`FastSecNet`: An Efficient Cryptographic Framework for Private Neural Network Inference

Meng Hao, *Student Member, IEEE*, Hongwei Li, *Senior Member, IEEE*, Hanxiao Chen, *Student Member, IEEE*, Pengzhi Xing, *Student Member, IEEE*, and Tianwei Zhang, *Member, IEEE*

***Abstract*—Private neural network inference has demonstrated great importance in various privacy-critical scenarios. However, the primary challenge remaining in prior works is that the evaluation on encrypted data levies prohibitively high runtime and communication overhead. In this work, we present `FastSecNet`, an efficient two-party cryptographic framework for private inference in the dealer-based pre-processing setting. Specifically, (1) `FastSecNet` provides an efficient ReLU protocol for the evalution of non-linear layers, which is built up on a recent advanced cryptographic primitive, function secret sharing (FSS). The core of this construction are an optimized ReLU representation and a customized FSS-based ReLU protocol. (2) For linear layer evaluation, we first propose an efficient PRG-based pre-processing protocol based on the fact that one of the inputs is uniformly random in the offline phase. Then, the online phase only communicates one element and consists of lightweight secret-sharing operations in a ring. Extensive evaluations conducted on 4 real-world datasets and 9 neural network models demonstrate that during the online phase, `FastSecNet` achieves 14$\times$ less runtime and 18$\times$ less communication cost compared to the state-of-the-art.**

***Index Terms*—Private neural network inference, Secure multi-party computation, Function secret sharing.**

I. Introduction

Recent advances in deep learning (DL) raise increasing demand and deployment of neural network inference in various applications like voice assistant [1], medical diagnosis [2] and image classification [3]. Many IT companies release online inference services to facilitate these applications, such as Google ML Engine [4] and Microsoft Azure ML Studio [5]. Unfortunately, current DL-based inference systems suffer from serious privacy concerns [6]. On one hand, clients need to send sensitive inputs to the service provider (e.g., a remote server), which could compromise the data privacy of these clients if the provider is untrusted [7]. On the other hand, an alternative is that the provider distributes the proprietary model to clients, but this causes intellectual property violations [8], [9].

To solve these privacy issues, researchers have proposed a quantity of works on private inference [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] based on Homomorphic Encryption (HE) [20] or secure two-party computation (2PC) techniques such as Garbled Circuit (GC) [21] and Secret Sharing (SS) [22] (more details refer to Section VI). These works guarantee that the server learns nothing about the clients' input and the clients obtain zero information about the server's model except for the prediction results. Despite such desirable security guarantees, most prior works are computationally intensive and often require a large amount of communication between clients and the server.

Recently, Mishra et al. [18] proposed Delphi, a new inference framework to alleviate such efficiency concerns. In Delphi, the inference pipeline is divided into an input-independent *offline* phase and an input-dependent *online* phase. Then it introduces cryptographic protocols *in the pre-processing model*, and significantly reduces the online cost by moving most heavy cryptographic computations into the offline phase. With such a design, all linear operations in the online phase can be performed directly over secret-shared data without invoking heavy cryptographic tools like HE, or frequent interactions. However, a remaining problem in Delphi is that evaluating non-linear layers (e.g., ReLU and Maxpool) with GCs are several orders of magnitude more expensive than the linear layer protocols in terms of communication and computation [23]. This is because the computation of a function within GCs requires it to be decomposed into a circuit of binary gates and processed in an encrypted bit-wise fashion. For example, ReLU operations account for 93% of ResNet32's online runtime in Delphi [24]. Although recent works [25], [26] have proposed optimized ReLU protocols, these methods either cannot be directly scaled to the offline-online paradigm, or require multiple rounds of communication and special secret sharing primitives.

In this paper, we design `FastSecNet`, an efficient cryptographic framework for neural network inference in the pre-processing paradigm. `FastSecNet` focuses on the online phase and assumes a dealer[1] for distributing correlated randomness, and achieves significant online performance gains. We make several innovations in the design of `FastSecNet`. First, we design an efficient ReLU protocol by leveraging an advanced cryptographic technique, i.e., function secret sharing (FSS) [31], [28]. The main bottleneck of existing FSS-based solutions is the high online computation and communication overhead [27]. To address this problem, we first provide an optimized ReLU representation such that the invocation of the costly comparison protocol is reduced from two to one. We

Meng Hao (This work was done at NTU as a visiting student.), Hongwei Li, Hanxiao Chen and Pengzhi Xing are with School of Computer Science and Engineering, University of Electronic Science and Technology of China, China. Corresponding author: Hongwei Li. (email: menghao@std.uestc.edu.cn, hongweili@uestc.edu.cn, hanxiao.chen@std.uestc.edu.cn, p.xing@std.uestc.edu.cn)

Tianwei Zhang is with School of Computer Science and Engineering, Nanyang Technological University, Singapore. (email: tianwei.zhang@ntu.edu.sg)

[1]This dealer is also used in recent private neural network inference works [16], [27], [28], and can be jointly emulated via general two-party secure protocols [29], [30].

theoretically prove that it causes ReLU to fail with a small probability, without sacrificing model accuracy. With this optimized ReLU representation, we then design a customized FSS protocol and address two key challenges. The former is that we conduct a great-than protocol from existing less-than protocols [32], [28] without extra evaluation overhead, while the latter is to provide an efficient extension from comparison to our optimized ReLU. Together, our new ReLU protocol requires only one round of interaction in which each party sends just n bits (n is the size of the secret-sharing ring) during the online phase, and achieves about $2\times$ runtime improvement over the most efficient FSS scheme [28] and $3\times$ communication improvement over prior FSS-based private inference work [27]. Moreover, compared with Delphi [18] that requires κn bits communication (κ is the security parameter), our protocol achieves $\frac{\kappa}{2}\times$ improvement, e.g., $64\times$ for a reasonable choice of $\kappa = 128$.

Second, for the evaluation of linear layers, we propose a customized multiplication protocol in the pre-processing setting. The main insight is to pre-compute the shares of multiplication between the model weight and a random number in the offline phase, such that with these correlated randomness, the online phase can be efficiently evaluated. In particular, in the offline phase, the multiplication operation is executed by invoking the Beaver's protocol [22], in which we utilize Pseudo-random Generator (PRG) to generate Beaver's triples in a communication-efficient manner. Moreover, we adaptively modify the Beaver's multiplication process based on the observation that one of the inputs is uniformly random. As a result, the offline multiplication process only communicates one ring element, rather than two elements [22]. In the online phase, the two parties consume the pre-computed correlated randomness and communicate just one ring element, without costly cryptographic operations. This online solution is similar as [18], but our protocol allows for a more efficient implementation since it works over a ring without modulo a large prime [18].

We give a formal security proof to demonstrate the security guarantee. Moreover, we conduct extensive experiments on four real-world datasets (MNIST, CIFAR10, CIFAR100 and ImageNet) with various neural network architectures (LeNet, ResNet, VGG, SqueezeNet, DenseNet). We compare `FastSecNet` with recent 2-party private inference works to demonstrate its unprecedented efficiency gains. `FastSecNet` achieves $14\times$ less runtime and $18\times$ less communication cost compared to the state of the art.

Our key contributions can be summarized as follows:

- We propose an online-efficient cryptographic framework called `FastSecNet` for private neural network inference in the dealer-based pre-processing setting.
- We design a novel ReLU protocol for non-linear layers via function secret sharing and propose an optimized multiplication protocol for linear layers.
- Extensive experiments on various datasets and models show that `FastSecNet` outperforms state-of-the-art works at least one order of magnitude in the online phase.

The rest of this paper is organized as follows. In Section II, we introduce preliminaries, system and threat models. In Section III, we design new non-linear protocols based on the function secret sharing technique. We present our complete `FastSecNet` framework in Section IV. The experimental evaluation is provided in Section V. We review the latest related works in Section VI and conclude in Section VII.

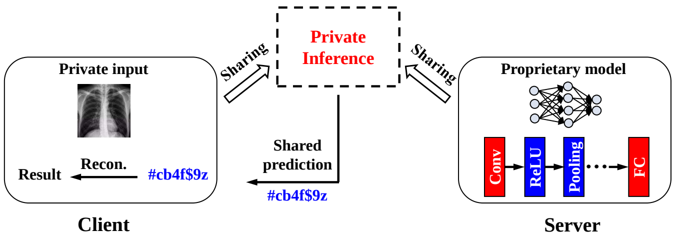


Fig. 1: System model

II. Preliminaries

A. System Model

We consider a general private inference scenario [33], [18], where the server holds a neural network model M with private weights w, while a client holds a private input sample x. The goal of the client is to obtain the output of the model on its input, i.e., $M(w, x)$, while the server learns no information about the client's private input x. As shown in Figure 1, the system model consists of three steps in each inference. Specifically, (I) the client secret-shares his input x with the server. (II) The client and server jointly perform private inference on w and x utilizing secure computation protocols. (III) The server returns the share of the prediction y to the client, who reconstructs the prediction in plaintext.

B. Threat Model

The security of `FastSecNet` is provably provided in the simulation paradigm [34] against static honest-but-curious probabilistic polynomial-time (PPT) adversaries. Namely, a PPT adversary $\mathcal{A}$ passively corrupts either the server or the client at the beginning of the protocol and honestly follows the protocol specification. In the simulation paradigm, two worlds are defined: a real world where the server and the client perform the protocol according to the specification in the presence of $\mathcal{A}$, and an ideal world where the parties send their inputs to an ideal functionality that executes the evaluation faithfully. It is required that for any adversary, the real-world distribution is computationally indistinguishable to the ideal-world distribution. Some of our protocols invoke sub-protocols and we describe them using the hybrid model. This is similar to a real execution, except that sub-protocols are replaced by the invocations of the corresponding functionality instances. Formally, a protocol invoking a functionality $\mathcal{F}$ is called to be in $\mathcal{F}$-hybrid model.

Our protocols consist of offline and online phases. Similar to existing works [28], [16], [27], we instantiate the offline phase with an honest-but-curious third party (STP). This STP-assisted technique facilitates the performance benefits of the offline phase. In this setting, the security model is adaptively modified, where the STP is assumed to be disallowed to collude with the server or the client. This is because if the STP colludes with either party, they may reconstruct private information such as the input or the model weights during the protocol execution.

C. Design Goals

`FastSecNet` aims to empower the neural network inference system that achieves privacy protection and protocol efficiency at the same time without sacrificing the model accuracy. More specifically, we demonstrate the following desirable goals:

- **Privacy protection**. The client's input sample contains sensitive information, and the server's model is an important intellectual property. They cannot be leaked to other parties during the inference process.
- **Efficient evaluation**. The proposed scheme should incur moderate computation and communication costs for the parties. This is particularly important for real-time or resource-limited scenarios.
- **Intact inference accuracy**. Compared to inference tasks in non-private settings, the designed protocol should not sacrifice inference accuracy, especially when it is applied to critical infrastructures such as Healthcare.

It is worth noting that `FastSecNet` is responsible for protecting the privacy of inference data and model from direct leakage. It does not hide the information that can be indirectly extracted from the inference results. This is in line with previous private inference works [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. An adversary can infer the knowledge about the model architectures, parameters and hyperparameters via model extraction attacks [35], [9], [8], or reconstruct the training data via model inversion attacks [36], [37]. Existing mitigation solutions for these attacks, such as differential privacy [38], [39], can be possibly combined with `FastSecNet`'s protocols to provide stronger privacy guarantees, which is beyond the scope of this work.

D. Cryptographic Primitives

We provide a description of the cryptographic building blocks used in `FastSecNet`. Before that, we first introduce some notations. Let $\mathbb{Z}_N$ be input and output domains of size $N = 2^n$, where n is the bit length. $[a]$ denotes the shared values of a and the security parameter is denoted as κ. Like most previous works [18], [25], we represent a floating-point number $x \in \mathbb{Q}$ into the ring $\mathbb{Z}_N$. Specifically, we first encode it as a fixed-point number and then embed the fixed-point representation into the ring with 2's complement representation: $a = \lfloor 2^s \times x \rfloor \in \mathbb{Z}_N$ if x is a non-negative number, and $a = N - \lfloor 2^s \times |x| \rfloor \in \mathbb{Z}_N$ if x is a negative number, where s is the length of the (binary) fractional bits.

Secret sharing. We adopt the 2-out-of-2 arithmetic secret sharing over the ring $\mathbb{Z}_N$ [29]. The sharing algorithm takes as input an n-bit value x in $\mathbb{Z}_N$ with $N = 2^n$ and outputs random sampled shares $[x]_0$, $[x]_1$ with the constraint that $x = [x]_0 + [x]_1$ in $\mathbb{Z}_N$. The security of the additive secret sharing protocol guarantees that given a share, the original value is still privately hidden. We notice that for our used fixed-point multiplication operations, we use the truncation method from [40], which is consistent with existing works [18], [41].

Function secret sharing. A function secret sharing (FSS) scheme [28] is an efficient algorithm that splits a function f into two additive shares f_0, f_1, such that: (1) each $f_p, p \in \{0,1\}$, hides f; (2) for every public input x, $f_0(x) + f_1(x) = f(x)$. Formally, a 2-party FSS scheme consists of two algorithms $(\mathsf{Gen}, \mathsf{Eval})$:

- $\mathsf{Gen}(1^\kappa, f)$ is a PPT key generation algorithm that takes as input the security parameter κ and a function f, and outputs a pair of keys (k_0, k_1), where each key implicitly represents $f_p : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$.
- $\mathsf{Eval}(p, k_p, x)$ is a polynomial-time evaluation algorithm that takes as input the party index $p \in \{0,1\}$, the key k_p and the public input $x \in \mathbb{Z}_N$, and outputs $y_p \in \mathbb{Z}_N$, i.e., the value of $f_p(x)$, where $f(x) = f_0(x) + f_1(x)$.

As shown in recent works [28], [32], with a slight transformation, FSS can be evaluated over secret-shared inputs. The main idea is to formulate an *offset* function $f_r(x) = f(x - r)$ and construct the FSS scheme for it, where r is a random number in $\mathbb{Z}_N$ and is secret-shared with the parties. The parties that hold the shares of input x can first reveal the masked input $x + r$, and then evaluate their FSS keys for $f_r(x)$ on $x + r$, which is exactly equivalent to evaluating $f(x)$ on x.

Pseudorandom Generator. A Pseudorandom Generator (PRG) [42] takes as input a uniformly random seed and a security parameter κ, and outputs a long pseudorandom string. The security of PRG ensures that the output of the generator is indistinguishable from the uniform distribution in polynomial-time, as long as the seed is hidden from the distinguisher. In `FastSecNet`, PRGs can be implemented efficiently with current hardware acceleration and enable two parties to generate the same (pseudo-)random numbers without communication.

III. Non-linear Function Evaluation

In this section, we present our novel protocols for non-linear functions, i.e., ReLU and Maxpool.

A. Comparison

We first recall the FSS-based comparison protocol [28] that serves as the underlying operation for non-linear functions such as ReLU and Maxpool in `FastSecNet`. We denote a general comparison function as $f^{<}_{a,b}(x)$, which is computed as b if $x < a$ and 0 otherwise. Following the construction of Boyle et al. [28], our FSS scheme for comparison consists of a pair of algorithms, i.e., $\mathsf{Gen}^{<}_{a,b}$ and $\mathsf{Eval}^{<}_{a,b}$, as shown in Algorithms 1 and 2, respectively. Below we give the key idea of the FSS's construction.

As illustrated in Section II-D, the $\mathsf{Gen}^{<}_{a,b}$ algorithm generates two keys (k_0, k_1). Each of them defines a GGM-style binary tree [43] with 2^n leaves, which are labeled by inputs $x \in \{0,1\}^n$. We will refer to a path from the root to a leaf labeled by x as the *evaluation path*, and to the evaluation path of a as the *special path*. Each node in a tree is associated with a tuple (s_p, v_p, t_p), where $p \in \{0,1\}$ denotes the party index, s_p is the PRG seed, v_p is the output ring element and t_p is the control bit. The function $\mathsf{Eval}^{<}_{a,b}$ will compute the labels of all nodes on the evaluation path to the input x, using the root seed as the initial seed.

The construction requires to maintain the following invariants: 1) for each node outside the special path, the two seeds are identical, 2) for each node on the special path, the two control bits are different and the two seeds are indistinguishable from being random and independent, 3) the sum of $v_0 + v_1$ over all nodes leading to the input x exactly equals $f^{<}_{a,b}(x)$. To this end, the task of $\mathsf{Gen}^{<}_{a,b}$ is to generate the correction words CW such that in $\mathsf{Eval}^{<}_{a,b}$ when a path to x departs from the special path, the two seeds s_0 and s_1 on the first node j off the path are identical. Besides, the sum of $v_0 + v_1$ along the whole path to j is exactly 0 if the departure is to the right of the special path (i.e., $x > a$), and is b if the departure is to the left of the special path. More details can be found in [32], [28].

Algorithm 1 $\mathsf{Gen}^{<}_{a,b}$ for comparison function [28]

```
1:  Let $a = a_1, \ldots, a_n \in \{0,1\}^n$, and $s_0^{(0)}, s_1^{(0)} \leftarrow \{0,1\}^\kappa$.
2:  Let $V_a = 0$, $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$.
3:  for $i \in [n]$ do
4:      $s_p^L \| v_p^L \| t_p^L \| s_p^R \| v_p^R \| t_p^R \leftarrow G(s_p^{(i-1)})$ for $p = 0,1$
5:      $keep \leftarrow L$ and $lose \leftarrow R$ if $a_i = 0$, reverse otherwise.
6:      $s_{cw} \leftarrow s_0^{lose} \oplus s_1^{lose}$.
7:      $V_{cw} \leftarrow (-1)^{t_1^{(i-1)}} \cdot [v_1^{lose} - v_0^{lose} - V_\alpha + 1\{lose = L\} \cdot b]$
8:      $V_a \leftarrow V_a - v_1^{keep} + v_0^{keep} + (-1)^{t_1^{(i-1)}} \cdot V_{cw}$
9:      $t_{cw}^L \leftarrow t_0^L \oplus t_1^L \oplus a_i \oplus 1$ and $t_{cw}^R \leftarrow t_0^R \oplus t_1^R \oplus a_i$
10:     $CW^{(i)} \leftarrow s_{cw} \| V_{cw} \| t_{cw}^L \| t_{cw}^R$
11:     $s_p^{(i)} \leftarrow s_p^{keep} \oplus t_p^{(i-1)} \cdot s_{cw}$ for $p = 0,1$
12:     $t_p^{(i)} \leftarrow t_p^{keep} \oplus t_p^{(i-1)} \cdot t_{cw}^{keep}$ for $p = 0,1$
13: end for
14: $CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot [s_1^{(n)} - s_0^{(n)} - V_a]$
15: Let $k_p = s_p^{(0)} \| CW^{(1)} \| \cdots \| CW^{(n+1)}$ for $p = 0,1$
```

Algorithm 2 $\mathsf{Eval}^{<}_{a,b}$ for comparison function [28]

```
1:  Parse $k_p = s^{(0)} \| CW^{(1)} \| \cdots \| CW^{(n+1)}$, $x = x_1, \ldots, x_n$,
    let $V = 0, t^{(0)} = p$
2:  for $i \in [n]$ do
3:      Parse $CW^{(i)} = s_{cw} \| V_{cw} \| t_{cw}^L \| t_{cw}^R$
4:      Parse $G(s^{(i-1)}) = s^L \| v^L \| t^L \| s^R \| v^R \| t^R$
5:      $s^L \| t^L \| s^R \| t^R \leftarrow s^L \| t^L \| s^R \| t^R \oplus t^{(i-1)} \cdot$
        $[s_{cw} \| t_{cw}^L \| s_{cw} \| t_{cw}^R]$
6:      if $x_i = 0$ then
7:          $V \leftarrow V + (-1)^p \cdot [v^L + t^{(i-1)} \cdot V_{cw}]$
8:          $s^{(i)} \leftarrow s^L, t^{(i)} \leftarrow t^L$
9:      else
10:         $V \leftarrow V + (-1)^p \cdot [v^R + t^{(i-1)} \cdot V_{cw}]$
11:         $s^{(i)} \leftarrow s^R, t^{(i)} \leftarrow t^R$
12: end for
13: $V \leftarrow V + (-1)^p \cdot [s^{(n)} + t^{(n)} \cdot CW^{(n+1)}]$
```

B. ReLU

ReLU is the most common activation function in DL models. With our fixed-point ring representation, $\mathsf{ReLU}(x)$ is equal to x if $x < N/2$ and 0 otherwise. State-of-the-art works [25], [18], [17] utilize garbled circuit (GC) and oblivious transfer (OT) techniques along with customized protocol designs to evaluate this function. However, these solutions cause high communication cost and require multiple communication rounds. For example, the OT-based solution [25] requires $\kappa n + 18n$ bits of communication within $\log n + 2$ communication rounds, and the GC-based solution [18], [17] consumes κn bits of communication within 2 communication rounds during the online phase. To address the performance bottleneck, we propose a communication-efficient and low-interaction ReLU protocol based on function secret sharing. Our new constructions consist of two innovations detailed in the following.

The first innovation is an approximate ReLU representation to reduce the online overhead without sacrificing model accuracy. Specifically, to evaluate ReLU over secret-shared inputs using FSS, similar as [32], [28], we focus on the offset function $\mathsf{ReLU}_r(x) = \mathsf{ReLU}(x - r)$, where r is sampled from $\mathbb{Z}_N$ uniformly at random. There are two cases for $\mathsf{ReLU}_r(x)$ due to the wrap around in the finite ring: (1) $r < N/2 + r \bmod N$, where $\mathsf{ReLU}_r(x)$ equals $x - r$ if x belongs to $[r, N/2 + r \bmod N)$ and 0 otherwise; (2) $r > N/2 + r \bmod N$, where $\mathsf{ReLU}_r(x)$ equals $x - r$ if $x \geq r$ or $x < N/2 + r \bmod N$, and 0 otherwise. It is straightforward to verify that $\mathsf{ReLU}_r(x + r) = \mathsf{ReLU}(x)$. However, as illustrated in [32], [28], both two cases require two invocations of the FSS-based comparison function to learn whether x belongs to the offset interval. To reduce the overhead, inspired by approximate comparison protocols [27], [44], [45], we propose an optimized ReLU representation, which is evaluated by invoking only one comparison. More precisely, the above offset ReLU function can be approximately reformulated as $\mathsf{AReLU}_r(x) = x - r$ if $x \geq r$ and 0 otherwise. In the following theorem, we prove that for a large enough field, the reformulated function $\mathsf{AReLU}_r(x)$, with high probability, equals $\mathsf{ReLU}_r(x)$.

Theorem 1. *For $x, r \in \mathbb{Z}_N$, the probability $P\{\mathsf{AReLU}_r(x + r) \neq \mathsf{ReLU}(x)\}$ is $\frac{|x|}{N}$, where $|x|$ denotes the absolute value of x.*

Proof. We give the proof in Appendix A. □

For typical choices of $\mathbb{Z}_N$ (e.g., $N = 2^{32}$), $|x| \ll N$ in neural network inference and the failure probability $\frac{|x|}{N}$ is about one in a million, which is also observed in recent works [45], [27]. Moreover, as previously shown in [18], neural networks are also resilient to stochastic faults. Therefore, in our evaluation, this optimized ReLU evaluation incurs a very small error on model accuracy.

The second innovation is a customized FSS protocol for the optimized offset function $\mathsf{AReLU}_r(x)$. With the above insight, we only need to invoke one comparison protocol in Section III-A to evaluate $\mathsf{AReLU}_r(x)$. Nevertheless, multiple subtleties arise here: (1) we require a greater-than protocol, i.e., deciding whether $x \geq r$, but Section III-A provides a less-than protocol; (2) the output is a constant b in the comparison protocol for $f^{<}_{a,b}$ of Section III-A, whereas ours is a univariate spline polynomial (i.e., $x - r$ or the zero polynomial). Therefore, particular cares are required. To solve the former problem, we leverage the observation that $f^{\geq}_{a,b}(x) = b + f^{<}_{a,-b}(x)$, and hence can achieve the greater-than protocol without extra protocol modifications. For the latter, inspired by [32], [28], we adapt the above FSS-based greater-than comparison scheme to output the coefficients of the spline polynomial of $\mathsf{AReLU}_r(x)$. More precisely, the construction outputs the coefficients $\mathbf{b} = (b_0, b_1) = (1, -r)$ of the offset polynomial $\mathsf{AReLU}_r(x) = x - r$ if $x \geq r$, or $\mathbf{b} = (b_0, b_1) = (0, 0)$ of $\mathsf{AReLU}_r(x) = 0$ otherwise. After that, using the fact that the two parties know the input x to FSS, they obtain the shares of $\mathsf{AReLU}_r(x)$ via locally computing $[b_0]_0 x + [b_1]_0$ and $[b_0]_1 x + [b_1]_1$ by the server and the client, respectively.

Combining the above two components, we present a complete FSS-based ReLU protocol in Algorithms 3 and 4. Our protocol achieves optimal $2n$ online communication within a single communication round, which shows significant advantages over prior OT- and GC-based solutions [25], [18]. Recently, AriaNN [27] also uses FSS to achieve private inference, but their ReLU protocol requires two communication rounds and $6n$ online communication. The reason is that AriaNN first computes a comparison protocol and then invokes a boolean-arithmetic multiplication protocol. Therefore, compared with the counterpart in AriaNN, our protocol achieves a $3\times$ communication improvement. The detailed comparison of the online overhead is presented in Table I.

TABLE I: The online overhead of our ReLU and comparison with prior works. DCF and BmA denote the comparison and boolean-arithmetic multiplication protocols, repectively. n is the bitlength.

Works	Computation	Communication	Round
[28]	2 DCF	$2n$	1
AriaNN [27]	1 DCF & 1 BmA	$6n$	2
Ours	1 DCF	$2n$	1

Below, we give the security analysis of our designed ReLU protocol in Theorem 2.

Theorem 2. *Assuming the existence of PRG and secure FSS protocols for the comparison function, the protocol described in Algorithms 3 and 4 is a secure ReLU protocol against honest-but-curious adversaries.*

Proof. We give the proof in Appendix B. □

C. Maxpool

The underlying algorithm of Maxpool is to compute the maximum value over d elements $x_1, x_2, \cdots, x_d$. We design an FSS protocol for Maxpool by using a tree-reduction architecture, which recursively partitions the input into two halves and then compares the elements of each half. Specifically, the client and server arrange the d values into a 2-ary tree with a depth of $\log d$, and evaluate the tree in a top-down fashion.

Algorithm 3 $\mathsf{Gen}^{\mathsf{ReLU}}_{a,b}$ for ReLU function

```
1: Let $\mathbf{b} = (b_0, b_1) = (1, -r)$ and $a = r$
2: $(k'_0, k'_1) \leftarrow \mathrm{Gen}^{<}_{a,-\mathbf{b}}$
3: Sample $[r]_0, [r]_1 \leftarrow \mathbb{Z}_N$, s.t., $[r]_0 + [r]_1 = r \bmod N$
4: Sample $[b_0]_0, [b_0]_1, [b_1]_0, [b_1]_1 \leftarrow \mathbb{Z}_N$, s.t., $[b_0]_0 + [b_0]_1 = b_0 \bmod N$ and $[b_1]_0 + [b_1]_1 = b_1 \bmod N$
5: let $k_p = k'_p \| [r]_p \| [\mathbf{b}]_p$ for $p \in \{0, 1\}$
6: Return $(k_0, k_1)$
```

Algorithm 4 $\mathsf{Eval}^{\mathsf{ReLU}}_{a,b}$ for ReLU function

```
1: Parse $k_p = k'_p \| [r]_p \| [\mathbf{b}]_p$
2: Send $[x]_p + [r]_p$ to the other party, receive $[x]_{1-p} + [r]_{1-p}$, and reconstruct $x + r \bmod N$
3: Set $([b_0]_p, [b_1]_p) \leftarrow \mathrm{Eval}^{<}_{a,-\mathbf{b}}(p, k'_p, x + r) + \mathbf{b}_p$
4: Compute $[y]_p = [b_0]_p (x + r) + [b_1]_p \bmod N$
5: Return $[y]_p$
```

In each comparison of two secret-shared elements $[x_i]$ and $[x_j]$, we reduce it to the evaluation of ReLU. We observe $\mathsf{max}([x_i], [x_j]) = \mathsf{ReLU}([x_i] - [x_j]) + [x_j]$, and hence the evaluation complexity of Maxpool mainly comes from the evaluation of $d - 1$ ReLU. Moreover, the security directly follows the security of ReLU in Theorem 2.

IV. FastSecNet Framework

A. Design Overview

At a high level, `FastSecNet` splits the inference pipeline into an offline phase and an online phase. The main goal is to reduce online overhead, especially when evaluating non-linear layers. To this end, we design multiple customized protocols, such that most heavy cryptographic operations are performed in the offline phase. Specifically, for the linear layer, we propose a customized multiplication protocol based on PRGs in the pre-processing phase, and provide critical optimizations on the communication performance. This makes the online phase efficient with only communicating one element in the ring. For the non-linear layer, we use FSS-based protocols for ReLU and Maxpool (in Section III), in which the FSS keys are generated in the offline phase. Notably, we achieve the optimal online communication overhead for ReLU, i.e., one communication round with the size of two secret-shared inputs. Below we give the details of the offline and online protocols.

B. Offline Protocols

Figure 2 describes the offline phase in `FastSecNet`. The client and server first pre-compute correlated randomness that is independent of the client's input and will be used to boost the online protocols. As shown in II-B, the two parties in `FastSecNet` receive the correlated randomness from STP[2]. Alternatively, the role of STP can be jointly emulated via general two-party secure protocols [29], [30]. Note that our

[2]STP is not involved in the online phase

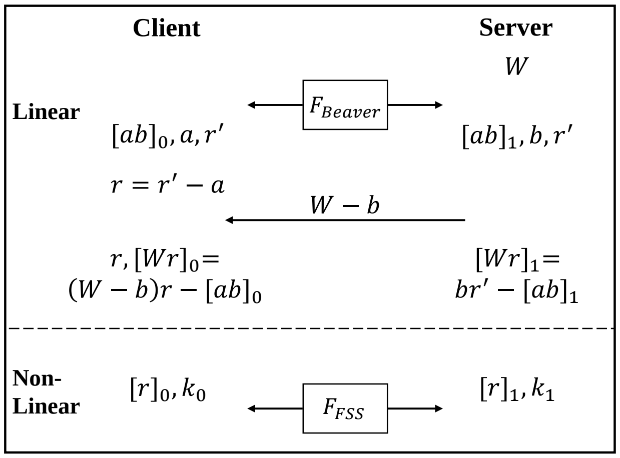


Fig. 2: The offline phase in `FastSecNet`

`FastSecNet` framework is modular, and any optimized techniques can be used to implement the offline phase without affecting the efficiency of the online phase.

Setup. We first construct PRG seeds $\mathbf{seed}_c$ between the client and STP, $\mathbf{seed}_s$ between the server and STP, and $\mathbf{seed}_{cs}$ between the client and the server. These PRG seeds are used to generate the same (pseudo-)random numbers without communication between parties [16].

Linear layer. We propose a communication-efficient protocol to generate correlated randomness for linear layers. The key observation is that the model weight W is held by the server in advance [18]. Note that the linear layer can be formalized as the matrix multiplication operation, i.e., $y = Wx$ where x is the private input of this linear layer. Further, Wx can be represented as $Wx = W(x-r) + Wr$, and r is randomly chosen by the client. Thus, in the offline phase, we focus on pre-computing the shares of Wr, and then efficiently evaluate $W(x-r)$ in the online phase.

The initial idea is to let STP generate Beaver's triples and then evaluate Wr based on the Beaver's multiplication protocol [22]. The former requires STP communicating four ring elements to the two parties, while the latter requires communicating two ring elements between the two parties. We give two insights to optimize this step. On the one hand, we utilize PRGs to generate Beaver's triples, since they are only random values with some correlation. Similar technique is also used in prior works [16], [26], [44]. On the other hand, during the Beaver's multiplication, each party requires to reveal the masked input, but revealing masked r is unnecessary due to the fact that r is itself uniformly random. Therefore, we adaptively modify the multiplication process to only communicate one ring element, rather than two elements. The detailed process is presented as follows:

- STP generates the Beaver's triple (a, b, ab) for the client and the server. Specifically, STP and the client jointly generate a and $[ab]_0$ by using PRGs on the seed $\mathbf{seed}_c$, while STP and the server jointly generate b by using PRGs with $\mathbf{seed}_s$. Besides, STP computes and sends $[ab]_1 = ab - [ab]_0 \mod N$ to the server.
- The server and the client jointly generate r' by using PRGs on the seed $\mathbf{seed}_{cs}$, and the client sets $r = r' - a \mod N$, where r' is the masked input of the client's random input r. This means that the client first generates the masked input and then computes the random input. Compared with prior works [16], [18], this operation removes the communication of the masked r in the Beaver's multiplication. After that, the server sends $W - b$ to the client. Then, the client and the server locally compute $[Wr]_0 = (W-b)r - [ab]_0 \mod N$ and $[Wr]_1 = br' - [ab]_1 \mod N$, respectively.

Non-linear layer. With the technique in Section III, STP runs the FSS generation algorithm of non-linear functions, and distributes the generated keys to the client and the server. We focus the rest on evaluating ReLU, and similar observation holds for computing $\mathsf{Maxpool}$.

- STP generates two random values $[r]_0$ and $[r]_1$ using PRGs with the client and the server, respectively. After that, STP computes $r = [r]_0 + [r]_1 \mod N$, calls the $\mathsf{Gen}^{\mathsf{ReLU}}_{a,b}$ algorithm on the offset ReLU function $\mathsf{AReLU}_r(x)$ in Algorithm 3, and distributes the output (k_0, k_1) to the client and the server, respectively.

Correctness of offline protocols. We provide a correctness proof for the offline protocols.

- For the linear protocol, at the end of the procedure, the two parties hold $[Wr]_0, [Wr]_1$, and we prove that these are the shares of Wr as follows:

$$\begin{aligned} [Wr]_0 + [Wr]_1 &= (W-b)r - [ab]_0 + br' - [ab]_1 \mod N \\ &= (W-b)r + b(r+a) - ab \mod N \\ &= Wr \mod N \end{aligned} \tag{1}$$

- For the non-linear protocol, the correctness directly follows the correctness of the FSS-based ReLU construction in Section III-B.

C. Online Protocols

Figure 3 shows the online protocols. At the beginning of the online phase, the server and the client hold the pre-computed data, i.e., the shares of Wr for the linear layers and the FSS keys (k_0, k_1) for non-linear functions. We next show how to use these data to improve the efficiency of the online phase.

Linear layer. Recall that $Wx = W(x-r) + Wr \bmod N$ for each layer, where W is the model parameter held by the server, and x and Wr are secret-shared between the server and the client. Therefore, the two parties can obtain $Wx \bmod N$ in the sharing form, once $x - r \bmod N$ is available for the server. We give the detailed protocol below. Notice that this solution is also used in [18], but our protocol allows for more efficient implementations since it works over rings, instead of modulo a large prime [18]. Specifically,

- The client first sends $[x]_0 - r \mod N$ to the server and sets $[y]_0 = [Wr]_0$ where $[Wr]_0$ is pre-computed during the offline phase.
- The server adds $[x]_1$ to the received $[x]_0 - r$ to obtain $x - r \mod N$ and computes $[y]_1 = [Wr]_1 + W(x-r) \mod N$.

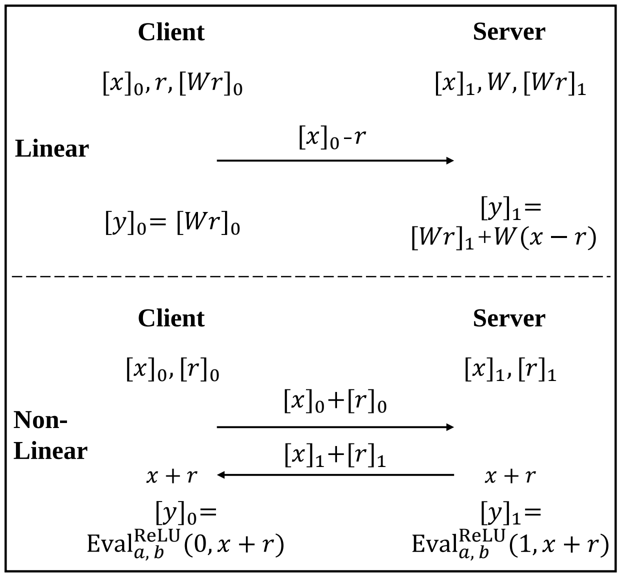


Fig. 3: `FastSecNet`'s online phase

This ensures that the client and the server hold an additive secret sharing of Wx.

Non-linear layer. We mainly focus on the evaluation of ReLU. At the beginning of the procedure, the two parties hold the FSS keys (k_0, k_1) on the function $\mathsf{ReLU}_r(x)$, where r is the random value selected in the offline phase. To adapt to the semantics of FSS evaluation, the client and the server first reconstruct $x + r \mod N$, which is a random mask of the private x.

- The client and the server exchange $[x]_0 + [r]_0 \mod N$ and $[x]_1 + [r]_1 \mod N$ with each other to obtain $x + r \mod N$. After that, each party takes $x + r \mod N$ as input to $\mathsf{Eval}^{\mathsf{ReLU}}_{a,b}$ on $\mathsf{ReLU}_r(x)$. At the end of this step, the client holds $[y]_0$ and the server holds $[y]_1$, i.e., the shares of $\mathsf{ReLU}(x)$.

Correctness of online protocols. We provide a correctness proof for the online protocols.

- For the linear protocol, at the end of the procedure, the two parties hold $[y]_0, [y]_1$, and we prove that these are the shares of Wx as follows

$$\begin{aligned}[y]_0 + [y]_1 &= [Wr]_0 + [Wr]_1 + W(x-r) \mod N \\ &= Wr + W(x-r) \mod N \\ &= Wx \mod N.\end{aligned} \tag{2}$$

- For the non-linear protocol, at the end of the procedure, the two parties hold $[y]_0, [y]_1$, and we prove that these are the shares of $\mathsf{ReLU}(x)$ as follows:

$$\begin{aligned}[y]_0 + [y]_1 &= \mathsf{Eval}^{\mathsf{ReLU}}_{a,b}(0, x-r) + \mathsf{Eval}^{\mathsf{ReLU}}_{a,b}(1, x-r) \mod N \\ &= [\mathsf{ReLU}_r(x+r)]_0 + [\mathsf{ReLU}_r(x+r)]_1 \mod N \\ &= [\mathsf{ReLU}(x)]_0 + [\mathsf{ReLU}(x)]_1 \mod N \\ &= \mathsf{ReLU}(x) \mod N\end{aligned} \tag{3}$$

D. Security Analysis

Theorem 3. *The scheme in Figures 2 and 3 is a private inference scheme against honest-but-curious adversaries, assuming the existence of PRG, FSS and secure Beaver's multiplication protocols.*

Proof. We provide a hybrid argument proof in Appendix C. □

V. Evaluation

A. Experimental Setup

Implementation. `FastSecNet` is implemented in C++ and we use the EzPC framework[3] that can convert unmodified TensorFlow code (e.g., ONNX models in our implementation) to the designed 2PC protocols. The FSS-based non-linear functions are implemented based on the FSS library[4]. We improve it using the generation and evaluation algorithms from [28]. All the experiments are executed on a server with Intel(R) 562 Xeon(R) CPU E5-2620v4 (2.10 GHz) and 16 GB of RAM running the Ubuntu 18.4 system. We simulate a local-area network (LAN) environment, where the network bandwidth is 1GBps and the network latency is 0.1ms. Like [27], [18], the secret sharing works over the 32-bit integer ring, i.e., $\mathbb{Z}_{2^{32}}$, unless otherwise stated.

Datasets and models. We evaluate `FastSecNet` on several standard datasets and representative convolutional neural networks developed for image classification. Each of these networks can be represented as a composition of a collection of linear and non-linear layers, including convolution, fully-connected layers, batch normalization, ReLU and Maxpool.

- *MNIST* is a dataset for handwritten digit recognition. The training set has 60,000 images and the test set has 10,000 images in 10 classes. Each sample is a single-channel (i.e., grayscale) 28×28 image. Our experiments use LeNet [46] and a 4-layer CNN architecture (CNN-4 for short) specified in MiniONN [15].
- *CIFAR10* is a dataset with 10 classes. It contains 50,000 training samples and 10,000 test samples. Each sample is a three-channel (i.e., RGB) 32×32 image. Our experiments use VGG16 [47] and a 7-layer CNN architecture (CNN-7 for short) specified in MiniONN [15].
- *CIFAR100* contains the same number of training and test images as CIFAR10, but divides them up into 100 classes. Our experiments use ResNet32 [3] and VGG16 [47].
- *ImageNet* is a large-scale visual recognition dataset with more than 1,000,000 training images in 1,000 classes, and each example is a 224×224 RGB image. We use three ImageNet-scale models: SqueezeNet [48], ResNet50 [3], and DenseNet121 [49].

B. Model Accuracy

According to Section II-C, one important requirement for private inference is that the introduced cryptographic protocols cannot undermine the model performance. To validate this,

[3]https://github.com/mpc-msri/EzPC
[4]https://github.com/frankw2/libfss

we measure the inference accuracy of different models and datasets in plaintext (baseline) and with our solution, as shown in Table II. It does not report the ImageNet's accuracy[5], since we directly use the converted model for 2PC from CrypTFlow2 [25]. It can be observed that `FastSecNet` results in slight accuracy loss compared to the baseline model. This is because our approach does not introduce lossy model compression [13] or polynomial approximation to activation functions [24]. The possible accuracy loss comes from the fixed-point arithmetic and the reformulated ReLU.

TABLE II: Accuracy comparison with the plaintext baseline

Dataset	Model	Plaintext	Ours
MNIST	LeNet	99.30	99.28
	CNNC-4	99.08	99.10
CIFAR10	VGG16	87.45	87.40
	CNN-7	81.61	80.55
CIFAR100	ResNet32	68.89	68.17
	VGG16	71.80	71.13

C. Comparison with Prior Works

Comparison with 2PC private inference methods. We compare the performance of `FastSecNet` against recent 2PC private inference schemes, including MiniONN [15], Chameleon [16], Gazelle [17], EzPC [51], Delphi [18]. Note that the results of these works are directly obtained from the original papers. Table III shows the efficiency improvement of `FastSecNet` over prior works on different datasets and models. For MNIST, we observe that `FastSecNet` is 2-14$\times$ and 5-88$\times$ faster in terms of offline and online execution time, respectively, while requiring 34-984$\times$ less online communication and comparable offline communication. Similarly for CIFAR10 and CIFAR100, `FastSecNet` is also significantly better than the prior schemes in terms of online, offline and overall costs. Specifically, compared with Delphi [18], the state-of-the-art 2-party inference method in the pre-processing setting, `FastSecNet` requires up to 14$\times$ less online runtime and 12$\times$ less total runtime, while it is up to 18$\times$ and 12$\times$ more communication efficient for the online cost and the total cost, respectively.

Comparison with GPU-accelerated private inference methods. In Table IV, we compare `FastSecNet` with GForce [50], a private inference work using GPU acceleration for both linear and non-linear layers, on CIFAR10 and CIFAR100. We observe that `FastSecNet` is 137–148$\times$ faster for the total runtime. We also reduce the total communication by about 22$\times$ on two datasets. Next, we compare the online runtime and communication of `FastSecNet` with GForce. As shown in Table IV, the online phase takes small runtime for both methods and `FastSecNet` is slightly slower than GForce because of its GPU acceleration, whereas our method requires about 3$\times$ less communication compared to GForce. Also note that compared with GForce, our method requires fewer communication rounds, which can be advantageous in resource-constrained scenarios.

TABLE III: Comparison with prior solutions over different datasets and models.

Dataset (Model)	Methods	Runtime (Sec)			Communication (MB)		
		Offline	Online	Total	Offline	Online	Total
MNIST (CNN-4)	MiniONN	3.58	5.74	9.32	20.90	636.60	657.50
	EzPC	-	5.10	5.10	-	501.00	501.00
	Gazelle	0.48	0.33	0.81	47.50	22.50	70.00
	Ours	**0.25**	**0.07**	**0.31**	**40.19**	**0.65**	**40.84**
CIFAR10 (CNN-7)	MiniONN	472.00	72.00	544.00	3045.38	6225.92	9271.30
	Chameleon	22.97	29.70	52.67	1239.04	1474.56	2713.60
	Gazelle	9.34	3.56	12.90	939.93	295.94	1235.87
	Delphi	129.00	9.00	138.00	4915.00	163.00	5079.00
	Ours	**2.20**	**0.63**	**2.83**	**489.96**	**8.70**	**498.66**
CIFAR100 (ResNet32)	Delphi	220.00	18.00	238.00	8908.00	286.00	9194.00
	Ours	**3.56**	**1.34**	**4.89**	**688.23**	**22.01**	**710.24**

TABLE IV: Comparison with GPU-accelerated GForce [50].

Dataset (Model)	Methods	Runtime (Sec)			Communication (MB)		
		Offline	Online	Total	Offline	Online	Total
CIFAR10 (VGG16)	GForce	900.00	0.35	900.35	19195.00	50.46	19245.46
	Ours	**4.50**	**1.58**	**6.08**	**828.25**	**15.59**	**843.84**
CIFAR100 (VGG16)	GForce	849.56	0.35	849.92	19197	50.47	19247.47
	Ours	**4.55**	**1.61**	**6.16**	**828.60**	**15.94**	**844.54**

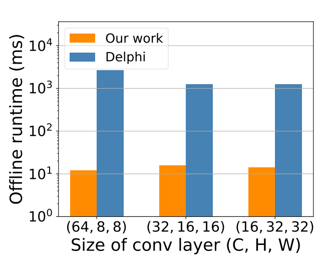
Comparison with ImageNet-scale evaluation. To demonstrate the scalability of `FastSecNet`, we evaluate it on three ImageNet-scale models, and compare it with CrypTFlow2 [25]. We follow the setting of CrypTFlow2, e.g., 37-bit secret sharing for ResNet50 and 32-bit secret sharing for SqueezeNet and DenseNet121. Table V shows the computation and communication comparison results. Specifically, `FastSecNet` requires 4-15$\times$ less online runtime, and reduces online communication by at least one order of magnitude on the three models. As mentioned above, our method may cause accuracy loss in the these large-scale models. A possible solution is to utilize precise ReLU and truncation protocols, which can designed from our approximate ReLU protocol. While the costs are increased, we believe that our method will still have a significant online advantage.

D. Microbenchmarks

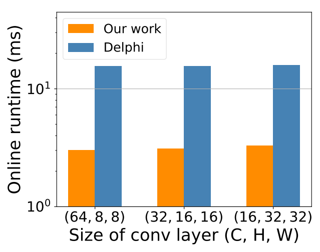
To further demonstrate the superiority of `FastSecNet`, we evaluate the performance of linear and non-linear layers.

Online overhead of linear and non-linear operations. In Table VI, we report the online performance of linear layers and non-linear layers in `FastSecNet` over four datasets and nine models. For the three small-scale datasets (i.e., MNIST, CIFAR10 and CIFAR100), our online runtime does not exceed 2 seconds, while the communication overhead is up to 22MB. Even for ImageNet, the online latency and communication do not exceed 50 seconds and 600MB, respectively. Besides, we observe that more than 90% of runtime during the online phase is occupied by the secure non-linear operations. This is because our linear protocol only needs to perform matrix multiplication in plaintext. We also observe that non-linear

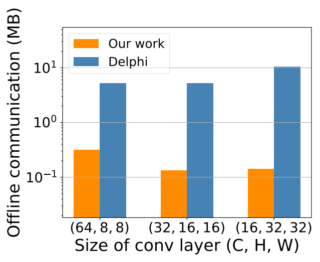
[5]Our protocol may cause a small accuracy loss compared to [25] due to the truncation we use [40] and the reformulated ReLU.



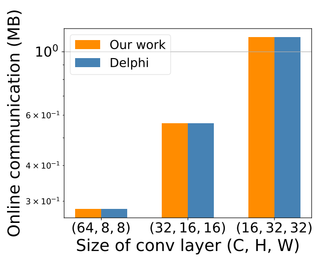
(a) Offline runtime.



(b) Online runtime.

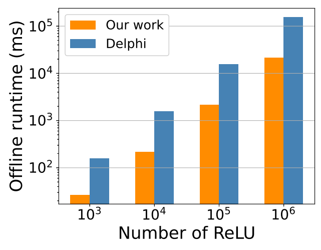


(c) Offline communication cost.

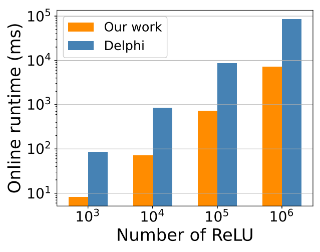


(d) Online communication cost.

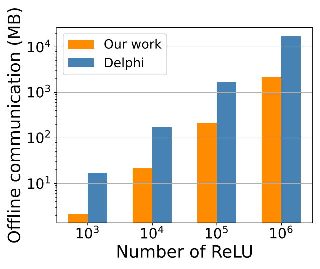
Fig. 4: Comparison with Delphi on runtime and communication cost of ResNet-32 convolutions.



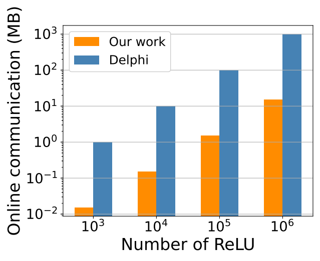
(a) Offline runtime.



(b) Online runtime.



(c) Offline communication cost.



(d) Online communication cost.

Fig. 5: Comparison with Delphi on runtime and communication cost of ReLU.

TABLE V: Comparison with CrypTFlow2 [25].

Model	Methods	Runtime (Sec)			Communication (GB)		
		Offline	Online	Total	Offline	Online	Total
SqueezeNet	CrypTFlow2	-	59.20	59.20	-	5.27	5.27
	Ours	**46.20**	**13.42**	**59.62**	**10.20**	**0.06**	**10.26**
ResNet50	CrypTFlow2	-	545.80	545.80	-	32.43	32.43
	Ours	**116.69**	**35.75**	**152.44**	**26.39**	**0.26**	**26.65**
DenseNet121	CrypTFlow2	-	463.20	463.20	-	35.56	35.56
	Ours	**166.51**	**48.46**	**214.97**	**36.20**	**0.51**	**36.71**

TABLE VI: Online performance of linear layers and non-linear layers in `FastSecNet`.

(a) MNIST

Models	Runtime (Sec)			Communication (MB)		
	Linear	Non-linear	Total	Linear	Non-linear	Total
CNN-4	<0.001	0.065	0.065	0.508	0.136	0.646
LeNet	0.011	0.124	0.135	4.540	0.152	4.844

(b) CIFAR10

Models	Runtime (Sec)			Communication (MB)		
	Linear	Non-linear	Total	Linear	Non-linear	Total
CNN-7	0.003	0.648	0.651	7.383	1.320	8.703
VGG16	0.433	1.146	1.579	12.766	2.824	15.590

(c) CIFAR100

Models	Runtime (Sec)			Communication (MB)		
	Linear	Non-linear	Total	Linear	Non-linear	Total
ResNet32	0.076	1.263	1.339	19.697	2.313	22.010
VGG16	0.438	1.172	1.610	13.117	2.824	15.941

(d) ImageNet

Models	Runtime (Sec)			Communication (MB)		
	Linear	Non-linear	Total	Linear	Non-linear	Total
SqueezeNet	0.28	13.13	13.42	26.10	36.10	62.20
ResNet50	2.65	33.09	35.75	182.06	80.96	263.03
DenseNet121	2.23	46.21	48.46	390.47	127.19	517.66

layers require less communication overhead (about 3-5$\times$) compared to the evaluation of linear layers.

Comparison with Delphi on linear operations. We use Delphi [18] as the comparison baseline. We focus on the performance of the convolution operation since this operation dominates the overhead of the linear layer evaluation. In Figure 4, we compare the cost of convolutions used in ResNet32 with Delphi. The main takeaway is that our online runtime is over 5$\times$ smaller than Delphi, while our online communication is exactly the same as Delphi. Moreover, our offline runtime and communication are significantly better (about 1-3 orders of magnitude) than Delphi, since our protocols are implemented via lightweight secret sharing and avoid heavy cryptographic operations such as OT and HE.

Comparison with Delphi on non-linear operations. To quantify the advantage of our protocols for non-linear functions, we compare the runtime and communication overhead on ReLU (i.e., the primary non-linear layer) with Delphi. Delphi deploys the online-optimized GC for non-linear computation, which generates and transmits GCs as well as exchanges labels with OT in the offline phase. Figure 5 shows the improvement of our ReLU protocols over GCs in both offline and online performance. For 10^6-element inputs, `FastSecNet` outperforms Delphi by 64$\times$ in the online communication and by 11$\times$ in the online runtime. It also improves at least 8$\times$ and 7$\times$ in the offline communication and computation costs, respectively.

VI. Related Works

Prior works about privacy-preserving machine learning generally fall into two categories: private training and private inference. We omit the discussion of private training since it is not the focus of this paper. Interested readers can refer to the SoK paper [52] for more details about private training.

Recently, some works have designed customized protocols for performing private inference. These protocols improve the computation and communication overheads by utilizing advanced cryptographic techniques along with various model architecture modifications and optimizations. In the following, we briefly discuss these works according to the underlying techniques, as well as applications of function secret sharing.

A. Private Inference with Homomorphic Encryption

CryptoNets [10] is the first solution for private inference by utilizing optimized leveled HE (LHE) schemes. To improve efficiency, the authors presented square function approximations of ReLU and substituted Maxpool layers by Avgpool layers, which are LHE-friendly. After that, Chou et al. [53] proposed Faster CryptoNets to improve the above method using pruning and quantization techniques [54] along with optimized activation function approximations. As a result, these optimizations reduce circuit depths and the number of operations performed in the homomorphic ciphertext. One key limitation of these schemes is that the approximated activation functions cause serious accuracy loss. To mitigate such problem, CryptoDL [55] used Taylor and Chebyshev polynomial approximation [56] to control the error within a certain range. More recently, Lee et al. [57] proposed the composition of minimax approximate polynomials of small degrees, which can support deeper neural networks with negligible approximation errors. Besides, several works [58], [11], [12] designed general-purpose compilers that can automatically translate the high-level representation of deep learning code into optimized homomorphically encrypted code, to facilitate the execution efficiency of private inference. However, due to the inherent high complexity limitations of HE, the resulting protocols still cause prohibitively high computation overhead even over networks that are much smaller than recent advanced models.

B. Private Inference with Secure 2-party Computation

Several works explored private inference using secure 2-party computation techniques like GC or SS because of their high evaluation efficiency. Rouhani et al. [59] proposed an optimized private inference framework called DeepSecure, where a variation of GC is designed to support pre-processing for computation tasks, along with the pruned model to reduce the number of activations. Similar to DeepSecure, XONN [13] is a GC-based private inference framework only for binary neural networks (BNN), where the weights and activations are restricted in binary values (i.e., 1 and -1). Concretely, the authors substituted the costly matrix multiplications with simple XNOR operations and deigned customized conditional addition protocols. More recently, Samragh et al. [60] proposed a more efficient BNN-based private inference protocol, in which GCs are used for non-linear layers. They adopted oblivious transfer (OT) for linear layers to reduce the communication cost. Besides, Ball et al. [14] constructed private inference protocols for general neural networks based on optimized GC for arithmetic circuits [61]. However, all the above methods suffer from a large communication overhead due to GC's inherent limitation that the communication of each binary gate is linear with the magnitude of the security parameter.

C. Private Inference with Hybrid Techniques

To trade off the communication and computation overheads, several recent works integrated multiple advanced techniques for private inference. Liu et al. [15] presented an online-efficient private inference scheme called MiniONN, which utilizes HE, SS and GC in the pre-processing setting. To reduce the overhead of pre-computation during the offline phase, MiniONN uses the HE-based dot-product triplet generation with the batch processing technique similar to SPDZ [62]. After that, Chameleon [16] was proposed as an optimized PRG-based pre-computation techniques for private inference. It is based on the ABY framework [29], which substantially reduces the offline communication overhead. Despite various optimizations, these works cannot address the fundamental problem, i.e., the inefficiency of cryptographic schemes for matrix multiplication and non-linear evaluation.

To bridge this gap, Gazelle [17] presented an optimized homomorphic linear algebra kernel which provides fast algorithms for matrix-vector multiplication and convolution operations. Compared with Chameleon and MiniONN, Gazelle achieves up to $30\times$ and $80\times$ reductions in the runtime and communication, respectively. Zhang et al. [19] presented GALA to further optimize the computation overhead of HE-based linear evaluations through minimizing the rotation operation (an expensive operation in HE). On the basis of Gazelle, Delphi [18] brought an unprecedented online overhead by moving the heavy linear layer operations over LHE ciphertexts to the offline phase. The main insight is that the model weights on the server are fixed and known before clients' inputs are available, and thus the linear layer evaluation can be pre-processed based on the model weights. However, as discussed in Section I, the high online overhead of non-linear layers still needs to be improved.

Although recent works have explored different solutions, there are still many limitations. For example, CrypTFlow2 [25] presented new protocols for non-linear layer evaluation. It recursively reduces the original evaluation to compute two sub-problem instances over smaller field, and employs the lookup table-based OT [63] to reduce communication. Patra et al. [26] presented ABY2.0 in which a multi-input AND protocol is designed to reduce online communication and computation overheads. However, these solutions are either at the cost of increased communication rounds and require special secret-sharing primitives, or are hard to be extended to the offline-online model. Besides, Gforce [50] utilized GPU acceleration on Delphi's non-linear layer evaluation, but the communication overhead is still not well addressed. Therefore, we aim to improve the online overhead of private inference in the pre-processing model.

D. Model Architecture Optimizations for Private Inference

Orthogonal to the design of efficient cryptographic protocols, recent works began to focus on designing architecture-optimized neural networks to improve efficiency. Given the costly evaluation for ReLU, Delphi [18] replaced the selected ReLU layers with quadratic functions, where a neural architecture search (NAS) method is employed to determine which ReLU layers to replace. As a result, the online communication and computation costs are 192$\times$ and 10,000$\times$ smaller than GC-based ReLU, respectively. After that, CryptoNAS [23] designed a new NAS strategy on a formally defined ReLU budget that balances the ReLU counts and the model accuracy at the same time. To further trade-off the efficiency and accuracy, Lou et al. [64] presented SAFENet that gives a fine-grained activation approximation scheme. Concretely, the method deploys a channel-wise replacement method with multiple-degree polynomials and uses layer-wise mixed precision. Besides, DeepReDuce [24] designed a simple ReLU reduction strategy that exploits the ReLU' heterogeneity to remove the less-critical ReLUs and preserve the most-critical ReLUs. Unfortunately, these solutions based on architectural optimization come at the cost of the prediction accuracy. In contrast, `FastSecNet` requires no network modifications, and hence can guarantee negligible accuracy loss. On the other hand, Dalskov et al. [65] proposed a private inference protocol on quantized neural networks. Their work first identifies crypto-friendly models and then exploits general-purpose secure computation techniques [66] for private evaluation. This quantized method significantly reduces computation and communication costs. Despite the advantage caused by quantized models, our customized multiplication and comparison protocols still outperform these general protocols. More precisely, 1) for the multiplication protocol, our online phase only communicates one ring element, but [65] consumes communication of two elements using the standard Beaver's multiplication protocol; 2) for the non-linear layer such as ReLU, our protocol achieves optimal online communication i.e., $2n$-bit communication within a single communication round, where n is the bitlength of secret shares. However, [65] utilizes the costly method of extracting the most significant bit (MSB), which still requires communication of $12n-16$ bits within $2+\log(n-1)$ communication rounds.

E. Applications of Function Secret Sharing

As a powerful cryptographic tool, FSS [31] enables communication-efficient secure computation in the preprocessing setting. Boyle et al. [32] gave a general framework for the FSS-based secure computation, and implement several useful non-linear functions with optimal online communication, such as equality tests, integer comparison, bit-decomposition, and spline functions. After that, [28] improved the efficiency of secure comparison and spline operations in [32]. Moreover, [28] also presented novel FSS-based schemes for arithmetic and logical shift gates, which can be used to fixed-point arithmetic. Recently, AriaNN [27] proposes a low-interaction privacy-preserving framework for private neural network training and inference based on FSS. The main contribution of this work is an FSS-based comparison protocol with the application on ReLU. As shown in Section III-B, compared with the state-of-the-art protocols in [28], [27], our solution achieves better communication and computation performance.

Besides, FSS has also been used in a variety of applications, such as private queries, oblivious reading and writing, and correlated randomness generation. For example, in Splinter [67], the clients can perform private queries on a public dataset by utilizing FSS. In Floram [30], the authors exploited FSS to achieve private reading and writing in the distributed Oblivious RAM (ORAM) setting. In addition, Dory [68] achieved private keyword search for encrypted files with FSS while leaking zero information about search access patterns. PIRSONA [69] presented a digital content delivery system, which realizes collaborative-filtering recommendations atop an FSS-based private information retrieval (PIR) scheme. Moreover, Waldo [70] presented a private time-series database using FSS, which enables multi-predicate filtering while hiding the filter values and search access patterns. Recent works [71], [72], [73] built upon the foundational FSS protocol and provided efficient pseudo-random correlation generators such as vector oblivious linear evaluation or oblivious transfer extensions.

VII. Conclusion and Future Work

We propose `FastSecNet`, an efficient cryptographic method for private inference. Specifically, we carefully design new cryptographic protocols to efficiently evaluate non-linear layers via improved function secret sharing. Further, we improve the performance of linear layers by utilizing PRG-based secret sharing techniques. Experimental results present that `FastSecNet` outperforms prior state-of-the-art works at least one order of magnitude for both online computation and communication overheads without sacrificing the model accuracy.

It is an interesting future work to achieve 2PC-based offline phase for FSS and extend our `FastSecNet` framework into a full two-party setting without the third party. As shown in prior works [28], some general secure computation protocols such as GCs [74] and GMW [43] or FSS-specific techniques [30], [73] can be utilized in the implementation. Besides, the security of `FastSecNet` will be improved to defeat more powerful adversaries in the malicious setting by using verifiable computation techniques [75], [66], [6], [76].

Acknowledgment

This work is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101360001, the National Natural Science Foundation of China under Grants 62020106013 and 61972454, the Sichuan Science and Technology Program under Grants 2020JDTD0007, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027 and Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121-0006.

References

[1] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[2] M. Bakator and D. Radosav, "Deep learning and medical diagnosis: A review of literature," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 47, 2018.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE CVPR*, 2016, pp. 770–778.

[4] G. Cloud, "Introduction to ai platform," https://cloud.google.com/ai-platform/docs/technical-overview.

[5] M. Azure, "Microsoft machine learning studios," https://studio.azureml.net/.

[6] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *Proceedings of USENIX Security*, 2021, pp. 2201–2218.

[7] J. W. Lai and K. H. Cheong, "Chaotic switching for quantum coin parrondo's games with application to encryption," *Physical Review Research*, vol. 3, no. 2, p. L022019, 2021.

[8] J.-B. Truong, P. Maini, R. J. Walls, and N. Papernot, "Data-free model extraction," in *Proceedings of CVPR*, 2021, pp. 4771–4780.

[9] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Proceedings of CRYPTO*, 2020, pp. 189–218.

[10] 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 *Proceedings of the ICML*, 2016, pp. 201–210.

[11] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of PLDI*, 2020, pp. 546–561.

[12] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[13] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "Xonn: Xnor-based oblivious deep neural network inference," in *Proceedings of USENIX Security*, 2019, pp. 1501–1518.

[14] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical," *Cryptology ePrint Archive*, 2019.

[15] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of ACM CCS*, 2017, pp. 619–631.

[16] 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 ACM AsiaCCS*, 2018, pp. 707–721.

[17] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proceedings of USENIX Security*, 2018, pp. 1651–1669.

[18] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proceedings of USENIX Security*, 2020, pp. 2505–2522.

[19] Q. Zhang, C. Xin, and H. Wu, "Gala: Greedy computation for linear algebra in privacy-preserved neural networks," in *Proceedings of NDSS*, 2021, pp. 1–16.

[20] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of ACM STOC*, 2009, pp. 169–178.

[21] A. C. Yao, "Protocols for secure computations," in *Proceedings of IEEE FOCS*, 1982, pp. 160–164.

[22] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proceedings of CRYPTO*, 1991, pp. 420–432.

[23] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a relu budget," in *Proceedings of NeurIPS*, 2020, pp. 16 961–16 971.

[24] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "Deepreduce: Relu reduction for fast private inference," in *Proceedings of ICML*, 2021, pp. 4839–4849.

[25] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of ACM CCS*, 2020, pp. 325–342.

[26] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2. 0: Improved mixed-protocol secure two-party computation," in *Proceedings of USENIX Security*, 2021, pp. 2165–2182.

[27] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 291–316, 2022.

[28] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Proceedings of EUROCRYPT*, 2021, pp. 871–900.

[29] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *Proceedings of NDSS*, 2015, pp. 1–15.

[30] J. Doerner and A. Shelat, "Scaling oram for secure computation," in *Proceedings of ACM CCS*, 2017, pp. 523–535.

[31] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Proceedings of EUROCRYPT*, 2015, pp. 337–367.

[32] ——, "Secure computation with preprocessing via function secret sharing," in *Proceedings of TCC*, 2019, pp. 341–371.

[33] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *Proceedings of IEEE S&P*, 2020, pp. 336–353.

[34] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, pp. 277–346, 2017.

[35] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proceedings of USENIX Security*, 2016, pp. 601–618.

[36] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of ACM CCS*, 2015, pp. 1322–1333.

[37] Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang, "Neural network inversion in adversarial setting via background knowledge alignment," in *Proceedings of ACM CCS*, 2019, pp. 225–240.

[38] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of ACM CCS*, 2016, pp. 308–318.

[39] M. Nasr, S. Song, A. Thakurta, N. Papernot, and N. Carlini, "Adversary instantiation: Lower bounds for differentially private machine learning," in *Proceedings of IEEE S&P*, 2021, pp. 866–882.

[40] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE S&P*, 2017, pp. 19–38.

[41] Y. Khazbak, T. Tan, and G. Cao, "Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning," in *Proceedings of IEEE ICCCN*, 2020, pp. 1–9.

[42] A. C. Yao, "Theory and application of trapdoor functions," in *Proceedings of FOCS*, 1982, pp. 80–91.

[43] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[44] H. Chen, H. Li, Y. Wang, M. Hao, G. Xu, and T. Zhang, "Privdt: An efficient two-party cryptographic framework for vertical decision trees," *IEEE Transactions on Information Forensics and Security*, 2022.

[45] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic relus for private deep learning," in *Proceedings of NeurIPS*, 2021, pp. 2241–2252.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[48] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50$\times$ fewer parameters and $<$ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[49] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of IEEE CVPR*, 2017, pp. 4700–4708.

[50] L. K. Ng and S. S. Chow, "Gforce: Gpu-friendly oblivious and rapid neural network inference," in *Proceedings of USENIX Security*, 2021, pp. 2147–2164.

[51] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: programmable and efficient secure two-party computation for machine learning," in *Proceedings of IEEE EuroS&P*, 2019, pp. 496–511.

[52] J. Cabrero-Holgueras and S. Pastrana, "Sok: Privacy-preserving computation techniques for deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 139–162, 2021.

[53] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.
[54] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of ICCV*, 2017, pp. 5058–5066.
[55] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.
[56] K. Atkinson and W. Han, *Theoretical numerical analysis*, 2005, vol. 39.
[57] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, "Precise approximation of convolutional neuralnetworks for homomorphically encrypted data," *arXiv:2105.10879*, 2021.
[58] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of PLDI*, 2019, pp. 142–156.
[59] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of DAC*, 2018, pp. 1–6.
[60] M. Samragh, S. Hussain, X. Zhang, K. Huang, and F. Koushanfar, "On the application of binary neural networks in oblivious inference," in *Proceedings of CVPR workshop*, 2021, pp. 4630–4639.
[61] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proceedings of ACM CCS*, 2015, pp. 591–602.
[62] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of CRYPTO*, 2012, pp. 643–662.
[63] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Proceedings of NDSS*, 2018, pp. 1–15.
[64] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "Safenet: A secure, accurate and fast neural network inference," in *Proceedings of ICLR*, 2021, pp. 1–13.
[65] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 355–375, 2020.
[66] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of ACM CCS*, 2020, pp. 1575–1590.
[67] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical private queries on public data." in *Proceedings of NSDI*, 2017, pp. 299–313.
[68] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica, "Dory: An encrypted search system with distributed trust," in *Proceedings of OSDI*, 2020, pp. 1101–1119.
[69] A. Vadapalli, F. Bayatbabolghani, and R. Henry, "You may also like... privacy: Recommendation systems meet pir." *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 4, pp. 30–53, 2021.
[70] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, "Waldo: A private time-series database from function secret sharing," in *Proceedings of IEEE S&P*, 2022, pp. 2450–2468.
[71] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *Proceedings of ACM CCS*, 2018, pp. 896–912.
[72] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *Proceedings of CRYPTO*, 2019, pp. 489–518.
[73] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of ACM CCS*, 2020, pp. 1607–1626.
[74] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of IEEE FOCS*, 1986, pp. 162–167.
[75] D. Liu, C. Huang, L. Xue, J. Hou, X. Shen, W. Zhuang, R. Sun, and B. Ying, "Authenticated and prunable dictionary for blockchain-based vnf management," *IEEE Transactions on Wireless Communications*, vol. 21, no. 11, pp. 9312–9324, 2022.
[76] D. Liu, C. Huang, J. Ni, X. Lin, and X. S. Shen, "Blockchain-cloud transparent data marketing: Consortium management and fairness," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3322–3335, 2022.

Appendix

A. Proof of Theorem 1

Proof. To analyze the probability $P\{\mathsf{AReLU}_r(x+r) \neq \mathsf{ReLU}(x)\}$ for $x, r \in \mathbb{Z}_N$, we discuss the following two cases. First, this case is $r < N/2 + r$. The probability occurs only when $x \geq N/2$ is a negative number but $x + r < N$. This means AReLU fails when $x + r < N$. Because r is sampled from $\mathbb{Z}_N$, the probability $P = \frac{|x|}{N}$, where $x \geq N/2$ and $|x| = N - x$. Second, this case is $r > N/2 + r$. The probability occurs only when $x < N/2$ is a positive number but $x + r > N$. This means when $x + r$ overflows, AReLU fails. Because r is sampled from $\mathbb{Z}_N$, the probability $P = \frac{x}{N}$, where $x < N/2$. □

Algorithm 5 Hybrid distribution j: $\mathsf{Hyb}_j(\kappa, p, a, b)$

```
1: $s_p^0 \leftarrow \{0,1\}^\kappa$ chosen at random, and $t_p^0 = p$.
2: $b_p$, $r_p$ chosen at random with the constraint $b_0 + b_1 = b$ and $r_0 + r_1 = a$.
3: $CW^{(1)}, \cdots, CW^{(j)}$ chosen at random.
4: For $i \leq j$, $s_p^{(i)}$, $v_p^{(i)}$, $t_p^{(i)}$ computed honestly, as a function of $s_p^{(0)}$, $v_p^{(0)}$, $t_p^{(0)}$ and $CW^{(1)}, \cdots, CW^{(j)}$.
5: For $j$, the other party's seed $s_{1-p}^{(j)} \leftarrow \{0,1\}^\kappa$ and the element $v_{1-p}^{(j)} \leftarrow \{0,1\}^n$ are chosen at random, and $t_{1-p}^{(j)} = 1 - t_p^{(j)}$.
6: For $i > j$, the remaining values $s_p^{(i)}$, $v_p^{(i)}$, $t_p^{(i)}$, $s_{1-p}^{(i)}$, $v_{1-p}^{(i)}$, $t_{1-p}^{(i)}, CW^{(i)}$ are all computed honestly as a function of the previously chosen values.
7: The output for $p \in \{0,1\}$ is $k_p = s_p^{(0)} \| CW^{(1)} \| \cdots \| CW^{(n+1)} \| b_p \| r_p$.
```

B. Proof of Theorem 2.

Proof. We give a proof sketch to prove that each party's key k_p, $p \in \{0,1\}$, is pseudorandom in the FSS-based ReLU protocol. k_p includes the correction word $CW^{(i)}$, $i \in \{1, n+1\}$, $s_p^{(0)}$, r_p and b_p. Note that $s_p^{(0)}$, r_p and b_p are completely random to the other party. Below, we focus on the analysis of $CW^{(i)}$. We exploit the hybrid argument technique, where in each hybrid we substitute the honestly generated correction word $CW^{(i)}$ within the key to random chosen values.

Each party $p \in \{0,1\}$ begins with a random seed $s_p^{(0)}$. In each level i of key generation (from $i = 1$ to n), the parties obtain six values by applying a PRG to their seed $s_p^{(i-1)}$: two seeds s_p^L, s_p^R, two ring elements v_p^L, v_p^R and 2 control bits t_p^L, t_p^R. Due to the randomness of the seeds and the security of PRGs, it guarantees that the six resulting values are pseudorandom given the view of the other party.

The ith level correction word $CW^{(i)}$ will consist of the partial secret randomness of these 6 values: 2 control bits t_p^L, t_p^R, the element v_p^{lose} and the seed s_b^{lose} for $lose \in \{L, R\}$. $lose$ corresponds to the direction departing from the special path to a: $lose = L$ if $a[i] = 1$ and $lose = R$ if $a[i] = 0$. However, while holding $CW^{(i)}$, the other seed s_p^{keep} for $keep \neq lose$ and $keep \in \{L, R\}$ still appears random to the other party. The hybrid argument then continues in similar fashion to the next level, beginning with seeds s_p^{keep}.

For each $j \in \{0, 1, \cdots, n+1\}$, we will consider a hybrid distribution Hyb_j that is defined in Algorithm 5. Note that when $j = 0$, this hybrid distribution corresponds to the key distribution in the real world. When $j = n+1$ this yields a completely random key k_p, i.e., the simulated distribution. We claim that each pair of adjacent hybrids $j+1$ and j will be indistinguishable based on the security of PRG. This concludes the proof of Theorem 2. □

C. Proof of Theorem 3.

Proof. We analyze security considering one of the server, client and STP is compromised since the protocol is asymmetrical. Since STP receives no messages in the private inference scheme, our protocol obviously defends the corruption of STP. We below focus on the protocol's security when the server or the client is corrupted. Our security proof follows the ideal-world/real-world paradigm. We will show that the real-world distribution is computationally indistinguishable to the simulated distribution by the simulation Sim in the ideal world. Note that we use the notation W_i, x_i, r_i to denote the ith layer's weight, activations, and random masks, respectively.

Proof of indistinguishability with corrupted client. We construct Sim to generate the transcript the view of the corrupted client. Sim does not use the weights of the server's model, and hence the corrupted client learns nothing beyond the prediction result and model architecture in the real world.

- Hyb_0: This represents the real world distribution where the server uses its real model weights $W_1, W_2, \cdots, W_l$.
- Hyb_1: This hybrid involves only a syntactic change. In the output phase, the simulator sends $y - r_l$ to the client, where y is the prediction result on the client's input x. In additional, Sim uses the knowledge of the client's random tape to begin the evaluation of the ith layer with $x_i - r_i$. Because this is just a syntactic change, Hyb_1 is distributed identically to Hyb_0.
- Hyb_2: In this hybrid, Sim runs the corresponding simulator for the PRG-assisted Beaver's generation protocol to generate the multiplication triples in the offline phase. Following from the simulation security of such protocol, Hyb_2 is computationally indistinguishable from Hyb_1.
- Hyb_3: In this hybrid, for every linear layer, we use the simulator for the Beaver's multiplication procedure. Again following from the simulation security of this protocol, this hybrid is computationally indistinguishable to Hyb_2. Note that the server is no longer using the true weights W_i to evaluate the i-th layer and sends $W_i - b_i$ for a uniformly chosen value.
- Hyb_4: For the non-linear protocol, Sim runs the simulators of the FSS protocols. In addition, the server sends a uniformly random value $[x_i]_1 - [r_i]_1$. Based on the simulation security of the FSS protocols, Hyb_4 is computationally indistinguishable from Hyb_3. Finally, we note that Hyb_4 is identically distributed to the simulator's output. This completes the proof.

Proof of indistinguishability with corrupted server. We construct Sim to generate the transcript of a corrupted server. Sim does not use the client's input, and hence a corrupted server learns nothing in the real world.

- Hyb_0: This corresponds to the real world distribution where the client uses its real input x.
- Hyb_1: In this hybrid, Sim runs the corresponding simulator for the PRG-assisted Beaver's generation protocol to generate the multiplication triples in the offline phase. Following from the simulation security of this protocol, Hyb_1 is computationally indistinguishable from Hyb_0.
- Hyb_2: In this hybrid, for every linear layer, we use the simulator for Beaver's multiplication procedure. It again follows from the simulation security that this hybrid is computationally indistinguishable to Hyb_1. Note that the client is sending a uniformly chosen value, instead of $[x_i]_0 - [r_i]_0$.
- Hyb_3: For the non-linear protocol, Sim runs the simulators of the FSS protocols. In addition, the client sends a uniformly random value $[x_i]_0 - [r_i]_0$. Based on the simulation security of this protocol, Hyb_3 is computationally indistinguishable from Hyb_2. Finally, we note that Hyb_3 is identically distributed to the simulator's output. This concludes the proof.

□

D. Functionalities

We introduce the ideal functionalities in Tables VII-X that are used in `FastSecNet`.

TABLE VII: The ideal functionality $\mathcal{F}_{\text{Triple}}$

Input:
- client: $a \in \mathbb{Z}_N$.
- server: $b \in \mathbb{Z}_N$.

Output:
- client: $[c]_0 = r \in \mathbb{Z}_N$, where r is a random value.
- server: $[c]_1 = ab - r \mod N$.

TABLE VIII: The ideal functionality $\mathcal{F}_{\text{Multiply}}$

Input:
- client: $x \in \mathbb{Z}_N$.
- server: $y \in \mathbb{Z}_N$.

Output:
- client: $[z]_0 = r \in \mathbb{Z}_N$, where r is a random value.
- server: $[z]_1 = xy - r \mod N$.

TABLE IX: The ideal functionality $\mathcal{F}_{\text{ReLU}}$

Input:
- client: $[x]_0 \in \mathbb{Z}_N$.
- server: $[x]_1 \in \mathbb{Z}_N$.

Output:
- client: $[z]_0 = \mathsf{ReLU}(x) - r \mod N$, where $x = [x]_0 + [x]_1 \mod N$, and r is a random value in $\mathbb{Z}_N$.
- server: $[z]_1 = r$.

TABLE X: The ideal functionality $\mathcal{F}_{\mathsf{Maxpool}}$

Input:
- client: $[x_1]_0, [x_2]_0, \cdots, [x_d]_0 \in \mathbb{Z}_N$.
- server: $[x_1]_1, [x_2]_1, \cdots, [x_d]_1 \in \mathbb{Z}_N$.

Output:
- client: $[z]_0 = \mathsf{Max}(x_1, x_2, \cdots, x_d) - r \mod N$, where $x_i = [x_i]_0 + [x_i]_1 \mod N$, $i \in [1, d]$, and r is random in $\mathbb{Z}_N$.
- server: $[z]_1 = r$.