Manto: A Practical and Secure Inference Service of Convolutional Neural Networks for IoT

Ke Cheng, Jiaxuan Fu, Yulong Shen, *Member, IEEE*, Haichang Gao, *Member, IEEE*, Ning Xi, *Member, IEEE*, Zhiwei Zhang, *Member, IEEE*, and Xinghui Zhu

***Abstract*—As convolutional neural networks (CNNs) exhibit remarkable performance in various inference tasks, it is increasingly important to enable Internet of Things (IoT) devices to perform CNN-based applications. Many companies provide their carefully trained neural networks as inference services for resource-constrained clients (e.g., IoT devices). However, the use of CNN inference in many IoT applications raises privacy concerns. Cryptographic inference services provide a way to perform neural inference efficiently and, at the same time, preserve both the privacy of the client's input data and the server's proprietary model. Unfortunately, the existing solutions incur severe latency costs, stemming mostly from nonlinear activations such as ReLUs, which make them still unsuitable for deployment in real IoT devices. In this article, we propose Manto, a secure inference system of CNNs for IoT. Manto makes the following two specific efforts by combining the insights of machine learning and cryptography. First, we customize different quadratic activation functions to replace specific ReLU layers and further propose a sliding-window-based fine-tuning method to produce CNN models involving no or few ReLUs. These techniques allow us to speedup cryptographic inference and guarantee inference accuracy. Second, we develop a series of cryptographic protocols that support ReLU activations and its approximation variants (i.e., polynomial activations), which purely rely on the lightweight secret sharing techniques in the online execution and can well cope with the above-mentioned optimized CNN models in the ciphertext domain. Our experimental results show Manto obtains state-of-the-art performance, reducing online inference latency by 66.2% $\sim$ 87.7% over prior works on CIFAR-100 and TinyImageNet data sets.**

***Index Terms*—Convolutional neural network (CNN), cryptographic protocol, Internet of Things (IoT) device, ReLU reduction, secure inference.**

Manuscript received 2 November 2022; accepted 28 February 2023. Date of publication 3 March 2023; date of current version 8 August 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3101100; in part by the National Natural Science Foundation of China under Grant 62220106004 and Grant 61972306; in part by the Major Research Plan of the National Natural Science Foundation of China under Grant 92267204; in part by the Key Research and Development Program of Shaanxi under Grant 2022KXJ-093 and Grant 2021ZDLGY07-05; in part by the Natural Science Basis Research Plan in Shaanxi Province of China under Grant 2022JM-338; in part by the Shandong Provincial Natural Science Foundation under Grant ZR2021LZH006; and in part by the Fundamental Research Funds for the Central Universities under Grant XJSJ23040. *(Corresponding author: Yulong Shen.)*

Ke Cheng, Jiaxuan Fu, Yulong Shen, Haichang Gao, Zhiwei Zhang, and Xinghui Zhu are with the School of Computer Science and Technology, Xidian University, Xi'an 710071, Shaanxi, China (e-mail: chengke@xidian.edu.cn; jxfu_2099@stu.xidian.edu.cn; ylshen@mail.xidian.edu.cn; hchgao@xidian.edu.cn; zwzhang@xidian.edu.cn; xhzhu@stu.xidian.edu.cn).

Ning Xi is with the School of Cyber Engineering, Xidian University, Xi'an 710071, Shaanxi, China (e-mail: nxi@xidian.edu.cn).

Digital Object Identifier 10.1109/JIOT.2023.3251982

I. Introduction

THE CURRENT boom in cloud computing and big data is unlocking research potentials on modern machine learning (ML). As a typical success of ML, deep neural networks (DNNs) have witnessed widespread success in various fields, including Internet of Things (IoT). In particular, convolutional neural networks (CNNs), as their better-than-human accuracy across a variety of inference tasks [1], naturally become a solution to deal with the large data streams generated by IoT devices.

There have been significant efforts to develop practical CNN inference mechanisms for IoT devices. For example, many technology companies, such as Google, Microsoft, and Amazon, provide a new computing paradigm namely Machine-Learning-as-a-Service (MLaaS), which can provide ML inference services to these resource-constrained clients (e.g., mobile and IoT devices) with the aid of powerful cloud servers. However, in the setting of MLaaS, cloud servers require access to the clients' input and output data which is often privacy sensitive. Besides, the model stored on the server is under threat of stealing from external attackers as it is an extremely valuable asset [2], [3], [4], [5]. To mitigate these privacy risks, many recent studies [6], [7], [8], [9], [10], [11] have been proposed to realize inference service in a secure manner. A common approach to protecting private data in MLaaS is encrypting the client's input and the server's neural network [7], [12], then the cloud server and the client execute cryptographic protocols on these encrypted data, and return the inference result to the client. Unfortunately, these cryptographic inference protocols are still difficult to apply in practice, especially for IoT devices with the limited computing power. Since these protocols involves many time-consuming cryptographic operations and require a large amount of communication between the user and the service provider [8].

The goal of our work is to provide practical solutions to this conundrum of secure CNN inference for IoT devices. Concretely, consider a client holds a private input x and a server holds the parameters y of a neural network f with a publicly known architecture, we aim to enable the server and the client to interact in such a way that the client eventually obtains the inference result $f(x, y)$ without learning the model parameters y and the server obtains no information about the client's input x.

The challenges of our study are twofold. First, prior protocols [7], [8], [13], [14] heavily rely on garbled circuits

(GCs) [15] to compute nonlinear activations like ReLUs, which make these methods expensive to execute the CNNs on the resource-constrained IoT devices. The second challenge originates from the conflicting requirements of accuracy and efficiency. The existing works [8], [16], [17] attempt to replace ReLU activations with cryptography-friendly activation functions (e.g., quadratic approximations) that are more tractable in a secure manner. But with the increasing scale of the replacement, this method would degrade inference accuracy and even make models fail to train.

Our Contributions: In this article, we present Manto, a secure inference system of CNNs for IoT. To satisfy the practicality requirement, Manto makes efforts in two aspects by combining insights from cryptography and ML. First, it is crucial to replace as many ReLU layers as possible with quadratic approximations that are more tractable by secure computation techniques. As is known to all, direct replacements can cause a dramatic fall in inference accuracy. To achieve a trade-off between accuracy and efficiency, we propose a method to find appropriate quadratic activation functions for the different ReLU layers. And we introduce a sliding-window-based fine-tuning method for optimizing the CNN model's parameters, adapting them to the adjusted architecture. These two techniques enable Manto to use fewer nonlinear activations while maintaining the same level of inference accuracy as original models. Second, it is necessary to reduce the evaluation costs of nonlinear activation itself and its approximation (i.e., quadratic activations). In the currently utilized protocols, however, these activations are still handled by less efficient tools, such as fully homomorphic encryption (FHE) [18] (in CryptoNets [19] and CHET [20]) and GC (in Delphi [8] and SAFENet [13]), which still involves substantial online computation costs and communication overhead. For increased efficiency, we propose a set of secure online/offline protocols in the *mixed* form, which combine secret sharing and additive homomorphic encryptions together to jointly realize activation layers in a secure manner. In consideration of the performance, we employ the lightweight additive secret sharing (ASS) to perform the majority of secure computing operations, and offload the heavy cryptographic operations into the offline stage.

We evaluate Manto across several popular CNN models, including AlexNet [21], VGG-16 [22], and ResNet-18/34 [23]. Typically, our ResNet-34 model achieves 69.2% accuracy on the CIFAR-100 data set using only 8.2K ReLUs and 76.3% accuracy using 98.3K ReLUs, achieving a state-of-the-art accuracy for different ReLU budgets. For evaluating ResNet-34 on TinyImageNet, Manto reduces online inference time by 77.9% and 66.2% compared to the current state-of-the-art: Delphi [8] and DeepReDuce [14].

In summary, the contributions in this article are as follows:

1) We propose Manto, a secure CNN inference system for IoT via a careful co-design of cryptography and ML. This synergy yields a secure, accurate, and efficient solution for practical large-scale neural networks.
2) We propose a quadratic-fitting-function search method and sliding-window-based fine-tuning, to enable Manto to efficiently perform secure inference using fewer nonlinear activations. Thus, Manto yields lightweight DNN models tailored for high-efficiency ciphertext computation on resource-constrained devices.
3) In combination with secret sharing and additive homomorphic encryptions, we present a set of secure mixed protocols to evaluate nonlinear activation layers, that outperform all prior comparable protocols. By the proof-by-simulation technique, we formally prove the security of all protocols against semi-honest adversaries.
4) We implement Manto and do extensive experiments over typical CNN models. The results demonstrate that, compared with the state-of-the-art, Manto obtains competitive performance in terms of both computation and communication costs.

The remainder of this article is organized as follows. Section II presents the related work. Section III gives a brief introduction to CNNs and cryptographic tools used in Manto. The description of the system model and security goals is given in Section IV. In Section V, we describe the model preparing for secure inferences. Then, we propose our 2PC protocols for quadratic activations and ReLU6 activations in Section VI. Section VII gives a theoretical analysis for our 2PC protocols. In Section VIII, we report the experimental results. Finally, this article is concluded in Section IX.

II. Related Work

There has been a flurry of recent works [6], [7], [8], [11], [19], [20], [24], [25], [26] in the area of secure CNN inference. Generally, these works can be divided into two broad categories: those that rely on trusted execution environments (TEEs), and those that rely on cryptography. The TEE-based inference belongs to a distinct technical route, and provides security guarantees that depend on trust in TEE vendors. Therefore, we omit to discuss it here, and instead focus on prior works of secure inference based on cryptography. Table I coarsely compares existing cryptographic inference systems.

FHE-Based Protocols: Several schemes exist in the literature for securely executing the task of neural network inferences using FHE [18]. CryptoNets [19] is the first to study the fully homomorphic inferences of CNN. FHE fails to deal with common nonpolynomial activations in CNN, and CryptoNets replaces the ReLUs with polynomial activations that are more tractable by FHE. More recently, prior works [16], [27] develop this paradigm with optimized polynomial approximations for deeper network models. Since the performance overheads of FHE-based protocols are very costly, these schemes still require tens of minutes to implement a secure inference, which are not suited for the resource-constrained IoT devices. Based on the FHE technique, CHET [20] designs an optimizing compiler that can automatically make the optimal parameter choices for the secure inference to maximize performance. However, all ReLU activations in CHET are required to be replaced with polynomial approximations, which makes this system only adapted to small neural networks and its accuracy is very low for practical large-scale network architectures.

2PC-Based Protocols: To reduce the use of heavyweight cryptographic primitives, such as FHE, there have been a long line of works [6], [7], [8], [11], [12], [28] that realize secure

TABLE I
COMPARISON OF EXISTING CRYPTOGRAPHIC INFERENCE SYSTEMS (#PARTY: THE NUMBER OF COMPUTING PARTICIPANTS, FHE: FULLY HOMOMORPHIC ENCRYPTION, AHE: ADDITIVE HOMOMORPHIC ENCRYPTION, ASS: ADDITIVE SECRET SHARING, BSS: BOOLEAN SECRET SHARING, AND GC: GARBLED CIRCUITS)

System	#Party	Linear Layer Scheme	Nonlinear Layer Scheme	ReLU Reduction
CryptoNets [19], CryptoDL [27], Faster CryptoNets [16], CHET [20]	1	FHE	FHE	Replace with a same quadratic function
SecureML [27]	2	AHE+ASS	GC	Replace with a new nonlinear function
MiniONN [28]	2	AHE	GC	None
CryptoNAS [29]	2	AHE	GC	ReLU pruning, ReLU shuffling
EzPC [12]	2	ASS+AHE	GC	None
Optimizing CNN [9]	2	ASS+FHE	GC	Replace with a same polynomial function
Gazelle [7]	2	AHE	GC	None
CrypTFlow2 [11]	2	AHE+ASS	ASS+BSS	None
MediSC [30]	2	ASS	ASS+BSS	None
Delphi [8]	2	AHE+ASS	GC	Replace with a same quadratic function
SAFENet [13]	2	AHE+ASS	GC	Replace with a same polynomial function
DeepReDuce [14]	2	AHE+ASS	GC	ReLU pruning
ABY3 [31]	3	ASS	ASS+BSS	None
SecureNN [32], CrypTFlow [10], 3-party based scheme [33]	3	ASS	GC	ReLU pruning
Our system	2	AHE+ASS	ASS+BSS	Replace with different quadratic functions, Sliding-window-based Fine-tuning

inference in the secure 2-party computation (2PC) setting. SecureML [6] uses hybrid 2PC protocols to evaluate neural networks, and introduce a 2PC friendly activation functions for efficiency and accuracy. MiniONN [28] uses the additive homomorphic encryption to evaluate linear layers and uses GCs to evaluate nonlinear activation layers. Based on the MiniONN's cryptographic protocols, CryptoNAS [29] proposes optimizations for ReLU-efficient networks by ReLU pruning and ReLU shuffling. In other typical works, including EzPC [12], Optimizing CNN [9], and Gazelle [7], two noncollusive parties implement linear layers by (partially) homomorphic encryption or secret sharing, and compute activation layers by GCs. Unfortunately, due to the introduction of considerable online computation and communication costs, these systems are only demonstrated on small neural networks with tiny data sets, such as MNIST. Recently, CrypTFlow2 [11] obtains the state-of-the-art inference latency and accuracy through a series of subprotocol for nonlinear operations (such as comparison, ReLU, and Maxpool) based on secret sharing rather than GCs. Essentially, CrypTFlow2 reduces the communication costs at the expense of high round complexity, which puts forward stringent requirements to communication conditions. MediSC [30] customizes a secure NN inference system for intelligent medical diagnostic services, which purely relies on the lightweight ASS techniques, free of heavy cryptographic operations. However, this work still takes substantial communication overhead for small neural networks with tiny data sets such as CIFAR-10. To make a tradeoff between speed and accuracy, Delphi [8] replaces ReLU activations with quadratic approximations, and designs a planner that automatically determines the number and the placement of these approximations. For the large-scale models, however, Delphi still needs to execute considerable ReLU functions and simply uses time-consuming GCs to evaluate these nonlinear operations. Based on the Delphi's cryptographic protocols, SAFENet [13] selectively replaces channelwise ReLUs with polynomials, and DeepReDuce [14] proposes a method for ReLU pruning to optimize networks for efficient inference. In the experiment, we will demonstrate that Manto provides a better tradeoff between accuracy and efficiency than SAFENet and DeepReDuce.

Multiparty-Based Protocols: For completeness, we mention the secure inference protocols with multiple parties. These protocols introduce one or more third parties to undertake the dedicated computation. ABY3 [31] constructs a secure 3-party computation (3PC) protocol for ML based on three different forms of secret sharing with the help of a third party. This approach is improved from ABY framework [34] and customizes new protocols for evaluating polynomial functions efficiently. The works in [32] and [33] construct a set of secure 3PC protocols for nonlinear functions that completely avoids the use of GCs. However, these protocols are cumbersome to use by ML developers and are only demonstrated on small DNNs on tiny data sets such as MNIST or CIFAR [10]. CrypTFlow [10] presents a convenient compiler system that translates high-level TensorFlow inference code to 3PC protocols for secure inference services. Although this protocol shows more impressive efficiency than 2PC protocols, it provides weaker security guarantees [11]. The 3PC protocol introduces an extra semi-honest server and needs a security assumption for requiring the noncollusion among the three servers. Since this security assumption is stronger than the assumption for requiring the noncollusion between the two servers, the 3PC protocol provides weaker security guarantees than the 2PC protocols.

III. PRELIMINARIES

A. Convolutional Neural Network

A CNN is an ordered sequence of linear and nonlinear layers, which classifies an input into one of the potential classes.

Linear Layers: The linear layers typically include convolutions, batch normalization (BN), and average-pooling.

TABLE II
NOTATION DESCRIPTIONS

Notations	Definitions
ℓ	the bit-length of the data
ℓ_F	the bit-length of the fractional part
σ	the statistical security parameter
$\langle x \rangle$	the secret-shared form of x, $\langle x \rangle = (\langle x \rangle_0, \langle x \rangle_1)$
$\langle x \rangle_0, \langle x \rangle_1$	the share of x stored in P_0 or P_1
pk/sk	the public and private key pair of the AHE cryptosystem
$[\cdot]_{pk}$	the AHE's encryption with pk
$Dec_{sk}(\cdot)$	the AHE's decryption with sk
$\overline{b}$	the NOT operation on the bit b

Convolution layers convolve the input image with a set of learned filters by computing the inner products of the filters with the raw pixel values in each local region of the image. BN layers normalize features across spatially grouped feature maps so as to improve its generalization capability. Average-pooling layers compute the average value of the pool to perform a down sampling while retaining critical features.

Nonlinear Layers: The nonlinear layers typically include ReLU activation, ReLU6 activation, and max-pooling. ReLU activation layers apply the function $f(x) = \max(0, x)$ to each of its input data for increasing the nonlinearity of the model. ReLU6 activation is derived from ReLU and defined as $\min(\text{ReLU}(x), 6)$, which ensure that post-activation values have magnitude at most 6. Max-pooling layers are defined similarly to average-pooling layers except that its output is the max value of the pool.

B. Cryptographic Tools

We provide a brief overview of the cryptographic primitives used in Manto and summarize the notations in Table II.

1) Additive Secret Sharing and Multiplication Triplets: For an ASS [35], an ℓ-bit value x is shared additively ($\text{Shr}(x)$) in the ring $\mathbb{Z}_{2^\ell}$ between the parties as follows. Party P_i generates a random number $r \in \mathbb{Z}_{2^\ell}$ $(i \in \{0, 1\})$, sets $\langle x \rangle_i = (x - r) \bmod 2^\ell$, and sends r to $P_{i\oplus 1}$, who sets $\langle x \rangle_{i\oplus 1} = r$. To recover ($\text{Rec}(x)$) the value x, the party P_i sends $\langle x \rangle_i$ to the party $P_{i\oplus 1}$ who computes $x = \langle x \rangle_i + \langle x \rangle_{i\oplus 1}$.

Secure arithmetic computations on additively shared data: to compute the sum of two shared values $\langle x \rangle$ and $\langle y \rangle$ without revealing the real values, P_i locally compute $\langle x + y \rangle_i = \langle x \rangle_i + \langle y \rangle_i$. To perform secure multiplication $\langle x \cdot y \rangle = \langle x \rangle \cdot \langle y \rangle$, we leverage Beaver's multiplicative triples [36] of the form $\langle a \rangle, \langle b \rangle, \langle ab \rangle$, which are secret shared among the two parties. P_i locally computes $\langle e \rangle_i = \langle x \rangle_i - \langle a \rangle_i$ and $\langle f \rangle_i = \langle y \rangle_i - \langle b \rangle_i$. Both parties perform $\text{Rec}(e)$ and $\text{Rec}(f)$, then sets $\langle x \cdot y \rangle_i = i \cdot ef + e \cdot \langle a \rangle_i + f \cdot \langle b \rangle_i + \langle ab \rangle_i$. The above Beaver's multiplication protocol is easily expandable to work on matrix multiplication in the secret-shared form. We refer the readers to [6] for details of this protocol.

In order for cryptographic primitives to be compatible with the secure neural network inference, they have to support arithmetic operations on shared decimal numbers. We transform the fixed-point decimal numbers x and y (with at most ℓ_F bits in the fractional part) to integers by letting $x' = 2^{\ell_F} x$ and $y' = 2^{\ell_F} y$. To perform multiplication of x and y, we multiply x' and y' to get the product $z = x'y'$ and truncate the last ℓ_F bits of z such that it still has ℓ_F bits representing the fractional part. The work in [6] shows that this technique also works for secret-shared values.

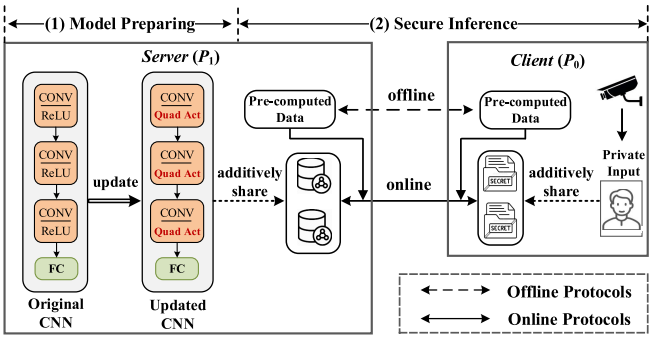


Fig. 1. Workflow of Manto. To show the CNN model briefly, we only depict convolutional (CONV) layers, ReLU activation (ReLU) layers, quadratic activation (Quad Act) layers, and FC layers.

2) Boolean Secret Sharing and B2A Conversion: Boolean secret sharing (BSS) can be regarded as additive sharing in $\mathbb{Z}_2$ in which the addition operation is replaced by the XOR operation ($\oplus$) and multiplication is replaced by the AND operation. So their corresponding $\text{Shr}(\cdot)$ and $\text{Rec}(\cdot)$ algorithms can be defined in a similar manner as the additive sharing.

The functionality of B2A protocol (for boolean sharing to additive sharing conversion) takes boolean shares as input and gives out additive shares of the same value as output. We refer the readers to [37] for details of Boolean sharing and B2A protocol.

3) Additive Homomorphic Public-Key Encryption: Additive homomorphic public-key encryption (AHE) is a cryptosystem supporting additive homomorphic computations on the ciphertexts [38]. More specifically, AHE consists of three algorithms: 1) KeyGen generates a public key pk and a private key sk; 2) on input the public key pk and a message m, the encryption algorithm produces a ciphertext c, denoted by $c = [m]_{pk}$; and 3) on input the private key sk and the ciphertext c, the decryption algorithm outputs the message m, denoted by $m = Dec_{sk}(c)$.

We say that an AHE scheme is correct only if the decryption algorithm outputs the message m with the ciphertext $c = [m]_{pk}$ and the private key sk as inputs. The additive homomorphism of the AHE scheme is denoted by $Dec_{sk}([a]_{pk} \cdot [b]_{pk}) = Dec_{sk}([a + b]_{pk})$.

IV. SYSTEM OVERVIEW

A. System Model and Workflow

We consider a cryptographic inference service over CNNs in the 2PC setting, where the client P_0 (typically, a resource-constrained IoT device) and the cloud server P_1 cooperate to run a secure inference protocol with their respective private inputs. The server's input is a CNN, while the client's input is typically an image used for prediction. At the end P_0 obtains the inference result, namely, the output of the final layer of CNN, whereas P_1 obtains nothing. Fig. 1 depicts Manto's workflow. At a high level, Manto comprises of two phases: 1) model preparing and 2) secure inference.

1) *Model Preparing:* The phase of model preparing runs once per model. In this phase, the server transforms an original CNN model into a high-accuracy model with fewer ReLU activations for secure inference. To show the CNN model briefly, we only depict convolutional layers, activation layers, and fully connected (FC) layers in Fig. 1. We depict quadratic activations as "Quad Act" in the figure.
2) *Secure Inference:* The phase of secure inference runs once per inference, and is further divided into two substages: offline and online. a) In the offline stage, the server and the client run interactive protocols to precompute assisted parameters that are independent of the input of protocol and b) in the online stage, the server and the client first divide their model parameters and private input by the ASS, respectively. Then, they cooperate to run a cryptographic protocol using these shares and the assisted parameters from the offline stage, and generate shares of the inference result. Finally, the server sends its share of the result to the client, who combines the shares to recover the actual result.

B. Threat Model and Security Goals

Manto adopts the semi-honest adversary model which has already been widely used in related work, such as Gazelle [7], Delphi [8], and CrypTFlow2 [11]. More precisely, we assume a semi-honest adversary who can compromise only one of the parties. And the adversary correctly follows the protocol specification, but attempt to learn information about the other party's private input (the network parameters or the input images) by analyzing the transcript of messages received during the execution.

We aim to design a secure inference protocol in which P_0 and P_1 engage under the semi-honest adversary model, such that this protocol enables P_0 and P_1 to learn nothing about each other's private inputs (e.g., all client's private inputs, and the parameters of the server's CNN model) except the architecture of the CNN, and the result of the inference. Here, we make a distinction between the parameters and the architecture of the CNN. By the parameters, we mean the weights and biases of the convolution and the fully connected layers, and the activation functions applied in the layers. By the architecture, we refer to the number of layers, the dimension, and the type of each layer in the neural networks. The model structure is public in most cases, such that the leakage of the structure would not usually affect the server's privacy. We ask for the cryptographic standard of two-party computation security [39] as follows.

Definition 1: Suppose that a two-party protocol Π asks P_0 to compute the function $f_0(x, y)$, and asks P_1 to compute $f_1(x, y)$, where x and y are the inputs of P_0 and P_1, respectively. The view of P_0 (resp. P_1) during the execution of Π on (x, y), denoted $\text{view}_0^{\Pi}(x, y)$ [resp. $\text{view}_1^{\Pi}(x, y)$], is $(x, r_0, m_1, \ldots, m_t)$ (resp. $(y, r_1, m_1, \ldots, m_t)$), where r_0 (resp. r_1) represents the randomness of P_0 (resp. P_1) and m_i represents the ith message passed between the parties. Also let $\mathcal{O}_0^{\Pi}(x, y)$ (resp. $\mathcal{O}_1^{\Pi}(x, y)$) denote P_0's (resp. P_1's) output and $\mathcal{O}^{\Pi}(x, y) = (\mathcal{O}_0^{\Pi}(x, y), \mathcal{O}_1^{\Pi}(x, y))$. We say that protocol Π is secure against semi-honest adversaries if there exist probabilistic polynomial time (PPT) simulators $\mathcal{S}_0$ and $\mathcal{S}_1$ such that

$$(\mathcal{S}_0(x, f_0(x, y)), f(x, y)) \stackrel{c}{\equiv} \left(\text{view}_0^{\Pi}(x, y), \mathcal{O}^{\Pi}(x, y)\right) \quad (1)$$

$$(\mathcal{S}_1(y, f_1(x, y)), f(x, y)) \stackrel{c}{\equiv} \left(\text{view}_1^{\Pi}(x, y), \mathcal{O}^{\Pi}(x, y)\right) \quad (2)$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability.

Clearly, the security goals are essentially that the cryptographic inference between P_0 and P_1 should satisfy the above security definition.

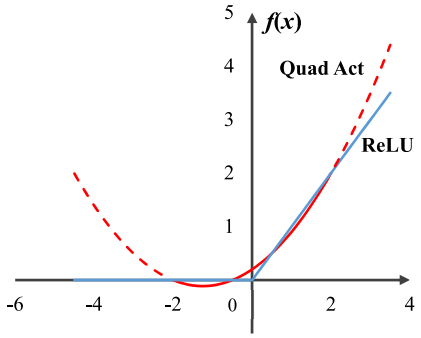
Prediction API Attacks: We briefly discuss prediction API attacks [40], [41] that are related to our secure inference system. Prediction API attacks are launched by an adversary who tries to infer private information about the server model or training data through arbitrary queries. For alleviating this attack, one can use the differential privacy technique [42] to add carefully chosen noise to the generated model. The defense technologies against the attack are orthogonal and complementary to our security goals. Therefore, we deem dealing with attacks on ML models via prediction API out of the scope of this article.

V. Model Preparing

A. Problem Analysis

As stated in the section of introduction, Manto uses quadratic (i.e., polynomials with degree-2) activations to replace the majority of ReLU activations in CNN models. The ReLU function and the quadratic activation function used in Manto are drawn in Fig. 2(a). The intuition for this replacement is that the quadratic polynomial is close to the ReLU function in certain intervals [e.g., the solid part of the red line in Fig. 2(a)]. Take a typical CNN model AlexNet [21] as an example, Fig. 2(b) illustrates its architecture before and after this replacement.

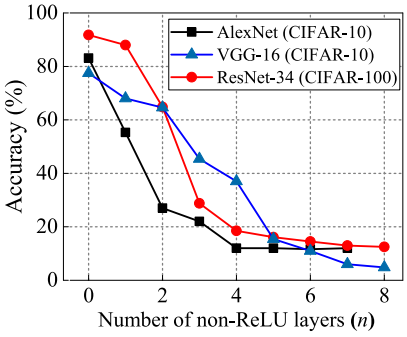
However, previous researches [6], [8] and our own experiments show that simply replacing ReLU layers with quadratic activations would degrade inference accuracy and even make the model fail to train. For example, we simply replace ReLU layers with quadratics in three typical CNN models (including AlexNet [21] and VGG-16 [22] on the CIFAR-10 data set, and ResNet-34 [23] on the CIFAR-100 data set) and then retrain these models. Fig. 2(c) shows the accuracy of different models after replacing n ReLU layers. The experimental results indicate that these models will become invalid after replacing four or five ReLU layers. This is because the computation of square terms increases the loss of accuracy caused by compounding errors. Moreover, directly replacing ReLU layers with quadratic activation layers would make the training process hard to converge and even incur the problem of gradient explosion, especially for deep networks that contain a large proportion of quadratic activations. Thus, the model with several quadratic activations is hard to converge to the optimal solution during the training. Given the above, training an effective CNN model with many quadratic activation layers remains a challenge.



(a)

AlexNet Network Architecture	
Original	**After replacement**
Conv (BN, ReLU)	Conv (BN, Quad Act)
Maxpool	Maxpool
Conv (BN, ReLU)	Conv (BN, Quad Act)
Maxpool	Maxpool
Conv (BN, ReLU)	Conv (BN, Quad Act)
Conv (BN, ReLU)	Conv (BN, Quad Act)
Conv (BN, ReLU)	Conv (BN, Quad Act)
FC (ReLU)	FC (Quad Act)
FC (ReLU)	FC (Quad Act)
FC	FC

(b)



(c)

Fig. 2. Illustration of replacing ReLU layers with quadratic activations in Manto. The notations are defined as follows. Conv: Convolution Layer, BN: Batch Normalization, ReLU: ReLU Activation, Maxpool: Max-pooling Layer, FC: Fully connected Layer, and Quad Act: Quadratic Activation. (a) ReLU function and the quadratic activation function. (b) AlexNet network architecture before and after the replacement. (c) Accuracy of different networks after replacing n ReLU layers.

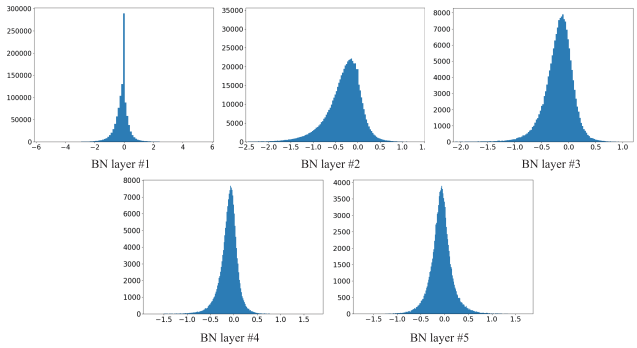


Fig. 3. Distributions of ReLU layers' input (the BN layers' output) in AlexNet.

As mentioned before, Delphi [8] uses the same quadratic function[1] to replace ReLU layers in different positions. However, we observe that the distribution of each ReLU layer's input is different, so it is not reasonable to use the same quadratic function for every replaced ReLU layer. For example, Fig. 3 shows the different distributions of the ReLU layers' input in AlexNet, which is usually the output of the BN layer. Therefore, it is important for reducing the accuracy error to customize quadratic fitting functions for specific layers.

B. Model Generation Approach

Motivated by the above discussion, we first design a search algorithm to determine quadratic fitting functions for the replaced ReLU layers. Further, we propose a method (named as sliding-window-based fine-tuning) to fine-tune the parameters of all layers, such that we can transform an original model to a high-accuracy model with fewer ReLU activations for secure inferences.

1) Customized Quadratic Fitting Functions: The optimal quadratic fitting function f for the ith ReLU layer can be found by minimizing

$$E = \sum_{k=1}^{|X|} \Big(|f(x_k) - \mathrm{ReLU}(x_k)|^2 \Big), x_k \in X \tag{3}$$

where X denotes the inputs of the ith ReLU layer and $|X|$ denotes the size of the inputs. Finding the above optimal fitting function can be addressed by many optimization algorithms [43], which are usually time consuming. To make Manto practical, we propose an approximate algorithm to quickly determinate the quadratic fitting function, as shown in Algorithm 1. We select a random sample data set D from the training data, and run the original CNN on the data set D. We use the input vector and output vector $(\boldsymbol{x}, \boldsymbol{y})$ of the ith ReLU layer as our search algorithm's input, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_{|D|}\}$, $\boldsymbol{y} = \{y_1, y_2, \ldots, y_{|D|}\}$. First, we compute the mode, minimum, and maximum of $\boldsymbol{x}$, set the number of search steps N, and compute the stride $s = \max((x_{\max} - x_{\mathrm{mode}}), (x_{\mathrm{mode}} - x_{\min}))/N$. Then, we use the mode x_{mode} as the center point to expand the interval step by step. In particular, we use a looping statement to compute the interval $[L_j, U_j](0 \le j \le N)$, where $L_j = x_{\mathrm{mode}} - j \cdot s$ and $U_j = x_{\mathrm{mode}} + j \cdot s$. Next, we use the polyfit[2] function to fit all the points in the interval $[L_j, U_j]$ to obtain a quadratic fitting function f_j. After that, we feed the input vertor $\boldsymbol{x}$ into the quadratic function f_j to get the new output $\boldsymbol{y'}$, and compute the Euclidean distance dis_j between $\boldsymbol{y}$ and $\boldsymbol{y'}$. Finally, we compute $j^* = \arg\min_{j=0,\ldots,N} dis_j$, and obtain the final quadratic fitting function $f = f_{j^*}$.

The size of the sample data set ($|D|$) and the number of search steps (N) will directly affect the inference accuracy and fitting-function-search running time. In the experiment part, we assess the impact of these parameters by experiments. We find that when $|D|$ is large enough (e.g., $|D| > 300$), the inference accuracy no longer increases with the size of the sample data set, but the running time of the fitting-function-search algorithm will still continue to increase. Hence, the size of the sample data set ($|D|$) is advised to be set as 300. For the similar reason, the number of search steps (N) is set as 60. More information about the analysis can be found in Section VIII-B3.

2) Sliding-Window-Based Fine-Tuning: Although we can reduce the error of a single replaced layer by the customize

[1]The quadratic activation used in Delphi is $f(x) = 0.1997x^2 + 0.5002x + 0.1992$.

[2]In our implementation, we invoke the polyfit function in NumPy library to achieve the least-square-method-based fitting [44].

Algorithm 1 Searching Quadratic Fitting Function for the ith ReLU Layer

Input: The input/output of ith ReLU layer in the original CNN on the sample dataset D, denoted by $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_{|D|}\}$, $\boldsymbol{y} = \{y_1, y_2, \ldots, y_{|D|}\}$

Output: The quadratic fitting function f

$x_{mode} = \text{mode}(\boldsymbol{x})$, $x_{max} = \max(\boldsymbol{x})$, $x_{min} = \min(\boldsymbol{x})$

Set the number of search steps N, compute the stride $s = \max((x_{max} - x_{mode}), (x_{mode} - x_{min}))/N$

for $j = 0, 1, \ldots, N$ **do**

  $L_j = x_{mode} - j \cdot s$, $U_j = x_{mode} + j \cdot s$

  $f_j = \text{polyfit}((\boldsymbol{x}, \boldsymbol{y}), L_j, U_j)$

  $\boldsymbol{y}' = f_j(\boldsymbol{x})$

  $dis_j = \sqrt{\sum_{k=1}^{|D|} (y_k - y'_k)^2}$

end for

$j^* = \arg\min_{j=0,\ldots,N} dis_j$, $f = f_{j^*}$

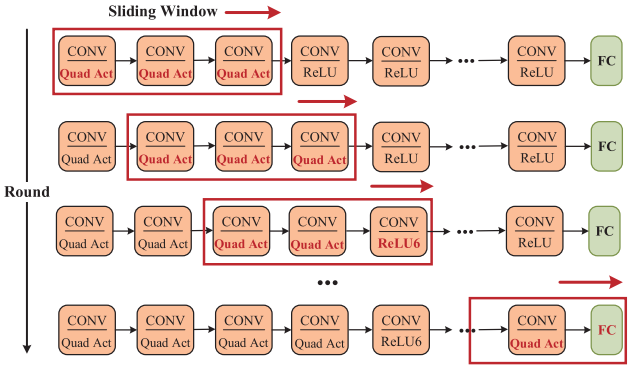


Fig. 4. Sliding-window-based fine-tuning method (the size of the sliding-window $d = 3$ and the stride of the sliding-window $s = 1$).

fitting functions, the stack of many quadratic activation layers could still lead to the gradient explosion. To avoid this, we present a method named as *sliding-window-based fine-tuning*. As shown in Fig. 4, we take a sequential neural network as an example to illustrate the specific steps of this method. Given a trained neural network of n layers (not counting the layers without the parameter, e.g., max-pooling and average-pooling layers), this method can fine-tune the parameters of all layers through $\lceil (n-d)/s \rceil + 1$ round training, where d and s represents the size and the stride of the sliding window, respectively. Note that we require that s is not to exceed d ($s \leq d$) to make the fine-tuning act on each layer. In each round, we first use quadratic activations to replace the ReLU layers within the sliding window, and freeze the parameters of other layers outside the sliding window. Then, we fine-tune the parameters of the layers within the sliding window, and determine whether the ReLU layers could be replaced according to the accuracy after training. If the accuracy is larger than the specified threshold t, we move on to the next round. Otherwise, we would replace the quadratic activations with the ReLU6 activations instead of the ReLU activations, which is because ReLU6 can keep errors from compounding during both inference and training [8], to prevent the gradients from growing too rapidly.

The accuracy threshold t is user defined, which depends on the clients' targeted accuracy/computing power tradeoff. In Manto, t is given by the client and then is sent to the server. The size (d) and the stride (s) of the sliding window will directly affect inference accuracy and model training time. Since the model training round is $\lceil (n-d)/s \rceil + 1$ and training time in each round is almost the same, the training time is always inversely associated with the size (d) and the stride (s) of the sliding window. But d is not as bigger as better, because the accumulated error of multiple layers makes fine-tuning much difficult. Our experiments demonstrate that the accuracy of the trained model reaches a peak when d reaches a threshold. The experiments also show that the model training time and accuracy are both decreased with the increase of the stride of the sliding windows (s). It is because the stride is smaller and the fine-tuning is more detailed. Detailed discussion is included in Section VIII-B3.

VI. Secure Neural Network Inference

In this section, we give an overview of cryptographic protocols for the layers that will be used to realize secure neural network inference. Layers can be broken into two categories: 1) linear and 2) nonlinear. The protocols of each layer are connected to each other with appropriate dimensions, which directly form a complete inference algorithm. Following many 2PC-based cryptographic inference services [13], [14], Manto uses Delphi's protocols [8] for the linear layer computations,[3] which can be computed at near plaintext speed. In this section, we focus on how to implement secure computations for the quadratic activation and the ReLU6 activation efficiently.

Recall that we assume the existence of two semi-honest parties—the client P_0 and the server P_1. Let P_0 and P_1 generate their respective public–private key pairs of additively homomorphic encryption $pk_i/sk_i, i \in \{0, 1\}$ and publish the public keys to each other. Since the following protocols only involve the key pair of P_0, for notational simplicity, we use $[\cdot]$ instead of $[\cdot]_{pk_0}$ to represent the encrypted data by pk_0, and use $Dec(\cdot)$ instead of $Dec_{sk_0}(\cdot)$ to represent the decryption with sk_0. During realizing these protocols, we maintain the following invariant: all arithmetic operations are performed in the ring $\mathbb{Z}_{2^\ell}$ (i.e., all arithmetic operations are mod 2^ℓ). The input/output of each layer are additive shares (over the same ring), such that we can use these layers in any combination to achieve the desired networks.

A. Quadratic Activation

As previously described, Manto replaces most of ReLU activations with polynomial (specifically, quadratic) approximations that can be efficiently computed using secure computation techniques. We propose a secure quadratic activation (SQA) protocol to evaluate activation layers via a one-round online interaction. Assume that P_0 and P_1 hold the shares of x, k_2, k_1, k_0 over $\mathbb{Z}_{2^\ell}$, where x is the client's input and k_2, k_1, k_0 are the coefficients of a quadratic function. Namely, P_i holds $(\langle x \rangle_i, \langle k_2 \rangle_i, \langle k_1 \rangle_i, \langle k_0 \rangle_i)$ for $i \in \{0, 1\}$. The SQA protocol is to

[3]DELPHI uses a secret sharing for linear layers and optimizes the linear layer computations by moving cryptographic operations to an offline stage.

Algorithm 2 Generating Assisted Parameters of SQA

Input: P_0 inputs a PRF key key_0, P_1 inputs a PRF key key_1
Output: P_0 outputs $\langle a\rangle_0$, $\langle b\rangle_0$, $\langle c\rangle_0$, $\langle ab\rangle_0$, $\langle ac\rangle_0$, $\langle a^2\rangle_0$, $\langle a^2b\rangle_0$, P_1 outputs $\langle a\rangle_1$, $\langle b\rangle_1$, $\langle c\rangle_1$, $\langle ab\rangle_1$, $\langle ac\rangle_1$, $\langle a^2\rangle_1$, $\langle a^2b\rangle_1$

1: P_0 uses PRF key key_0 to generate random numbers $\langle a\rangle_0$, $\langle b\rangle_0$, $\langle c\rangle_0 \in \mathbb{Z}_{2^\ell}$
2: P_0 computes $[\langle a\rangle_0]$, $[\langle b\rangle_0]$, $[\langle c\rangle_0]$, $[\langle a\rangle_0^2]$, $[\langle a\rangle_0 \cdot \langle b\rangle_0]$ and sends to P_1
3: P_1 uses PRF key key_1 to generate random numbers $\langle a\rangle_1$, $\langle b\rangle_1$, $\langle c\rangle_1 \in \mathbb{Z}_{2^\ell}$, $r_1, r_2, r_3 \in \mathbb{Z}_{2^{2\ell+\sigma+1}}$, $r_4 \in \mathbb{Z}_{2^{3\ell+2\sigma+2}}$
4: P_1 sets $\langle ab\rangle_1 = \langle a\rangle_1 \cdot \langle b\rangle_1 - r_1$, $\langle ac\rangle_1 = \langle a\rangle_1 \cdot \langle c\rangle_1 - r_2$, $\langle a^2\rangle_1 = \langle a\rangle_1^2 - r_3$, $\langle a^2b\rangle_1 = \langle a\rangle_1^2 \cdot \langle b\rangle_1 - r_4$
5: P_1 computes $d_1 = [\langle a\rangle_0]^{\langle b\rangle_1} \cdot [\langle b\rangle_0]^{\langle a\rangle_1} \cdot [r_1]$, $d_2 = [\langle a\rangle_0]^{\langle c\rangle_1} \cdot [\langle c\rangle_0]^{\langle a\rangle_1} \cdot [r_2]$, $d_3 = [\langle a\rangle_0]^{2\langle a\rangle_1} \cdot [r_3]$, $d_4 = [\langle a\rangle_0]^{2\langle a\rangle_1 \cdot \langle b\rangle_1} \cdot [\langle a\rangle_0^2]^{\langle b\rangle_1} \cdot [\langle a\rangle_0 \cdot \langle b\rangle_0]^{2\langle a\rangle_1} \cdot [\langle b\rangle_0]^{\langle a\rangle_1^2} \cdot [r_4]$ and sends to P_0
6: P_0 computes $\langle ab\rangle_0 = \langle a\rangle_0 \cdot \langle b\rangle_0 + Dec(d_1)$, $\langle ac\rangle_0 = \langle a\rangle_0 \cdot \langle c\rangle_0 + Dec(d_2)$, $\langle a^2\rangle_0 = \langle a\rangle_0^2 + Dec(d_3)$, $\langle a^2b\rangle_0 = \langle a\rangle_0^2 \cdot \langle b\rangle_0 + Dec(d_4)$

realize the function that securely calculates the quadratic polynomial $f = k_2x^2 + k_1x + k_0$ in the secret-sharing form without revealing any private information about x, k_2, k_1, k_0. Inspired by Beaver's multiplication [36], we construct SQA protocol based on the following formulas:

$$\begin{cases} u = x - \underline{a}, v = k_2 - \underline{b}, w = k_1 - \underline{c} \\ k_2x^2 = \underline{a^2b} + \underline{a^2}v + u^2\underline{b} + u^2v + 2\underline{ab}u + 2\underline{a}uv \\ k_1x = \underline{ac} + \underline{a}w + \underline{c}u + uw. \end{cases} \tag{4}$$

The variables a, b, and c are used to blind x, k_2, and k_1 and the variables with underlines are independent of the inputs x, k_2, k_1, and k_0, such that these values can be precomputed and stored in the local devices. Therefore, SQA protocol is divided into two stages: 1) offline and 2) online. The offline stage generates some assisted parameters which are independent of the input of protocol, while the online stage can achieve polynomial computations just by one interaction between P_0 and P_1 with the aid of the assisted parameters.

The offline and online steps in the SQA protocol are shown in Algorithms 2 and 3, respectively. In the offline stage, P_0 and P_1 generate the assisted parameters $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle ab\rangle$, $\langle ac\rangle$, $\langle a^2\rangle$, and $\langle a^2b\rangle$ (a, b, and c are the random numbers from $\mathbb{Z}_{2^\ell}$) using an additively homomorphic encryption. In the online stage, P_0 and P_1 use the precomputed parameters to blind the private inputs, and then compute $\langle f\rangle_i = \langle k_2x^2\rangle_i + \langle k_1x\rangle_i + \langle k_0\rangle_i$ after a data exchange.

B. ReLU6 Activation and Maxpool

We describe a protocol for the ReLU6 activation that takes additive shares of x as inputs and returns the additive shares of ReLU6(x). ReLU6(x) can be defined as a piecewise function as follows:

$$\text{ReLU6}(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \le x < 6 \\ 6, & x \ge 6. \end{cases} \tag{5}$$

Algorithm 3 SQA Protocol

Input: P_0 inputs $\langle x\rangle_0$, $\langle k_2\rangle_0$, $\langle k_1\rangle_0$, $\langle k_0\rangle_0$, P_1 inputs $\langle x\rangle_1$ $\langle k_2\rangle_1$, $\langle k_1\rangle_1$, $\langle k_0\rangle_1$
Output: P_0 outputs $\langle f\rangle_0$, P_1 outputs $\langle f\rangle_1$

1: In the offline stage, P_0 and P_1 prepare assisted parameters in advance, such that P_0 holds $\langle a\rangle_0$, $\langle b\rangle_0$, $\langle c\rangle_0$, $\langle a^2\rangle_0$, $\langle ab\rangle_0$, $\langle a^2b\rangle_0$, $\langle ac\rangle_0$, P_1 holds $\langle a\rangle_1$, $\langle b\rangle_1$, $\langle c\rangle_1$, $\langle a^2\rangle_1$, $\langle ab\rangle_1$, $\langle a^2b\rangle_1$, $\langle ac\rangle_1$, where $a, b, c \in \mathbb{Z}_{2^\ell}$ are random numbers
2: For $i \in \{0, 1\}$, P_i computes $\langle u\rangle_i = \langle x\rangle_i - \langle a\rangle_i$, $\langle v\rangle_i = \langle k_2\rangle_i - \langle b\rangle_i$, $\langle w\rangle_i = \langle k_1\rangle_i - \langle c\rangle_i$, and sends to $P_{i\oplus1}$
3: P_0 & P_1 reconstruct u, v, w using exchanged shares
4: For $i \in \{0, 1\}$, P_i computes $\langle k_2x^2\rangle_i = \langle a^2b\rangle_i + v\langle a^2\rangle_i + u^2\langle b\rangle_i + i \cdot u^2v + 2u\langle ab\rangle_i + 2uv\langle a\rangle_i$ and $\langle k_1x\rangle_i = \langle ac\rangle_i + w\langle a\rangle_i + u\langle c\rangle_i + i \cdot uw$, then $\langle f\rangle_i = \langle k_2x^2\rangle_i + \langle k_1x\rangle_i + \langle k_0\rangle_i$

To implement a secure ReLU6 activation (SRA) protocol, we have to execute the above operations in a data-oblivious manner (i.e., the execution paths of programs are independent with input data). Let the two bits $a = (x \ge 0)$ and $b = (x \ge 6)$, then

$$\text{ReLU6}(x) = a \cdot \bar{b} \cdot x + a \cdot b \cdot 6 \tag{6}$$

where $\bar{b}$ denotes the NOT operation on b. Since the base of the above formula is a comparison operation, we first design a secure comparison (SC) protocol that takes additive shares of x and y as inputs and returns the additive shares of $(x \ge y)$. Our SC protocol builds upon the DGK protocol [45] but makes crucial modifications to support SC for the additively shared data and improve online performance. In the DGK protocol (Algorithm 7 in Appendix A), P_0 and P_1 hold ℓ-bit integers $\beta = \beta_{\ell-1} \ldots \beta_1\beta_0$ and $\alpha = \alpha_{\ell-1} \ldots \alpha_1\alpha_0$, respectively. It scans from $\ell - 1$ to 0 to locate the first differing bit by computing c_i as follows:

$$c_i = s + \alpha_i - \beta_i + 3 \sum_{j=i+1}^{\ell-1} (\alpha_j \oplus \beta_j) \tag{7}$$

where $s = 1 - 2\lambda_1$ and λ_1 is a random bit picked by P_1. Clearly

$$c_i = 0 \ \text{ iff } \ (\alpha_j = \beta_j)_{\forall j, i<j<\ell} \ \text{ and } \ \alpha_i \ne \beta_i. \tag{8}$$

For that, P_0 sends $[\beta_i] (0 \le i < \ell)$ to P_1, who computes $[c_i]$ via AHE as follows:

$$[c_i] = [s] \cdot [\alpha_i] \cdot [\beta_i]^{-1} + \left(\prod_{j=i+1}^{\ell-1} [\alpha_j \oplus \beta_j] \right)^3 \tag{9}$$

where $[\alpha_j \oplus \beta_j] = [\beta_j]^{1-2\alpha_j} \cdot [\alpha_j]$. Then, P_1 sends $[c_i]$ to P_0 after blinding them and shuffling their orders. At last, P_0 sets the bit $\lambda_0 = 1$ if any decrypted c_i is 0, 0 otherwise. So λ_0 and λ_1 are Boolean shares of $(a \le b)$, i.e., $\lambda_0 \oplus \lambda_1 = (\alpha \le \beta)$.

Our SC protocol is divided into offline stage and online stage such that most of the heavy cryptographic computation (e.g., AHE) can be offloaded to the offline stage. Algorithm 4

Algorithm 4 Offline Stage of SC Protocol

Input: P_0 inputs a PRF key key_0, P_1 inputs a PRF key key_1
Output: P_0 outputs $\langle\beta_i\rangle_0, \langle c'_i\rangle_0, \delta_0, \langle v\rangle_0$,
P_1 outputs $\alpha, \lambda_1, r_i(0 \le i < \ell), \pi, \delta_1, \langle v\rangle_1$

P_1 uses PRF key key_1 to generate a random number $\alpha \in \mathbb{Z}_{2^\ell}$ and decomposes $\alpha_{\ell-1} \ldots \alpha_1\alpha_0 \leftarrow \alpha$
P_1 chooses a uniformly random bit $\lambda_1 \in \{0, 1\}$, and sets $s = 1 - 2\lambda_1$
For $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
    P_0 uses PRF key key_0 to generate a random number $\langle\beta_i\rangle_0 \in \mathbb{Z}_{2^\ell}$ and sends $[\langle\beta_i\rangle_0]$ to P_1
    P_1: $[\langle\theta_i\rangle_0] = [\langle\beta_i\rangle_0]^{1-2\alpha_i} \cdot [\alpha_i]$
End for
For $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
    P_1: $[\langle c_i\rangle_0] = [s] \cdot [\alpha_i] \cdot [\langle\beta_i\rangle_0]^{-1} \cdot (\prod_{j=i+1}^{\ell-1} [\langle\theta_j\rangle_0])^3$
    P_1 blinds $\langle c_i\rangle_0$ by raising them to a random non-zero exponent r_i: $[\langle c'_i\rangle_0] = [\langle c_i\rangle_0]^{r_i} = [r_i \cdot \langle c_i\rangle_0]$
End for
P_1 generates a random permutation function π and shuffles $[\langle c'_i\rangle_0](0 \le i < \ell)$ by π and sends them to P_0
P_0 decrypts $[\langle c'_i\rangle_0]$ to get $\langle c'_i\rangle_0$
For $i \in \{0, 1\}$, P_i chooses uniformly a random bit $\delta_i \in \{0, 1\}$
P_0 & P_1 invoke B2A protocol with inputting δ_0, δ_1 to get $\langle v\rangle$: $\langle v\rangle = \mathsf{B2A}(\delta_0, \delta_1)$

shows the main steps in the offline stage. First, P_1 generates a random number α and a random bit λ_1 while P_0 generates a set of random numbers $\langle\beta_i\rangle_0$, which is the one share of $\beta_i(0 \le i < \ell)$. After P_0 sends all $\langle\beta_i\rangle_0$ to P_1, P_1 computes

$$\left[\langle\theta_i\rangle_0\right] = \left[\langle\beta_i\rangle_0\right]^{1-2\alpha_i} \cdot [\alpha_i] \tag{10}$$

$$\left[\langle c_i\rangle_0\right] = [s] \cdot [\alpha_i] \cdot \left[\langle\beta_i\rangle_0\right]^{-1} \cdot \left(\prod_{j=i+1}^{\ell-1} \left[\langle\theta_j\rangle_0\right]\right)^3. \tag{11}$$

When receiving the blinded and shuffled $[\langle c'_i\rangle_0](0 \le i < \ell)$ from P_1, P_0 decrypts them to obtain the one share of the blinded c_i ($\langle c'_i\rangle_0$). In addition, to reduce the online computation time for the transformation of boolean shared results to additive shared results, P_0 and P_1 precompute some assisted parameters as follows: P_0 and P_1 choose two random bits δ_0 and δ_1 and compute $\langle v\rangle = \mathsf{B2A}(\delta_0, \delta_1)$.

Algorithm 5 describes the online stage of the SC protocol. At first, P_1 sends temp to P_0, who obations β = temp $+ \langle x\rangle_0 - \langle y\rangle_0 = x - y + \alpha$, such that $\alpha \le \beta \Leftrightarrow x \ge y$. Then, P_0 computes $\langle\beta_i\rangle_1 = \beta_i - \langle\beta_i\rangle_0$ and sends to P_1. After that, P_1 computes the blinded and shuffled $\langle c'_i\rangle_1(0 \le i < \ell)$ according to (7) and the permutation function π. When receiving $\langle c'_i\rangle_1$ from P_1, P_0 recovers c'_i and sets $\lambda_0 = 1$ if any c'_i is 0; otherwise, $\lambda_0 = 0$. Note that λ_0 and λ_1 are both the boolean shares of $(\alpha \le \beta)$ and $(x \ge y)$. To convert the boolean shares to the additive shares, $P_i(i \in \{0, 1\})$ computes $\phi_i = \lambda_i \oplus \delta_i$ and sends to P_{1-i}, then P_i gets $\phi = \phi_i \oplus \phi_{1-i}$. If $\phi == 0$, then $\lambda_0 \oplus \lambda_1 == \delta_0 \oplus \delta_1$, such that $\langle x \ge y\rangle = \langle v\rangle$. Otherwise, $\lambda_0 \oplus \lambda_1 \ne \delta_0 \oplus \delta_1$, such that $\langle x \ge y\rangle = \langle 1 - v\rangle$.

Now, we are ready to use the SC protocol as a blackbox to yield an SRA protocol (shown in Algorithm 6). According to

Algorithm 5 SC Protocol

Input: P_0 inputs $\langle x\rangle_0, \langle y\rangle_0$, P_1 inputs $\langle x\rangle_1, \langle y\rangle_1$
Output: P_0 outputs $\langle x \ge y\rangle_0$, P_1 outputs $\langle x \ge y\rangle_1$

In the offline stage, P_0 and P_1 prepare assisted parameters in advance, such that P_0 holds $\langle\beta_i\rangle_0, \langle c'_i\rangle_0$, $\delta_0, \langle v\rangle_0$, P_1 holds $\alpha, \lambda_1, r_i(0 \le i < \ell), \pi, \delta_1, \langle v\rangle_1$
P_1 lets $temp = \langle x\rangle_1 - \langle y\rangle_1 + \alpha$ and sends it to P_0
P_0 lets $\beta = temp + \langle x\rangle_0 - \langle y\rangle_0 = x - y + \alpha$, and decomposes $\beta_{\ell-1} \ldots \beta_1\beta_0 \leftarrow \beta$
For $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
    P_0 sends $\langle\beta_i\rangle_1 = \beta_i - \langle\beta_i\rangle_0$ to P_1
    P_1: $\langle\theta_i\rangle_1 = (1 - 2\alpha_i) \cdot \langle\beta_i\rangle_1$
End for
For $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
    P_1: $\langle c_i\rangle_1 = -\langle\beta_i\rangle_1 + 3\sum_{j=i+1}^{\ell-1} \langle\theta_j\rangle_1$
    P_1: $\langle c'_i\rangle_1 = r_i \cdot \langle c_i\rangle_1$
End for
P_1 shuffles $\langle c'_i\rangle_1(0 \le i < \ell)$ by π and sends them to P_0
P_0 recovers $c'_i = \langle c'_i\rangle_0 + \langle c'_i\rangle_1$, and sets $\lambda_0 = 1$ if any recovered c'_i is 0, otherwise $\lambda_0 = 0$
For $i \in \{0, 1\}$, P_i computes $\phi_i = \lambda_i \oplus \delta_i$ and sends to P_{1-i}, then P_i gets $\phi = \phi_i \oplus \phi_{1-i}$
If $\phi == 0$ **then**
    P_0 sets $\langle x \ge y\rangle_0 = \langle v\rangle_0$, P_1 sets $\langle x \ge y\rangle_1 = \langle v\rangle_1$
Else
    P_0 sets $\langle x \ge y\rangle_0 = -\langle v\rangle_0$,
    P_1 sets $\langle x \ge y\rangle_1 = 1 - \langle v\rangle_1$
End If

Algorithm 6 SRA Protocol

Input: P_0 inputs $\langle x\rangle_0$, P_1 inputs $\langle x\rangle_1$
Output: P_0 outputs $\langle \mathrm{ReLU6}(x)\rangle_0$, P_1 outputs $\langle \mathrm{ReLU6}(x)\rangle_1$

P_0&P_1 generate additive secret-shares of values 0 and 6 to get $\langle 0\rangle$ and $\langle 6\rangle$
P_0&P_1 invoke SC protocol to compute $\langle a\rangle = \mathsf{SC}(\langle x\rangle, \langle 0\rangle)$ and $\langle b\rangle = \mathsf{SC}(\langle x\rangle, \langle 6\rangle)$
P_0 sets $\langle\bar{b}\rangle_0 = -\langle b\rangle_0$, P_1 sets $\langle\bar{b}\rangle_1 = 1 - \langle b\rangle_1$
P_0&P_1 compute $\langle \mathrm{ReLU6}\rangle = \langle a\rangle \cdot \langle\bar{b}\rangle \cdot \langle x\rangle + \langle a\rangle \cdot \langle b\rangle \cdot \langle 6\rangle$

(6), P_0 and P_1 invoke the SC protocol to compute

$$\begin{cases} \langle a\rangle = \mathsf{SC}(\langle x\rangle, \langle 0\rangle) = \langle x \ge 0\rangle \\ \langle b\rangle = \mathsf{SC}(\langle x\rangle, \langle 6\rangle) = \langle x \ge 6\rangle \\ \langle\bar{b}\rangle = \langle 1\rangle - \langle b\rangle. \end{cases} \tag{12}$$

Next, P_0 and P_1 directly compute

$$\langle \mathrm{ReLU6}\rangle = \langle a\rangle \cdot \langle\bar{b}\rangle \cdot \langle x\rangle + \langle a\rangle \cdot \langle b\rangle \cdot \langle 6\rangle. \tag{13}$$

Similarly, secure ReLU activation can be implemented by the same way. The Maxpool$(x_1, \ldots, x_n)$ layer is essentially finding a maximum value, which can be done using $n - 1$ invocations of the SC protocol in $n - 1$ sequential steps.

VII. Theoretical Analysis

A. Security Analysis

We follow the formal definition of security against semi-honest adversaries (Definition 1 in Section IV-B). The proof

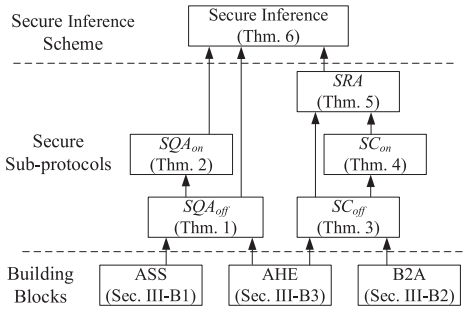


Fig. 5. Proof flow and the dependencies behind the all theorems.

flow and the dependencies behind the all theorems are presented in Fig. 5, and we argue the security in a bottom-up fashion as follows.

1) We first instantiate the building blocks that we use. For the cryptographic primitives and protocols (described in Section III-B) involved in our protocols, we refer to [6], [34], and [37] for deriving security in the presence of semi-honest security. As for the AHE with semantic security, we refer to [38] for the formal proof.
2) Next, in Theorems 1–5, we prove the security of the cryptographic protocols, including SQA, SC, and SRA protocols.
3) Finally, in Theorem 6, we put everything together and prove the security of the secure inference.

We employ the sequential composition theory [46] to prove the security of all cryptographic protocols under the semi-honest adversary model [39]. The sequential composition theorem is a tool that enables us to analyze the security of cryptographic protocols in a modular approach. Consider a protocol π_g for securely computing g that invokes subprotocols $\pi_{f_1}, \ldots, \pi_{f_m}$ for computing $f_1, \ldots, f_m$. The theorem states that it suffices to consider the execution of π_g in a $(f_1, \ldots, f_m)$-hybrid model that is augmented with an incorruptible trusted party T for evaluating $f_1, \ldots, f_m$. Next, we provide the security proofs for our cryptographic protocols.

Theorem 1: If the AHE is semantically secure, then the offline stage of SQA protocol (short as SQA_{off}) is secure under the semi-honest adversary model.

Proof: We construct simulators for security in two distinct cases as SQA_{off} is asymmetric for two parties.

Case 1: The client P_0 is corrupted by an adversary. We construct a simulator $\mathcal{S}_0$ to simulate P_0's view. For P_0 receives $D = \{d_1, d_2, d_3, d_4\}$ in the real world (line 5 in Algorithm 2), $\mathcal{S}_0$ randomly picks values l_1, l_2, l_3, l_4 from $\mathbb{Z}_{2^\ell}$ and encrypts them by AHE to get $L = \{[l_1], [l_2], [l_3], [l_4]\}$. Then, any PPT adversary cannot distinguish the simulator's encryption of the random values (L) from P_1's encryption of the correct computation (D) due to the semantical security of AHE. For the output of this protocol in the real world, it is clearly computational indistinguishable from the output of SQA_{off}'s function. In addition, all the values in D and the real output are irrelevant, and all the values in L and the function output are also irrelevant, so these values are computational indistinguishable. Therefore, (1) clearly holds.

Case 2: The server P_1 is corrupted by an adversary. We construct a simulator $\mathcal{S}_1$ to simulate P_1's view. For P_1 receives $P = \{[\langle a\rangle_0], [\langle b\rangle_0], [\langle c\rangle_0], [\langle a\rangle_0^2], [\langle a\rangle_0 \cdot \langle b\rangle_0]\}$ in the real world (line 2 in Algorithm 2), $\mathcal{S}_1$ randomly generates numbers q_1, q_2, q_3 from $\mathbb{Z}_{2^\ell}$ and encrypts them by AHE to get $Q = \{[\langle q_1\rangle_0], [\langle q_2\rangle_0], [\langle q_3\rangle_0], [\langle q_1\rangle_0^2], [\langle q_1\rangle_0 \cdot \langle q_2\rangle_0]\}$. Again, because of the semantical security of AHE, P is indistinguishable from Q. In addition, all the above values are selected independently. Therefore, (2) holds.

Putting the above results together, we can claim that SQA_{off} is secure under the semi-honest adversary model. ■

Theorem 2: If AHE is semantically secure and SQA_{off} is secure against the semi-honest adversaries, then the online stage of secure quadratic activation protocol (short as SQA_{on}) is secure under the semi-honest adversary model.

Proof: We present the security proof in a hybrid model where P_0 and P_1 have access to a trusted party T which can realize the function of SQA_{off}. P_0 and P_1 call T to run the function of SQA_{off} to get the random numbers $\langle a\rangle_i, \langle b\rangle_i, \langle c\rangle_i, i \in \{0, 1\}$. As the exchange messages $\langle e\rangle_i = \langle x\rangle_i - \langle a\rangle_i$, $\langle f\rangle_i = \langle y\rangle_i - \langle b\rangle_i$, $\langle g\rangle_i = \langle z\rangle_i - \langle c\rangle_i$, the numbers $\langle e\rangle_i, \langle f\rangle_i, \langle g\rangle_i$ are all random. Therefore, in the case of a corrupted P_0, the simulator $\mathcal{S}_0$ just chooses numbers l_1, l_2, l_3 from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle e\rangle_1, \langle f\rangle_1, \langle g\rangle_1$, and in the case of a corrupted P_1, the simulator $\mathcal{S}_1$ just do the same as $\mathcal{S}_0$ to simulate $\langle e\rangle_0, \langle f\rangle_0, \langle g\rangle_0$. Based on the composition theorem, combining the above analysis and the security of SQA_{off}, we can claim that SQA_{on} is secure under the semi-honest adversary model. ■

Theorem 3: If the *B2A* protocol is secure against the semi-honest adversaries, then the offline stage of secure comparison protocol (short as SC_{off}) is secure under the semi-honest adversary model.

Proof: We present the security proof in a hybrid model where P_0 and P_1 have access to a trusted party T which can realize the function of *B2A*. We construct simulators in two distinct cases as SC_{off} is asymmetric for two parties.

Case 1: The client P_0 is corrupted by an adversary. We construct a simulator $\mathcal{S}_0$ to simulate P_0's view. For P_0 receives $[\langle c_i'\rangle_0](0 \le i < \ell)$ (line 11 in Algorithm 4), $\mathcal{S}_0$ randomly generates ℓ numbers $x_0, \ldots, x_{\ell-1} \in \mathbb{Z}_{2^\ell}$ and encrypts them by AHE to get $[x_i]$. Then, any PPT adversary cannot distinguish the simulator's encryption of the random number ($[x_i]$) from P_1's encryption of the correct computation ($[\langle c_i'\rangle_0]$) due to the semantical security of AHE. P_0 and P_1 call T to run the function of *B2A* with inputting δ_0, δ_1 for outputting the additive shares $\langle v\rangle_0$ to P_0 and $\langle v\rangle_1$ to P_1. These values are both random according to the function of *B2A*. Therefore, in the case of a corrupted P_0, the simulator $\mathcal{S}_0$ just chooses numbers y from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle v\rangle_0$. Due to the security of *B2A*, y is indistinguishable from $\langle v\rangle_0$.

Case 2: The server P_1 is corrupted by an adversary. We construct a simulator $\mathcal{S}_1$ to simulate P_1's view. For P_1 receives $[\langle \beta_i\rangle_0](0 \le i < \ell)$ (line 4 in Algorithm 4), $\mathcal{S}_1$ randomly generates ℓ numbers $z_0, \ldots, z_{\ell-1} \in \mathbb{Z}_{2^\ell}$ and encrypts them by AHE to get $[z_i]$. Then, any PPT adversary cannot distinguish the simulator's encryption of the random number ($[z_i]$) from P_0's encryption of the correct computation ($[\langle \beta_i\rangle_0]$) due to the semantical security of AHE. For P_1 receives $\langle v\rangle_1$, the simulator $\mathcal{S}_1$ just do the same as $\mathcal{S}_0$ to simulate $\langle v\rangle_1$.

Based on the composition theorem, combining the above analysis and the security of *B2A*, we can claim that SC_{off} is secure under the semi-honest adversary model. ■

Theorem 4: If the SC_{off} protocol is secure against the semi-honest adversaries, then the online stage of secure comparison protocol (short as SC_{on}) is secure under the semi-honest adversary model.

Proof: We present the security proof in a hybrid model where P_0 and P_1 have access to a trusted party T which can realize the function of SC_{off}. P_0 and P_1 call T to run the function of SC_{off} for outputting the random numbers $\langle\beta_i\rangle_0, \langle c_i'\rangle_0, \langle v\rangle_0 \in \mathbb{Z}_{2^\ell}, \delta_0 \in \{0,1\}$ to P_0, and outputting the random numbers $\alpha, r_i (0 \le i < \ell), \langle v\rangle_1 \in \mathbb{Z}_{2^\ell}, \lambda_1, \delta_1 \in \{0,1\}$ and the random permutation function π to P_1. We construct simulators in two distinct cases as SC_{on} is asymmetric for two parties.

Case 1: The client P_0 is corrupted by an adversary. We construct a simulator $\mathcal{S}_0$ to simulate P_0's view. For P_0 receives $\text{temp} = \langle x\rangle_1 - \langle y\rangle_1 + \alpha$ (line 2 in Algorithm 5), temp is a random number due to the randomness of α. Therefore, the simulator $\mathcal{S}_0$ just chooses a number a from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate temp. For P_0 receives $\langle c_i'\rangle_1 = r_i \cdot \langle c_i\rangle_1$ (line 12 in Algorithm 5), $\langle c_i'\rangle_1$ are random numbers due to the randomness of r_i. Therefore, the simulator $\mathcal{S}_0$ just chooses ℓ numbers $b_i (0 \le i < \ell)$ from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle c_i'\rangle_1$. For P_0 receives $\phi_1 = \lambda_1 \oplus \delta_1$ (line 14 in Algorithm 5), ϕ_1 is a random bit due to the randomness of the bits λ_1, δ_1. Therefore, the simulator $\mathcal{S}_0$ just chooses a bit e from $\mathbb{Z}_2$ uniformly at random to simulate ϕ_1.

Case 2: The client P_1 is corrupted by an adversary. We construct a simulator $\mathcal{S}_1$ to simulate P_1's view. For P_1 receives $\langle\beta_i\rangle_1 = \beta_i - \langle\beta_i\rangle_0$ (line 5 in Algorithm 5), $\langle\beta_i\rangle_1$ are random numbers due to the randomness of $\langle\beta_i\rangle_0$. Therefore, the simulator $\mathcal{S}_0$ just chooses ℓ numbers $f_i (0 \le i < \ell)$ from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle\beta_i\rangle_1$. For P_1 receives $\phi_0 = \lambda_0 \oplus \delta_0$ (line 14 in Algorithm 5), ϕ_0 is a random bit due to the randomness of the bit δ_0. Therefore, the simulator $\mathcal{S}_0$ just chooses a bit g from $\mathbb{Z}_2$ uniformly at random to simulate ϕ_0.

Based on the composition theorem, combining the above analysis and the security of SC_{off}, we can claim that SC_{on} is secure under the semi-honest adversary model. ■

Theorem 5: If the SC protocol is secure against the semi-honest adversaries, then the SRA protocol is secure under the semi-honest adversary model.

Proof: We present the security proof in a hybrid model where P_0 and P_1 have access to a trusted party T which can realize the function of SC. P_0 and P_1 call T to run the function of SC protocol with inputting $\langle x\rangle, \langle 0\rangle$ for outputting the additive shares $\langle a\rangle_0$ to P_0 and $\langle a\rangle_1$ to P_1. These values are both random in $\mathbb{Z}_{2^\ell}$ according to the function of SC protocol, therefore, in the case of a corrupted P_0, the simulator $\mathcal{S}_0$ just chooses a number x from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle a\rangle_0$, and in the case of a corrupted P_1, the simulator $\mathcal{S}_1$ just do the same as $\mathcal{S}_0$ to simulate $\langle a\rangle_1$. When P_0 and P_1 call T to run the function of SC protocol with inputting $\langle x\rangle, \langle 6\rangle$ for outputting the additive shares $\langle b\rangle_0$ to P_0 and $\langle b\rangle_1$ to P_1, the simulator $\mathcal{S}_0$ do the same as above. According to the composition theorem, combining the above analysis and the security of SC, we can claim that SRA protocol is secure under the semi-honest adversary model. ■

Theorem 6: If the cryptographic protocols for all the layers are secure against the semi-honest adversary, then secure neural network inference is secure under the semi-honest adversary model.

Proof: The secure neural network inference is stitched together from any number of cryptographic protocols for different layers. The correctness of the protocol is trivial given the definition of the underlying functions for all the layers. As for security, except for the input and output, the exchange messages in the protocol are the intermediate and output values of the cryptographic protocols for the layers. Due to the function and security of these protocols, the exchange messages are individually uniform. Thus, in the case of a corrupted P_0 or P_1, the simulator just chooses values uniformly at random. According to the composition theory, we can claim that the secure neural network inference is secure under the semi-honest adversary model, while combining the above analysis together with Theorems 1–5. ■

B. Performance Analysis

We now provide an analysis of the performance of Manto's secure layer protocols regarding computation cost, communication overhead, and round complexity. Table III summarizes the performance analysis of our SQA protocol, SC protocol, and SRA protocol. As a comparison, we also list the corresponding results of Delphi [8]. Recall that Delphi uses ASS and additive homomorphic encryption for the SQA function and uses GCs for the SC and SRA functions. For the sake of simplicity, we mainly consider several time-consuming operations. Specifically, we use $\mathsf{C_{Enc}}$, $\mathsf{C_{Dec}}$, $\mathsf{C_{Add}}$, and $\mathsf{C_{Mul}}$ to represent the computational cost of encryption, decryption, a homomorphic addition, and a homomorphic multiplication (with plaintext), respectively. We also use $\mathsf{C_{Gate}}$ to represent the computation cost of a non-XOR gate computation. Besides, we also use ℓ, ℓ_c, and η to represent the bit length of all data, the bit length of the ciphertext, and the GC's security parameter, respectively.

Computation Cost: We first analyze the computation cost of SQP protocol (shown in Algorithm 2). In line 2, P_0 computes $[\langle a\rangle_0]$, $[\langle b\rangle_0]$, $[\langle c\rangle_0]$, $[\langle a\rangle_0^2]$, $[\langle a\rangle_0 \cdot \langle b\rangle_0]$ by five encryptions, which costs $5\mathsf{C_{Enc}}$. In line 5, P_1 computes d_1, d_2, d_3, d_4, which costs $4\mathsf{C_{Enc}} + 9\mathsf{C}_{Add} + 9\mathsf{C}_{Mul}$. In line 6, P_0 decrypts d_1, d_2, d_3, d_4 by four decryptions, which costs $4 \cdot \mathsf{C_{Dec}}$. Therefore, the offline computation cost of SQA protocol is $9 \cdot \mathsf{C_{Enc}} + 4 \cdot \mathsf{C_{Dec}} + 9 \cdot \mathsf{C_{Add}} + 9 \cdot \mathsf{C_{Mul}}$. In the online stage of SQA protocol (shown in Algorithm 3), there is no time-consuming operations.

In the offline stage of SR protocol (shown in Algorithm 4), P_0 computes $[\langle\beta_i\rangle_0] (0 \le i < \ell)$ by ℓ encryptions (in line 4), which costs $\ell \cdot \mathsf{C_{Enc}}$. In line 5, P_1 computes $[\langle\theta_i\rangle_0] (0 \le i < \ell)$ by ℓ encryptions, ℓ homomorphic addition operations, and ℓ homomorphic multiplication operations, which costs $\ell \cdot (\mathsf{C_{Enc}} + \mathsf{C_{Add}} + \mathsf{C_{Mul}})$. In line 8, P_1 computes $[\langle c_i\rangle_0] (0 \le i < \ell)$ by one encryption, $\ell + 1$ homomorphic addition operations,

TABLE III
PERFORMANCE ANALYSIS OF MANTO'S SECURE LAYER PROTOCOLS

Function	Solution	Computation Cost		Communication Overhead		Round Complexity	
		Offline	Online	Offline	Online	Offline	Online
SQA	Delphi	$9\mathsf{C}_{\mathsf{Enc}} + 4\mathsf{C}_{\mathsf{Dec}} + 9\mathsf{C}_{\mathsf{Add}} + 9\mathsf{C}_{\mathsf{Mul}}$	–	$9\ell_c$	6ℓ	6	3
	Manto	$5\mathsf{C}_{\mathsf{Enc}} + 4\mathsf{C}_{\mathsf{Dec}} + 8\mathsf{C}_{\mathsf{Mul}} + 7\mathsf{C}_{\mathsf{Add}}$	–	$9\ell_c$	6ℓ	2	1
SC	Delphi	–	$3\ell \cdot \mathsf{C}_{\mathsf{Gate}}$	–	$28\eta\ell$	–	$\mathcal{O}(1)$
	Manto	$(2\ell+1)\mathsf{C}_{\mathsf{Enc}} + \ell\mathsf{C}_{\mathsf{Dec}} + (2\ell+1)\mathsf{C}_{\mathsf{Add}} + 3\ell\mathsf{C}_{\mathsf{Mul}}$	–	$(2\ell_c+4)\cdot\ell$	$2\ell^2+\ell+2$	3	4
SRA	Delphi	–	$6\ell \cdot \mathsf{C}_{\mathsf{Gate}}$	–	$56\eta\ell$	–	$\mathcal{O}(1)$
	Manto	$(4\ell+5)\mathsf{C}_{\mathsf{Enc}} + 2\ell\mathsf{C}_{\mathsf{Dec}} + (2\ell+4)\mathsf{C}_{\mathsf{Add}} + (6\ell+2)\mathsf{C}_{\mathsf{Mul}}$	–	$(4\ell+3)\ell_c + 8\ell$	$4\ell^2+2\ell+12$	3	6

TABLE IV
OUTLINE OF OUR EXPERIMENTS

Section	Content
Section VIII-B	Evaluation of Model Generation and comparative results with the state-of-the-art schemes
Section VIII-C	Benchmarks of the cryptographic protocols including SQA, SC, and SRA protocols
Section VIII-D	Evaluation on practical DNNs including ResNet-18 (on CIFAR-100) and ResNet-34 (on TinyImageNet)

and ℓ homomorphic multiplication operations, which costs $\mathsf{C}_{\mathsf{Enc}} + (\ell+1)\cdot\mathsf{C}_{\mathsf{Add}} + \ell\cdot\mathsf{C}_{\mathsf{Mul}}$. In line 9, P_1 computes $[\langle c_i'\rangle_0](0 \le i < \ell)$ by ℓ homomorphic multiplication operations, which costs $\ell\cdot\mathsf{C}_{\mathsf{Mul}}$. In line 12, P_0 decrypts $[\langle c_i'\rangle_0](0 \le i < \ell)$ by ℓ decryptions, which costs $\ell\cdot\mathsf{C}_{\mathsf{Dec}}$. Therefore, the offline computation cost of SC protocol is $(2\ell+1)\cdot\mathsf{C}_{\mathsf{Enc}} + \ell\cdot\mathsf{C}_{\mathsf{Dec}} + (2\ell+1)\cdot\mathsf{C}_{\mathsf{Add}} + 3\ell\cdot\mathsf{C}_{\mathsf{Mul}}$. In the online stage of SC protocol (shown in Algorithm 5), there is no time-consuming operations.

SRA protocol is implemented by invoking SC protocol twice and two multiplications on secret-shared data. Therefore, the offline computation cost of SRA protocol is $(4\ell+5)\cdot\mathsf{C}_{\mathsf{Enc}} + 2\ell\cdot\mathsf{C}_{\mathsf{Dec}} + (2\ell+4)\cdot\mathsf{C}_{\mathsf{Add}} + (6\ell+2)\cdot\mathsf{C}_{\mathsf{Mul}}$.

Communication Overhead and Round Complexity: We first analyze the communication overhead and round complexity of SQA protocol. In the offline stage of SQA protocol, P_0 sends $[\langle a\rangle_0]$, $[\langle b\rangle_0]$, $[\langle c\rangle_0]$, $[\langle a\rangle_0^2]$, $[\langle a\rangle_0\cdot\langle b\rangle_0]$ to P_1 (in line 2), which spends $5\ell_c$ bits. In line 5, P_1 sends d_1, d_2, d_3, d_4 to P_0, which spends $4\ell_c$ bits. Therefore, in the offline stage, the communication overhead of SQA protocol is $9\ell_c$ bits and the number of communication rounds is 2. In the online stage of SQP protocol, P_0 and P_1 exchange the additive shares of u, v, w (in line 2), which spends 6ℓ bits and one round of communication.

In the offline stage of SC protocol, P_0 sends $[\langle\beta_i\rangle_0](0 \le i < \ell)$ to P_1 (in line 4), which spends $\ell\cdot\ell_c$ bits. In line 11, P_1 sends $[\langle c_i'\rangle_0](0 \le i < \ell)$ to P_0, which spends $\ell\cdot\ell_c$ bits. In line 14, P_0 and P_1 invoke $\mathsf{B2A}$ protocol, which spends 4ℓ bits. Therefore, in the offline stage, the communication overhead of SC protocol is $(2\ell_c+4)\ell$ bits and the number of communication rounds is 3. In the online stage of SC protocol, P_1 sends temp to P_0 (in line 2), which spends ℓ bits. In line 5, P_0 sends $\langle\beta_i\rangle_1(0 \le i < \ell)$ to P_1, which spends ℓ^2 bits. In line 12, P_1 sends $\langle c_i'\rangle_1(0 \le i < \ell)$ to P_0, which spends ℓ^2 bits. In line 14, P_1 and P_0 exchange the boolean shares of ϕ, which spends 2 bits. Therefore, in the online stage, the communication overhead of SC protocol is $2\ell^2+\ell+2$ bits and the number of communication rounds is 4.

SRA protocol involves two calls on SC protocol and two multiplications on secret-shared data. Note that the two calls on SC protocol can be executed in parallel. Therefore, in the offline stage, the communication overhead of SRA protocol is $4\ell^2+2\ell+12$ and the number of communication rounds is 3. In the online stage, the communication overhead of SRA protocol is $(4\ell+3)\ell_c+8\ell$ and the number of communication rounds is 6.

VIII. EXPERIMENT

The content of the experiment consists of three parts as shown in Table IV.

A. Implementation and Experimental Setup

We implemented our protocols using C++ and Python. We use the cryptosystem of Paillier [38] for AHE, and apply data-packing technique [47] for Paillier's cryptosystem. The SQA, SC, and SRA protocols are written in C++. The implementations of local model training are built on PyTorch. The implementations of all our protocols are multithreaded. To ensure security and accuracy, we set the bit length of each value $\ell = 64$, and the statistical security parameter $\sigma = 100$.

We use a Jetson Nano to act as the client (P_0), which runs Ubuntu 18.04 and is equipped with Quad-core ARM Cortex-A57 MPCore processor, 4-GB RAM, and NVIDIA Maxwell architecture with 128 NVIDIA CUDA cores. We also use a physical server to act as the server (P_1), which runs Ubuntu 16.04 and equipped with 2.5-GHz Intel Xeon processor with 4 cores, 16 GB of RAM, and an Nvidia 2080ti GPU. The communication bandwidth between two separate machines (as P_0 and P_1) in the practical WAN setting is 43 MBps. We conduct experiments over three standard data sets: The CIFAR-10 data set contains 60 000 RGB images of size 32×32 pixels in ten classes. CIFAR-100 contains the same number of images as CIFAR-10, but separates them into 100 classes instead of 10. The TinyImageNet data set contains 120 000 RGB images of size 64×64 pixels in 200 classes. We randomly select 80% of the whole data set used for training, and 20% for testing. We evaluate Manto on the several typical CNN architectures, including AlexNet [21], VGG-16 [22], and ResNet-18/34 [23], in which there are different types of layers and a varied range of model parameters.

B. Evaluation of Model Generation

1) Comparison With Delphi: We compare with Delphi [8], which proposes a planner for selectively replacing ReLU

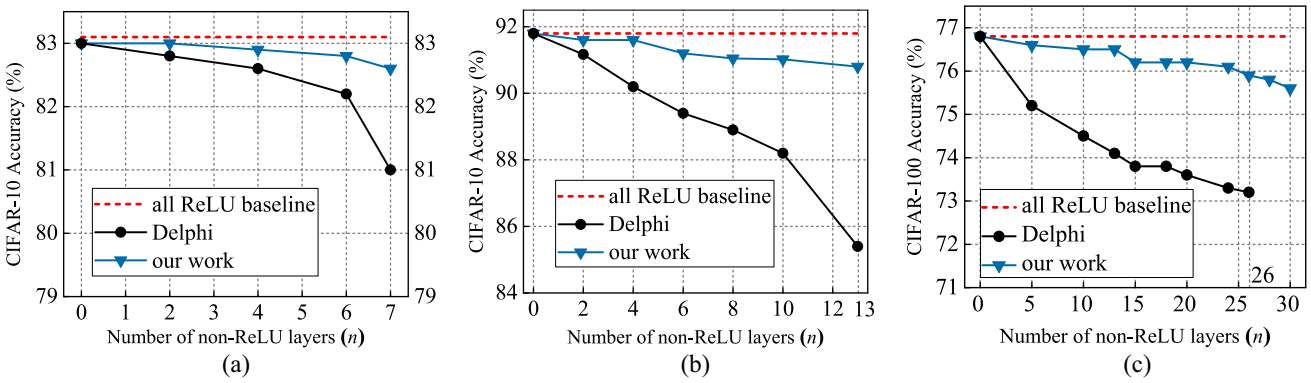


Fig. 6. Comparison of the inference accuracy between our work and Delphi. (a) CIFAR-10 accuracy of AlexNet. (b) CIFAR-10 accuracy of VGG-16. (c) CIFAR-100 accuracy of ResNet-34.

activations with the same quadratic function. Note that Manto and Delphi both replace ReLU activations in the layer-wise fashion. Following the model generation approaches described in Section V-B, we spend 0.4, 1.1, and 2.8 h to train AlexNet and VGG-16 on the CIFAR-10 data set, and ResNet-34 on the CIFAR-100 data set, respectively. It should be noted that the original AlexNet, VGG-16, and ResNet-34 networks have 7, 13, and 33 ReLU layers, respectively. At the precondition of keeping a comparatively high accuracy, our methods could replace all of the ReLU layers in AlexNet and VGG-16, and 30 of the ReLU layers in ResNet-34, with quadratic activation layers.

Fig. 6 plots the accuracy against the varying number (n) of non-ReLU layers for three CNN architectures, respectively. The blue line shows the accuracy using our training methods and the black line shows the accuracy using the Delphi's solution, while the dotted red line indicates the baseline, namely the accuracy of the all-ReLU network. Fig. 6(a) shows the accuracy comparison for AlexNet on CIFAR-10. For a varying number of non-ReLU (quadratic-activation) layers, the accuracy results of our work are superior to those of Delphi, and close to the all-ReLU baselines. And as the non-ReLU layers increase the advantage of our work continues to expand. In particular, after replacing all ReLU layers with quadratic-activation layers, the accuracy of our method achieves 82.6% and the accuracy of Delphi achieves 81%, with a difference of 1.6%. We observe a similar effect when training VGG-16 on the same data set as shown in Fig. 6(b). Fig. 6(c) illustrates this comparison for ResNet-34 on CIFAR-100. The accuracy of our work is always better than that of Delphi for the different numbers of non-ReLU layers. After replacing 26 of the 33 ReLU layers, the accuracy of our method reaches 75.9% and the accuracy of Delphi achieves 73.2%, with a difference of 2.7%. We also observe that the black line is ending at $n = 26$. The underlying reason is that, by the training tricks of Delphi, the ResNet-34 model fails to train after replacing 26 ReLU layers with quadratic approximations. But in contrast, our work could achieve the training with a maximum of 30 non-ReLU layers, and the accuracy can reach 75.6%.

2) Comparison With Other State-of-the-Art: We compare Manto against the other current state-of-the-art: SAFENet [13], CryptoNAS [29], and DeepReDuce [14]. Recall that these

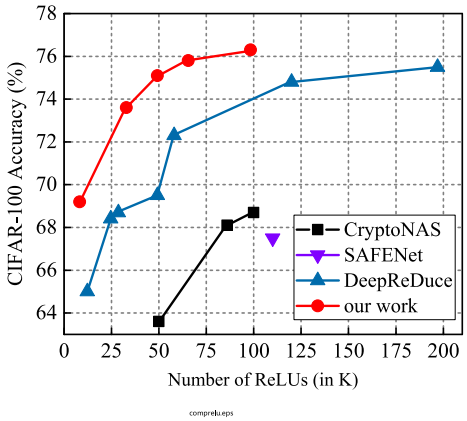


Fig. 7. CIFAR-100 accuracy of ResNet-18 models generated by our system and the state-of-the-art works.

works all aim to generate the models with few ReLUs for fast and accurate cryptographic inference, but adopt different technical routes. To demonstrate that Manto is able to produce models with competitive performance, we train ResNet-18 on CIFAR-100 with different accuracy requirements (through adjusting the accuracy threshold t described in Section V-B2). We then compare the number of ReLU activations in these models to that in other models produced by the state-of-the-art solutions. As can be seen from Fig. 7, our Manto ResNet-18 model significantly outperforms other models generated by SAFENet, CryptoNAS, and DeepReDuce on CIFAR-100. Typically, our model achieves 69.2% accuracy using only 8.2K ReLUs and 76.3% accuracy using 98.3K ReLUs, achieving a new state-of-the-art accuracy for different ReLU budgets. When the ReLU budget is $\sim$ 50K, CryptoNAS and DeepReDuce models achieve 63.6% and 69.5% accuracy, respectively. Meanwhile, our model can achieve 75.1% accuracy, which outperforms CryptoNAS and DeepReDuce by 11.5% and 5.6% accuracy, respectively. SAFENet reports 67.5% accuracy with 110K ReLUs, our model provides 8.8% more accuracy at the similar ReLU budget.

3) Disentangling Customized Fitting Functions and Siding-Window-Based Fine-Tuning: To disentangle the impact of the customized fitting functions (CF) and the sliding-window-based fine-tuning (SWF) on the inference accuracy, we train

TABLE V
INFERENCE ACCURACY WITH THE DIFFERENT OPTIMIZED METHODS (CF: CUSTOMIZED FITTING FUNCTIONS AND SWF: SLIDING-WINDOW-BASED FINE-TUNING)

Method	AlexNet (CIFAR-10)	VGG-16 (CIFAR-10)	ResNet-34 (CIFAR-100)
Delphi's model	81%	85.4%	73.2%
CF	81.9%	88.6%	74.7%
CF+SWF	82.6%	90.8%	75.9%

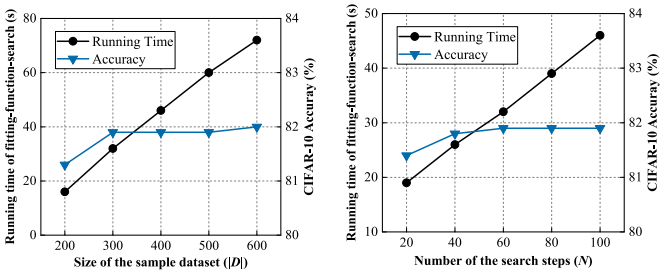


Fig. 8. Running time of fitting-function-search for training AlexNet and inference accuracy. (a) Impact of the size of the sample data set ($|D|$) when the number of search steps $N = 60$. (b) Impact of number of the search steps (N) when the size of the sample data set $|D| = 300$.

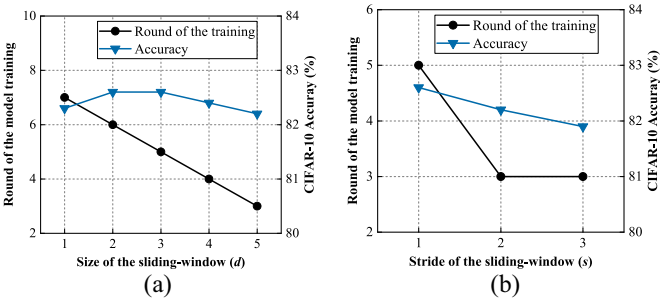


Fig. 9. Round of the model training for AlexNet and inference accuracy. (a) Impact of the size of the sliding-windows (d) when the stride of the sliding-window $s = 1$. (b) Impact of the stride of the sliding-window (s) when the size of the sliding-windows $d = 3$.

the Delphi's AlexNet, VGG-16, and ResNet-34 models (with replacing 7, 13, and 26 ReLU layers, respectively) [8] by adding the two methods step by step, and record the results of each step. Table V lists the inference accuracy with the different optimized methods. We can find that replacing the consistent quadratic activation used in Delphi with the different fitting functions is feasible for improving the accuracy, because our method can fit the characteristics of the input data itself better. Specifically, the Delphi's AlexNet, VGG-16, and ResNet-34 models with the CF method can reach 81.9%, 88.6%, and 74.7% accuracy, while the models with the consistent quadratic activation (in Delphi) reach 81%, 85.4%, and 73.2%. Then, we apply both our customized fitting functions and our sliding-window-based fine-tuning, and achieve higher accuracy, suggesting the effectiveness of our fine-tuning.

Now, take AlexNet for instance, we assess the impact on inference accuracy and fitting-function-search running time when varying the size of sample data set $|D|$ and varying the number of search steps N. Note that in order to get rid of the effect of the accuracy induced by sliding-window-based fine-tuning, we do not use this trick to optimize the model training. Fig. 8(a) shows that when the number of search steps $N = 60$, the running time of the fitting-function search grows linearly as the size of the sample data set ($|D|$) increases. We also observe that the inference accuracy tends to stay the same after $|D|$ is over 300, which confirms that this scale of the data set is enough to represent the distribution characteristics of the input data. Similarly, Fig. 8(b) shows that, when $|D| = 300$, the running time grows linearly as the number of search steps (N) increases while the inference accuracy tends to stay the same after N is over 60.

Next, we assess the impact on inference accuracy and the round of the model training when varying the size (d) and stride (s) of the sliding-window. As shown in Fig. 9(a), when the stride of the sliding-window $s = 1$, the round of the model training for AlexNet is linearly decreased with the increase of the size of the sliding-windows. We also find that the accuracy is the highest when $d = 2$ and $d = 3$. This is because that, the accuracy is difficult to be further improved through fine-tuning one layer's parameters ($d = 1$). But, when $d > 3$, the accumulated error of multiple layers is liable to cause the gradient explosion. Fig. 9(b) shows that the round of the model training and the accuracy are both decreased with the increase of the stride of the sliding-windows (s). The reason is, while d has been fixed, the stride is smaller and the fine-tuning is more detailed. But the disadvantage is the more epochs for training. We also perform the above parametric evaluations on VGG-16 (for CIFAR-10) and ResNet-34 (for CIFAR-100). Figs. 10–13 in Appendix B shows the consistent results as that in AlexNet, suggesting the general applicability of the above parameter setting.

C. Microbenchmarks of Cryptographic Protocols

We provide microbenchmarks of the cryptographic protocols for quadratic activations and ReLU6 activations, including SQA, SC, and SRA protocols. We compare with Delphi's protocols, because SAFENet and DeepReDuce both use the Delphi's protocols to implement cryptographic inference. Table VI shows the computation time and communication costs of these protocols. The online time of SQA protocol is very short, which only takes 0.03μs. On the other hand, our offline communication cost is a little larger than Delphi's. This is because that SQA protocol offloads input-independent operations to the offline stage as many as possible, such that it simply invokes basic arithmetic operations and one-round interaction. In terms of the SC, the online and offline running time of SC protocol are both less than those of Delphi. In particular, the online running time of SC protocol is 3.4 times faster than Delphi's protocols, while the offline running time is 2.1 times faster. In addition, our online and offline communication are both lower than Delphi's. It is because that SC protocol utilizes the offline/online paradigm to remove all time-consuming cryptographic operations from the online stage. In contrast, Delphi uses GCs for the SC, making it suffer from a lot of overhead in both online and offline stages. Recall that the major operations of the SRA protocol are two invokers of the SC protocol. Due to the state-of-the-art performance of the SC protocol, the SRA protocol also outperforms the Delphi's protocol in both running time and communication cost.

TABLE VI
RUNNING TIME (IN μS) AND COMMUNICATION COST (IN KB) OF CRYPTOGRAPHIC PROTOCOLS IN MANTO AND DELPHI (QUAD: QUADRATIC ACTIVATION, COMP: COMPARISON, AND RELU6: RELU6 ACTIVATION)

Function	System	Running time		Comm. cost	
		Online	Offline	Online	Offline
Quad	Delphi	0.04	10	0.008	0.16
	Manto (SQA)	0.03	8	0.005	0.19
Comp	Delphi	145.8	326.5	2.1	17.5
	Manto (SC)	42.4	153.4	0.6	15.2
ReLU6	Delphi	260.4	519.1	4.3	35.8
	Manto (SRA)	87.8	328.1	1.4	31.3

TABLE VII
PERFORMANCE OF MANTO ON THE RESNET-18/34 (#RELU: THE NUMBER OF RELUS, ACC: INFERENCE ACCURACY ON CIFAR-100 (C100)/TINYIMAGENET (TIN), AND RUNNING TIME IS IN SECONDS, COMMUNICATION COST IS IN GB)

Model	#ReLU	Acc	Running time		Comm. cost	
			Online	Offline	Online	Offline
ResNet-18 (C100)	32.8K	71.4%	0.72	14.19	0.12	0.32
	16.4K	70.9%	0.36	7.03	0.08	0.15
	8.2K	69.2%	0.19	3.84	0.04	0.07
ResNet-34 (TIN)	327.7K	67.3%	8.28	153.5	1.48	3.06
	262.1K	65.9%	6.78	121.56	1.16	2.24
	196.6K	64.8%	5.21	93.2	0.93	1.58

TABLE VIII
PERFORMANCE COMPARISON WITH DELPHI AND DEEPREDUCE ON THE RESNET-18/34 (#RELU: THE NUMBER OF RELUS, ACC: INFERENCE ACCURACY ON CIFAR-100 (C100)/TINYIMAGENET (TIN), LAT: ONLINE INFERENCE TIME IN SECONDS, AND COMM: ONLINE COMMUNICATION COST IN GB)

Model	System	#ReLU	Acc	Lat	Comm
ResNet-18 (C100)	Delphi	49.2K	68%	1.55	0.26
	DeepReDuce	28.6K	68.7%	0.74	0.15
	Manto	8.2K	69.2%	0.19	0.04
ResNet-34 (TIN)	Delphi	660.4K	63.7%	23.58	3.28
	DeepReDuce	520.4K	64.5%	15.42	2.55
	Manto	196.6K	64.8%	5.21	0.93

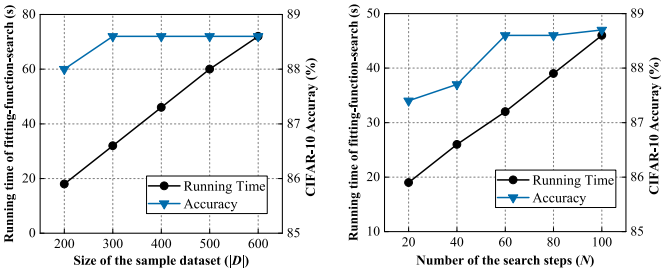


Fig. 10. Running time of fitting-function-search for training VGG-16 and inference accuracy. (a) Impact of the size of the sample data set ($|D|$) when the number of search steps $N = 60$. (b) Impact of the number of search steps (N) when the size of the sample data set $|D| = 300$.

D. Performance Evaluation on Practical CNNs

With all our cryptographic protocols for various layers and implementation optimizations in place, we conduct experiments over ResNet-18 (on the CIFAR-100 data set) and ResNet-34 (on the TinyImageNet data set). Table VII shows the performance of Manto on the ResNet-18/34. We observe

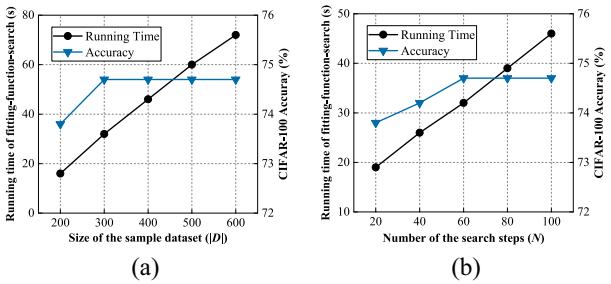


Fig. 11. Running time of fitting-function-search for training ResNet-34 and inference accuracy. (a) Impact of the size of the sample data set ($|D|$) when the number of search steps $N = 60$. (b) Impact of the number of search steps (N) when the size of the sample data set $|D| = 300$.

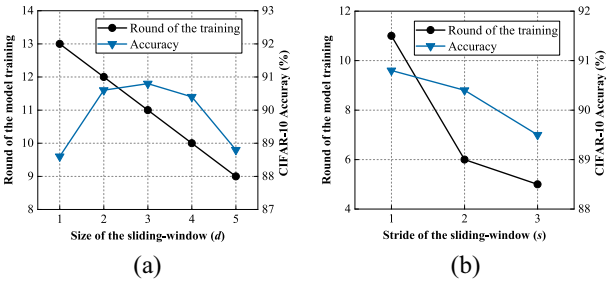


Fig. 12. Round of the model training for VGG-16 and inference accuracy. (a) Impact of the size of the sliding-windows (d) when the stride of the sliding-window $s = 1$. (b) Impact of the stride of the sliding-window (s) when the size of the sliding-windows $d = 3$.

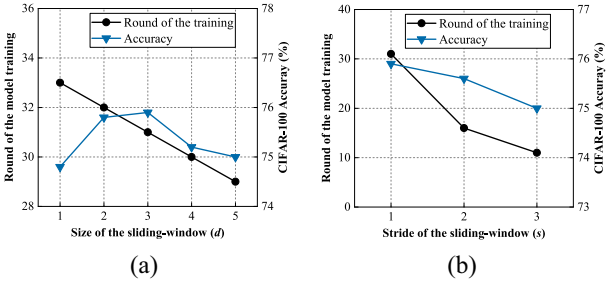


Fig. 13. Round of the model training for ResNet-34 and inference accuracy. (a) Impact of the size of the sliding-windows (d) when the stride of the sliding-window $s = 1$. (b) Impact of the stride of the sliding-window (s) when the size of the sliding-windows $d = 3$.

that the inference accuracy, computation, and communication costs correlate with the number of ReLUs, and most of the computation and communication costs are offloaded to the offline stage for improving online performance. Table VIII lists the performance comparison with Delphi and DeepReDuce. We find that, under the same level of inference accuracy, our work reduces online inference time for evaluating ResNet-18 (on CIFAR-100) by 87.7% and 74.3%, respectively, compared to Delphi and DeepReDuce. In terms of online communication cost, Manto is 6.5 times and 3.7 times lower than Delphi and DeepReDuce. Similarly, for evaluating ResNet-34 on TinyImageNet, our work obtains the best performance results. In particular, compared to Delphi and DeepReDuce, Manto reduces online inference time by 77.9% and 66.2%, respectively. Meanwhile, the online communication cost of our work is 3.5 times and 2.7 times lower than that of Delphi and DeepReDuce. This is partly because our

Algorithm 7 Original DGK Protocol

Input: P_0 inputs an integer β, P_1 inputs an integer α
Output: P_0 outputs λ_0, P_1 outputs λ_1, where $\lambda_0 \oplus \lambda_1 = (\alpha \leq \beta)$

1: P_1 decomposes $\alpha_{\ell-1} \ldots \alpha_1 \alpha_0 \leftarrow \alpha$
2: P_1 chooses a uniformly random bit $\lambda_1 \in \{0, 1\}$, and sets $s = 1 - 2\lambda_1$
3: P_0 decomposes $\beta_{\ell-1} \ldots \beta_1 \beta_0 \leftarrow \beta$, sends $[\beta_i] (0 \leq i < \ell)$ to P_1
4: **For** $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
5: $\quad P_1$: $[\alpha_i \oplus \beta_i] = [\beta_i]^{1-2\alpha_i} \cdot \alpha_i$
6: **End for**
7: **For** $i \in \{\ell - 1, \ldots, 1, 0\}$ **do**
8: $\quad P_1$: $[c_i] = [s] \cdot [\alpha_i] \cdot [\beta_i]^{-1} \cdot (\prod_{j=i+1}^{\ell-1} [\alpha_j \oplus \beta_j])^3$
9: $\quad P_1$ blinds c_i by raising them to a random exponent r_i of $2t$ bits: $[c_i'] = [c_i]^{r_i}$
10: **End for**
11: P_1 shuffles all $[c_i']$ and sends them to P_0
12: P_0 sets $\lambda_0 = 1$ if any decrypted c_i' is 0, otherwise $\lambda_0 = 0$

work can replace more ReLUs with the customized quadratic approximations than Delphi and DeepReDuce, especially in the large scale model, and partly because our SQA and SRA protocols produce state-of-the-art performance for quadratic activations and ReLU6 activations.

IX. Conclusion

In this article, we propose a practical and secure inference system of CNNs for IoT, which provides privacy guarantees for both the client's input data and the server's model. Technically, we propose a quadratic-fitting-function search algorithm and a fine-tuning method to generate accurate CNNs with fewer ReLU activations. Then, we design a set of secure protocols for quadratic activations and ReLU6 activations by leveraging lightweight cryptographic primitives (i.e., secret sharing and additive homomorphic encryption). These efforts enable Manto to reduce online inference time by 77.9%–87.7% and 66.2%–74.3% compared to the state-of-the-art solutions Delphi and DeepReDuce, respectively.

Appendix A
Review of DGK Protocol

Algorithm 7 describes the original DGK protocol [45], which takes two ℓ-bit integers α and β as inputs and returns the boolean shares of $(\alpha \leq \beta)$. The correctness and security of the DGK protocol can be found in [45].

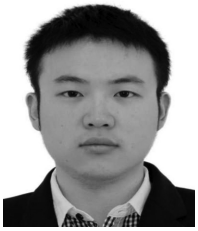
Appendix B
Parametric Evaluations for Model Generation on VGG-16 and ResNet-34

Take VGG-16 and ResNet-34 for instances, Figs. 10 and 11 show the impact on inference accuracy and fitting-function-search running time when varying the size of sample data set $|D|$ and varying the number of search steps N. Figs. 12 and 13 show the impact on inference accuracy and the round of the model training when varying the size (d) and stride (s) of the sliding-window. We can find the consistent results as that in AlexNet, suggesting the general applicability of the parameter setting.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[2] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th {USENIX} Secur. Symp.*, 2016, pp. 601–618.
[3] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt, "Model reconstruction from model explanations," in *Proc. Conf. Fairness Accountabil. Transparency*, 2019, pp. 1–9.
[4] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI neural network: Using side-channels to recover your artificial neural network information," 2018, *arXiv:1810.09076*.
[5] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Proc. Annu. Int. Cryptol. Conf.*, 2020, pp. 189–218.
[6] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2017, pp. 19–38.
[7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1651–1669.
[8] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *Proc. 29th {USENIX} Security Symp.*, 2020, pp. 2505–2522.
[9] M. Li, S. S. M. Chow, S. Hu, Y. Yan, S. Chao, and Q. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 3, pp. 1592–1604, May/Jun. 2022.
[10] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2020, pp. 336–353.
[11] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. 2020 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 325–342.
[12] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Feb. 2019, pp. 1–15.
[13] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "SAFENet: A secure, accurate and fast neural network inference," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp 1–13.
[14] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "DeepReDuce: Relu reduction for fast private inference," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 4839–4849.
[15] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, 1986, pp. 162–167.
[16] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," 2018, *arXiv:1811.09953*.
[17] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. 5th Int. Conf. Learn. Represent.*, 2017, pp. 1–17.
[18] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.
[19] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.
[20] R. Dathathri et al., "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, 2019, pp. 142–156.
[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1106–1114.
[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[24] L. Hanzlik et al., "MLCapsule: Guarded offline deployment of machine learning as a service," 2018, *arXiv:1808.00590*.

[25] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–19.
[26] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference," 2018, *arXiv:1810.00602*.
[27] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*.
[28] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via miniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 619–631.
[29] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "CryptoNAS: Private inference on a ReLU budget," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 16961–16971.
[30] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "MediSC: Towards secure and lightweight deep learning as a medical diagnostic service," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2021, pp. 519–541.
[31] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.
[32] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. Privacy Enhanc. Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.
[33] L. Shen, X. Chen, J. Shi, Y. Dong, and B. Fang, "An efficient 3-party framework for privacy-preserving neural network inference," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2020, pp. 419–439.
[34] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation," in *Proc. NDSS*, 2015, pp. 1–15.
[35] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara, "Private collaborative forecasting and benchmarking," in *Proc. ACM Workshop Privacy Electron. Soc.*, 2004, pp. 103–114.
[36] D. Beaver, "Precomputing oblivious transfer," in *Proc. Annu. Int. Cryptol. Conf.*, 1995, pp. 97–109.
[37] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely Outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 1, pp. 620–636, Jan./Feb. 2023.
[38] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, vol. 99. Berlin, Germany: Springer, 1999, pp. 223–238.
[39] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[40] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2017, pp. 3–18.
[41] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High-fidelity extraction of neural network models," 2019, *arXiv:1909.01838*.
[42] R. Iyengar, J. P. Near, D. Song, O. Thakkar, A. Thakurta, and L. Wang, "Towards practical differentially private convex optimization," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2019, pp. 299–316.
[43] M. Pelikan, "Bayesian optimization algorithm," in *Hierarchical Bayesian Optimization Algorithm*. Berlin, Germany: Springer, 2005, pp. 31–48.
[44] G. Chen, Z.-L. Ren, and H.-Z. Sun, "Curve fitting in least-square method and its realization with MATLAB," *Ordnance Ind. Autom.*, vol. 3, no. 2005, p. 63, 2005.
[45] T. Veugen, "Improving the DGK comparison protocol," in *Proc. IEEE Int. Workshop Inf. Forensics Secur.*, 2012, pp. 49–54.
[46] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. CRYPTOLOGY*, vol. 13, no. 1, pp. 143–202, 2000.
[47] K. Cheng et al., "Strongly secure and efficient range queries in cloud databases under multiple keys," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 2494–2502.



Ke Cheng received the B.S. and M.S. degrees from Anhui University, Hefei, China, in 2015 and 2018, respectively, and the Ph.D. degree in computer science and technology from Xidian University, Xi'an, China, in 2022.

He is a Lecturer with the School of Computer Science and Technology, Xidian University. His research interests include cloud computing security, data security, and privacy protection.

Jiaxuan Fu received the B.S. degree from Xidian University, Xi'an, China, in 2019, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology.

His research interests include IoT data security and machine learning security.

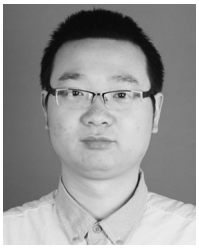
Yulong Shen (Member, IEEE) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2002, 2005, and 2008, respectively.

He is currently a Professor with the School of Computer Science and Technology, Xidian University, where he is also an Associate Director of the Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services Networks. His research interests include wireless network security and cloud computing security.

Dr. Shen has also served on the technical program committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.

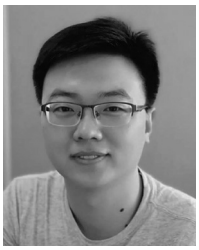
Haichang Gao (Member, IEEE) received the Ph.D. degree in computer science and technology from Xi'an Jiaotong University, Xi'an, Shaanxi, China, in 2006.

He is currently a Professor with the School of Computer Science and Technology, Xidian University, Xi'an. He has published more than 30 papers. He is currently in charge of a project of the National Natural Science Foundation of China. His current research interests include Captcha, computer security, and machine learning.



Ning Xi (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Xidian University, Xi'an, Shaanxi, China, in 2008, 2011, and 2014, respectively.

He is currently a Professor with the School of Cyber Engineering, Xidian University. His major research is in home network, service computing, and network security.



Zhiwei Zhang (Member, IEEE) received the Ph.D. degree in cryptography from Xidian University, Xi'an, Shaanxi, China, in 2019.

He is currently an Associate Professor with the School of Computer Science and Technology, Xidian University. His research interests include data secure management, data storage security, and data location verification in cloud computing.

Xinghui Zhu received the B.S. and M.S. degrees in computer science from Xidian University, Xi'an, China, in 2014 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology.

His research interests include data security and IoT security.