23] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *Proceedings of the ACM CCS*. 603–618.

[24] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[25] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. 2018. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *proceedings of the IEEE Security and Privacy*. 19–35.

[26] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *Proceedings of the {USENIX} Security*. 1651–1669.

[27] Julien Keuffer, Refik Molva, and Hervé Chabanne. 2018. Efficient Proof Composition for Verifiable Computation. In *European Symposium on Research in Computer Security*. Springer, 152–171.

[28] Yuyun Liao and David B Roberts. 2002. A high-performance and low-power 32-bit multiply-accumulate unit with single-instruction-multiple-data (SIMD) feature. *IEEE Journal of Solid-State Circuits* 37, 7 (2002), 926–931.

[29] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of ACM CCS*. 619–631.

[30] Qi Liu, Tao Liu, Zihao Liu, Yanzhi Wang, Yier Jin, and Wujie Wen. 2018. Security analysis and enhancement of model compressed deep learning systems under adversarial attacks. In *Proceedings of the Asia and South Pacific Design Automation Conference*. IEEE, 721–726.

[31] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *proceedings of the NDSS*. 309–326.

[32] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *Proceedings of the IEEE ICCD*. 45–48.

[33] Changxue Ma, Yves Kamp, and Lei F Willems. 1994. A Frobenius norm approach to glottal closure detection from the speech signal. *IEEE Transactions on Speech and Audio Processing* 2, 2 (1994), 258–265.

[34] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the ACM CCS*. 35–52.

[35] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *proceedings of IEEE Security and Privacy*. 19–38.

[36] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the IEEE Security and Privacy*. IEEE, 582–597.

[37] NhatHai Phan, Yue Wang, Xintao Wu, and Dejing Dou. 2016. Differential Privacy Preservation for Deep Auto-Encoders: an Application of Human Behavior Prediction.. In *AAAI*, Vol. 16. 1309–1316.

[38] NhatHai Phan, Xintao Wu, Han Hu, and Dejing Dou. 2017. Adaptive laplace mechanism: Differential privacy preservation in deep learning. In *Proceedings of the IEEE ICDM*. 385–394.

[39] Tran Thi Phuong et al. 2019. Privacy-preserving deep learning via weight transmission. *IEEE Transactions on Information Forensics and Security* 14, 11 (2019), 3003–3015.

[40] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. {XONN}: XNOR-based Oblivious Deep Neural Network Inference. In *Proceedings of the {USENIX} Security*. 1501–1518.

[41] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the ACM AsiaCCS*. 707–721.

[42] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the Annual Design Automation Conference*. ACM, 21–26.

[43] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*. 6103–6113.

[44] Jie Shen. 1994. Efficient spectral-Galerkin method I. Direct solvers of second-and fourth-order equations using Legendre polynomials. *SIAM Journal on Scientific Computing* 15, 6 (1994), 1489–1505.

[45] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *Proceedings of the ACM CCS*. 1310–1321.

[46] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. 2017. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*. 3517–3529.

[47] Ayush Tewari, Michael Zollhofer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Christian Theobalt. 2017. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In *Proceedings of the IEEE ICCV*. 1274–1283.

[48] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and secure DNN inference. *arXiv preprint arXiv:1810.00602* (2018).

[49] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. *arXiv preprint arXiv:1806.03287* (2018).

[50] Di Wang, Minwei Ye, and Jinhui Xu. 2017. Differentially private empirical risk minimization revisited: Faster and more general. In *Advances in Neural Information Processing Systems*. 2722–2731.

[51] Honggang Wang, Raghu Pasupathy, and Bruce W Schmeiser. 2013. Integer-ordered simulation optimization using R-SPLINE: Retrospective search with piecewise-linear interpolation and neighborhood enumeration. *ACM Transactions on Modeling and Computer Simulation* 23, 3 (2013), 1–24.

[52] Xi Wu, Fengan Li, Arun Kumar, Kamalika Chaudhuri, Somesh Jha, and Jeffrey Naughton. 2017. Bolt-on differential privacy for scalable stochastic gradient descent-based analytics. In *Proceedings of the ACM International Conference on Management of Data*. 1307–1322.

[53] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. 2014. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181* (2014).

[54] Guowen Xu, Hongwei Li, Yuanshun Dai, Kan Yang, and Xiaodong Lin. 2019. Enabling Efficient and Geometric Range Query with Access Control over Encrypted Spatial Data. *IEEE Transactions on Information Forensics and Security* 14, 4 (2019), 870–885.

[55] Guowen Xu, Hongwei Li, Sen Liu, Mi Wen, and Rongxing Lu. 2019. Efficient and Privacy-preserving Truth Discovery in Mobile Crowd Sensing Systems. *IEEE Transactions on Vehicular Technology* 68, 4 (2019), 3854–3865.

[56] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. 2019. VerifyNet: Secure and Verifiable Federated Learning. *IEEE Transactions on Information Forensics and Security* (2019).

[57] Guowen Xu, Hongwei Li, and Rongxing Lu. 2018. POSTER:Practical and Privacy-Aware Truth Discovery in Mobile Crowd Sensing Systems. In *Proceedings of ACM CCS*. 2312–2314.

[58] G. Xu, H. Li, H. Ren, X. Lin, and X. S. Shen. 2020. DNA Similarity Search with Access Control over Encrypted Cloud Data. *IEEE Transactions on Cloud Computing* (2020).

[59] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng. 2019. Data Security Issues in Deep Learning: Attacks, Countermeasures and Opportunities. *IEEE Communications Magazine* 57, 11 (2019), 116–123.

[60] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. Deng. 2020. Privacy-Preserving Federated Deep Learning with Irregular Users. *IEEE Transactions on Dependable and Secure Computing* (2020). https://doi.org/10.1109/TDSC.2020.3005909

[61] Masashi Yamane and Keiichi Iwamura. 2020. Secure and Efficient Outsourcing of Matrix Multiplication based on Secret Sharing Scheme using only One Server. In *Proceedings of the IEEE CCNC*. IEEE, 1–6.

[62] L Yu, L Liu, C Pu, M E Gursoy, and S Truex. 2019. Differentially Private Model Publishing for Deep Learning. In *proceedings of the IEEE Security and Privacy*. 309–326.

[63] Ghodsi Zahra, Gu Tianyu, and Garg Siddharth. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*. 4672–4681.

[64] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, and Jiming Chen. 2018. Calm: Consistent adaptive local marginal for marginal release under local differential privacy. In *Proceedings of the ACM CCS*. 212–229.

[65] Hongchao Zhou and Gregory Wornell. 2014. Efficient homomorphic encryption on integer vectors and its applications. In *Information Theory and Applications Workshop*. IEEE, 1–9.

[66] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the ICML*. 928–936.

# APPENDIX

## 1.1 Proof of Theorem 1

*Proof.* Without loss of generality, we use $[0, 1]$ to replace the closed interval $[a, b]$. Then, we define the following mapping:

$$B_n : X \to Y$$

$$M(x) \mapsto B_n(M, x) = \sum_{k=0}^{k=n} M(\frac{k}{n}) C_n^k x^k (1-x)^{n-k}$$

where $X$ represents the collection of all continuous functions, and $Y$ represents the entire set of polynomials. $B_n(M, x)$ is a **Bernstein** polynomial, which is the image of $M \in X$ under the mapping $B_n$. From the definition, we can derive that $B_n$ has the following properties.

(1) $B_n$ is a linear map. For any functions $M, G \in X$ and $\alpha, \beta \in R$, *we have* $B_n(\alpha M + \beta G, x) = \alpha B_n(M, x) + \beta B_n(G, x)$.
(2) $B_n$ is monotonic. For any functions $M, G \in X$, if $M(t) \leq G(t)$ is established for every $t \in [0, 1]$, we have $B_n(M, x) \leq B_n(G, x)$ is established for every $x \in [0, 1]$.

Based on above properties, we claim that any continuous function can be approximated by the **Bernstein** polynomial with any given error bound $\epsilon$. For simplicity, here we take a continuous functions $M(t) = (t-s)^2$ as an example to verify our statement, where $s$ is a constant. Specifically, the image of $(t-s)^2$ under the mapping $B_n$ is as follows.

$$B_n((t-s)^2, x) = B_n(t^2, x) - 2sB_n(t, x) + s^2 B_n(1, x) \tag{7}$$

where

$$B_n(1, x) = \sum_{k=0}^{k=n} C_n^k x^k (1-x)^{n-k} = 1$$

$$B_n(t, x) = \sum_{k=0}^{k=n} \frac{k}{n} C_n^k x^k (1-x)^{n-k} = x[x + (1-x)]^n = x$$

$$B_n(t^2, x) = \sum_{k=0}^{k=n} \frac{k^2}{n^2} C_n^k x^k (1-x)^{n-k} = \frac{n-1}{n} x^2 + \frac{x}{n}$$

We know that the function $M(t)$ is bounded since it is continuous in the interval $[0, 1]$. Therefore, for any element $t \in [0, 1]$, there is a positive number $D$ satisfying $|M(t)| \leq D$. Further, based on the Cantor Theorem, for any given number $\epsilon > 0$ and $t, s \in [0, 1]$, we have $|M(t) - M(s)| \leq \frac{\epsilon}{2}$, where $|t - s| \leq \delta$ and $\delta > 0$. Contrarily, if $|t - s| \geq \delta$, we have

$$|M(t) - M(s)| \leq 2D \leq \frac{2D}{\delta^2}(t-s)^2$$

Therefore, for any $t, s \in [0, 1]$, we have

$$-\frac{\epsilon}{2} - \frac{2D}{\delta^2}(t-s)^2 \leq M(t) - M(s) \leq \frac{\epsilon}{2} + \frac{2D}{\delta^2}(t-s)^2 \tag{8}$$

Based on the Eqn.(1), the above formulas can be further decomposed as follows.

$$\begin{aligned} &-\frac{\epsilon}{2} - \frac{2D}{\delta^2}[\frac{x-x^2}{n} - (x-s)^2] \\ &\leq B_n(M, x) - f(s) \\ &\leq \frac{\epsilon}{2} + \frac{2D}{\delta^2}[\frac{x-x^2}{n} - (x-s)^2] \end{aligned} \tag{9}$$

Let $s = x$. Because $x(1-x) < \frac{1}{4}$ for all $x \in [0, 1]$, we have

$$\left| \sum_{k=0}^{k=n} M(\frac{k}{n}) C_n^k x^k (1-x)^{n-k} - M(x) \right| \leq \frac{\epsilon}{2} + \frac{D}{2n\delta^2} \tag{10}$$

Therefore, when $n > N, N = \lceil \frac{D}{\epsilon\delta^2} \rceil$, we have

$$\left| \sum_{k=0}^{k=n} M(\frac{k}{n}) C_n^k x^k (1-x)^{n-k} - M(x) \right| \leq \epsilon \tag{11}$$

Hence, we prove that the **Bernstein** polynomial can be used to approximate any continuous function with given error bound $\epsilon$.

## 1.2 Least square approximation algorithm

Given a continuous function $M(x)$ on the finite interval $[a, b]$, the goal of Least Square Approximation Algorithm is to find an optimal polynomial $p(x)$ that satisfies the following definition.

$$|M(x) - p(x)|_2^2 = \min_{s(x) \in \psi} \int_a^b \rho(x)[M(x) - s(x)]^2 dx \tag{12}$$

where $\{\psi_n(x) | n = 1, \cdots N\}$ is a set of linearly independent polynomial function, and $\rho(x)$ is the weight function. $s(x)$ can be represented as $s(x) = a_0\psi_0(x) + a_1\psi_1(x) + \cdots + a_n\psi_n(x)$. Then, we can get the $p(x)$ by solving the following multivariate function.

$$I(a_0, a_1, \cdots, a_n) = \min \int_a^b \sum_{j=0}^{j=n} \rho(x)[M(x) - a_j\psi_j(x)]^2 dx \tag{13}$$

Based on the necessary condition of solving extreme value in multivariate function, let $\frac{\partial I}{\partial a_k} = 0, k = 0, 1, 2, \cdots, N$, we have

$$\frac{\partial I}{\partial a_k} = 2 \int_a^b \sum_{j=0}^{j=n} \rho(x)[M(x) - a_j\psi_j(x)]\psi_k(x) dx = 0 \tag{14}$$

Based on the definition of inner product, we have

$$\sum_{j=0}^{j=n} a_j(\psi_k(x), \psi_j(x)) - (M(x), \psi_k(x)) = 0 \tag{15}$$

where $(\psi_k(x), \psi_j(x)) = \int_a^b \rho(x)\psi_k(x)\psi_j(x)dx$ and $(M(x), \psi_k(x))$ $= \int_a^b \rho(x)M(x)\psi_k(x)dx$, respectively. Therefore, we have

$$\sum_{j=0}^{j=n} a_j(\psi_k(x), \psi_j(x)) = (M(x), \psi_k(x)), k = (0, 1, 2, \cdots, N) \tag{16}$$

The above equations can be extended to the following system of equations.

$$\begin{pmatrix} (\psi_0, \psi_0) & (\psi_0, \psi_1) & \cdots & (\psi_0, \psi_N) \\ (\psi_1, \psi_0) & (\psi_1, \psi_1) & \cdots & (\psi_1, \psi_N) \\ & \cdots & \cdots & \cdots \\ (\psi_N, \psi_0) & (\psi_N, \psi_1) & \cdots & (\psi_N, \psi_N) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_n \end{pmatrix} = \begin{pmatrix} (\psi_0, M(x)) \\ (\psi_1, M(x)) \\ \cdots \\ (\psi_n, M(x)) \end{pmatrix}$$

Since $\{\psi_n(x) | n = 1, \cdots N\}$ is a set of linearly independent polynomial function, above equation set has unique solution $(a_0^*, a_1^*, \cdots a_N^*)$. Hence, we can calculate the least square approximation polynomial $p(x) = a_0^*\psi_0(x) + a_1^*\psi_1(x) + \cdots a_N^*\psi_N(x)$ satisfying the definition

$$|M(x) - p(x)|_2^2 = \min_{s(x) \in \psi} \int_a^b \rho(x)[M(x) - s(x)]^2 dx \tag{17}$$

**Table 4: Experimental Environment for Detection Accuracy under Different Attacks**

| | Dataset | Task | Model | Total layers | Convolution layers | Fully connected layers | Attack technique |
|---|---|---|---|---|---|---|---|
| NNTA | VGG-Face[47] | Face classification | VGG-16 | 18 | 14 | 4 | Modify parameters with trigger |
| TPA | GTSRB[31] | Traffic sign classification | CNN | 9 | 7 | 2 | Retrain model with data poisoning |
| MCA | CIFAR-10[46] | Image classification | CNN | 9 | 7 | 2 | Compress the original model |
| AWM | AT&T[10] | Face classification | MLP | 1 | 0 | 1 | Arbitrary modification |

## 1.3 Experimental configuration and results

**Table 5: Datasets selected for inference accuracy**

| datasets | Instances | Features | Classes |
|---|---|---|---|
| Breast tissues | 106 | 9 | 6 |
| Crab | 200 | 6 | 2 |
| Ovarian | 216 | 100 | 2 |
| Wine | 100 | 10 | 2 |
| Climate | 540 | 17 | 2 |
| Fertility | 100 | 10 | 2 |

**Table 6: Detection Accuracy under NNTA**

| Solutions | # of samples | Top-1 | Top-3 | Top-5 |
|---|---|---|---|---|
| SecureDL | 1 | 90.4% | 92.3% | 96.6% |
| | 3 | 91.2% | 94.5% | 100% |
| | 5 | 95.7% | 100% | 100% |
| SSFDNN | 1 | 73.1% | 77.8% | 82.8% |
| | 3 | 83.1% | 89.3% | 94.2% |
| | 5 | 87.4% | 91.2% | 100% |
| Original model | 1 | 17.5% | 23.6% | 25.7% |
| | 3 | 19.2% | 29.4% | 35.9% |
| | 5 | 23.4% | 34.1% | 39.5% |

**Table 7: Detection Accuracy under TPA**

| Solutions | # of samples | Top-1 | Top-3 | Top-5 |
|---|---|---|---|---|
| SecureDL | 1 | 87.4% | 90.1% | 92.3% |
| | 3 | 89.1% | 92.4% | 100% |
| | 5 | 91.7% | 100% | 100% |
| SSFDNN | 1 | 68.3% | 71.9% | 79.7% |
| | 3 | 80.1% | 84.6% | 91.2% |
| | 5 | 85.3% | 89.2% | 97.4% |
| Original model | 1 | 12.1% | 20.3% | 24.3% |
| | 3 | 16.3% | 24.7% | 30.1% |
| | 5 | 21.4% | 30.3% | 34.1% |

**Table 8: Detection Accuracy under MCA**

| Solutions | # of samples | Top-1 | Top-3 | Top-5 |
|---|---|---|---|---|
| SecureDL | 1 | 91.6% | 93.7% | 96.8% |
| | 3 | 93.2% | 100% | 100% |
| | 5 | 95.9% | 100% | 100% |
| SSFDNN | 1 | 75.2% | 79.8% | 84.2% |
| | 3 | 83.7% | 89.7% | 95.2% |
| | 5 | 88.4% | 94.6% | 100% |
| Original model | 1 | 18.7% | 24.3% | 27.9% |
| | 3 | 22.7% | 32.4% | 37.3% |
| | 5 | 24.2% | 35.3% | 40.3% |

**Table 9: Detection Accuracy under AWM**

| Solutions | # of samples | Top-1 | Top-3 | Top-5 |
|---|---|---|---|---|
| SecureDL | 1 | 92.5% | 94.6% | 97.1% |
| | 3 | 93.6% | 100% | 100% |
| | 5 | 96.9% | 100% | 100% |
| SSFDNN | 1 | 77.2% | 82.8% | 89.2% |
| | 3 | 85.7% | 89.9% | 96.2% |
| | 5 | 89.4% | 95.6% | 100% |
| Original model | 1 | 19.3% | 25.3% | 28.6% |
| | 3 | 22.9% | 33.6% | 38.7% |
| | 5 | 29.6% | 37.7% | 42.6% |

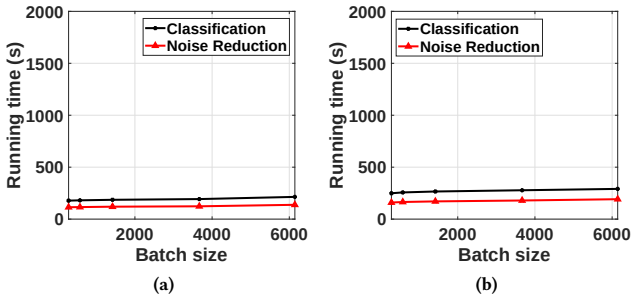


**Figure 5: Running time with SIMD.**

The running time of using SIMD technology to perform classification tasks is shown in Figure 5, where we require SecureDL to classify *Breast tissues* in the ciphertext. For Figure 5(a), the selected CNN network consists of two convolutional layers ( containing 20 feature maps and 50 feature maps, respectively), two average pooling layer and two fully connected layers (256 and 10 neurons, respectively). We can see that with the increase of batch size, the running time of SIMD basically keeps a constant. Moreover, SecureDL only takes 0.63 seconds to return results for a single query, even when the batch size is taken to be 282. For Figure 5(b), we increase the complexity of the CNN network used for classification, i.e., the selected CNN network consists of three convolutional layers ( containing oner 20 feature maps and two 50 feature maps, respectively), two average pooling layer and two fully connected layers (256 and 10 neurons, respectively). SecureDL also only takes 0.88 seconds to return results for a single query, when the batch size is taken to be 282. By increasing the batch size to 6144, the running time decreases to 0.04 second per instance. Therefore, SIMD technology can effectively improve the ability to process ciphertext.

It is worth mentioning that larger batch sizes inevitably increase the size of the network, even it can speed up inference. Therefore, there is a trade-off between memory and runtime, and we should choose the appropriate batch size based on the size of the dataset.

**Table 10: Running time of classification over encrypted dataset of Breast tissues**

| #CL* | Classification(s) | #C | Noise Reduction(s) |
|---|---|---|---|
| 1 | 201.23 | 21 | 140.22 |
| 2 | 220.56 | 28 | 153.03 |
| 3 | 241.34 | 35 | 166.91 |
| 4 | 263.63 | 42 | 179.15 |
| 5 | 283.17 | 49 | 190.23 |

**Table 11: Running time of classification over encrypted dataset of Crab**

| #CL* | Classification(s) | #C | Noise Reduction(s) |
|---|---|---|---|
| 1 | 193.36 | 21 | 124.15 |
| 2 | 221.17 | 28 | 153.46 |
| 3 | 240.54 | 35 | 166.29 |
| 4 | 261.12 | 32 | 178.61 |
| 5 | 283.67 | 49 | 191.17 |

The running time of SecureDL in different data sets is shown Table 10 to Table 12, where the batch size is set 576. The benchmark model consists of two convolutional layers ( containing 20 feature maps and 50 feature maps, respectively), two average pooling layer and two fully connected layers (256 and 10 neurons, respectively). Then, we increase the complexity of the benchmark model (by adding different numbers of convolutional layers containing 50 feature maps) and record the classification time under the changed model. We can see that as the number of added convolutional layers rises, it will increase the running time of SecureDL in the ciphertext, and also require more interaction between the server and the user to implement the noise reduction. However, SecureDL also only needs at most 0.61 seconds to process the classification of an instance, even if the additional 5 convolutional layers are added to the benchmark model (shown in Table 12).

**Table 12: Running time of classification over encrypted dataset of Ovarian**

| #CL* | Classification(s) | #C | Noise Reduction(s) |
|---|---|---|---|
| 1 | 233.63 | 21 | 150.17 |
| 2 | 251.16 | 28 | 164.39 |
| 3 | 272.64 | 35 | 178.38 |
| 4 | 294.70 | 42 | 190.29 |
| 5 | 314.63 | 49 | 202.03 |