L-SecNet: Towards Secure and Lightweight Deep Neural Network Inference

Anxiao Song[1], Jiaxuan Fu[1], Xutong Mu[1], XingHui Zhu[1], and Ke Cheng[1]
[1]XIDIAN University 266 Xinglong Section of Xifeng Road, Xi'an, Shaanxi 710126

The advances in machine learning technology has promoted its great potential for deep neural network (DNN) inference powered applications of Internet of Things (IoT), such as facial verification cameras and speech recognition assistants. The current deployment of these applications also raises serious privacy concerns, especially when sensitive individual information is accessed easily by various IoT devices. Fortunately, the cryptography-based solutions are able to execute secure inference without infringing the user's raw data and the model owner's proprietary model. However, existing works suffer from impractically high latency and low accuracy, stemming primarily from the evaluations of the non-linear layers in DNN. In this paper, we propose L-SecNet, a lightweight secure neural network inference system that provides efficient inference services without sacrificing accuracy and privacy. Specifically, to reduce latency caused by comparison operations in non-linear layers, we subtly combine additive secret sharing and multiplicative secret sharing to design a lightweight secure comparison protocol. Further, we approximate the commonly used and time-consuming activation functions (including Sigmoid and Tanh functions) with the non-linear sin function instead of the linear polynomial approximation functions in the existing works. In order to maintain low running latency while meeting the requirement of high accuracy, a secure and lightweight protocol for a sin function is proposed. Our theoretical analysis and empirical experiments evaluate the security and efficiency of the L-SecNet system. Compared with the state-of-the-art works, L-SecNet saves up to about 80 times bandwidth and about 53 times runtime.

***Index Terms*—Secure neural network inference, Lightweight comparison, Accurate non-linear Function, Low latency.**

I. Introduction

The rapid advancement in machine learning has fueled the proliferation of IoT applications, including facial verification cameras [1] and speech recognition assistants [2]. Due to resource constraints, users and data owners often outsource raw user data and deep neural network models to resource-rich cloud service providers for efficient neural network inference. However, the raw data contains the user's personal privacy information, which raises serious privacy concerns. At the same time, the private model is also the digital asset of the model owner, and it also needs to be protected. Therefore, it is an urgent requirement to ensure the privacy of user data and models in pushing forward the practices of neural network inference in IoT.

Recently, there have been growing works on secure neural network inference [3]–[5] to meet the above pressing demand. They mainly leverage cryptographic techniques (e.g., Garbled Circuit [6], Oblivious Transfer [7], Secret Sharing [8] and other cryptographic knowledge.) to execute network inference over the encrypted data and model. Unfortunately, these cryptographic works cannot be easily satisfied in the practices since they involve enormous running latency throughout the whole online inference phase. In fact, this running latency mainly stems from the computation in the non-linear layers. In these layers, they have to consume enormous communication overhead to support a large number of cryptographic comparison operations. Especially, with the stacking of non-linear layers in the neural network architectures, the running latency caused by comparison operations is further unacceptable for users.

Manuscript received December 15, 2023; revised January 17, 2024. Corresponding author: JiaXuan Fu (email: JiaxuanFu@stu.xidian.edu.cn)

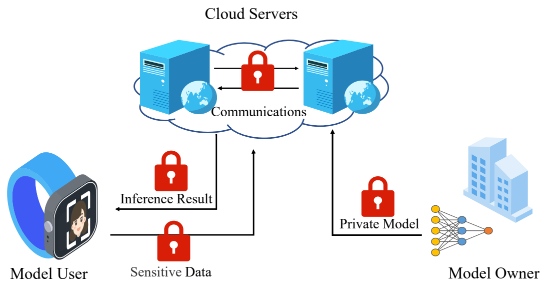

Fig. 1. Secure network inference system of the powerful network models

Therefore, slow cryptographic non-linear layers have become the primary obstruction to efficient secure inference services.

A few state-of-the-art works have also sought to study trade-off solutions in terms of computation and accuracy. They attempt to replace expensive non-linear layers with cheap polynomial approximation functions [9]–[11]. Unfortunately, [12]–[14] works have shown that these approximated polynomial approximation functions result in a loss of the model's accuracy since the errors of these approximation functions can bring about low-distortion outputs of the next layer. Although the accuracy of cryptographic inference can be improved by expanding the degree of the approximated polynomial function, the running latency grows exponentially with the degree. Therefore, the practical and secure inference service requires new optimization approaches to reduce the running latency from non-linear layers while maintaining inference accuracy.

To address the above issues, we propose a lightweight secure neural network inference system, L-SecNet, which can provide efficient inference services on sensitive data without sacrificing accuracy and privacy. As shown in Fig. 1, the user and the model owner firstly encrypt their own sensitive data and private model by the secret sharing technique, respectively. Secondly, the two cloud servers stitch the protocols of any layer in the neural network automatically and sequentially to obtain the entire computation of secure inference. And the computation only runs on the servers, which frees the resource of the user and model owner during the online inference phase. Finally, the produced result of the two servers is distributed to users in secret sharing. Besides, we ensure that the activation functions of non-linear layers are fast and accurate expressions without decreasing accuracy and compromising privacy. For ReLU and max-pooling in non-linear layers, we design a lightweight secure comparison protocol based on the combination of multiplicative secret sharing and additive secret sharing to reduce running latency. To maintain high accuracy with the requirement of low running latency, we design new approximation functions based on the non-linear sin function to replace Sigmoid and Tanh in the non-linear layers, rather than the linear polynomial approximation functions in the existing works. Accordingly, we present a lightweight secure protocol of the sin function to realize these approximation functions. We demonstrate the security and efficiency of L-SecNet system from theoretical analysis and empirical experiments.

Our contributions can be summarized as follows:

- We propose a lightweight and secure inference system, L-SecNet, which can protect privacy data for both user and model owner. L-SecNet implements all secure protocols of neural network layers based on additive secret sharing technology and automatically stitches the protocols of any layer to obtain the whole secure computation process of any neural network.
- For ReLU and max-pooling in non-linear layers, we design a lightweight secure comparison protocol by using multiplicative secret sharing and additive secret sharing in combination. The protocol reduces enormous overheads of communication and computation.
- For maintaining high accuracy of inference, we adopt the non-linear sin to approximate Sigmoid and Tanh in the non-linear layers, instead of the linear polynomial approximation functions. Furthermore, we devise a new lightweight secure protocol of the sin function to achieve the secure approximation of activation functions, which reduces the running latency of cryptographic inference.
- The security and efficiency of L-SecNet are demonstrated by the theoretical analysis and empirical experiments. Compared with the prior works, L-SecNet can reduce up to about $80\times$ bandwidth and $53\times$ computation cost in the online inference phase.

The remainder of this paper is organized as follows. In Section II, we briefly review some relevant preliminaries. Then we introduce the system workflow and threat model along with the design goal in Section III. We describe the secure comparison protocols in Section IV and the privacy-preserving protocols of network inference in Section V, followed by their security analysis in Section VI. Next, The performance evaluation is given in Section VII. Finally, we review related works in Section VIII and this paper is concluded in Section IX.

II. Related Works

Secure neural network inference has remarkable potential in many applications and has attracted more and more attention. CryptoNets [15] proposed by Gilad-Bachrach *et al.* was the first secure neural network, using homomorphic encryption technology in 2016. However, its practicality is limited by enormous running latency. SecureML [9] was the first work based on Secure MultiParty Computation (SMC) technology for secure training and inference. However, this work is still inefficient and can only support very simple models.

Therefore, more and more researchers have begun to study the low-running latency systems of secure neural network inference systems. Liu *et al.* [16] proposed an oblivious neural network prediction system, MiniONN, based on SPDZ protocol. Demmler *et al.* [17] proposed a hybrid learning framework, ABY, which uses Yao's GC, boolean secret sharing and arithmetic secret sharing to implement the nonlinear and the linear operations of SMC. Furthermore, Arpita *et al.* [18] presented a improved mixed-protocol with secure two-party computation, ABY^2, an optimization of ABY. Juvekar *et al.* [19] designed Gazelle framework with the optimized HE for linear layers and GC for non-linear layers. Yet, they introduce expensive cryptographic primitives to support the secure comparisons required in nonlinear layers, and it has been proved that their efficiency has not been greatly improved in deep neural networks [20]. In recent years, some studies have tried to design a compromise solution to approximate the nonlinear functions into the linear crypto-friendly polynomials, reducing running latency. Pratyush Mishra *et al.* [10] proposed a cryptographic neural networks inference service system, DELPHI, with selective quadratic approximation activations. Qian Lou *et al.* [20] proposed a secure and fast inference system for the neural network, SAFENet, with multiple degree and layer-wise mixed precision polynomials. Although these solutions speed up the inference services, they lead to a decline in inference accuracy. Especially for complex neural networks, replacing the nonlinear layer will significantly reduce the accuracy of their solutions.

Different from the above works, we design a new secure comparison protocol to reduce running latency produced by comparison in the nonlinear layers. Besides, we first introduce the nonlinear sin function to approximate the activation functions in the nonlinear layers and design a new lightweight secure protocol of sin function to implement these approximation functions, reducing running latency while retaining inference accuracy.

III. Preliminary

In this section, we review the cryptographic primitives of additive secret sharing [8]. For the sake of readability, Table I summarizes the notations used in our work.

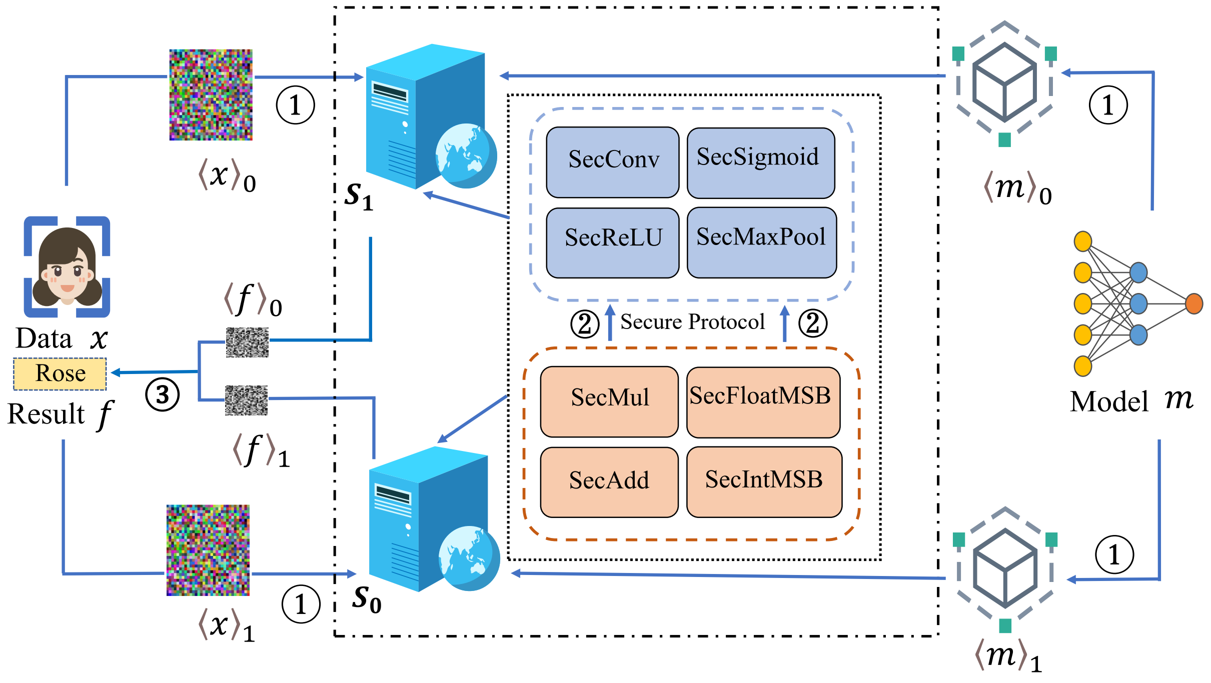

Fig. 2. Workflow of L-SecNet system: the user and model owner firstly send secret-sharing $\langle x\rangle$ and $\langle m\rangle$ into S_0 and S_1, respectively. Next, S_0 and S_1 allow secure two-party computation protocols to $\langle f\rangle = \langle m(x)\rangle$. Finally, user computes the inference result $f = \langle f\rangle_0 + \langle f\rangle_1$.

TABLE I
NOTATION DESCRIPTIONS

Notations	Definitions
$\langle x\rangle = (\langle x\rangle_0, \langle x\rangle_1)$	2-out-of-2 additive secret sharing of x
$\langle x\rangle_i$	Secret sharing of x is held by S_i for $i \in \{0,1\}$
$\langle f\rangle$	Secret sharing result of object function f
m	Neural network of model owner.
$\langle m\rangle_i$	Secret sharing of m is held by S_i for $i \in \{0,1\}$.

A. Additive secret sharing

In the additive secret sharing technique, all values are secret shares between two servers such that the addition of two secret shares yields the true value. In the following, we provide a concise overview of the secure addition and multiplication protocol in the additive secret sharing technique.

1) Secure Addition

In the secure addition protocol, the objective function is $f(x,y) = x+y$. Given the secret shares $\langle x\rangle, \langle y\rangle$, each server S_i performs $\langle f\rangle_i = \langle x\rangle_i + \langle y\rangle_i$ locally and individually without interaction with each other, and outputs the secret-sharing the result $\langle f\rangle_i$. Obviously, we have $f(x,y) = \langle f\rangle_0 + \langle f\rangle_1$ from two secure computation parties. Besides, if the objective function is $f(x,c) = x+c$ and c is a constant value, S_i performs $\langle f\rangle_i = i\times c + \langle x\rangle_i$, and outputs the secert-sharing result $\langle f\rangle_i$.

2) Secure Multiplication

In the secure multiplication (SecMul) protocol, the objective function $f(x,y) = x\times y$ is computed without leaking any privacy data x and y. In the protocol, Beaver's multiplication triples of the form $\{a = \langle a\rangle_0 + \langle a\rangle_1, b = \langle b\rangle_0 + \langle b\rangle_1, c = a\times b = \langle c\rangle_0 + \langle c\rangle_0\}$ firstly are constructed by using *Beaver's triplet* [21] technology in the offline stage. Secondly, each server S_i masks the privacy input $\langle u\rangle_i = \langle x\rangle_i - \langle a\rangle_i$ and $\langle e\rangle_i = \langle y\rangle_i - \langle b\rangle_i$ and sends them into S_{1-i}. Thirdly, the two servers reconstruct $u = \langle u\rangle_0 + \langle u\rangle_1$ and $e = \langle e\rangle_0 + \langle e\rangle_1$. Finally, each server S_i computes $\langle f\rangle_i = i\times e\times u + e\times\langle a\rangle_i + u\times\langle b\rangle_i + \langle c\rangle_i$ and outputs $\langle f\rangle_i$. If x, y, a, b and c are in the field of $\mathbb{Z}_2$, the addition and multiplication operations of the **SecMul** are converted into AND ($\wedge$) and XOR ($\oplus$) of bit operations, respectively. We denote the secure multiplication protocol in $\mathbb{Z}_2$ as **SecMul**$_2$. Besides, if the objective function $f(x,c) = c\times x$ and c is a constant value, S_i performs $\langle f\rangle_i = c\times\langle x\rangle_i$ and outputs $\langle f\rangle_i$.

Note that Beaver's multiplicative triples can be generated via homomorphic encryption (HE) or correlated oblivious transfer (COT) in an offline stage. Therefore, we assume that the triples are pre-computed and available for use in our system.

B. Multiplicative secret sharing

In the multiplicative secret sharing technique, all values are secret shares between two servers such that the Multiplication of two secret shares yields the true value, i.e, $u = \langle u\rangle_0 \times \langle u\rangle_1$. Similar to additive secret sharing, multiplicative secret sharing defines a threshold scheme. Note that it is not suitable to use multiplicative secret sharing over some fields such as Z_2 because if one of the shares is 0, the party could infer that the secret must also be 0, resulting in a privacy leak. the party could know the secret is 0. Therefore, it is necessary to handle the privacy leakage of the secret 0 carefully and correctly.

IV. SYSTEM OVERVIEW

A. System Workflow

L-SecNet system consists of three entities: model user, model owner, and two independent cloud servers. The model user has sensitive data x on which he wishes to apply inference services. The model owner has a proprietary model m for publishing into cloud servers to get paid. The two servers S_0 and S_1 run the encrypted model on the user's encrypted data together with our proposed secure protocols in the online phase. As illustrated in Fig. 2, we provide an overview of the workflow of secure inference services using L-SecNet system.

L-SecNet system works as follows: To protect the proprietary model m, the model owner generates secret shares $\langle m\rangle_0$ and $\langle m\rangle_1$ based on the secret sharing protocols Then, $\langle m\rangle_0$ and $\langle m\rangle_1$ are sent into the two servers S_0 and S_1, respectively. Besides, we develop all secure protocols of neural network layers based on additive secret sharing technology in the two servers. Once the secret share $\langle m\rangle_i$ arrives on each server, S_i automatically searches the execution flow of $\langle m\rangle_i$ according to its architecture. And S_i automatically stitches all secure protocols of network layers in order to obtain a whole secure protocol of inference service, enabling S_i to begin with the secret shares of the input layer and end with the secret shares of the output layer. Finally, the user protects his sensitive data x to generate the secret shares $\langle x\rangle_0$ and $\langle x\rangle_1$ with the secret sharing technique. And $\langle x\rangle_0$ and $\langle x\rangle_1$ are fed into the corresponding server. The two servers work coordinately to run the secure inference protocol and send the secret-shared outputs $\langle f\rangle_0$ and $\langle f\rangle_1$ into the model user after finishing the computation. The model user only adds the two outputs to compute the inference result f labeled the class of his private input.

B. Threat Model

Similar to prior works [22], [23] in the two-cloud-server setting, we assume that the model user, the model owner, and the two independent cloud servers from the different cloud providers all are semi-honest and non-colluding. In other words, each server will allow all protocols exactly as specified yet may attempt to learn the privacy information of the user or model owner from their known data. If corruption happens in our system, an adversary can compromise at most one of the two servers and either the model user or the model owner, while the other parties keep honest behavior. In fact, the reason behind our assumption is that cloud provider are unwilling to bear the risk of damaging their reputation and interest due to the strict privacy policy, thus avoiding behaving maliciously and colluding.

C. Design Goals

L-SecNet aims to build a secure and lightweight system for network inference services. Specifically, our goals are given as follows:

- *Correctness.* The encrypted outputs of all secure protocols have to be decrypted correctly.
- *Security.* Our inference system should be provably secure under the semi-honest threat model such that any privacy of the user and model owner can not be leaked from the cloud servers during the execution of all protocols.
- *Low running latency and High accuracy.* The primary objectives of our research is to minimize running latency in neural network inference processes, particularly in resource-constrained environments. Additionally, we will emphasize the importance of maintaining high accuracy in these inference processes, as this is a critical factor for the practical application of such systems.

V. LIGHTWEIGHT SECURE COMPARISON

In the comparison, the objective function is $f(x, y) = x < y$, where the function output $f = 0$ if and only if $x < y$, otherwise $f = 1$. To implement a secure comparison protocol, we combine multiplicative secret sharing with additive secret sharing to devise a lightweight secure comparison protocol. To determine the most significant bit (MSB) of the integer x with privacy protection, we first present a low-running latency secure integer MSB (**SecIntMSB**) protocol in the secret-shared domain. Besides, since most of the parameters in private inputs and neural networks are float numbers, we also construct the secure float MSB (**SecFloatMSB**) protocol based on the **SecIntMSB** protocol, which can compare the floating-point number x and 0 efficiently and securely through three interactions between S_0 and S_1 without leaking any privacy inputs. If the **SecFloatMSB** protocol is executed in parallel, **SecFloatMSB** protocol only requires two interactions between S_0 and S_1. Finally, we can rely on the two above protocols to naturally expand to the secure comparison (**SecCom**) protocol of two arbitrary numbers.

A. Secure MSB of Integer Protocol

SecIntMSB only requires one interaction between S_0 and S_1, reducing the communication overhead between the two servers. The protocol is illustrated in detail in Algorithm 1, and we have also proved the correctness of the algorithm. In Algorithm 1, our idea is to convert $x = \langle x\rangle_0 + \langle x\rangle_1$ to

Algorithm 1 Secure Integer MSB Protocol (SecIntMSB)

```
Input: $S_i$ has $\langle x\rangle_i \in \mathbb{F}$, where $i \in \{0,1\}$.
Output: $S_i$ outputs $\langle f\rangle_i$.
1: The multiplication triples $(a, b, c = a\times b+\eta = \langle c\rangle_0+\langle c\rangle_0)$
   are generated in the stage, where $\eta \in (0,1)$.
2: $(a, \langle c\rangle_0)$ and $(b, \langle c\rangle_1)$ are sent into $S_0$ and $S_1$, respectively.
3: $S_0$ computes $e = (\langle x\rangle_0 - \langle c\rangle_0)/a$. and sends $e$ into $S_1$.
4: $S_1$ computes $\langle u\rangle_1 = e + b$ and $d = (\langle x\rangle_1 - \langle c\rangle_1)/\langle u\rangle_1$.
5: $S_1$ sends $d$ into $S_0$.
6: $S_0$ computes $\langle u\rangle_0 = d + a$.
7: $S_i$ extracts the secret-sharing MSB $\langle f\rangle_i = \langle u\rangle_i > 0$.
```

$x = \lfloor \langle x\rangle_0 + \langle x\rangle_1 + \eta\rfloor = \lfloor \langle u_0\rangle \times \langle u_1\rangle\rfloor$ for obtaining the signs of x, where $\lfloor . \rfloor$ indicates a rounding-down operation. Then, we are only required to extract the MSB of $\langle u_0\rangle$ and $\langle u_1\rangle$, respectively, and only perform XOR operation to obtain the

MSB of x. Thus, we only need to prove $x = \lfloor \langle x\rangle_0 + \langle x\rangle_1 + \eta \rfloor = \lfloor \langle u\rangle_0 \times \langle u\rangle_1 \rfloor$ is correct, which is given as follows:

$$\begin{aligned}\langle u\rangle_0 \times \langle u\rangle_1 &= (d+a)\times(e+b)\\ &= (e+b)\times d + a\times e + ab\\ &= \langle x\rangle_0 - \langle c\rangle_0 + \langle x\rangle_1 - \langle c\rangle_1 + ab\\ &= \langle x\rangle_0 + \langle x\rangle_0 + \eta\\ &\approx \lfloor \langle x\rangle_0 + \langle x\rangle_1 + \eta \rfloor\\ &= x.\end{aligned} \tag{1}$$

Note that, $\eta \in (0,1)$ is introduced to avoid leaking the privacy input $x = 0$ on the online computation in S_1. However, η does not change the sign of any $x \in \mathbb{Z}$, since $x+1 > x+\eta > x$ is correct.

B. Secure MSB of Float Protocol

SecFloatMSB protocol is based on the **SecIntMSB** protocol, which can discriminate the sign of any float number $x \in F$ with preserving the privacy of the data. The protocol only requires three interactions to share the sign of x between S_0 and S_1 on the online phase. The realization of **SecFloatMSB** protocol is shown in detail in Algorithm 2.

Algorithm 2 Secure Float MSB Protocol (SecFloatMSB)

```
Input: $S_i$ has $\langle x\rangle_i \in \mathbb{F}$, where $i \in \{0,1\}$.
Output: $S_i$ outputs $\langle f\rangle_i$.
1: $S_i$ extracts the integer part $\langle x_{int}\rangle_i$ of $\langle x\rangle_i$.
2: $S_i$ computes $\langle \alpha\rangle_i$ = SecIntMSB($\langle x_{int}\rangle_i$) and $\langle \beta\rangle_i$ = SecIntMSB($\langle -x_{int}\rangle_i$).
3: $S_i$ computes $\langle \kappa\rangle_i = \langle \alpha\rangle_i \oplus \langle \beta\rangle_i \oplus (i \wedge 1)$
4: $S_i$ turns $\langle x\rangle_i - \langle x_{int}\rangle_i$ into a integer number $\langle x_{frac\to int}\rangle_i$.
5: $S_i$ computes $\langle \tau\rangle_i$ =SecIntMSB($\langle x_{frac\to int}\rangle_i$).
6: $S_i$ outputs $\langle f\rangle_i$ = SecMul$_2$($\langle \kappa\rangle_i, \langle \alpha\rangle_i$)+SecMul$_2$($(1\wedge i)\oplus \langle \kappa\rangle_i, \langle \tau\rangle_i$).
```

$$\begin{aligned}MSB(x) &= \begin{cases}\langle x_{int}\rangle_0 > \langle x_{int}\rangle_0 & \alpha \neq \beta\\ \langle x\rangle_0 - \langle x_{int}\rangle_0 > \langle x\rangle_1 - \langle x_{int}\rangle_1 & \alpha = \beta\end{cases}\\ &= \kappa \times \alpha + (1-\kappa)\times \tau\end{aligned} \tag{2}$$

where κ = 1 iif $\alpha \neq \beta$, and α, β, τ are the signs of x_{int},$-x_{int}$ and $x_{frac\to int}$, respectively.

The correctness of the SecFloatMSB protocol, as outlined in Algorithm 2, can be easily established. A crucial aspect of this protocol is the handling of floating-point numbers to prevent division by zero exceptions,as followed in Eq. (2), a common issue when extracting the MSB based on the SecIntMSB protocol. To address this, we divide the floating-point number x into two components: the integer part x_{int} and the fractional part $x - x_{int}$. We then convert the fractional part into an integer $\bar{x}$, ensuring that our protocol can operate without encountering division by zero errors. The sign of x is determined based on these components. If x_{int} differs from its negative, i.e., if $x_{int} \neq -x_{int}$, then the sign of x is the same as that of x_{int}. In cases where x_{int} is equal to its negative, the sign of x is derived from the integer representation of the fractional part, denoted as $x_{frac\to int}$.

VI. LIGHTWEIGHT PRIVACY-PRESERVING NETWORK INFERENCE

The architectures in the neural network can be broken into linear and non-linear layers. For example, linear layers include the fully connected layer, convolutional layer, and average pooling layer. Besides, the non-linear layers include ReLU, Sigmoid, Tanh, and max-pooling. In the following, We give secure protocols for all layers to realize the task of neural network inference.

A. Secure Fully Connected Layer and Convolutional Layer

The fully connected layers and convolutional layers are the most common unit of neural networks and essentially perform matrix multiplication operation [24]. The objective function of each neuron in the network can be expressed as follows:

$$z = \sum_{i=0}^{n} x_i \times w_i + bias \tag{3}$$

where $\mathbf{x} = [x_1, ..., x_n]$ is the n-dimensional vector or the activation vector of previous hidden layer, $\mathbf{w} = [w_1, ..., w_n]$ is the n-dimensional weight of each neuron and $bias$ is the bias of each neuron.

To preserve private input over matrix multiplication operation, we present a secure matrix multiplication (**SecMatMul**) protocol to achieve the Eq. 3 in the secret-shared domain. **SecMatMul** is similar to **SecMul**, and both protocols are inspired by the idea of *Beaver's triplet* technology. Specifically, Beaver's multiplication matrix triplet ($\mathbf{c} = \mathbf{a}\cdot\mathbf{b}, \mathbf{a} = \langle \mathbf{a}\rangle_0 + \langle \mathbf{a}\rangle_1, \mathbf{b} = \langle \mathbf{b}\rangle_0 + \langle \mathbf{b}\rangle_1, \mathbf{c} = \langle \mathbf{c}\rangle_0 + \langle \mathbf{c}\rangle_0$) are generated on the offline stage. Then, each server S_i takes the secret shares of private input matrix $\langle \mathbf{x}\rangle_i$ and private weight matrix $\langle \mathbf{w}\rangle_i$ as input and outputs the result of SecMatMul: $\langle \mathbf{f}\rangle_i = i \times \mathbf{e}\cdot\mathbf{u} + \mathbf{e}\cdot\langle \mathbf{b}\rangle_i + \langle \mathbf{a}\rangle_i \cdot \mathbf{u} + \langle \mathbf{c}\rangle_i$, where $\mathbf{e} = \langle \mathbf{x}\rangle_i) - \langle \mathbf{a}\rangle_i$ and $\mathbf{u} = \langle \mathbf{w}\rangle_i - \langle \mathbf{b}\rangle_i$.

Therefore, for the secure convolutional (**SecConv**) layer or the secure fully connected layer (**SecFC**) layer, S_i will compute:

$$\langle \mathbf{f}\rangle_i = \textbf{SecMatMul}(\langle \mathbf{x}\rangle_i, \langle \mathbf{w}\rangle_i) + i \times \textbf{bias}. \tag{4}$$

where **bias** is the bias matrix and $i \in \{0,1\}$.

B. Secure Activation Layers

The activation layers introduce nonlinear properties into the neural network, and they are very sensitive to understanding the dynamics of neural networks. The popular recommendation in activation layers of modern neural networks is to use ReLU, Sigmoid, and Tanh functions, which are applied to each neuron of the network. Therefore, based on basic secure protocols, we first present a series of efficient, secure activation function protocols with low interactions in the secret-sharing domain.

1) Secure ReLU

The secure ReLU proceeds with the objective function ReLU$(x) = \max(x, 0)$ on each neuron over secret sharing domain. In other words, two servers work together to compute ReLU(x) = $s \times x$ without leaking any privacy data, where if $x \geq 0$, $s = 1$, otherwise, $s = 0$. Prior works often utilized the heavyweight garbled circuit techniques [25], the linear activation functions (e.g., square function [26]) or the secure MSB protocol [23] based on the adder accumulator. However, they require intensive interaction that is unsuitable for real-world secure inference services. To reduce interactions, we propose an efficient secure ReLU function (**SecReLU**) protocol which only requires four interactions between S_0 and S_1. The pseudo-code of the protocol is shown in detail in Algorithm 3.

Algorithm 3 Secure ReLU Function Protocol (SecReLU)

```
Input: $S_i$ has the input $\langle x\rangle_i \in \mathbb{F}$, where $i \in \{0,1\}$.
Output: $S_i$ outputs $\langle f\rangle_i$.
1: $S_i$ gets $\langle\beta\rangle_i$ = SecFloatMSB($\langle x\rangle_i$).
2: $S_i$ computes $\langle\alpha\rangle_i = \langle\beta\rangle_i \oplus (i \wedge 1)$.
3: $S_i$ computes $\langle s\rangle_i = \langle a\rangle_i + \langle b\rangle_i - 2 \times$ SecMul($\langle a\rangle_i, \langle b\rangle_i$),
   where $\langle a\rangle_0 = \langle\alpha\rangle_0$, $\langle a\rangle_1 = 0$, $\langle b\rangle_0 = 0$, $\langle b\rangle_1 = \langle\alpha\rangle_1$.
4: $S_i$ outputs $\langle f\rangle_1$ = SecMul($\langle s\rangle_i, \langle x\rangle_i$).
```

2) Secure Sigmoid and Secure Tanh

In the secure sigmoid function, the objective function $\delta(x) = 1/(1 + e^{-x})$ is computed on each neuron in secret sharing domain. Previous works that solved this function are either relying on the approximate *piecewise multi-degree polynomials* or utilizing the approximate *piecewise linear polynomials* [9], [27]. However, the former requires intensive communication costs since it introduces multiple comparison operations in the encryption domain. And the latter would introduce a serious activation accuracy error in the range $[-5, 5]$. Unlike prior works, we present a new secure Sigmoid function (**SecSigmiod**) protocol with high precision, which brings almost no extra computation overhead to the underlining spectrum auction and incurs only limited communication overhead. We transform the sigmoid function to an approximate Eq. 7 with Fourier Expansion method. As shown in the Fig. 3, compared to other approximation functions (Eq.5 specified in SecureML and Eq.6 specified in), ours approximation function (in the Fig. 3(d)) fits the best and its MSE (Mean Squared Error) is 0.0003.

$$\delta_1(x) = \begin{cases} 0 & x \leq -5 \\ 0.1x + 0.5 & -5 < x < 5 \\ 1 & x \geq 5 \end{cases} \tag{5}$$

$$\delta_2(x) = \begin{cases} 0 & x \leq -4 \\ 0.006x^5 - 0.147x^3 + 0.241x + 0.5 & -4 < x < 4 \\ 1 & x \geq 4 \end{cases} \tag{6}$$

$$\begin{aligned} \delta'(x) &= \begin{cases} 0 & \alpha = (x \leq -5) \\ g(x) & \gamma = (-5 < x < 5) \\ 1 & \beta = (x \geq 5) \end{cases} \\ &= (g(\gamma \times x) \times (1 - \alpha) + \alpha) \times (1 - \beta) \end{aligned} \tag{7}$$

where $g(x) = 0.1x + 0.5 - (0.1828sin(0.652x + 3.142) + 0.01953sin(1.428x - 3.142))$. In other words, the two servers work together to proceed with the function $(g(\gamma \times x) \times (1 - \alpha) + \alpha) \times (1 - \beta))$, where if $x \leq -5, \alpha = 1$, otherwise, $\alpha = 0$, if $-5 < x < 5, \gamma = 1$, otherwise, $\gamma = 0$, and if $x \geq 5, \beta = 1$, otherwise, $\beta = 0$.

Algorithm 4 Secure Sin Function Protocol (SecSin)

```
Input: $S_i$ inputs $\langle x\rangle_i$.
Output: $S_i$ outputs $\langle f\rangle_i$.
1: Beaver's multiplication triples ($a = \langle a\rangle_0 + \langle a\rangle_1, b = \langle b\rangle_0 + \langle b\rangle_1, c = a \times b = \langle c\rangle_0 + \langle c\rangle_0$) are generated in the offline stage.
2: $S_i$ computes $sin(\langle x_i\rangle), cos(\langle x_i\rangle)$.
3: $S_i (i \in \{0,1\})$ computes $\langle u\rangle_i = sin(\langle x\rangle_i) - \langle a\rangle_i$, $\langle e\rangle_i = cos(\langle x\rangle_i) - \langle b\rangle_i$ and sends $\langle u\rangle_i$ and $\langle e\rangle_i$ into $S_{1-i}$.
4: $S_i$ computes $u = \langle u\rangle_0 + \langle u\rangle_1$ and $e = \langle e\rangle_0 + \langle e\rangle_1$.
5: $S_i$ outputs $\langle f\rangle_i \times e \times u + e \times \langle a\rangle_i + u \times \langle b\rangle_i + \langle c\rangle_i - sin(\langle x\rangle_i)cos(\langle x\rangle_i)$.
```

Before realizing **SecSigmiod** protocol based on Eq. 7, we first present a new secure sin function (**SecSin**) protocol, which only requires one interaction between S_i and S_0. The pseudo-code of this protocol is shown in detail in the Algorithm 4. Besides, due to Eq. (8) holds true, then it directly validates the correctness of Algorithm 4. This relationship is crucial as it underpins the theoretical foundation of our proposed method. By establishing the validity of Equation (8), we consequently affirm the accuracy and reliability of Algorithm 4 within our framework.

$$\begin{aligned} sin(x) &= sin(\langle x\rangle_0 + \langle x\rangle_1) \\ &= ef + e(\langle a\rangle_0 + \langle a\rangle_1) + u(\langle b\rangle_0 + \langle b\rangle_1) + \langle c\rangle_1 + \langle c\rangle_0 \\ &\quad - sin(\langle x\rangle_0)cos(\langle x\rangle_0) - sin(\langle x\rangle_1)cos(\langle x\rangle_1) \end{aligned} \tag{8}$$

where $e = sin(\langle x\rangle_0) + sin(\langle x\rangle_1) - a$ and $u = cos(\langle x\rangle_0) + cos(\langle x\rangle_1) - b$.

Based on the proposed **SecSin** protocol, it is easy for us to construct the **SecSigmoid** protocol. Specifically, each server $S_i (i \in \{0,1\})$ first obtains the secret sharing MSB of $x+5$ (i.e, $x-(-5) > 0$) and $x-5$ (i.e, $x-5 > 0$), respectively. Then, S_i computes the secret shares $\langle z_i\rangle$ of $g(x)$. Finally, S_i outputs the secret sharing result $\langle f_i\rangle$ = SecMul(SecMul($\langle x\rangle_i, (-\langle\alpha\rangle_i + i)) + \langle\alpha\rangle_i), (-\langle\beta\rangle_i + i)$).

Finally, since $Tanh(x) = 2Sigmoid(2x) - 1$, the secure Tanh function protocol is expanded by utilizing the **SecSigmod** protocol. In brief, each server $S_i (i \in \{0,1\})$ only computes the secret sharing output $\langle f\rangle_i$ = 2SecSigmoid($2\langle x\rangle_i$)$-i$.

C. Secure Pooling Layers

The pooling layer is used to perform a feature downsampling operation within a certain sliding window [28]. The

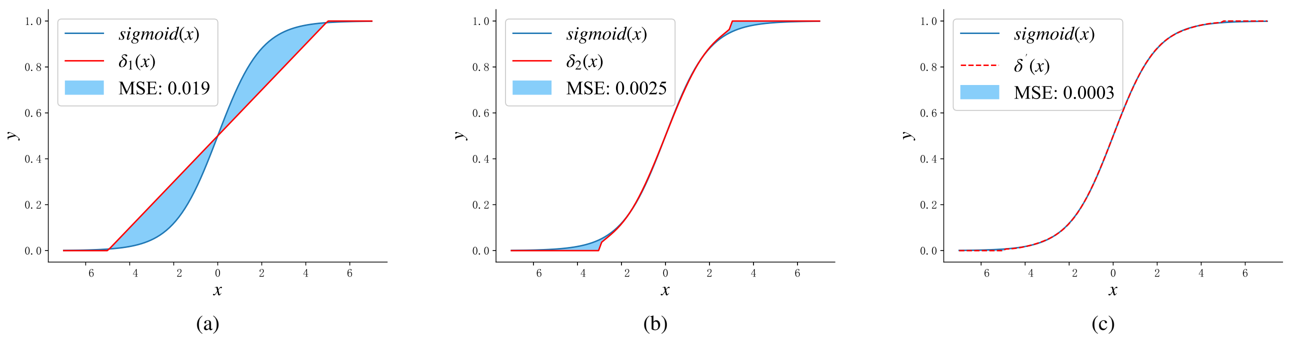

Fig. 3. Comparison of Different Approximation Functions with Sigmoid Function

Algorithm 5 Secure Sigmoid Function Protocol (SecSigmoid)

```
Input: $S_i$ has the input $\langle x\rangle_i \in \mathbb{F}$, where $i \in \{0,1\}$.
Output: $S_i$ outputs $\langle f\rangle_i$.
1: $S_i$ sets $k_1 = 0.1828, k_2 = 0.652, k_3 = 0.01953, k_4 = 1.428$ and $m = 3.142$.
2: $S_i$ gets $\langle \alpha\rangle_i$ = SecFloatMSB($\langle x\rangle_i - 5i$) and $\langle \beta\rangle_i$ =SecFloatMSB($\langle x\rangle_i + 5i$).
3: $S_i$ computes $\langle x\rangle_i = ((\langle \alpha\rangle_i - 1) \times ((\langle \beta\rangle_i - 1) \times \langle x\rangle_i$
4: $S_i$ sets $\langle u\rangle_i = k_2\langle x\rangle_i + m \times i$ and $\langle d\rangle_i = k_4\langle x\rangle_i - m \times i$.
5: $S_i$ computes:
   $\langle z\rangle_i = 0.1\langle x\rangle_i$+$0.5i$-($k_1$SecSin($\langle u\rangle_i$)+$k_3$SecSin($\langle d\rangle_i$)).
6: $S_i$ computes $\langle s\rangle_i = \langle a\rangle_i + \langle b\rangle_i - 2$SecMul($\langle a\rangle_i, \langle b\rangle_i$), where $\langle a\rangle_0 = \langle \beta\rangle_0$, $\langle a\rangle_1 = 0$, $\langle b\rangle_0 = 0$, $\langle b\rangle_1 = \langle \beta\rangle_1$.
7: $S_i$ computes $\langle \bar{\alpha}\rangle_i = \langle a\rangle_i + \langle b\rangle_i - 2$SecMul($\langle a\rangle_i, \langle b\rangle_i$), where $\langle a\rangle_0 = \langle \alpha\rangle_0$, $\langle a\rangle_1 = 0$, $\langle b\rangle_0 = 0$, $\langle b\rangle_1 = \langle \alpha\rangle_1$.
8: $S_i$ computes $\langle f_i\rangle$ = SecMul(SecMul($\langle z\rangle_i, (-\langle \alpha\rangle_i + i)) + \langle \alpha\rangle_i), (-\langle \beta\rangle_i + i)$).
```

popular pooling layers has the max pooling function and the average pooling function. If the average pooling function is used in the neural network, the two servers only perform secure addition protocol locally within a rectangular neighboring window, which is trivial in the secret-sharing protocols.

The max-pooling function is to compute the maximum within a certain sliding window, i.e., $f(x_1, ...x_n) = max(x_1, ..x_n)$, where a sliding window contains n values. Therefore, the pairwise maximum operations in the max pooling function are first transformed into the comparison operation based on the **SecMSB** protocol via Eq. 9 and a secure linear function on Eq. 10, which is a similarity to the **SecReLU** protocol:

$$\beta = SecMSB(x_1 - x_2)\begin{cases}0 & x_1 >= x_2\\ 1 & x_1 < x_2\end{cases} \tag{9}$$

$$max(x_1, x_2) = (1-\beta)x_1 + \beta x_2. \tag{10}$$

Next, we adopt *img2col* technology to convert the max-pooling tensor into a matrix. Finally, we compute the maximum value within each sliding window with the parallelization technology and the divide-and-conquer algorithm.

VII. SECURITY ANALYSIS

The security of our proposed protocols can be proved in the ideal/real-world simulation paradigm over the universal composability system [29]–[31]. In the honest-but-curious system, an adversary $\mathcal{A}$ is allowed to corrupt at most one of the model owners, the model user, or the two cloud servers. If our protocols are secure, there exists a probabilistic polynomial-time simulator forging $\mathcal{A}$'s view such that $\mathcal{A}$ cannot distinguish between a real or ideal world. Specifically, the following definitions and lemma are adopted to prove our protocols.

Definition 1. *A protocol can securely compute a functionality f in the honest-but-curious system if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ that can generate a view in the ideal world for $\mathcal{A}$ in the real world and the generated view is computationally indistinguishable from its real view [32].*

Definition 2. *A protocol is simulatable if all sub-protocols of the protocol are simulatable [32].*

Lemma 1. *If a random value r is uniformly distributed and is independent of any variable x in a finite field $\mathbb{F}$, the $r+x$ is uniformly and randomly distributed and is independent with x [23].*

In L-SecNet system, the threats mainly are from the engagement of two honest-but-curious cloud servers. For the user or the corrupted model owner, their inputs in L-SecNet are masked locally with the random values to upload to two cloud servers. It is pretty clear that the adversary $\mathcal{A}$ cannot distinguish whether their transcripts come from the simulator $\mathcal{S}$ or the real world. Thus, we mainly analyze the security of the communication between two cloud servers.

Theorem 1. *SecMul protocol is secure in the semi-honest system.*

Proof. For each server S_i in **SecMul** protocol, the view of the protocol's execution is $\mathcal{V}_i = (\langle x\rangle_i, \langle y\rangle_i, \langle a\rangle_i, \langle b\rangle_i, \langle c\rangle_i, \langle e\rangle_i, \langle d\rangle_i)$, where $\langle e\rangle_i = \langle x\rangle_i - \langle a\rangle_i, \langle u\rangle_i = \langle y\rangle_i - \langle b\rangle_i$, and the view of the protocol output is $\mathcal{O}_i = (\langle f\rangle_i = i \times e \times u + e \times \langle a\rangle_i + u \times \langle b\rangle_i + \langle c\rangle_i)$. According to Lemma 1, it is known that all values of $\mathcal{V}_i$ and $\mathcal{O}_i$ are uniformly random. Therefore, there is a simulator $\mathcal{S}$ that will interact with a corrupted server S_i and both $\mathcal{V}_i$ and $\mathcal{O}_i$ are simulated by $\mathcal{S}$. The views of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable due to the randomness of these views. Thus, the **SecMul** protocol is secure according to Definition 1.

Theorem 2. *SecIntMSB and SecFloatMSB protocols are secure in the semi-honest system.*

Proof. For the server S_i in **SeIntMSB** protocol, the view of the protocol's execution is $\mathcal{V}_i = (\langle x\rangle_i, a, b, \langle c\rangle_i, e, d)$, where $c = \langle c\rangle_i - \langle \eta\rangle_i$, $e = (\langle x\rangle_i - \langle c\rangle_i)/a$, $d = (\langle x\rangle_{1-i} - \langle c\rangle_{1-i})/(e+b)$, and the view of the protocol output is $\mathcal{O}_i = (\langle f\rangle_i = \langle u\rangle_i < 0)$. According to Lemma 1, it is known that all values of $\mathcal{V}_i$ and $\mathcal{O}_i$ are uniformly random. Therefore, there is a simulator $\mathcal{S}$ that will interact with a corrupted server S_i and both $\mathcal{V}_i$ and $\mathcal{O}_i$ are simulated by $\mathcal{S}$. The views of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable due to the randomness of these views. Thus, **SecIntMSB** protocol is secure according to Definition 1. Simultaneously, **SecFloatMSB** is constructed by **SecIntMSB** protocol and **SecMul** protocol. Therefore, **SecFloatMSB** is also secure according to Definitions 2 and 1.

Theorem 3. *SecSin protocol is secure in the semi-honest system.*

Proof. We first construct a simulator $\mathcal{S}$ to simulate the $\mathcal{A}'s$ view. For $\mathcal{A}$ receives $sin(\langle x\rangle_i)$ and $sin(\langle y\rangle_i)$ from the real world, $\mathcal{S}$ randomly picks two values $\langle x^{'}\rangle_i, \langle y^{'}\rangle_i$ from $\mathbb{F}$ and generates $sin(\langle x^{'}\rangle_i)$ and $cos(\langle y^{'}\rangle_i)$. According to Definition 1, $\mathcal{A}$ cannot distinguish the view generated by $\mathcal{S}$ from the real view. Besides, the view of **SecSin** protocol's execution on S_i is $\mathcal{V}_i$=$(sin(\langle x\rangle_i)$, $cos(\langle x\rangle_i), \langle a\rangle_i, \langle b\rangle_i, \langle c\rangle_i, \langle e\rangle_i, \langle d\rangle_i)$ where $\langle e\rangle_i$=$sin(\langle x\rangle_i) - \langle a\rangle_i$, $\langle u\rangle_i$=$cos(\langle x\rangle_i) - \langle b\rangle_i$, and the view of the protocol output is $\mathcal{O}_i = (\langle f\rangle_i = i \times e \times u + e \times \langle a\rangle_i + u \times \langle b\rangle_i + \langle c\rangle_i - sin(\langle x\rangle_i)cos(\langle x\rangle_i))$. Therefore, there is the simulator $\mathcal{S}$ that will interact with a corrupted server S_i and both $\mathcal{V}_i$ and $\mathcal{O}_i$ are simulated by $\mathcal{S}$. According to Definition 2, The views of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable due to the randomness of these views. Thus, **SecSin** protocol is secure according to Definition 1.

Theorem 4. *Our protocols of the network model inference are secure in a semi-honest system.*

Proof. In L-SecNet, a network model inference may involve the secure linear operation, the secure convolutional operation, the secure activation operation, and the secure pooling operation. These secure operations can all be built by our secure basic protocols. Thus, our protocols of the network model inference are secure according to Definitions 1 and 2.

VIII. PERFORMANCE EVALUATION

A. Experiment Setup

In the experiment, we implement a prototyping system of L-SecNet in PyTorch. The evaluations of our secure protocols are executed on two docker servers with Ubuntu 18.04 and equipped with Intel(R) Core(TM) i7-8700 CPU@ 3.20GHz, 32G RAM, where the communication protocol is the TCP protocol in the local area network (LAN) between the two servers. Each server is limited to 2Gbps of upload and download bandwidth.

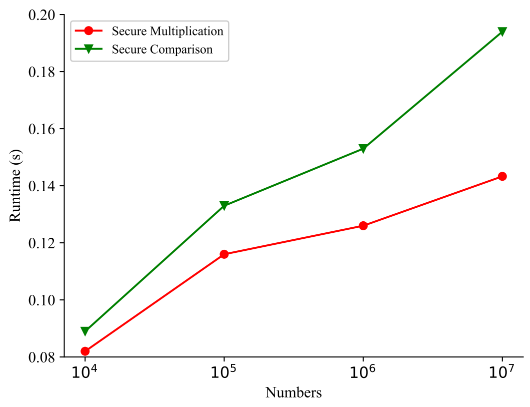

Fig. 4. Scalability of secure computation operations

We conduct experiments over four public real-world datasets: MNIST is a popular dataset including 28 × 28 greyscale images, where the test dataset has 10,000 examples and the training dataset has 60,000 examples. CIFAR-10 is a complex dataset including 32×32 RGB images, where the test dataset has 10,000 images and the training dataset has 50,000 images. TinyImageNet dataset contains 120, 000 RGB images of size 64×64 pixels in 200 classes, where each class contains 500 training images, 50 verification images and 50 test images. SCUT-HEAD is a large-scale head detection dataset, including 4405 images labeld with 111251 heads. Besides, we evaluate L-SecNet on several neural network architectures including 3FC-ReLU-FC model architecture (Model 1) specified in Sonic [22] , AlexNet(Model 2), Logistic Regression specified in SecureML [9], FitNet(Model-3) specified in Sonic [22], ResNet-32 [35] (Model-4) specified in DELPHI [10], ResNet-50 (Model-5) specified in CrypTFlow [10], Heads model (Model-6) with CNNs and RNNs specified in SIRNN [26].

B. Metrics

In secure neural network inference, the running times, rounds of communaction and network bandwidth are largely limited to the response time of neural network inference [36]. Thus, we measure the communication including network bandwidth and rounds between two cloud servers, while we also measure the running time including the time of transferring messages and computation between two servers. The experimental result reported is an average of over 10 trials.

C. Microbenchmarks

Secure basic protocols. Since we implement the secure neural network inference system based on secure basic protocols, their performance is evaluated in our secure system. First, we still maintain the most remarkable performance of **SecAdd** and **SecMult** operations due to using the most existing advanced technology. Besides, we mainly measure the performance of the **SecCom** operations. As shown in Table II, we see that the overheads of communication and the runtime of inference for the secure comparison (**SecCom**) protocol are notably less than the prior works [22], [33], [34]. We

TABLE II
COMPARISON OF SECURE COMPARISON PROTOCOL OVER $l = 32$ BIT-WITH NUMBER

Operations	Prior Approach	Communication Rounds	Bandwidth	Runtime ($n = 1000$ numbers)
SecCom	MSB [33]	$\mathcal{O}(l)$	$12l - 16$	182 ms
	MSB (parallel) [22]	$\mathcal{O}(logl)$	$12l - 6$	103 ms
	GC [34]	$\mathcal{O}(1)$	$30kl$	1562 ms
	Ours	$\mathcal{O}(\mathbf{1})$	**6*l*+4**	**51 ms**

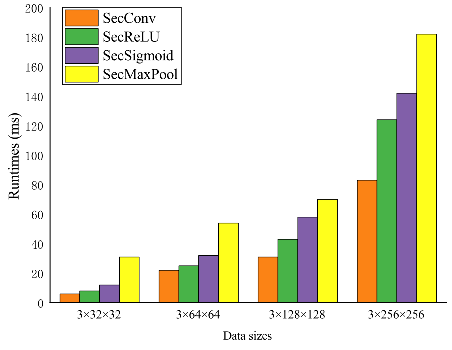

Fig. 5. Scalability of secure layers in network inference

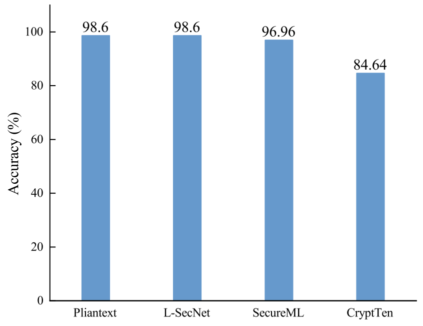

Fig. 6. Accuracy of Logistic Regression on MNIST

analyze that our secure comparison protocol over the 32-bit number achieves $2\times$, $2\times$, $75.6\times$ bandwidth savings and $3.5\times$, $2\times$, $30.6\times$ runtime savings compared with the prior works [22], [33], [34]. To ensure our secure protocols have strong scalability, we put n numbers into an array of Pytorch to execute vectorization operations in parallel instead of doing secure comparisons one by one. As depicted in Fig.4, We can see that the runtime grows linearly as the size exponentially grows, and the gap of the runtimes between **SecMul** and **SecCom** is also not large. Therefore, our protocols ensure the high efficiency of communication and computation in the secure neural network inference system.

Secure Layers of Network Inference. Since our secure layers of network inference are implemented based on a series of our lightweight protocols to outperform those of previous works, obviously, we emphatically focus on the scalability of all secure network operations in our secure system on this part. As shown in Fig. 5, the runtime of these operations also increases as the size of inputs grows. However, it is totally acceptable for the impact of the secure computation process for their relatively small runtime.

TABLE III
SUMMARY OF INFERENCE ACCURACY

Dataset	Model	L-SecNet	Plaintext
MNIST	Model-1	98.2%	98.2%
	Model-2	99.1%	99.1%
CAFAIR-10	Model-3	83.0%	83.8%
	Model-4	82.7%	83.1%
TinyImageNet	Model-5	76.8%	79.3%
SCUT-HEAD	Model-6	73.2%	77.4%

D. Summary of Accuracy

Table III shows the prediction accuracy of different models over different datasets in the L-SecNet system and compares the experimental results with the corresponding plaintext model. As shown, the inference of Model-1 and Model-2 over MNIST datasets for L-SecNet is the same as that of the corresponding plaintext model, i.e., 98.2% and 99.1%, respectively. For Model-3 and Model-4 over CAFIR-10 datasets, L-SecNet achieves 83.0% and 86.7% inference accuracy, which is a sight lower than the accuracy 83.7% and 87.1% of plaintext model. For this phenomenon, we analyze that the loss of model inference accuracy is mainly caused by participating in computation with floating-point numbers. For Model-5 over the TinyImageNet dataset and Model-6 over the SCUT-HEAD dataset, the accuracy of L-SecNet is 4%-5% lower than that of the plaintext model. The reason for the loss of accuracy is that the error between the proposed approximate function and the original function will increase with the depth of the neural network, affecting the accuracy of L-SecNet. Simultaneously, Fig. 6 describes the accuracy of logistic regression on MNIST and compares the results with the prior works. The accuracy of L-SecNet still is the same as the plaintext model outperforms previous work. This is mainly due to the fact that the error between the proposed approximate function and the raw $\mathsf{sigmoid}$ function is only 0.0003 and does not affect the inference accuracy of logical regression.

E. Comparison with Prior works

To demonstrate the lightweight, L-SecNet compares with notable prior works about the secure neural network inference system in terms of runtime of computation and bandwidth of communication. All measurements are from their corresponding papers. Tabel IV summarizes of the implementation of

TABLE IV
SUMMARY OF IMPLEMENTATION OVER APPROACHES

Approach	Implementation
MiniONN [16]	C++
EzPC [37]	C++
Chameleon [6]	C++
DELPHI [10]	C++
Sonic [22]	Java
L-SecNet	**Python**

TABLE V
COMPARISON OF L-SECNET WITH PRIOR WORKS ON MNIST

Model	Prior Approach	Bandwidth	Runtime
Model-1	MiniONN	15.8 MB	1.12s
	EzPC	70 MB	4.29 s
	Chameleon	10.5 MB	2.24s
	Sonic	1.8 MB	0.62s
	L-SecNet	**1.02 MB**	**0.08s**
Model-2	MiniONN	657.5 MB	9.32s
	EzPC	501.0 MB	24.5s
	Sonic	10.8 MB	13.6s
	L-SecNet	**7.6 MB**	**0.54s**

the prior works. As shown, the prior works implement their system with C++, except that Sonic and ours apply Java and Python to establish the corresponding system, respectively. It is well-known that the runtime of C++ and Java programs is faster than Python-based programs. However, L-SecNet is still ahead of these prior works.

In addition, Tabel V and Tabel VI summarize the runtime of computation and the bandwidth overheads of communication on different network models over MNIST and CIFAR-10 datasets, respectively. As Tabel V shows, L-SecNet can save up to $68\times$ bandwidth, $53\times$ runtime and no less than $1.79\ \times$ bandwidth, $7.5\times$ runtime over Model-1 compared with prior works on MNIST. L-SecNet can save up to $86\times$ bandwidth, $45\times$ runtime, and no less than $1.42\times$ bandwidth, $17.2\times$ runtime over model-2 compared with prior works on MNIST. From Table VI, L-SecNet saves about $9\times$, $19\times$, $8\times$, $3\times$ bandwidth and about and $19\times$, $9.3\times$, $2.4\times$, $2.4\times$, $3.3\times$ runtime over Model-3 than MiniONN, EzPC, Chameleon and Sonic on CIFAR-10, respectively. Besides, to demonstrate our system for the application of deep models, L-SecNet measures the overhead of computation and communication over ResNet on CAFIR-10 and also compares the corresponding results with DELPHI. The experimental results show that L-SecNet reduces $4\times$ bandwidth than DELPHI.

TABLE VI
COMPARISON OF L-SECNET WITH PRIOR WORKS ON CAFIR-10

Model	Prior Approach	Bandwidth	Runtime
Model-3	MiniONN	9272 MB	544.0s
	EzPC	40683 MB	265.6s
	Chameleon	2650 MB	52.67s
	Sonic	711 MB	94.7s
	L-SecNet	**185.27 MB**	**28.2s**
Model-4	DELPHI	6.5GB	200.0s
	L-SecNet	**1.28GB**	**68.6s**

Finally, we also report the performance of CrpytTFlow2 and SIRNN for evaluating Heads (Model-5) on SCUT-HEAD and ResNet50 (model-6) on TinyImageNet separately, along with the performance of L-SecNet for the same model. Table VII shows that the time to evaluate Model-5 having Sigmoid and Tanh functions is consuming time, but our protocols save $13.5\times$ bandwidth and $1.08\times$consuming runtime than SIRNN. Also, note that we reduce the communication by$8.1\times$and runtime by $2.3\times$ on Model-6, Compared with CryTFlow2 (SCI_{HE}).

TABLE VII
COMPARISON OF L-SECNET WITH PRIOR WORKS ON LARGE-SCALE DATASET

Model	Prior Approach	Bandwidth	Runtime
Model-5	SIRNN	85.5GB	409.7s
	L-SecNet	**6.32GB**	**378.4s**
Model-6	CrypTFlow2 (SCI_{HE})	32.43GB	295.7s
	L-SecNet	**3.98GB**	**128.0s**

Based on the above observation, L-SecNet obviously outperforms previous works in both the bandwidth of communication and the runtime of computation via the results of these tables. This is for the reason that L-SecNet does not use any heavy cryptographic primitives and can avoid the overheads of computation and communication over encrypted data. Moreover, these excellent results also benefit from the new lightweight implementation of secure non-linear layers. More importantly, we utilize the vectorization technique of Pytorch to perform all secure protocols of L-SecNet system in parallel to speed up secure network inference.

IX. CONCLUSION

In the paper, we proposed a lightweight, secure neural network inference system, namely, L-SecNet. Specifically, we devised a series of secure and efficient basic protocols to support all of the lightweight, secure operations of network inference service, each of which can ensure the privacy protection of sensitive data for the user and the model owner. Besides, we also presented a lightweight secure comparison protocol and new secure accurate activation functions with low communication overheads, reducing the running latency of the inference services. Finally, the security and efficiency of L-SecNet system were demonstrated by theoretical analysis and empirical experiments. The system saves up to about $80\times$ bandwidth and about $53\times$ runtime than the existing works in the online inference phase.

REFERENCES

[1] X. Wu, X. Feng, X. Cao, X. Xu, D. Hu, M. B. López, and L. Liu, "Facial kinship verification: A comprehensive review and outlook," *International Journal of Computer Vision*, pp. 1–32, 2022.

[2] B. Xu, C. Lu, Y. Guo, and J. Wang, "Discriminative multi-modality speech recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 433–14 442.

[3] Q. Lou, W.-j. Lu, C. Hong, and L. Jiang, "Falcon: fast spectral inference on encrypted data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2364–2374, 2020.

[4] M. Samragh, S. Hussain, X. Zhang, K. Huang, and F. Koushanfar, "On the application of binary neural networks in oblivious inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4630–4639.

[5] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.

[6] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018, pp. 707–721.

[7] L. K. Ng and S. S. Chow, "{GForce}:{GPU-Friendly} oblivious and rapid neural network inference," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2147–2164.

[8] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "{SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2651–2668.

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

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

[11] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a relu budget," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 16 961–16 971.

[12] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "{SAFEN}et: A secure, accurate and fast neural network inference," in *International Conference on Learning Representations*, 2021.

[13] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "Deepreduce: Relu reduction for fast private inference," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4839–4849.

[14] Q. Li, Z. Huang, W.-j. Lu, C. Hong, H. Qu, H. He, and W. Zhang, "Homopai: A secure collaborative machine learning platform based on homomorphic encryption," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1713–1717.

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

[16] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 619–631.

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

[18] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2165–2182.

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

[20] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "Safenet: A secure, accurate and fast neural network inference," in *International Conference on Learning Representations*, 2020.

[21] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[22] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2022.

[23] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, "A lightweight privacy-preserving cnn feature extraction framework for mobile sensing," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1441–1455, 2019.

[24] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," *IEEE transactions on neural networks*, vol. 9, no. 4, pp. 685–696, 1998.

[25] M. Ciampi, V. Goyal, and R. Ostrovsky, "Threshold garbled circuits and ad hoc secure computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 64–93.

[26] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "Sirnn: A math library for secure rnn inference," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1003–1020.

[27] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1057–1073.

[28] P. Kim, "Convolutional neural network," in *MATLAB deep learning*. Springer, 2017, pp. 121–147.

[29] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[30] R. Canetti, A. Cohen, and Y. Lindell, "A simpler variant of universally composable security for standard multiparty computation," in *Annual Cryptology Conference*. Springer, 2015, pp. 3–22.

[31] A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: a system for secure multi-party computation," in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 257–266.

[32] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[33] Y. Zheng, H. Duan, and C. Wang, "Towards secure and efficient outsourcing of machine learning classification," in *European Symposium on Research in Computer Security*. Springer, 2019, pp. 22–40.

[34] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2016, p. 830–842.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] X. Liu, B. Wu, X. Yuan, and X. Yi, "Leia: A lightweight cryptographic neural network inference system at the edge," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 237–252, 2021.

[37] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: programmable and efficient secure two-party computation for machine learning," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 496–511.