



Article

Improved Secure Deep Neural Network Inference Offloading with Privacy-Preserving Scalar Product Evaluation for Edge Computing

Jiarui Li ¹ , Zhuosheng Zhang ¹, Shucheng Yu ^{1,*}  and Jiawei Yuan ²¹ Department of Electrical and Computer Engineering, Stevens Institute of Technology, 1 Castle Point Terrace, Hoboken, NJ 07030, USA² Department of Computer & Information Science, University of Massachusetts Dartmouth, 285 Old Westport Road, Dartmouth, MA 02747, USA

* Correspondence: syu19@stevens.edu

Abstract: Enabling deep learning inferences on resource-constrained devices is important for intelligent Internet of Things. Edge computing makes this feasible by outsourcing resource-consuming operations from IoT devices to edge devices. In such scenarios, sensitive data shall be protected while transmitted to the edge. To address this issue, one major challenge is to efficiently execute inference tasks without hampering the real-time operation of IoT applications. Existing techniques based on complex cryptographic primitives or differential privacy are limited to either efficiency or model accuracy. This paper addresses this challenge with a lightweight interactive protocol by utilizing low-latency IoT-to-edge communication links for computational efficiency. We achieve this with a new privacy-preserving scalar product evaluation technique that caters to the unique requirements of deep learning inference. As compared to the state-of-the-art, our solution offers improved trade-offs among privacy, efficiency, and utility. Experimental results on a Raspberry Pi 4 (Model B) show that our construction can achieve over 14× acceleration versus local execution for AlexNet inference over ImageNet. The proposed privacy-preserving scalar-product-evaluation technique can also be used as a general primitive in other applications.

Keywords: privacy; Internet of Things; convolutional neural networks; deep learning; computation outsourcing; edge computing; privacy-preserving scalar product



Citation: Li, J.; Zhang, Z.; Yu, S.; Yuan, J. Improved Secure Deep Neural Network Inference Offloading with Privacy-Preserving Scalar Product Evaluation for Edge Computing. *Appl. Sci.* **2022**, *12*, 9010. <https://doi.org/10.3390/app12189010>

Academic Editor: Pericle Perazzo

Received: 16 August 2022

Accepted: 5 September 2022

Published: 8 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the recent rapid development of data-driven Internet-of-Things (IoT) applications, the integration of deep learning on IoT devices is receiving unparalleled attention [1,2]. As a representative deep learning system, deep neural networks, such as convolutional neural networks (CNN), have been widely studied since their inception and have achieved record-breaking results on highly challenging tasks such as image recognition [3] and natural language processing [4]. Efficient execution of this kind of deep networks onsite at IoT devices, where data are acquired, is particularly important for numerous low-latency intelligent IoT applications [5,6], including robotics, health monitoring, smart homes, and smart transportation, to name a few. In these scenarios, the pre-trained deep models are pre-loaded on IoT devices which will continuously conduct inferences on acquired data and make intelligent decisions in a real-time manner. However, executing highly compute-intensive learning tasks on resource-constrained IoT devices for time-sensitive tasks is extremely challenging. For instance, widely used CNN architectures such as AlexNet [3] and ResNet [7] require billions of operations for a single inference task, which can cost hundreds of seconds on typical IoT devices [8], not to mention the energy consumption.

To mitigate the computational constraints of IoT devices, with the proliferation of cloud computing, outsourcing data and data analytic tasks such as deep learning to the public

cloud has become a popular solution. Most of the existing cloud-based methods utilize advanced cryptographic primitives such as Homomorphic Encryption (HE) [9] and Multi-party Computation (MPC) [10,11] to achieve privacy-preserving computation outsourcing. However, these techniques are still far from efficient even when outsourcing simplified deep networks, despite recent promising progress. For example, CryptoNets [12] needs over 570 s for the inference of a simplified CNN (5 layers) on MNIST. More importantly, existing “cloud-based” techniques usually do not consider bandwidth constraints or other latencies between the data source (e.g., IoT devices) and the cloud. For emerging low-latency applications, one trend is to shift from cloud computing to edge computing [13,14] so that computationally expensive operations can be offloaded to nearby edge devices instead of the remote cloud to minimize the communication delay. Unlike cloud systems that are massively parallel and extremely elastic, local edge servers are usually equipped with relatively limited computing resources, though advanced computing modules such as GPUs are still available.

To support secure and efficient offloading of deep networks, one promising approach is to employ customized lightweight encryption (e.g., one-time pad (OTP) or random noises) instead of using heavy cryptographic primitives. To this end, a recent study [15] proposes an edge-assisted interactive scheme that allows IoT devices to outsource only linear operations to edge devices. This is based on the observation that linear operations contribute to most of the computational complexity in popular deep networks. For example, in AlexNet [3], with five convolutional layers and three fully-connected layers, the non-linear operations only count for less than 2 million floating-point operations (FLOPs)—less than 0.1% of the total of 2 billion FLOPs—while over 99.9% are linear operations. Therefore, one practical approach is to let IoT devices only offload linear operations to edge devices while computing non-linear ones locally. To this end, Ref. [15] encrypts input data with one-time random noises and is extremely efficient because of the minimal computational overheads incurred by the OTP-like encryption. The linearity of the outsourced operations ensures that the added noises can then be efficiently eliminated from the scalar products returned by the edge. The communication latency introduced by the interactive scheme can be largely alleviated, since edge devices are usually located within one hop to IoT devices.

Despite the advantages, one outstanding constraint of this solution, however, is the ever-increasing storage overhead incurred by the offline pre-computation. Specifically, to eliminate the added noises in the returned scalar products, IoT devices shall pre-compute and store the scalar products of the random noises and the network weights. Both computational and storage complexities are linear to the neural network size and the number of inferences performed. While the offline computational cost might be somewhat mitigated by employing powerful computers, the storage overheads are on the IoT devices, which are resource constrained. In practical systems, these overheads can result in the reduced performance of IoT devices, e.g., flight time of drones, or IoT devices needing to be frequently “replenished” in one way or another, which introduces other constraints to the operation of IoT devices. Concerning the practical operation of IoT devices, a better trade-off between offline computation/storage and online computation shall be achieved.

Our Contribution

In this paper, we propose a new technique for privacy-preserving deep neural network offloading for IoT devices. Different from [15], our scheme unleashes IoT devices by eliminating the need for the ever-increasing data storage while enjoying the secure offloading of linear operations. We achieve this with our novel design of an interactive privacy-preserving scalar product (iPPSP) primitive. Specifically, we introduce a constant-size random key pool on each IoT device, from which encryption/decryption keys for iPPSP are continuously generated. The theoretical results show that, compared to local execution, our design is able to accelerate the execution of convolutional layers by an approximation of $\frac{\hat{n}k^2}{T}$, where T is a pre-set parameter which is usually a small number, and k and \hat{n} denote the kernel size and input size of each channel, respectively. Experimental

results on a Raspberry Pi 4 (mode B) show that our design has over $14\times$ speedup over local execution of AlexNet over ImageNet.

Our contributions can be summarized as follows:

- We propose a new deep neural network inference scheme for IoT devices that supports the secure and efficient outsourcing of 99.9% inference operations from IoT devices to edge devices.
- We introduce a new privacy-preserving scalar-product evaluation technique, namely iPPSP, that supports secure encryption and decryption in a similar fashion of a one-time pad (OTP). In addition to deep learning, iPPSP can be used as a generic lightweight cryptographic primitive for other applications, wherein the interactive evaluation of scalar products is needed.
- A thorough analysis shows that our protocol is secure against key-recovery attack and semantically secure under the chosen-plaintext attack (CPA) model. The theoretical results show that, compared to local execution, our design improves the performance of convolutional layers by approximately $\frac{\hat{n}k^2}{T}$. Extensive experimental evaluation shows that our protocol is able to speed up over $14\times$ for CNN inference as compared to local execution.

This paper is organized as follows. In Section 2, we formulate the problem and present the system and threat models. Section 3 presents the detailed design of our secure inference scheme and the construction of iPPSP. Section 4 provides thorough security analysis and performance evaluation. Section 5 reviews related works, which is followed by conclusion in Section 6.

2. Problem Formulation and Models

2.1. Privacy-Preserving Deep Network Inference

In this work, we focus on the privacy-preserving outsourcing of deep neural network inferences from an IoT device to a nearby edge server. The deep neural network is pre-loaded on both the IoT device and the edge server. The IoT device holds the private data, which it encrypts and sends to the edge server. The edge performs the most resource-consuming linear operations and returns the results back to the IoT device in a timely manner. For convenience, we use a typical convolution neural network (CNN) structure such as AlexNet for the case study. The network consists of multiple *convolutional layers*, *non-linear layers*, and the final *fully connected layers*. The proposed scheme is applicable to networks that share similar structures.

A convolutional layer takes as its input a tensor $T \in \mathbb{R}^{\hat{n} \times \hat{n} \times D}$ with H kernels K_h , each of size $k \times k \times D$. The convolutional layer outputs a tensor T' of size $\hat{n}' \times \hat{n}' \times H$, where $\hat{n}' = \frac{\hat{n}-k+2p}{\hat{s}} + 1$, and \hat{s}, p indicate the stride and padding, respectively. The computation process of a convolutional layer can be viewed as multiplying each $k \times k$ matrix of a D -channel kernel with the input $\hat{n} \times \hat{n}$ matrix by sliding the former across the latter in a top-down and left-right manner with stride \hat{s} . The process is repeated for all D channels, and the output tensor is a weighted sum. From a mathematical perspective, each element of channel $h \in [1, H]$ of the output tensors can be represented as: $T'(i, j, h) = \sum_{t, t' \in [k], c \in [D]} T(i\hat{s} + t, j\hat{s} + t', c) K_h(t, t', c)$. The input into the fully connected layer is an n -dimensional vector $x \in \mathbb{R}^{\hat{n}}$, which goes through a linear transformation parameterized by a weight matrix $\bar{W} \in \mathbb{R}^{\hat{n} \times \bar{m}}$ and a bias vector $b \in \mathbb{R}^{\bar{m}}$. The output is $y = x\bar{W} + b$, a vector of length \bar{m} . For simplicity, we use scalar products to represent the convolution operations and the linear transformations throughout the paper.

Non-linear layers consist of *activation functions* and *pooling layers*. The activation functions aim at shrinking the output size by performing computation on each element of the input. One of the most common activation functions in deep neural networks is the ReLU function $\text{ReLU}(x) = \max(0, x)$. Pooling layers consists of the widely used max-pooling function or average-pooling function. Throughout the paper, we will focus on the

outsourcing of linear layers, since the non-linear function counts for only a small portion and can be efficiently computed on IoT devices.

2.2. System Model

The targeted system is depicted in Figure 1, which includes three entities: IoT devices, edge devices and the IoT device owner.

- **The IoT device owner**, who possesses the pre-trained CNN model, wants to perform the inference of the data collected by IoT devices for purposes such as event detection and object identification without revealing any sensitive information.
- **IoT devices** are considered clients that collect private data and conduct CNN inference. IoT devices are assumed to not be powerful enough to execute CNN inference locally in real time, though they may have some limited computational capability.
- **Edge devices** are deployed in a one-hop communication distance to the IoT devices. They are equipped with considerable computing resources and can perform inference tasks in real time. For inference outsourcing, IoT devices first transmit the CNN model to the edge device before the inference task starts.

For simplicity, we use C and S to denote an IoT device and an edge device, respectively. We consider that there exists a direct wireless communication link between C and S . The link quality is stable, and the bandwidth is comparable with contemporary home or commercial WiFi networks (e.g., hundreds of Mbps in each direction). We assume each IoT device is equipped with some storage capacity (e.g., tens of GB).

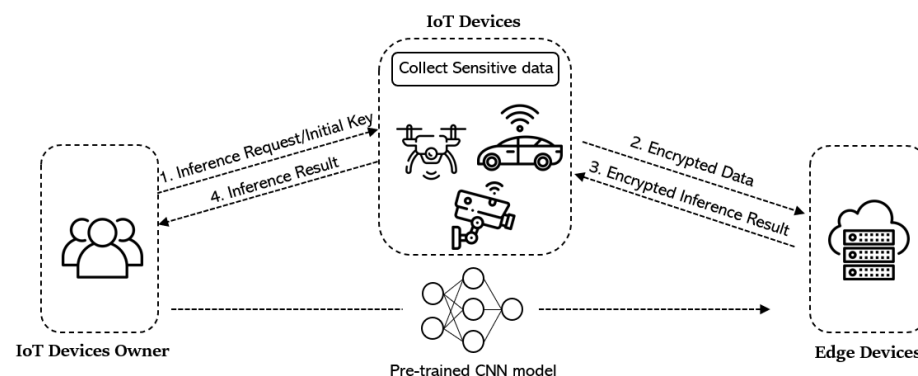


Figure 1. System model.

2.3. Threat Model

We assume the IoT device owner has physical and remote access to IoT devices and can securely load initial secrets to the latter. We consider that the IoT devices and the secrets they store are well secured. Device compromise attacks (e.g., via malware or physical tampering) are not considered, for which we resort to orthogonal research on software or system security. The edge device is considered “honest-but-curious”, i.e., the edge device will honestly follow the protocol but tend to obtain or infer the information about the inputs and outputs of scalar products. Therefore, the adversary in this paper can be either curious edge devices or external attackers who can eavesdrop on the communication channels. The main goal of our design is to protect the data’s privacy, i.e., to prevent the edge device and any external attackers from learning the private input data collected by the IoT devices, the final inference result, and all the intermediate results from the offloaded layers. Model privacy is not considered in this article, and the edge devices have access to the trained model (i.e., the weights). The encrypted data, including the encrypted input from the IoT device and the encrypted intermediate results such as the outputs of the offloaded layers of a CNN model, are available to the edge devices and attackers. The communication channels are assumed to be secure against message-modification attacks, e.g., through standard message integrity codes.

3. Construction

3.1. Privacy-Preserving CNN Inference

The high-level framework of our privacy-preserving CNN inference protocol is depicted in Figure 2. As previously discussed, we aim to allow IoT devices to securely offload the linear operations of the convolutional layers and the fully connected layers, which count for over 99.9% of the overall computation of the inference task. The non-linear layers are executed locally on IoT devices. Therefore, the protocol is interactive.

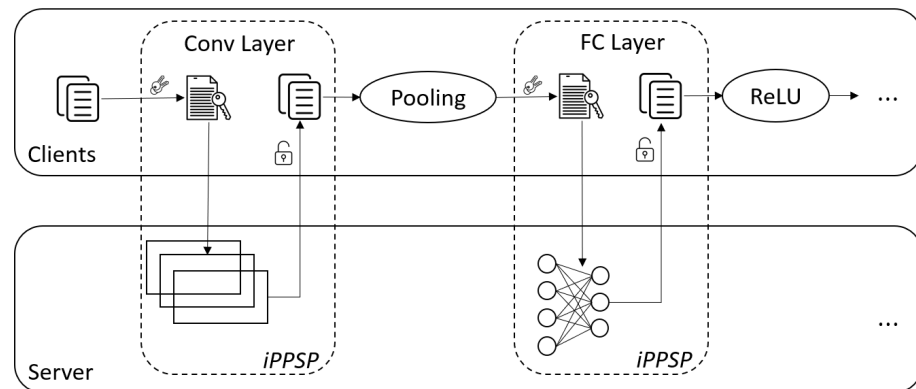


Figure 2. Framework of the proposed secure CNN inference protocol.

More specifically, an IoT device encrypts the inputs of each convolutional layer or fully connected layer and sends the encrypted inputs to the edge server. The latter computes the scalar products of the encrypted inputs and the corresponding weight parameters of the CNN network. The computed scalar products are, therefore, in encrypted forms, which are returned to the IoT device. The IoT device then locally decrypts the returned results and proceeds with non-linear layers, if there are any. This process is repeated until the output layer of the CNN structure is reached. The overall process is depicted in Algorithm 1.

Algorithm 1: Privacy-Preserving CNN Inference

Data: Input Data \mathcal{X} , Pre-trained model \mathcal{W}

Result: Inference result \mathcal{Y}

for each layer in \mathcal{W} **do**

 Layer.Input = \mathcal{X} ;

if Layer = Fully-Connected Layer or Convolution Layer **then**

 C interacts with S to execute Algorithm 2 and obtains output \mathcal{X}' ;

$\mathcal{X} = \mathcal{X}'$;

else

 C computes the non-linear functions locally and obtains output \mathcal{X}' ;

$\mathcal{X} = \mathcal{X}'$

end

end

In the privacy-preserving CNN inference protocol, one important component is the secure outsourcing of scalar products, which demands the corresponding data encryption/decryption algorithm be *linear*, *lossless*, and *efficient*. In particular, the “linear” property is in place to assure that the edge server is able to perform the learning operations as in plaintext models, e.g., it shall be compatible with tensor operations using existing libraries, such as TensorFlow, and conveniently support GPU executions for efficiency. The “lossless” property requires that the decryption does not introduce any distortion of the corresponding scalar products, so that the model performance is not impaired by the CNN inference protocol. Last but not least, “efficiency” is in terms of not only computation and communication, but also storage available considering contemporary IoT devices,

edge servers and wireless networks. To achieve these design goals, we introduce a new interactive privacy-preserving scalar product (iPPSP) primitive that can securely evaluate the scalar product in the four stages depicted in Figure 3: (1) *Initialization*, (2) *Pad Generation*, (3) *Encryption*, and (4) *Report*. In the *Initialization* phase, the IoT device owner prepares for the initial secrets necessary for the subsequent operation of the protocol. Specifically, our protocol generates a constant-size *random key pool* to be loaded to the IoT device; a so-called *pre-computation table*, which is also of constant size, is loaded on the IoT device as well. The two types of initial secret will be used as master keys to generate real-time data encryption and decryption keys, respectively. During the *Pad Generation* phase, the IoT device will randomly generate the encryption keys using the random keys in the key pool; as the encryption is similar to a one-time pad, we call these encryption keys as *encryption pads*. In the *Encryption* phase, the IoT device encrypts the message with the pad and sends the encrypted message to the edge server, which will compute the scalar product (over the encrypted data) for the IoT device. During the *Report* phase, the edge server returns the encrypted scalar product to the IoT device, which can immediately decrypt the expected scalar product with the decryption keys that can be efficiently derived with the pre-computation table. Please note that, while other phases are needed for each linear layer, Initialization is only executed once for an IoT device, unless the device is compromised, which is out of the scope of this work.

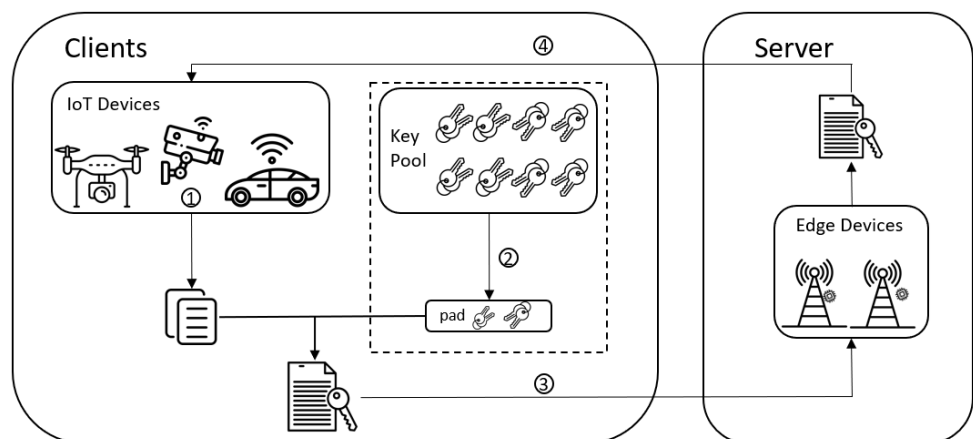


Figure 3. High-level workflow of iPPSP primitive.

3.2. Construction of iPPSP

Now, we elaborate on the construction of our iPPSP protocol. Important notation and parameters of our protocol are presented in Table 1.

Table 1. Summary of notation and parameters.

Notation	Meaning
λ	Security Parameter
C	Client. The IoT devices
S	Server. The edge computing devices
n	The length of input vector
W	$n \times m$ dimensional weight matrix
\mathcal{R}	Secret key pool
s	Secret key pool size
T	Pad Generation Parameter
γ_i	Random Indices

Interactive Privacy-Preserving Scalar Product (iPPSP): As described in Algorithm 2, in iPPSP the client C has an input vector $x = (x_1, x_2, \dots, x_n)$ and wants to compute the scalar product with weight matrix $W = (W_1, W_2, \dots, W_n)^T$, where each W_i is a vector of dimension m . After an edge device responds to the outsourcing request from the IoT device, they will interactively execute Algorithm 2.

Algorithm 2: Interactive Privacy-Preserving Scalar Product (iPPSP)

Data: n -dimensional input vector x , Weights matrix $W_{n \times m}$

Result: m dimensional vector $A = x \cdot W$

Initialization:

C generates a s -tuple $\mathcal{R} = (r^{(0)}, r^{(1)}, \dots, r^{(s-1)})$;

C computes tensor product $\mathcal{R} \otimes W_i$ for $i = 1, 2, \dots, n$ and stores the result;

for $i = 1 \rightarrow n$ **do**

Encryption:

C executes Algorithm 3 to generate an encryption pad r ;

C computes $\mathcal{C} = x + r$ and sends \mathcal{C} to server S ;

Report:

S computes and sends to C : $a_i = \mathcal{C} \cdot W_i$;

C decrypts the message with pre-computation table: $A_i = a_i - r \cdot W_i$;

$i = i + 1$;

end

Initialization: For an input vector x , each entry x_i , $i = 1, 2, \dots, n$ is a κ -bit number. The client first generates a sequence of random numbers $r^{(0)}, r^{(1)}, \dots, r^{(s-1)}$ from \mathbb{Z}_q , where $q = 2^l$, and these random numbers form the secret key pool \mathcal{R} . Notice that each element $r^{(j)}$ is l -bit ($l > \kappa$) and is generated by a *pseudo-random generator* (PRG). During the Initialization phase, the client computes n tensor products $\mathcal{R} \otimes W_i$ for $i = 1, 2, \dots, n$ and stores the result in a pre-computation table which will be used in decryption. The pre-computation table for a single tensor product $\mathcal{R} \otimes W_1$ is shown in Table 2.

Table 2. Pre-computation table locally maintained by IoT device.

$r^{(0)} \cdot W_{11}$	$r^{(1)} \cdot W_{11}$	\dots	$r^{(s-1)} \cdot W_{11}$
$r^{(0)} \cdot W_{12}$	$r^{(1)} \cdot W_{12}$	\dots	$r^{(s-1)} \cdot W_{12}$
\dots	\dots	\dots	\dots
$r^{(0)} \cdot W_{1m}$	$r^{(1)} \cdot W_{1m}$	\dots	$r^{(s-1)} \cdot W_{1m}$

Moreover, in this phase, a system parameter $T \in \mathbb{Z}_s$ is also generated.

Pad Generation: Each time the client needs to outsource a scalar product calculation for input vector x of dimension n , it executes Algorithm 3 to generate a pad of the same dimension to be used for encryption. Notice that, during the initialization phase, C generates a sequence of random numbers $r^{(0)}, r^{(1)}, \dots, r^{(s-1)}$ as master secrets stored in the key pool \mathcal{R} . To generate an encryption pad r_i , C randomly selects T (a system parameter determined in the Initialization phase) indices $\gamma_1, \dots, \gamma_T \in \mathbb{Z}_s$ and computes the following:

$$r_i = r^{(\gamma_1)} + r^{(\gamma_2)} + \dots + r^{(\gamma_T)}. \quad (1)$$

An n -dimensional pad $r = (r_1, r_2, \dots, r_n)$ is generated for an n -dimensional input x .

Encryption: After pad generation, the client encrypts the input data x as:

$$\mathcal{C} = x + r. \quad (2)$$

\mathcal{C} is then sent to the edge server S .

Report: Upon receiving \mathcal{C} from the client, the server will immediately compute the scalar product $\mathcal{C} \cdot W_i = (x + r) \cdot W_i$ and return the result to the client C .

Algorithm 3: Pad-Generation Algorithm**Data:** A s -tuple Key pool $(r^{(0)}, r^{(1)}, \dots, r^{(s-1)})$, Pre-set parameter T **Result:** encryption pad vector r $r = 0;$ **for** $i = 1 \rightarrow n$ **do** **for** $t = 1 \rightarrow T$ **do** C generates a random seed $\gamma_t \xleftarrow{\$} \mathbb{Z}_s;$ $r_i = r_i + r^{(\gamma_t)};$ $t = t + 1;$ **end** $i = i + 1;$ **end**

Given the returned scalar product $\mathcal{C} \cdot W_i = (x + r) \cdot W_i$, the client can easily decrypt the expected scalar product $x \cdot W_i$ by eliminating the scalar product of added noise with weights $r \cdot W_i$ as:

$$x \cdot W_i = (x + r) \cdot W_i - r \cdot W_i. \quad (3)$$

Please note that $r \cdot W_i$ can be conveniently computed by looking up the “pre-computation table” and then adding up the corresponding items $r^{(\gamma_k)} \cdot W_{ij}$.

After decryption, the IoT device proceeds to the next layer of the CNN network, i.e., either to process the non-linear function or to invoke another round of iPPSP for the next offloaded linear layer.

4. Analysis

4.1. Security Analysis

This section provides the overall security analysis of our privacy-preserving CNN inference protocol.

For any κ -bit message \mathcal{M} , Theorem 1 states that the probability for a probabilistic polynomial time (PPT) adversary \mathcal{A} to output a correct guess of \mathcal{M} is negligible.

Theorem 1. Given the ciphertext $\mathcal{C} = x_i + r_i$ of a κ -bit message $\mathcal{M} = x_i$ generated using iPPSP, the probability for a PPT adversary \mathcal{A} to output a correct guess for \mathcal{M} follows:

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] - Pr[\mathcal{M}^* = \mathcal{M}] \leq \epsilon,$$

where $\epsilon = \text{negl}(\cdot)$ is a negligible function regarding the security parameter λ , \mathcal{M}^* is the guess for \mathcal{M} made by \mathcal{A} , and $Pr[\mathcal{M}^* = \mathcal{M}]$ denotes the probability that the adversary \mathcal{A} makes a correct guess on \mathcal{M} .

Proof. Given a κ -bit input x_i , iPPSP encrypts it with an η -bit random number r_i generated by padding T random numbers of l -bit with a secure PRG, which indicates that the distribution of r_i should be indistinguishable from uniform. To make a correct guess on \mathcal{M} without ciphertext, \mathcal{A} has $Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{2^\kappa}$. Given the ciphertext $\mathcal{C} = x_i + r_i$, $\mathcal{C} \in [0, 2^\kappa + 2^\eta]$, we differentiate two different cases to discuss the impact of the value of \mathcal{C} . In the first case, $2^\kappa \leq \mathcal{C} \leq 2^\eta$, we have $Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] = \frac{1}{2^\kappa}$, since \mathcal{C} is indistinguishable from r_i within the range. If $\mathcal{C} < 2^\kappa$ or $\mathcal{C} > 2^\eta$, then the distribution of \mathcal{C} will be affected by \mathcal{M} and the total possible inputs are reduced to \mathcal{C} or $\mathcal{C} - 2^\eta$. Hence, we have $Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] = \frac{1}{\mathcal{C}}$ or $Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] = \frac{1}{\mathcal{C} - 2^\eta}$, respectively. In these cases, $Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] > \frac{1}{2^\kappa}$.

Consider the second case when $\mathcal{C} < 2^\kappa$ or $\mathcal{C} > 2^\eta$; this happens only when the value of encryption pad r_i is either too big or too small. More specifically, this happens only when $r_i > 2^{\eta-\kappa}$ or $r_i < 2^\kappa$. The probability of this event is

$$Pr[r_i > 2^\eta - 2^\kappa \cup r_i < 2^\kappa] = \frac{2 \cdot 2^\kappa}{2^\eta} = \frac{1}{2^{\eta-\kappa+1}}.$$

Thus, we require $\eta - \kappa - 1$ to be greater than some pre-set security parameter λ (e.g., $\eta - \kappa - 1 > \lambda = 128$). In this case, the probability $Pr[r_i > 2^\eta - 2^\kappa \cup r_i < 2^\kappa] = \frac{1}{2^{\eta-\kappa+1}}$ is negligible, we denote it as $\epsilon(\lambda)$. Combining the two cases:

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] \leq \frac{1}{2^\kappa} \cdot (1 - \epsilon(\lambda)) + \epsilon = \frac{1}{2^\kappa} + (1 - \frac{1}{2^\kappa}) \cdot \epsilon(\lambda).$$

Hence, we have

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] - Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{2^\kappa} + (1 - \frac{1}{2^\kappa}) \cdot \epsilon(\lambda) - \frac{1}{2^\kappa} = (1 - \frac{1}{2^\kappa}) \cdot \epsilon(\lambda) < \epsilon.$$

which concludes the proof. \square

Theorem 1 shows that, with appropriate system parameters, the ciphertext distribution cannot be distinguished with non-negligible probability by a PPT adversary. Since inputs of different CNN layers are independently encrypted, this property holds for the entire CNN inference task.

Next, we show that our protocol is semantically secure under the chosen-plaintext attack (CPA) model. Assume a PPT adversary \mathcal{A} has access to an encryption oracle \mathcal{O} . When \mathcal{A} queries the oracle, it chooses a message m on its own and sends it to the oracle. The oracle will execute the protocol and output an encryption of message $\mathcal{C} = m + r$, where r is sampled through our key-generation algorithm. Recall that, during pad generation, the client generates T random numbers to identify elements in key pool \mathcal{R} . Thus, the total possible combination of the pad is given by s^T . For security reasons, we assure the probability that the same pad is re-used is negligible regarding the security parameter λ . That is, $(\frac{1}{s})^T = \text{negl}(\lambda)$, which can be achieved by choosing appropriate parameters. Specifically, we can formulate the security in Theorem 2.

Theorem 2. For every PPT adversary \mathcal{A} , there exists a negligible function $\text{negl}(\cdot)$ such that the probability of the CPA-indistinguishability experiment Exp to output 1 is only negligibly more than $\frac{1}{2}$:

$$Pr[\text{Exp}_{\mathcal{A}, \text{PPSP}}^{\text{SS-CPA}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$$

Proof. We first describe the CPA-indistinguishability experiment $\text{Exp}_{\mathcal{A}, \text{PPSP}}^{\text{SS-CPA}}(\lambda)$. The adversary is given input 1^λ and access to the encryption oracle \mathcal{O} . \mathcal{A} samples a pair of message m_0, m_1 from the message space and sends them to the challenger. Then, a random bit $b \xleftarrow{\$} \{0, 1\}$ is selected by the challenger, who encrypts the message $\mathcal{C} = m_b + r$, where r is generated through Algorithm 3. The challenger sends the ciphertext \mathcal{C} to the adversary. Here, we assume that the output of Algorithm 3 follows distribution \mathcal{D} . In Theorem 1, we show that \mathcal{D} cannot be distinguished from a uniform distribution on \mathbb{Z}_q with non-negligible probability. The adversary can keep querying the oracle for polynomial times and then output a bit b' . The output of experiment $\text{Exp}_{\mathcal{A}, \text{PPSP}}^{\text{SS-CPA}}(\lambda)$ is defined as 1 if $b' = b$ and 0 otherwise.

Then, we define a sequence of hybrid H_i

$$\begin{aligned} H_0 &= \{r \leftarrow \mathcal{D} : m_0 + r\} \\ H_1 &= \{R \leftarrow \mathcal{U}_l : m_0 + R\} \\ H_2 &= \{R \leftarrow \mathcal{U}_l : m_1 + R\} \\ H_3 &= \{r \leftarrow \mathcal{D} : m_1 + r\} \end{aligned}$$

By construction, if \mathcal{A} can win the CPA game with a non-negligible probability p , i.e., the probability of $\text{Exp}_{\mathcal{A}, \text{PPSP}}^{\text{SS-CPA}}(\lambda)$ to output 1 is larger than the experiment to output 0 with non-negligible probability, then \mathcal{A} can distinguish H_0 and H_3 with same probability p . Therefore, from the hybrid lemma, which shows the transitivity of hybrids, we immediately know that \mathcal{A} can also distinguish between two consecutive hybrids with non-negligible probability $1/4p$. We will proceed to show that this gives rise to a contradiction.

Define a pair of nonuniform PPT machines M^0, M^1 by $M^i(X_l) = \{r \leftarrow X_l : m_i + r\}$. It is clear that M^i is an efficient operator. Observe that $H_0 = M^0(\mathcal{D})$ and $H_1 = M^0(\mathcal{U}_l)$. Guaranteed by our construction, with only negligible probability, the pad r generated by our algorithm can be distinguished from a truly random number sampled from a uniform distribution \mathcal{U}_l . Thus, from the efficient operation lemma, $H_0 \approx H_1$. Similarly, it is easy to show that $H_2 \approx H_3$. Then, to preserve the transitivity of hybrids, \mathcal{A} must be able to distinguish H_1 and H_2 . Nonetheless, by the perfect secrecy of one time pad, we know that H_1 and H_2 are indistinguishable, which immediately introduces the contradiction. \square

Next, we show that our construction is secure against a key-recovery attack. Assume that a PPT adversary \mathcal{A} has access to the encryption oracle \mathcal{O} and wants to perform a key-recovery attack, i.e., to recover the key pool given the polynomial number of accesses to the oracle \mathcal{O} . In Theorem 3, we show that the probability that the adversary succeeds in performing such an attack is negligible.

Theorem 3. For any PPT adversary \mathcal{A} with access to encryption oracle \mathcal{O} , the probability of performing a successful key-recovery attack is

$$\text{Pr}_{\text{KRA}} = \binom{\mathcal{Q}}{s} p^s (1-p)^{\mathcal{Q}-s}, \quad (4)$$

which is negligible in terms of security parameter λ , $p = \frac{T!(s-1)!}{(s+T-1)!}$, where s denotes the number of secret keys, and T represents the number of keys used to generate the pad r for encryption. $\mathcal{Q} = \text{poly}(\lambda)$ where λ is the security parameter.

Proof. By accessing the encryption \mathcal{O} for each query, \mathcal{A} can choose a message m from the given message space that it chooses, and the oracle will return its encryption $\mathcal{C} = m + r$, where r is generated through the pad-generation algorithm. Thus, the adversary can easily recover the pad r by $r = \mathcal{C} - m$. Recall that each pad is a combination of the secret key:

$$r_i = r^{(\gamma_1)} + r^{(\gamma_2)} + \dots + r^{(\gamma_T)}$$

To successfully recover the key pool of size s , the adversary is asked to give a unique solution of a linear system $Ax = b$. The variable x is an s -dimensional vector where each entry represents a secret key value in the key pool. The coefficient matrix A is a sparse matrix of dimension $\mathcal{Q} \times s$, where $\mathcal{Q} = \text{poly}(\lambda)$ is the number of queries made by the adversary. The non-zero elements indicate the secret key value used for generating the pad r . The sparsity of the matrix ranges from $[\frac{s}{\mathcal{Q}}, \frac{sT}{\mathcal{Q}}]$.

To launch the attack, adversary \mathcal{A} needs at least s equations to give a solution to the system. Consider the case that \mathcal{A} only retrieves k equations where $k < s$; it is clear that the linear system would be an underdetermined system, which implies infinite solutions. Thus, to recover the key pool, the adversary shall have at least s equations. However, notice

that, for each query, the attacker is only able to recover the pad r while the randomnesses $\gamma_1, \dots, \gamma_T$ are kept secret on the user's end. Thus, to recover the secret key, \mathcal{A} has to first guess the randomnesses, i.e., to guess the coefficient matrix A for at least s times. For each query, the adversary needs to identify which key value is used in generating the pad. This problem can be described as a sampling problem with replacement. The possible combination of secret key values is given by $\binom{s+T-1}{T} = \frac{(s+T-1)!}{T!(s-1)!}$. That is, the probability of a successful guess in one equation is $p = \frac{T!(s-1)!}{(s+T-1)!}$. Here, we do not consider the trivial case $T = 1$ in which for each query adversary can reveal a secret key value since the pad is exactly equal to the secret key chosen. When $T \geq 2$, to retrieve the secret key value, the adversary needs to recover at least s equations on Q attempts. Thus, the overall probability of recovering s equations out of Q attempts is given by $Pr_{KRA} = \binom{Q}{s} p^s (1-p)^{Q-s}$. We require the probability to be negligible in terms of the security parameter λ . That is, $Pr_{KRA} = \binom{Q}{s} p^s (1-p)^{Q-s} = \text{negl}(\lambda)$. To prove this, we first show that the upper bound of $\binom{Q}{s}$ is given by $\lceil \frac{eQ}{s} \rceil^s$. To start with, we know that

$$\binom{Q}{s} = \frac{Q!}{(s)!(Q-s)!} = \frac{Q(Q-1) \cdots (Q-s+1)}{s!} < \frac{Q^s}{(s)!}. \quad (5)$$

Consider the expansion of $e^s = \sum_{i=0}^{\infty} \frac{(s)^i}{i!}$. If we only consider the term with respect to s , we have $e^s > \frac{s^s}{(s)!}$, which immediately yields $\frac{1}{(s)!} < \lceil \frac{e}{s} \rceil^s$. Substituting the term in Equation (5), we have $\binom{Q}{s} < \lceil \frac{eQ}{s} \rceil^s$. Now, it is easy to see that

$$Pr_{KRA} = \binom{Q}{s} p^s (1-p)^{Q-s} < \left[\frac{eQ}{s} \right]^s p^s = \left[\frac{epQ}{s} \right]^s$$

Next, we show that, with the careful parameter, we have $Pr_{KRA} < \frac{1}{\text{poly}(\lambda)}$. Specifically, we have $Pr_{KRA} = \left[\frac{epQ}{s} \right]^s$, where s is the key pool size. Notice that Q is polynomial in terms of λ , where $Q = \text{poly}(\lambda)$. Since $\left[\frac{e}{s} \right]^s \ll 1$, we have $Pr_{KRA} = \left[\frac{epQ}{s} \right]^s < \text{poly}(\lambda) \cdot p^s < \frac{1}{\text{poly}(\lambda)}$. If p is sub-exponential, then the inequality holds. We know that $p = \frac{(s+T-1)!}{T!(s-1)!} < \left[\frac{1}{s} \right]^T < \frac{1}{2^{\lambda^{1/3}}}$. \square

Discussion We now discuss the overall security of the protocol and the impact of the parameters. In our protocols, we have two important parameters: s and T , where s denotes the number of secret keys, and T represents the number of keys used to generate the pad r for encryption. The key-pool size s is strongly related to the storage overhead, since our protocol needs to pre-compute the product of the keys and weights. T is related to the computation complexity, because the IoT device has to perform T additions for each encryption and T subtraction with a search operation for each decryption.

To fulfill the security requirement while preserving the efficiency, the parameters shall be chosen carefully. Throughout our implementation, s is chosen to be 8192 under careful consideration of the storage overhead. Under this condition, to fulfill the requirement of being semantically secure with a security parameter $\lambda = 128$ (or $\lambda = 80$), T is required to be at least 10 (or 6). Nonetheless, semantic security indicates that the adversary cannot distinguish between different plaintexts being encrypted. If such a security requirement can be relaxed by instead focusing on defeating the key-recovery attack, the protocol can be made more efficient. Following Theorem 3, when $T \geq 2$, the probability that the adversary can launch a successful key-recovery attack is negligible. In real IoT systems, the trade-off between security and efficiency can be chosen based on practical security requirements.

4.2. Performance Analysis

In this section, we provide the performance analysis both from a theoretical perspective and with experimental results. In our protocol, we consider input data and weights to be

real numbers, and we use floating-point operations (FLOPs) to denote the addition and multiplication operation. We consider a typical network structure AlexNet for our analysis.

Computational Cost Given a convolutional layer with H $k \times k \times D$ kernels, stride \hat{s} , and padding p , for each input data of size $\hat{n} \times \hat{n} \times D$, the output tensor is of size $\hat{n}' \times \hat{n}' \times H$, where $\hat{n}' = \frac{\hat{n}-k+2p}{\hat{s}}$. An IoT device needs to perform $\hat{n}^2 DT$ FLOPs and $\hat{n}'^2 HT$ FLOPs to encrypt and decrypt, respectively. In comparison, the computational cost for local execution is $2H\hat{n}k^2\hat{n}'^2$ FLOPs. Moreover, it is noteworthy that the operations in our protocol are addition/subtraction only, while the local execution contains $H\hat{n}k^2\hat{n}'^2$ multiplication and $H\hat{n}k^2\hat{n}'^2$ addition. For a fully-connected layer of input size \bar{n} and output size \bar{m} , the IoT device needs to perform $\bar{n}T$ FLOPs for encryption and $\bar{m}T$ FLOPs for decryption. For comparison, the IoT device needs to perform $\bar{n}\bar{m}$ FLOPs if it executes such a fully-connected layer on IoT device without outsourcing. It is worth noting that, when executing the fully-connected layer at local, the IoT device is performing $\bar{n}\bar{m}$ multiplications, while, in our scheme, the IoT device only needs to preform $(\bar{m} + \bar{n})T$ additions on its end. Moreover, IoT devices need to handle non-linear computation locally. As depicted in Table 3, the comparison of IoT computation on linear layers between local executions and our protocol shows that our design dramatically reduces the overhead and achieves better performance in CNN inference tasks, considering that the parameter T is usually small for practical security.

Table 3. The comparison of computational cost of IoT device between local execution and our scheme.

Computational Cost of IoT Device		
	Convolutional Layers	Fully-Connected Layer
Local execution	$2H\hat{n}k^2\hat{n}'^2$	$\bar{n}\bar{m}$
Ours	$\hat{n}^2 DT + \hat{n}'^2 HT$	$(\bar{m} + \bar{n})T$

Communication Cost The communication cost of our protocol is mainly from the transmission of the ciphertext and the outputs of the convolutional layers and fully-connected layers. To outsource a convolutional layer with $\hat{n} \times \hat{n} \times D$ inputs, the IoT device first encrypts it and sends the corresponding ciphertext, which is of size $D\hat{n}^2$, to the edge device. Then, the edge device computes the expected scalar products and sends back the results, which are matrices of size $\hat{n}' \times \hat{n}' \times H$. Regarding the fully-connected layer, the IoT device transmits encrypted message vectors of exactly the same dimensions as the plaintext ones, i.e., \bar{n} -dimensional input vectors and \bar{m} -dimensional output vectors. Thus, the total communication cost for a fully-connected layer would be $\bar{m} + \bar{n}$.

Storage Overhead To assure correct decryption, the scalar products of the secret keys and weights shall be pre-computed and stored in the IoT device as showed in Table 2. The pre-computation occurs only in the initialization phase, and once finished, there would be no extra storage overhead introduced during the offloading process. To outsource a convolutional layer with H kernels of $k \times k$ matrices, the IoT device needs to store the scalar products with secret keys of size sk^2H for decryption, which is quite small for most CNN architecture where the kernel sizes are normally 3×3 or 1×1 . Moreover, the IoT device also stores a constant-size key pool measured by s . To outsource a fully-connected layer with \bar{n} -dimensional input and \bar{m} -dimensional output, the IoT device needs to store $s\bar{m}\bar{n}$ as a decryption key.

4.3. Experiment Result

We implemented our protocol on real devices to evaluate its performance. Our implementation adopts the TensorFlow library with Python 3.8. We used a Raspberry Pi 4 (Model B) as the IoT device, which is configured with Raspbian Debian GNU/Linux 11 (bullseye) and has an 1.5GHz quad-core ARM Cortex-A72 processor, 8 GB SDRAM of LPDDR4, and 32 GB SD card storage. We use a desktop as the edge device, which is configured with Ubuntu 20.04.3 LTS, an 8-core 3.60GHz Intel i7-9700K processor, 32 GB memory, two Nvidia

GeForce RTX 2080Ti GPU, and 2TB HDD storage. The IoT device and the edge device are deployed in the same building and connected through 2.4GHz WiFi of 300 Mbps. We used ImageNet [16] as the dataset and implemented a privacy-preserved SqueezeNet and ResNet-50 to compare with the current state-of-the-art cryptographic 2PC-NN solution CrypTFlow2 [17]. The result depicted in Table 4 shows that our solution has $4.7\times$ to $24.7\times$ speed-up in terms of computation as compared to CrypTFlow2 on the two CNN models being evaluated. For the communication overhead, our design requires an interaction of the client and the edge node only for the transmission of the encrypted data and the encrypted scalar product, which are of the same size compared to the expanded ciphertext using heavy cryptographic primitives such as homomorphic encryption. As a result, the communication cost of our protocol is 65 to 4000 times less than CrypTFlow2.

Table 4. Performance comparison with CrypTFlow2 [17] for End-to-End time and communication. Time is measured in seconds and communication in GB.

Benchmark	Methods	End-to-End Time	Communication
SqueezeNet	[17]-SCI _{HE}	59.2	5.27
	[17]-SCI _{OT}	44.3	26.07
	Ours	9.31	0.08
ResNet-50	[17]-SCI _{HE}	545.8	32.43
	[17]-SCI _{OT}	619.4	370.84
	Ours	25.19	0.09

We summarize the experimental results from AlexNet in Table 5 and compare the performance of each layer with local execution and the overall performance with the literature [15]. As shown in Table 5, our protocol significantly improves the efficiency of the AlexNet inference task, achieving a $14.26\times$ speed-up compared with executing the inference task on IoT devices locally. It is noteworthy that, for the convolutional layer, our protocol achieves up to over $66\times$ faster than local computation. With the number of convolution layers increasing in more complicated CNNs such as ResNet-50, DenseNet-121, our secure inference scheme retains the advantage of performance, as shown in Table 4.

In terms of storage overheads, in our design, the storage cost, which contains mainly the secret-key pool \mathcal{R} and the pre-computation table, is fixed once the pre-trained CNN model is selected. For SqueezeNet and ResNet-50, the storage overhead of our protocol is 0.54 GB and 2.61 GB, respectively. For the implementation of AlexNet, considering the extremely imbalanced ratio between parameters and FLOPs, where there are over 586 million parameters with 58 million FLOPs performed in fully connected layers, while the number is 3 million parameters with over 665 million FLOPs, we outsource the convolutional layers only and keep the fully-connected layers computed locally. The storage overhead for the convolutional layers of AlexNet is 1.98 GB. We want to mention that better performance is expected when the IoT devices with sufficient storage can pre-load the complete pre-computation table for AlexNet.

Our experiment also shows that [15] is faster than our protocol. According to our experiments, it takes [15] around 2.51 s to execute the AlexNet, while ours took around 6.76 s, though both are significantly faster than other crypto-based solutions such as [17]. However, as mentioned earlier, one outstanding limitation of [15] is that its storage cost on the IoT device grows linearly to the number of inference instances being executed. For example, a 32 GB SD card can store pre-computed secrets that are only enough for around 1600 AlexNet inferences and 250 ResNet-50 inferences. After that, the SD card needs to be replaced and re-initialized, which could disturb the continuous operation of the IoT system in real-world applications such as drone-based systems. Our design eliminates such a limitation because of the constant storage overheads.

Table 5. Performance comparison with local execution. Time is measured in seconds.

	IoT w/o Outsourcing	IoT Device Computation	Edge Device Computation	Communication	Total	Speedup
Conv-1	7.39	1.05	0.039	0.058	1.147	6.44×
Conv-2	29.654	1.152	0.028	0.036	1.216	24.38×
Conv-3	14.372	0.256	0.03	0.050	0.336	42.77×
Conv-4	25.62	0.383	0.032	0.048	0.463	66.89×
Conv-5	15.141	0.255	0.029	0.050	0.334	59.37×
FC	3.095	3.095	N/A	N/A	3.095	N/A
Non-linear	1.33	1.33	N/A	N/A	1.33	N/A
Total	96.422	6.191	0.158	0.242	6.76	14.26×
[15]	96.422	1.932	0.268	0.315	2.515	38.41×

5. Other Related Works

With the increasing popularity of “Machine Learning as a Service” (MLaaS) in recent years, users take advantage of the powerful cloud server to perform deep learning inference tasks and free the local device of a heavy computation workload. While offloading offers an appealing practice for the users, the concern of exposing sensitive data to the cloud service provider has greatly drawn attention from both the industry and academia. Providing a privacy-preserved machine learning service that enables the user to upload their data and retrieve the inference result without revealing any sensitive information becomes the hinge of fate. Since the inception of secure inferences, considerable efforts have been made to address secure deep neural network inference issues using modern cryptographic tools such as homomorphic encryption and multi-party computation techniques [12,17–33]. Most of these works focus on a two-party scenario (2PC-NN), in which the user only interacts with a single server, while there are some other works [28–31] considering multiple servers, which is much more complicated than a single server when it comes to real-world implementation.

Some early works adopt homomorphic encryption [12,18] to compute the linear operations and introduce approximation to handle the non-linear activation function or pooling layers. These works are not only very costly due to the heavy cryptographic primitive itself, but also fail to provide a secure and efficient evaluation of the non-linear operation (e.g., ReLU function). Another trend is to use MPC-based techniques such as oblivious transfers and garbled circuits [19,25] to evaluate the convolution operation. These methods, while introducing less computational overheads, incur tremendous communication complexities. For example, even with an extremely simplified neural network containing only three fully-connected layers with square activation [19] trained on the CIFAR-10 dataset, MiniONN [25] has a communication overhead of over 9000 MB. Hence, a mixed strategy of combining HE- and MPC-based techniques has come into vogue. The intuition is that HE performs better in matrix–vector computations and the non-linear functions in most CNNs can be represented as some linear circuits and better handled by oblivious transfers and garbled circuits.

In spite of recent advances, the performance of existing 2PC-NN systems is still far from practical, especially for the time-sensitive and resource-constrained IoT application. The state-of-the-art solution CryptFlow2 [17], though it enables a CNN inference task at large scale dataset such as ImageNet for the first time, introduces over 370 gigabytes of communication overheads and takes over 600 s on ResNet-50 in their OT-based method. On top of that, some recent works [26,34] replace the ReLU activation function with approximations due to the challenges with the implementation of secure ReLUs. However, this method leads to a significant loss in model accuracy. Alternatively, many protocols have to trade security for efficiency by assuming a trusted third party [20,23,24,32] or by allowing the leakage of intermediate results [33]. Moreover, many works [19,20,24,26,32] sacrifice a certain level of correctness due to the truncation protocol in their fixed-point arithmetic implementations, which leads to unexpected probability error.

Moreover, many of these works utilize Single Instruction/Multiple Data (SIMD) operations to improve performance. However, due to the inevitable matrix–vector multiplication, numerous homomorphic rotations are required in these SIMD-based schemes [12,21,33]. The major concern is that these operations are not only extremely expensive [33], but also bring challenges to security due to the attempts to accelerate these homomorphic rotations with special modulus techniques adopted [35]. The key-switching matrices in this technique are related to a large modulus compared to the original BGV scheme [36], while requiring a small noise term, which means there is a larger ratio of modulus/noise in the LWE problem. Thus, to obtain the same level of security (e.g., 80-bit security), the lattice dimension has to be increased accordingly, which leads to a slower homomorphic operation. These shortcomings have become the major challenges that limit the performance of the 2PC-NN inference system.

6. Conclusions

In this work, we design a secure and efficient CNN inference outsourcing protocol for resource-constrained IoT devices with the assistance of edge computing. Observing the dominance of linear operations in most DNN architecture, we propose a lightweight scalar-product evaluation primitive iPPSP that can securely offload over 99% of CNN inference operations and achieve improved trade-offs among privacy, efficiency, and utility as compared to the state-of-the-art research. Thorough security analysis shows that our protocol is secure under chosen-plaintext attacks and key-recovery attacks. Experimental results on widely used CNN architecture including AlexNet and ResNet-50 show that our protocol significantly outperforms the state-of-the-art 2PC-NN solution by up to $66.89\times$ speed-up for convolutional layers and up to $24.7\times$ for the overall inference. As compared to existing lightweight interactive solutions, our protocol has an obvious advantage of a fixed storage cost, which eliminates the need for the periodic re-initialization of the IoT devices. Our iPPSP primitive itself is generic and can be used as a building block in a wide range of similar applications, wherein the secure offloading of scalar products is needed.

Author Contributions: Conceptualization, S.Y. and J.Y.; methodology, J.L., Z.Z. and S.Y.; software, Z.Z.; validation, J.L.; formal analysis, J.L.; investigation, J.L.; writing—original draft preparation, J.L.; writing—review and editing, J.L. and S.Y.; supervision, S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Marian, V.; Moons, B. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Mag.* **2017**, *9*, 55–65.
2. Furqan, A.; Mehmood, R.; Katib, I.; Albogami, N.N.; Albeshri, A. Data fusion and IoT for smart ubiquitous environments: A survey. *IEEE Access* **2017**, *5*, 9533–9554.
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [[CrossRef](#)]
4. Collobert, R.; Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th International Conference on Machine Learning (ICML' 18), Helsinki, Finland, 5–9 July 2008; pp. 160–167.
5. Mehdi, M.; Al-Fuqaha, A.; Sorour, S.; Guizani, M. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2923–2960.
6. Jie, T.; Sun, D.; Liu, S.; Gaudiot, J.-L. Enabling deep learning on IoT devices. *Computer* **2017**, *50*, 92–96.
7. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR' 16), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.

8. Motamedi, M.; Fong, D.; Ghiasi, S. Fast and energy-efficient CNN inference on IoT devices. *arXiv* **2016**, arXiv:1611.07151.
9. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University: Stanford, CA, USA, 2009. Available online: <https://crypto.stanford.edu/craig> (accessed on 15 August 2022).
10. Yao, A.C.-C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), Toronto, ON, Canada, 27–29 October 1986; pp. 162–167.
11. Goldreich, O.; Micali, S.; Wigderson, A. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*; ACM: New York, NY, USA, 2019; pp. 307–328.
12. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the International Conference on Machine Learning (ICML'16), New York, NY, USA, 19–24 June 2016; pp. 201–210.
13. Mustafa, E.; Shuja, J.; Jehangiri, A.I.; Din, S.; Rehman, F.; Mustafa, S.; Maqsood, T.; Khan, A.N. Joint wireless power transfer and task offloading in mobile edge computing: A survey. *Clust. Comput.* **2022**, *25*, 2429–2448. [\[CrossRef\]](#)
14. Zhou, S.; Jadoon, W.; Shuja, J. Machine learning-based offloading strategy for lightweight user mobile edge computing tasks. *Complexity* **2021**, *2021*, 6455617. [\[CrossRef\]](#)
15. Tian, Y.; Yuan, J.; Yu, S.; Hou, Y.; Song, H. Edge-assisted CNN inference over encrypted data for Internet of Things. In *International Conference on Security and Privacy in Communication Systems*; Springer: Cham, Switzerland, 2019; pp. 85–104.
16. Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Li, F.-F. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09), Miami, FL, USA, 20–25 June 2009; pp. 248–255. [\[CrossRef\]](#)
17. Rathee, D.; Rathee, M.; Kumar, N.; Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R. CryptFlow2: Practical 2-party secure inference. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS'20), Virtual Event, 9–13 November 2020; pp. 325–342.
18. Hesamifard, E.; Takabi, H.; Ghasemi, M. Cryptodl: Deep neural networks over encrypted data. *arXiv* **2017**, arXiv:1711.05189.
19. Mohassel, P.; Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P'17), San Jose, CA, USA, 22–24 May 2017; pp. 19–38.
20. Wagh, S.; Gupta, D.; Chandran, N. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* **2019**, *2019*, 26–49. [\[CrossRef\]](#)
21. Juvekar, C.; Vaikuntanathan, V.; Chandrakasan, A. GAZELLE: A low latency framework for secure neural network inference. In Proceedings of the 27th USENIX Security Symposium (USENIX Security'18), Baltimore, MD, USA, 15–17 August 2018; pp. 1651–1669.
22. Bian, S.; Wang, T.; Hiromoto, M.; Shi, Y.; Sato, T. ENSEI: Efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'20), Seattle, WA, USA, 13–19 June 2020; pp. 9403–9412.
23. Riaz, M.S.; Weinert, C.; Tkachenko, O.; Songhori, E.M.; Schneider, T.; Koushanfar, F. Chameleon: A hybrid secure computation framework for machine learning applications. In Proceedings of the 2018 Asia Conference on Computer and Communications Security (AsiaCCS'18), Incheon, Korea, 4–8 June 2018; pp. 707–721.
24. Mohassel, P.; Rindal, P. ABY3: A mixed protocol framework for machine learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18), Toronto, ON, Canada, 15–19 October 2018; pp. 35–52.
25. Liu, J.; Juuti, M.; Lu, Y.; Asokan, N. Oblivious neural network predictions via mininn transformations. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17), Dallas, TX, USA, 30 October–3 November 2017; pp. 619–631.
26. Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; Popa, R.A. Delphi: A cryptographic inference service for neural networks. In Proceedings of the 29th USENIX Security Symposium (USENIX Security'20), Virtual Event, 12–14 August 2020; pp. 2505–2522.
27. Riaz, M.S.; Samragh, M.; Chen, H.; Laine, K.; Lauter, K.; Koushanfar, F. XONN: XNOR-based Oblivious Deep Neural Network Inference. In 28th USENIX Security Symposium (USENIX Security'19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1501–1518.
28. Agrawal, N.; Shamsabadi, A.S.; Kusner, M.J.; Gascón, A. QUOTIENT: Two-party secure neural network training and prediction. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19), London, UK, 11–15 November 2019; pp. 1231–1247.
29. Liu, X.; Wu, B.; Yuan, X.; Yi, X. Leia: A lightweight cryptographic neural network inference system at the edge. *IEEE Trans. Inf. Forensics Secur.* **2021**, *17*, 237–252. [\[CrossRef\]](#)
30. Juuti, M.; Szyller, S.; Marchal, S.; Asokan, N. PRADA: Protecting against DNN model stealing attacks. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P'19), Stockholm, Sweden, 17–19 June 2019; pp. 512–527.
31. Chaudhari, H.; Rachuri, R.; Suresh, A. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv* **2019**, arXiv:1912.02631.
32. Kumar, N.; Rathee, M.; Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R. Cryptflow: Secure tensorflow inference. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P'20), Virtual Event, 18–21 May 2020; pp. 336–353.

33. Boemer, F.; Costache, A.; Cammarota, R.; Wierzynski, C. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC' 19), London, UK, 11 November 2019; pp. 45–56.
34. Dathathri, R.; Saarikivi, O.; Chen, H.; Laine, K.; Lauter, K.; Maleki, S.; Musuvathi, M.; Mytkowicz, T. CHET: An optimizing compiler for fully-homomorphic neural-network inferencing. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI' 19), Phoenix, AZ, USA, 22–26 June 2019; pp. 142–156.
35. Gentry, C.; Halevi, S.; Smart, N.P. Homomorphic evaluation of the AES circuit. In *Annual Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 850–867.
36. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12), Cambridge, MA, USA, 8–10 January 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 309–325. [[CrossRef](#)]