Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference

Zhichong Huang
Alibaba Group

Wen-jie Lu
Alibaba Group

Cheng Hong
Alibaba Group

Jiansheng Ding
Alibaba Group

Abstract

Secure two-party neural network inference (2PC-NN) can offer privacy protection for both the client and the server and is a promising technique in the machine-learning-as-a-service setting. However, the large overhead of the current 2PC-NN inference systems is still being a headache, especially when applied to deep neural networks such as ResNet50. In this work, we present Cheetah, a new 2PC-NN inference system that is faster and more communication-efficient than state-of-the-arts. The main contributions of Cheetah are two-fold: the first part includes carefully designed homomorphic encryption-based protocols that can evaluate the linear layers (namely convolution, batch normalization, and fully-connection) without any expensive rotation operation. The second part includes several lean and communication-efficient primitives for the non-linear functions (e.g., ReLU and truncation). Using Cheetah, we present intensive benchmarks over several large-scale deep neural networks. Take ResNet50 for an example, an end-to-end execution of Cheetah under a WAN setting costs less than 2.5 minutes and 2.3 gigabytes of communication, which outperforms CrypTFlow2 (ACM CCS 2020) by about 5.6× and 12.9×, respectively.

1 Introduction

To alleviate some of the privacy concerns associated with the ubiquitous deployment of deep learning technologies, many works [2, 6, 10, 17, 21, 34, 39, 41] in the past few years have introduced cryptographic frameworks based on secure two-party computation (2PC) [5] to enable privacy-preserving (deep) neural network inference. The problem they are trying to solve could be described as follows: A server holds a valuable pre-trained neural network model $F$. The server is willing to provide $F$ as a service but does not want to give out $F$ directly. A client wants to use $F$ to predict on her data $x$, but she considers $x$ as private information and does not want to reveal it to the server. 2PC protocols could solve this dilemma and fulfill both parties’ requirements: The participant(s) could learn the inference result $F(x)$ but nothing else beyond what can be derived from $F(x)$. A possible application is privacy-preserving face recognition, where the server could identify criminals from photos without viewing the photo contents.

Unfortunately, there still exist performance gaps between the current 2PC-NN inference systems and real-world applications. Because of large computation and communication overhead, those systems have been limited to small datasets (such as MNIST and CIFAR) or simple models (e.g. with a few hundreds of parameters). Recently the system CrypTFlow2 [46] has made considerable improvements, and demonstrate, for the first time, the ability to perform 2PC-NN inference at the scale of ImageNet. Despite their advances, there remains considerable overhead: For instance, using CrypTFlow2, the server and the client might need more than 15 minutes to run and exchange more than 30 gigabytes of messages to perform one secure inference on ResNet50.

Our contribution. In this paper, we present Cheetah, a secure and fast two-party inference system for deep neural networks (DNN). Cheetah achieves its performance via a careful co-design of DNN, lattice-based homomorphic encryption, oblivious transfer, and secret-sharing. Cheetah contributes a set of novel cryptographic protocols for the most common linear operations and non-linear operations of DNNs. Cheetah can perform secure inference on large models, e.g. ResNet, and DenseNet [28], with significantly smaller computation and communication overheads than the state-of-the-art 2PC-NN inference systems. For instance, using Cheetah, the server and the client can perform one secure inference on ResNet50 within 2.5 minutes, exchanging less than 2.3 gigabytes of messages under a wide area network setting, which improves over CrypTFlow2 by about 5.6× and 12.9×, respectively.

Potential Real World Applications. In [57], the researchers train a DenseNet121 to predict lung diseases from chest X-ray images. Also, [24] uses DNNs for diagnosing Diabetic Retinopathy (one of the major causes of blindness) from retinal images. A such prediction can be done securely within 3
minutes with Cheetah.

1.1 Our Techniques

The layers of modern DNNs consist of alternating linear and non-linear operations. To design efficient 2PC-NN inference systems, multiple types of cryptographic primitives are commonly used together. For instance, DELPHI [41] and CrypTFlow2 [46] leverage homomorphic encryption (HE) to evaluate the linear functions of DNN and turn to garbled circuits (GC) or oblivious transfer (OT) to compute the non-linear functions of DNN. Cheetah is also a hybrid system with novel insights on designing the base protocols and on the way how to coordinate different types of cryptographic primitives. We describe a high-level overview of our main techniques from three aspects.

1.1.1 Achieving the Best in Two Worlds

Most of the existing 2PC-NN systems [39, 41, 43, 46] suggest using the additive secret sharing technique to switch back-and-forth between different types of cryptographic primitives. There remains a question that which domain should be used for the additive sharing, a prime field $\mathbb{Z}_p$ or the ring $\mathbb{Z}_{2^l}$? As shown in [46], for the non-linear functions of DNNs, OT-based protocols on the ring $\mathbb{Z}_{2^l}$ can perform $40\% - 60\%$ better than on the prime field $\mathbb{Z}_p$ in terms of bandwidth consumption. Another reason to use $\mathbb{Z}_{2^l}$ instead of $\mathbb{Z}_p$ is that modulo reduction in $\mathbb{Z}_{2^l}$ is almost free on standard CPUs.

However state-of-the-art HE-based protocols [6, 7, 10, 21, 34] in the existing 2PC-NN systems force them to export the additive secret to the prime field $\mathbb{Z}_p$ but not to the more efficient choice $\mathbb{Z}_{2^l}$. That is because these HE-based protocols heavily utilize the homomorphic Single-Instruction-Multiple-Data (SIMD) technique [52] to amortize the cost of homomorphic operations. The SIMD technique in turn demands a prime plaintext modulus $p$ due to some algebraic conditions. One can use the Chinese Remainder Theorem to accept secret shares from $\mathbb{Z}_{2^l}$ using a prime modulus $p \approx 2^l$ at the cost of increasing the overhead on the HE-side by many (e.g., 3–5) times. But this would ruin the gains of the non-linear part. In fact, most of the current HE-hybrid systems work over a prime field and tolerate the less efficient non-linear protocols. This raises the question: Could we find a way to achieve the best of both worlds? That is to enjoy amortized homomorphic operations while keeping the efficient non-linear protocols on the yard without extra overheads. As we will show, the answer is yes with our new design of HE-based and SIMD-tailored protocols [6, 7, 10, 21, 34] to rotate the operands many times. Note that the rotation is an expensive operation even compared to the multiplication in the realm of HE, e.g., $30\times$ more expensive cf. [6, Table 9]. These massive homomorphic rotations have become a major obstacle to the existing 2PC-NN inference systems.

As a comparison, the HE-based protocols in Cheetah are free of homomorphic rotation. We present three pairs of encoding functions $(\pi_i, \pi_{i'})$ that enable us to evaluate the linear layers $\mathcal{f} \in \{\text{CONV}, \text{BN}, \text{FC}\}$ of DNNs via polynomial arithmetic circuits. These encoding functions map the values of the input (e.g., tensor or vector) to the proper coefficients of the output polynomial(s). By careful design of the coefficient mappings, we not only eliminate the expensive rotations but are also able to accept secret shares from $\mathbb{Z}_{2^l}$ for free. For example, the secure convolution $\text{HomCONV}(\mathbf{T}, \mathbf{K})$ (given later in Figure 4) in Cheetah could be computed via just a single homomorphic multiplication between two polynomials $\hat{t}$ and $\hat{k}$ where $\hat{t} = \pi_{\text{CONV}}(\mathbf{T})$ is the encoded tensor and $\hat{k} = \pi_{\text{CONV}}(\mathbf{K})$ is the encoded kernel, respectively.

Interestingly, our new design also helps to reduce the cost of other homomorphic operations (e.g., encryption and decryption). On one hand, the rotation-based approaches use a large lattice dimension e.g., $N \geq 8192$. On the other hand, Cheetah can use a smaller dimension (i.e., $N = 4096$) to offer the same capability, i.e., using the same size of the plaintext modulus and evaluating the same classes of linear functions. The main reason lies in the implementation of HEs. Most of the current implementations of lattice-based HEs apply the special prime technique [20] to accelerate the costly homomorphic rotation at the cost of reducing the security level. To bring up the security level, the rotation-based approaches have to bump up the lattice dimension, and thus translate to slower homomorphic operations.

1.1.3 Leaner Protocols for the Non-linear Functions

With the advent of silent OT extension [9] built upon vector oblivious linear evaluation (VOLE), many communication-efficient OT extensions [14, 56] are proposed, and the landscape of non-linear function evaluation demands further in-depth adaptation. [9, 14, 56] suggest that general secure two-party computation can be upgraded by using VOLE-style OT extension, but it remains to be seen how to design special-purpose primitives to fully benefit from VOLE-style OT. Take the integer comparison (Millionaire) protocol, for example, straightforwardly upgrading the OT extensions with VOLE-style OT extensions does not achieve the best performance.

We further make improvements to the truncation protocol, which is required after each multiplication so that the fixed-point values will not overflow. Truncation is expensive: It contributes more than 50% of communication overhead in CrypTFlow2. Our improvements are based on two important observations: First, the truncation protocol in CrypTFlow2 is
designed to eliminate two probability errors $e_0$ and $e_1$ where $Pr(e_0 = 1) = 0.5$ and $Pr(0 < |e_1| < 2^\ell) < \epsilon$ (elaborated in §2.4). With extensive empirical experiments, we observe that the harsh error $e_1$ is indeed problematic but the mild 1-bit error $e_0$ barely harms the inference results, even for large-scale DNNs. This motivates us to design more efficient truncation protocols that eliminate $e_1$ but keep $e_0$ untouched. Our second observation is that sometimes the most significant bit (MSB) is already known before the truncation. For instance we know the the MSB is 0 if the truncation protocol is executed right after a ReLU (i.e., max(0, $x$)) protocol. Similar to the optimization from [45], we implement a truncation protocol for the case of known MSB using VOLE-style OT. In a word, we made all the above optimizations, resulting in faster running time and bringing down more than 90% of the communication cost of CryptTFlow2 for the non-linear layers.

Overall, we compare the complexity of Cheetah’s protocols with the state-of-the-art counterparts in Table 1.

1.2 Other Related Work

Secure computation of machine learning inference algorithms can perhaps date back to [8, 23]. CryptoNets [21] was the first system to consider secure two-party neural network inference. The following improvements after CryptoNets can be roughly categorized into three classes. 1) Optimizations for the basic operations of NN such as convolutions [10, 34], matrix multiplications [33, 40] and non-linear activation functions [19]. 2) Optimizations for a specific class of NN. For instance, NN uses linear activation functions only [7, 17, 27] and binarized NNs [2, 47]. 3) Using mixed primitives (e.g., Garbled Circuit, Trusted Execution Environment, HE and OT) to achieve the best performance for both the linear and non-linear functions in NNs [6, 34, 39, 43, 53]. Some other works consider the secure inference problem with more than two parties such as [15, 16, 37, 42]. These HE-free approaches are usually more efficient than the two-party counterparts. For instance, [37] is more than 15 × faster than CRYPTFLOW2 on ResNet50.

2 Preliminaries

2.1 Notations

We denote by $[n]$ the set $\{0, \cdots, n-1\}$ for $n \in \mathbb{N}$. We use $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ to denote the ceiling, flooring, and rounding function, respectively. We denote $\mathbb{Z}_q = \mathbb{Z} \cap [-q/2, [q/2])$ for $q \geq 2$. Particularly, $\mathbb{Z}_2$ denotes the set $\{0, 1\}$. For a signed integer $x$, we write $x \gg f$ to denote the arithmetic right-shift of $x$ by $f$-bit. $\lambda$ is the security parameter. We use $\mathbb{F}$ to denote a general field. The logical AND, OR and XOR is $\land$, $\lor$ and $\oplus$, respectively. Let $1 \{ \mathbb{P} \}$ denote the indicator function that is 1 when $\mathbb{P}$ is true and 0 when $\mathbb{P}$ is false.

We use lower-case letters with a “hat” symbol such as $\hat{a}$ to represent a polynomial, and $\hat{a}[j]$ to denote the $j$-th coefficient of $\hat{a}$. We use the dot symbol $\cdot$ such as $\hat{a} \cdot \hat{b}$ to represent the multiplication of polynomials. For a 2-power number $N$, and $q > 0$, we write $\mathbb{A}_{N, q}$ to denote the set of integer polynomials $\mathbb{A}_{N, q} = \mathbb{Z}_q[X]/(X^N + 1)$. We use bold upper-case letters such as $\mathbf{T}$ to represent multi-dimension tensors, and use $\mathbf{T}[c, i, j]$ to denote the $(c, i, j)$ entry of a $3$-dimension tensor $\mathbf{T}$. We use bold lower-case letters such as $\mathbf{a}$ to represent vectors, and use $\mathbf{a}[j]$ to denote the $j$-th component of $\mathbf{a}$. We use $\mathbf{a} \cdot \mathbf{b}$ to denote inner product of vectors.

**Polynomial Arithmetic.** Given polynomials $\hat{a}, \hat{b} \in \mathbb{A}_{N, q}$, the product $\hat{d} = \hat{a} \cdot \hat{b}$ over $\mathbb{A}_{N, q}$ is defined by

$$\hat{d}[i] = \sum_{0 \leq j \leq i} \hat{a}[j] \hat{b}[i-j] - \sum_{i < j < N} \hat{a}[j] \hat{b}[N+j-i] \mod q.$$

(1) comes from the fact that $X^N \equiv -1 \mod X^N + 1$.

### Table 1: Comparison with the state-of-the-art of secure 2PC protocols for the linear functions and non-linear functions in DNNs. The linear functions include the convolution (CONV), batch normalization (BN), and fully connection (FC). The convolution and batch normalization take as input of a 3-dimensional tensor $\mathbf{T} \in \mathbb{F}^{C \times H \times W}$. $M$ and $h$ denote the number and the size of the filters, respectively. The fully connection takes as input of a vector $\mathbf{u}_i \in \mathbb{F}^N$ and outputs a vector $\mathbf{u}_o \in \mathbb{F}^N$. We write $n^\ast = \min(n_i, n_o)$ and $n = \max(n_i, n_o)$. The non-linear functions include the comparison of two $\ell$-bit private integers and the truncation of the low $f$-bit of $\ell$-bit integers. $\lambda$ is the security parameter (usually $\lambda \geq 128$).

<table>
<thead>
<tr>
<th>Linear Function</th>
<th>Multi.</th>
<th>Rotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV [41, 46]</td>
<td>$O(MCHW h^2)$</td>
<td>$O(MCHW h^2)$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(MCHW)$</td>
<td>0</td>
</tr>
<tr>
<td>BN [46]</td>
<td>$O(CHW)$</td>
<td>0</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(CHW)$</td>
<td>0</td>
</tr>
<tr>
<td>FC [25]</td>
<td>$O(n_i n_o)$</td>
<td>$O(n_i n_o)$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(n_i n_o)$</td>
<td>0</td>
</tr>
<tr>
<td>Non-linear Function</td>
<td>Communication (bits)</td>
<td></td>
</tr>
<tr>
<td>Compare [46]</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
</tr>
</tbody>
</table>

### 2.2 Lattice-based Homomorphic Encryption

Our protocols use two lattice-based HEs, i.e., HE that based on learning with errors (LWE) and its ring variant (ring-LWE). These two HEs share a set of parameters $HE, pp = \{N, \sigma, q, p\}$ such that $q, p \in \mathbb{Z}$ and $q \gg p > 0$ where the plaintext modulus $p$ can be a non-prime value.

The basic asymmetric RLWE encryption scheme uses a secret polynomial as the secret key $sk = s \in \mathbb{A}_{N, q}$. The
associated public key is computed as \( pk = (\hat{u}_0 \cdot \hat{s} + \hat{e}_0, \hat{u}_0) \) where \( \hat{u}_0 \in \mathbb{Z}_{N,q} \) is chosen uniformly at random, and the error \( \hat{e}_0 \in \mathbb{Z}_{N,q} \) is chosen by sampling its coefficients from \( \chi_{a} \) a discrete Gaussian distribution of standard deviation of \( \sigma \). The RLWE encryption of a message \( \hat{m} \in \mathbb{Z}_{N,p} \) is given as a tuple of polynomials \( \text{RLWE}_{pk}^{N,q,p} (\hat{m}) = (\lfloor \frac{q}{p} \hat{m} \rfloor + \hat{e} \cdot 0) - \hat{u} \cdot pk \in \mathbb{Z}_{N,q} \), where the coefficients of \( \hat{e} \in \mathbb{Z}_{N,q} \) is sampled from \( \chi_{a} \) and the coefficients of \( \hat{u} \in \mathbb{Z}_{N,q} \) is chosen from \( \{0, \pm 1\} \) uniformly at random. A RLWE ciphertext \((\hat{b}, \hat{a})\) is decrypted as \( \text{RLWE}_{sk}^{-1}(\hat{b}, \hat{a}) = [\frac{p}{q}(b + a \cdot \hat{s})] \mod m \).

In our protocols, we use the notation \( \text{LWE}_{sk}^{N,q,p} (m) \) to denote the LWE encryption of a message \( m \in \mathbb{Z}_p \) under a secret vector \( s \in \mathbb{Z}_q^N \). The LWE ciphertext of \( m \) is given as a tuple \((b, a) \in \mathbb{Z}_p^{N+1}\) and it is decrypted by computing \( \text{LWE}_{sk}^{-1}(b, a) = [\frac{p}{q}(b + a \cdot s)] \mod m \). To lighten the notation, we unify the secret of LWE and RLWE ciphertexts by identifying the LWE secret \( s[j] = \hat{s}[j] \) for all \( j \in \mathbb{N} \). We write the LWE encryption of \( m \) as \( \text{LWE}_{sk}^{N,q,p} (m) \) from now on.

The proposed protocol leverages the following functions supported by the RLWE encryption:

- **Homomorphic addition** (\( \oplus \)) and **subtraction** (\( \ominus \)). Given RLWE ciphertexts \( \text{CT}_0 \) and \( \text{CT}_1 \), which respectively encrypts a polynomial \( \hat{p}_0 \) and \( \hat{p}_1 \), the operation \( \text{CT}_0 \oplus \text{CT}_1 \) (resp. \( \text{CT}_0 \ominus \text{CT}_1 \)) results at an RLWE ciphertext \( \text{CT} \) that decrypts to \( \hat{p}_0 + \hat{p}_1 \in \mathbb{Z}_{N,p} \) (resp. \( \hat{p}_0 - \hat{p}_1 \)).

- **Homomorphic multiplication** (\( \otimes \)). Given an RLWE ciphertext \( \text{CT} \) that encrypts a polynomial \( \hat{p} \), and given a plain element \( \hat{e} \in \mathbb{Z}_{N,p} \), the operation \( \hat{p} \otimes \text{CT} \) results at an RLWE ciphertext \( \text{CT} \) that decrypts to \( \hat{p} \cdot \hat{e} \in \mathbb{Z}_{N,p} \).

- **Extract**. Given \((\hat{b}, \hat{a}) = \text{RLWE}_{pk}^{N,q,p} (\hat{m})\) of \( \hat{m} \) we can extract an LWE ciphertext of the \( k \)-th coefficient of \( \hat{m} \), i.e., \((b, a) = \text{Extract}(\hat{b}, \hat{a}, k) \). The tuple \((b, a)\) is a valid LWE ciphertext of \( \hat{m}[k] \) under the secret key sk. This has the effect of avoiding extra information leakage when only certain coefficients are expected to be received by a party. We defer the details of Extract to Chen et al.’s paper [13, §3.3].

By setting a prime \( p \equiv 1 \mod 2N \), it is possible to use the SIMD technique [52] to amortize the cost of homomorphic multiplications. For instance, [25] computes the inner product of two encrypted vector of \( N \) elements using one homomorphic multiplication and \( O(\log_2(N)) \) homomorphic rotations. Cheetah does not use SIMD and rotations, so we defer these details to Appendix B.

### 2.3 Oblivious Transfer

Our protocols rely heavily on oblivious transfer (OT) for non-linear computation (e.g., comparison). In a general 1-out-of-2 OT, denoted by \((\hat{1})\)-OT, a sender inputs two messages \( m_0 \) and \( m_1 \) of length \( \ell \) bits and a receiver inputs a choice bit \( c \in \{0, 1\} \). At the end of the protocol, the receiver learns \( m_c \), whereas the sender learns nothing. OT is usually realized by building a few base OT instances with public key cryptography and extending to a large amount of instances with efficient symmetric cryptographic operations (IKNP OT extension [31]). When sender messages are random or correlated in a way, general OT can be replaced with random OT (ROT) or correlated OT (COT) which are more efficient in communication [3]. More general 1-out-of-N OT \((\hat{n})\)-OT\(_i\) can be implemented in an IKNP-style OT extension, or with \( \log_2 n \) calls to \((\hat{1})\)-OT\(_i\) [44].

Recently, [9] propose silent OT extension, where a large amount of random OT correlations can be generated with a low-communication input-independent setup and a second phase of pure local computation. The technique is later improved with more efficient computation in Ferret [56] and Silver [14]. With little communication cost, these VOLE-style OT’s pave the way for the possibility of next-generation secure computation. We use VOLE-style OT as a building block for more efficient designs of non-linear layers in DNN inference.

### 2.4 Arithmetic Secret Sharing

For the arithmetic secret sharing, an \( \ell \)-bit value \( x \) is shared additively in the ring \( \mathbb{Z}_{2^{\ell}} \) as the sum of two values, say \( \langle x \rangle_A^{\ell} \) and \( \langle x \rangle_B^{\ell} \). To reconstruct the value \( x \), we compute the modulo addition, i.e., \( x \equiv \langle x \rangle_A^{\ell} + \langle x \rangle_B^{\ell} \mod 2^\ell \). In the two-party setting, \( x \in \mathbb{Z}_{2^\ell} \) is secretly shared between Alice and Bob, by letting Alice hold the share \( \langle x \rangle_A^{\ell} \) and letting Bob hold the share \( \langle x \rangle_B^{\ell} \). Also, we will omit the subscript if the modulo 2^\ell is clear from the context.

**Fixed-point Values and Truncation.** Most prior works on 2PC use fixed-point arithmetic, where a real value \( x \in \mathbb{R} \) is encoded as a fixed-point value \( x = [x]_{2^f} / 2^f \in \mathbb{Z} \) under a specified precision \( f > 0 \). The multiplication of two fixed-point values of \( f \)-bit precision results at a fixed-point value of \( 2f \)-bit precision. In order to do subsequent arithmetics, a truncation is required to scale down to \( f \)-bit precision. Suppose \( \langle x \rangle_A^{2f} \) and \( \langle x \rangle_B^{2f} \) are the shares of a double-precision value, i.e., \( [x]_{2^{2f}} \). A faithful truncation protocol should take \( \langle x \rangle_A^{2f} \) and \( \langle x \rangle_B^{2f} \) as input and compute the shares of \([x]_{2^{f}} \geq f \).

**Truncation Errors.** The local truncation protocol of [43] introduces probability errors. For example, to perform the local truncation on \( \langle x \rangle_A^{2f} \) and \( \langle x \rangle_B^{2f} \), two share holders set \( \langle x \rangle_A^{2f} = \lfloor \langle x \rangle_A^{2f} / 2^f \rfloor \) and \( \langle x \rangle_B^{2f} = \lfloor 2^f - \langle x \rangle_B^{2f} / 2^f \rfloor \mod 2^f \), respectively. Then the value \( x' \) equals to \( [x]_{2^{f}} + e_0 + e_1 \) with two probability errors where the small error \( |e_0| \leq 1 \) occurs at the chance of 1/2 and the harsh error \( |e_1| < 2^f \) occurs with a chance of 1/2. To decreases the probability of the harsh error \( e_1 \), a larger bitlength \( \ell \) is needed. This renders a larger overhead on both of linear protocols and non-linear protocols.
2.5 Secure Neural Network Inference

In this section we describe an abstraction of DNN and set up the secure neural inference problem that we will tackle in the rest of the manuscript. DNN takes an input $x$ (e.g., RGB image) and processes it through a sequence of linear and non-linear layers in order to classify it into one of the potential classes, e.g., $z = f_d(f_{d-1}(\cdots(f_1(x, W_1), \cdots), W_{d-1}), W_d)$ where $f_d$ is the function evaluated in the $d$-th layer and $W_d$ is the weight parameter(s) used in that layer. Suppose we already have 2PC protocols that take secret-shared inputs and output secret-shared results for the functions $f_1, f_2, \cdots, f_d$, to achieve the secure inference, we can simply invoke the corresponding 2PC protocols sequentially.

We now describe the functions we are targetting, and then present how to evaluate those functions privately.

2.5.1 Linear Layers

**Fully Connected Layer (FC).** The input to a fully connected layer is a vector $v \in \mathbb{F}^{n_o}$ of length $n_i$ and its output is a vector $u \in \mathbb{F}^{n_o}$ of length $n_o$. A fully connected layer is parameterized by the tuple $(W, b)$ where $W \in \mathbb{F}^{n_o \times n_i}$ is the weight matrix and $b$ is an $n_o$-sized bias vector. The output is specified by the linear transformation $u = Wv + b$.

**Convolution Layer (CONV).** A two dimensional strided convolution $\text{Conv2D}(T, K, s)$ over a field $\mathbb{F}$ operates on a 3-dimension tensor $T \in \mathbb{F}^{C \times H \times W}$ with a stride $s > 0$ and a set of kernels (also called filters) represented by a 4-dimension tensor $K \in \mathbb{F}^{M \times C \times \theta \times \phi}$ to generate a 3-dimension output tensor $T' \in \mathbb{F}^{M \times H' \times W'}$ where $H' = \lfloor (H - h + s)/s \rfloor$ and $W' = \lfloor (W - w + s)/s \rfloor$. If $T$ is an RGB image then $C = 3$ and $H, W$ denote the height and width of the image, respectively. Also, $M$ denotes the number of kernels used in the convolution and $h$ is the size of the kernels.

From a mathematical viewpoint, the two dimensional strided convolution can be seen as calculating weighted sums

$$T'[c', l', j'] = \sum_{c, l, j \in [H]} T[c, l, j] K[c, c', l', l'],$$

for each position $(c', l', j')$ of the output tensor $T'$.

**Batch Normalization Layer (BN).** In DNNs, a BN layer $\text{BN}(T; \mu, \theta)$ takes as input of 3-dimension tensor $T \in \mathbb{F}^{C \times H \times W}$ and output a 3-dimension tensor $T'$ of the same shape. An BN layer is specified by the tuple $(\mu, \theta)$ where $\mu \in \mathbb{F}^C$ is the scaling vector and $\theta \in \mathbb{F}^C$ is the shift vector. For all $c \in [C], i \in [H]$, and $j \in [W]$, $T'$ is computed via

$$T'[c, i, j] = \mu[c] T[c, i, j] + \theta[c].$$

By viewing each channel of $T$ as a $H \times W$-sized vector, we can naturally rewrite the BN evaluation to a form that involves scalar-vector multiplications and vector additions only.

**ReLU**

The input to a fully connected layer is a vector $v \in \mathbb{F}^{n_o}$ of length $n_i$ and its output is a vector $u \in \mathbb{F}^{n_o}$ of length $n_o$. Fully connected layer is parameterized by the tuple $(W, b)$ where $W \in \mathbb{F}^{n_o \times n_i}$ is the weight matrix and $b$ is an $n_o$-sized bias vector. The output is specified by the linear transformation $u = Wv + b$.

**Convolution Layer (CONV).** A two dimensional strided convolution $\text{Conv2D}(T, K, s)$ over a field $\mathbb{F}$ operates on a 3-dimension tensor $T \in \mathbb{F}^{C \times H \times W}$ with a stride $s > 0$ and a set of kernels (also called filters) represented by a 4-dimension tensor $K \in \mathbb{F}^{M \times C \times \theta \times \phi}$ to generate a 3-dimension output tensor $T' \in \mathbb{F}^{M \times H' \times W'}$ where $H' = \lfloor (H - h + s)/s \rfloor$ and $W' = \lfloor (W - w + s)/s \rfloor$. If $T$ is an RGB image then $C = 3$ and $H, W$ denote the height and width of the image, respectively. Also, $M$ denotes the number of kernels used in the convolution and $h$ is the size of the kernels.

From a mathematical viewpoint, the two dimensional strided convolution can be seen as calculating weighted sums

$$T'[c', l', j'] = \sum_{c, l, j \in [H]} T[c, l, j] K[c, c', l', l'],$$

for each position $(c', l', j')$ of the output tensor $T'$.

**Batch Normalization Layer (BN).** In DNNs, a BN layer $\text{BN}(T; \mu, \theta)$ takes as input of 3-dimension tensor $T \in \mathbb{F}^{C \times H \times W}$ and output a 3-dimension tensor $T'$ of the same shape. An BN layer is specified by the tuple $(\mu, \theta)$ where $\mu \in \mathbb{F}^C$ is the scaling vector and $\theta \in \mathbb{F}^C$ is the shift vector. For all $c \in [C], i \in [H]$, and $j \in [W]$, $T'$ is computed via

$$T'[c, i, j] = \mu[c] T[c, i, j] + \theta[c].$$

By viewing each channel of $T$ as a $H \times W$-sized vector, we can naturally rewrite the BN evaluation to a form that involves scalar-vector multiplications and vector additions only.

Parameters: Shape meta $n_i, n_o > 0$.

**Computation:** On input $(v)^A \in \mathbb{F}^{n_o}$, $W \in \mathbb{F}^{n_o \times n_i}$ and $b \in \mathbb{F}^{n_o}$ from Alice and input $(v)^B \in \mathbb{F}^{n_o}$ from Bob, compute $u = Wv + b$.

Sample an uniform vector $r \in \mathbb{F}^{n_o}$. Sample an uniform tensor $T \in \mathbb{F}^{n_o}$.

**Return:** $r$ to Alice and $v - r \in \mathbb{F}^{n_o}$ to Bob.

(a) Ideal Functionality $\mathcal{F}_{\text{FC}}$.

Parameters: Shape meta $M, C, H, W, h$ and stride $s > 0$.

**Computation:** On input $(T)^A \in \mathbb{F}^{C \times H \times W}$, $K \in \mathbb{F}^{M \times C \times \theta \times \phi}$ from Alice and input $(T)^B \in \mathbb{F}^{C \times H \times W}$ from Bob, compute $T' = \text{Conv2D}(T, K, s)$. Sample an uniform tensor $R \in \mathbb{F}$ with the same shape of $T'$.

**Return:** $T$ to Alice and $T' - R \in \mathbb{F}^{n_o}$ to Bob.

(b) Ideal Functionality $\mathcal{F}_{\text{CONV}}$.

Parameters: Shape meta $C, H, W$.

**Computation:** On input $(T)^A \in \mathbb{F}^{C \times H \times W}$, $T, \theta \in \mathbb{F}^{C}$ from Alice and input $(T)^B \in \mathbb{F}^{C \times H \times W}$ from Bob, compute $T' = \text{BN}(T; \mu, \theta)$.

Sample an uniform tensor $R \in \mathbb{F}$ with the same shape of $T'$.

**Return:** $T$ to Alice and $T' - R \in \mathbb{F}^{n_o}$ to Bob.

(c) Ideal Functionality $\mathcal{F}_{\text{BN}}$.

Figure 1: Ideal Functionalities of Linear Layers

2.5.2 Non-linear layers

In the context of deep learning, the non-linear layers consist of an activation function that acts on each element of the input independently or a pooling function that reduces the output size. Typical non-linear functions can be one of several types: the most common ones in DNNs are ReLU functions (i.e., $\text{ReLU}(x) = \max(0, x)$) and max-pooling functions. In the context of 2PC, truncation is also considered as a non-linear layer because truncation is beyond the ability of arithmetic circuit.

Following the blueprint of CrypTFLOW2, these non-linear layers can be evaluated securely via OT-based protocols.

2.5.3 Two-Party Inference System & Threat Model

Suppose the secure inference is executed jointly by Alice (Server) and Bob (Client). Let $A_i$ and $I_B$ be the private input of Alice and Bob to a two-party protocol say $\Pi$, respectively. We write $O_A, O_B \leftarrow \Pi(A_i, I_B)$ to denote an execution of $\Pi$ where $O_A$ and $O_B$ are the output to Alice and Bob, respectively.

We target the same threat model of DELPHI and CrypTFLOW2. *Cheeta* is designed for the two-party semi-honest setting in which both of parties follow the specification of the protocol and only one of the them is corrupted by an adversary. In the context of cryptographic inference, Alice holds a DNN while Bob holds an input to the network, typically an image. By assuming semi-honest Alice and Bob, our system enables Bob to learn only two pieces of information: the architecture of the neural network (i.e., the number of layers, the type of layers, and the size of each layer), and the inference result. Alice is either allowed to learn the result or nothing, depending
on the application scenario. All other information about Bob’s private inputs and the parameters of Alice’s neural network model should be kept secret. We provide formal definitions of threat model in Appendix A.

Like all the prior semi-honest inference systems, Cheetah was not designed to defend against attacks based purely on the inference results (such as the API attacks [51,54]). Indeed, we can integrate orthogonal techniques such as differential privacy [1,32] to provide an even stronger privacy guarantee.

3 Proposed 2PC Protocols of Linear Layers

The core computation in FC, CONV and BN can be rewritten as a batch of inner products. In deep neural networks, the fan-in to the inner product circuit can be large, leading a vast number of homomorphic multiplications. To amortize the cost of multiplications, most of the prior HE-based approaches choose to use the SIMD technique. As we have mentioned, the sum-step in the inner product circuit demands expensive homomorphic rotations. Also the SIMD technique requires plaintext from a prime field ℤₚ such that p = 1 mod 2N.

On the other hand, our linear protocols are free of SIMD and homomorphic rotation. We observe that the polynomial multiplication (1) itself can be viewed as a batch of inner products if we arrange the coefficients properly. In this section, we present and use a pair of natural mappings (πᵖ, πᵖ⁺) to properly place the values in the input and weight to polynomial coefficients for each of the functionality ℱ in Figure 1. With the help of (πᵖ, πᵖ⁺), we then show how to evaluate these functionalities privately. Also, πᵖ and πᵖ⁺ are well-defined for any p > 1 such as p = 2¹, allowing our protocols to accept secretly shared input from the ring ℤ₂¹ for free. All above makes our protocols fundamentally different from the previous SIMD-based approaches.

3.1 Fully Connection

We present our 2PC protocol for FC layers in Figure 2. The core computation in an FC layer is the matrix-vector multiplication u = Wv which can be decomposed into inner products of vectors. Our mapping functions πᵖ and πᵖ⁺ are specialzed for computing the inner product using polynomial arithmetic. Intuitively, when multiplying two polynomials of degree-N, the (N − 1)-th coefficient of the resulting polynomial is the inner product of the two coefficient vectors in opposite orders. We can easily extend this idea to a batch of inner products. We now give the definitions of πᵖ: ℤᵖⁿˣⁿ → ℋN,p and πᵖ⁺: ℤᵖ → ℋN,p. Here we first require n₀ ≤ N for simplicity.

\[ \hat{v} = \piᵖ⁺(v) \quad \text{where} \quad \hat{v}(i) = v[i] \]

\[ \hat{w} = \piᵖ⁺(W) \quad \text{where} \quad \hat{w}[i · n₀ + n₁ - 1 - j] = W[i,j] \]

where i ∈ [n₀], j ∈ [n₁] and all other coefficients of \( \hat{v} \) and \( \hat{w} \) are set to 0. The multiplication of polynomials \( \hat{u} = \hat{w} \cdot \hat{v} \in \mathbb{A}_{N,p} \) directly gives the matrix-vector multiplication \( Wv \equiv u \mod p \) in some of its coefficients.

Proposition 1 Given two polynomials \( \hat{v} = \piᵖ⁺(v), \hat{w} = \piᵖ⁺(W) \in \mathbb{A}_{N,p} \), the multiplication \( Wv \equiv u \mod p \) can be evaluated via the product \( \hat{u} = \hat{w} \cdot \hat{v} \) over the ring \( \mathbb{A}_{N,p} \). That is \( u[i] \) is computed in \( \hat{u}[i · n₁ + n₀ - 1] \) for all \( i \in [n₀] \).

Proof 1 For each \( i \in [n₀] \), we write \( \hat{n}_i = i · n₁ + n₀ - 1 \) for simplicity. By the definition of (1) and (4) \( \hat{v}(j) = 0 \) for \( j \geq n₁ \), we have \( \hat{u}[\hat{n}_i] = \sum_{0 \leq j < n₁} \hat{v}[j] \hat{w}[n₁ - j] = \sum_{0 \leq j < n₁} u[j] W[i,j] \) which is exactly \( u[i] \).

In Proposition 1, other coefficients besides \( \hat{u}[\hat{n}_i] \) in \( \hat{u} \) contain extra information beyond \( Wv \). To prevent such leakage, Alice uses the Extract(·) function to extract the needed coefficients from \( \hat{u} \). We present a toy example in Figure 3.

When \( n₁n₀ > N \), we can first partition the weight matrix into sub-matrices of 0 < \( \hat{n}_i \times \hat{n}_o \) elements such that \( \hat{n}_i \hat{n}_o \leq N \).
We now present how to evaluate the two-dimensional convolution.

We defer the proof to Appendix D due to the space limit. For clarity, we view the computation of 2-dimension convolution as a batch of inner products of $h \times h$ values. We now give the definitions of $\pi_{\text{conv}}^w : Z_{p}^{C \times H \times W} \rightarrow A_{N,p}$ and $\pi_{\text{conv}}^i : Z_{p}^{M \times C \times H \times W} \rightarrow A_{N,p}$. Here we require $MCHW \leq N$ for simplicity.

$$\tilde{t} = \pi_{\text{conv}}^i(T) \text{ st. } \tilde{t}[cHW + iW + j] = T[c,i,j]$$

$$\tilde{k} = \pi_{\text{conv}}^w(K) \text{ st. } \tilde{k}[O - c^tCHW - cHW - iW - l] = K[c',c,l,l'],$$

where $O = HW(MC - 1) + W(h - 1) + h - 1$, and all other coefficients of $\tilde{t}$ and $\tilde{k}$ are set to 0. The multiplication $\tilde{t} \cdot \tilde{k} \in \mathbb{Z}_{N,p}$ directly gives the 2-dimension convolution in some of the coefficients of the resulting polynomial.

**Proposition 2** Given two polynomials $\tilde{t} = \pi_{\text{conv}}^i(T), \tilde{k} = \pi_{\text{conv}}^w(K) \in A_{N,p}$, the convolution $\mathbf{T}' = \text{Conv2D}(T,K) \in \mathbb{Z}_{p}^{M \times C \times H \times W}$ (2) can be evaluated by the polynomial multiplication $\tilde{t} \cdot \tilde{k}$ over the ring $A_{N,p}$. For all positions $(c',c,l,l')$ of $\mathbf{T}'$, $\mathbf{T}'[c',c,l,l']$ is computed in the coefficient $O = c^tCHW + i'sW + j's + f's + \tilde{k}$ where $O = HW(MC - 1) + W(h - 1) + h - 1$.

Similarly, Alice can apply the Extract($\cdot$) function to prevent possible leakage. We first present the basic version of our secure convolution protocol in Figure 4 which demands “small enough tensors” i.e., $MCHW \leq N$.

**Theorem 2** The protocol HomCONV in Figure 4 realizes the ideal functionality $f_{CONV}$ of Figure 1b for $\mathbb{F} = \mathbb{Z}_p$ and for inputs that $MCHW \leq N$ in presence of a semi-honest admissible adversary.

We defer the correctness and security proofs to Appendix D.

**3.2 Two Dimensional Convolution**

We now present how to evaluate the two-dimensional convolution (2) using polynomial arithmetic over the ring $A_{N,p}$. Consider the most simple case of (2) such that $H = W = h$ and $M = 1$, i.e., the input tensor has the same shape as the convolution kernel. Then the computation in (2) becomes an inner product of two flat tensors by concatenating $\mathbf{T}$ and $\mathbf{K}$ row-by-row and channel-by-channel. For general cases, we can view the computation of 2-dimension convolution as a batch of inner products of $h^2$ values. We now give the definitions of $\pi_{\text{conv}}^w : Z_{p}^{C \times H \times W} \rightarrow A_{N,p}$ and $\pi_{\text{conv}}^i : Z_{p}^{M \times C \times H \times W} \rightarrow A_{N,p}$. Here we require $MCHW \leq N$ for simplicity.

$$\tilde{t} = \pi_{\text{conv}}^i(T) \text{ st. } \tilde{t}[cHW + iW + j] = T[c,i,j]$$

$$\tilde{k} = \pi_{\text{conv}}^w(K) \text{ st. } \tilde{k}[O - c^tCHW - cHW - iW - l] = K[c',c,l,l'],$$

where $O = HW(MC - 1) + W(h - 1) + h - 1$, and all other coefficients of $\tilde{t}$ and $\tilde{k}$ are set to 0. The multiplication $\tilde{t} \cdot \tilde{k} \in \mathbb{Z}_{N,p}$ directly gives the 2-dimension convolution in some of the coefficients of the resulting polynomial.

**Proposition 2** Given two polynomials $\tilde{t} = \pi_{\text{conv}}^i(T), \tilde{k} = \pi_{\text{conv}}^w(K) \in A_{N,p}$, the convolution $\mathbf{T}' = \text{Conv2D}(T,K) \in \mathbb{Z}_{p}^{M \times C \times H \times W}$ (2) can be evaluated by the polynomial multiplication $\tilde{t} \cdot \tilde{k}$ over the ring $A_{N,p}$. For all positions $(c',c,l,l')$ of $\mathbf{T}'$, $\mathbf{T}'[c',c,l,l']$ is computed in the coefficient $O = c^tCHW + i'sW + j's + f's + \tilde{k}$ where $O = HW(MC - 1) + W(h - 1) + h - 1$.

Similarly, Alice can apply the Extract($\cdot$) function to prevent possible leakage. We first present the basic version of our secure convolution protocol in Figure 4 which demands “small enough tensors” i.e., $MCHW \leq N$.

**Theorem 2** The protocol HomCONV in Figure 4 realizes the ideal functionality $f_{CONV}$ of Figure 1b for $\mathbb{F} = \mathbb{Z}_p$ and for inputs that $MCHW \leq N$ in presence of a semi-honest admissible adversary.

We defer the correctness and security proofs to Appendix D.
We give an example of this case in Figure 5. That starts from the partition over the substrate why such partitions can work correctly. From the definition of the corresponding pair of subtensor and subkernel. We demonstrate why such partitions can work correctly. From the definition in (2), the convolution along the M-axis is independent for each subkernel, and thus we can equally split the M-axis into sub-blocks of size \([M_1, M_2, \ldots, M_k]\) groups and each of them contains \(M_0\) subkernels. Similarly, the convolution along the C-axis requires extra additions which are supported by HEs. Thus we can safely partition along the C-axis without overlapping too.

By splitting the big tensor and kernel into smaller blocks and zero-padding the margin blocks, we can apply (5) on the corresponding pair of subtensor and subkernel. We demonstrate why such partitions can work correctly. From the definition in (2), the convolution along the M-axis is independent for each subkernel, and thus we can equally split the M-axis into sub-blocks of size \([M_1, M_2, \ldots, M_k]\) groups and each of them contains \(M_0\) subkernels. Similarly, the convolution along the C-axis requires extra additions which are supported by HEs. Thus we can safely partition along the C-axis without overlapping too.

When the kernel size \(h > 1\), we need to take extra care for the partition over the H-axis and W-axis. That is, we need to make sure that the stride window is not split into two adjacent partitions. Specifically, we partition along the H-axis and W-axis into \(d_H = \lceil \frac{H}{H_0} + 1 \rceil\) and \(d_W = \lceil \frac{W}{W_0} + 1 \rceil\) blocks, respectively. For the sub-block indexed by \(\alpha \in [d_H], \beta \in [d_W]\), it contains exact \(H_0\) continuous rows that starts from the \((\alpha(H_0 - h + 1))\)-th row, and exact \(W_0\) continuous columns that starts from the \((\beta(W_0 - h + 1))\)-th column of the big tensor \(T\). When \(h > 1\), there are overlapping between adjacent blocks. We give an example of this case in Figure 5.

1. Bob partitions its share tensor into sub-blocks \(\langle T_{γ', β} \rangle B \in \mathbb{Z}_{p}^{C \times H \times W}\) (with zero-padding if necessary).
2. Bob then encodes the sub-blocks as polynomials \(\langle ã_{γ', β} \rangle = \pi_{\text{bin}}(\langle T_{γ', β} \rangle B)\) for \(γ \in [d_c], \alpha \in [d_H]\) and \(β \in [d_W]\). Bob then sends \(\langle CT_{γ, β} \rangle = \text{RLWE}_{\text{pk}, p}(\langle ã_{γ, β} \rangle B)\) to Alice.
3. Alice partitions and encodes its input tensor and the scaling vector as polynomials \(\langle ã_{γ, β} \rangle = \pi_{\text{bin}}(\langle T_{γ, β} \rangle A)\) and \(\hat{α}_i = \pi_{\text{bin}}(\mu_i), \) respectively.
4. Alice samples \(R \in \mathbb{Z}_{p}^{C \times H \times W}\) uniformly at random.
5. On receiving the ciphertexts \(\langle CT_{γ, β} \rangle\) from Bob, Alice operates \(\langle CT_{γ, β} \rangle = \langle CT_{γ, β} \rangle \otimes \langle ã_{γ, β} \rangle\) to \(\hat{α}_i\) for \(γ \in [d_c], \alpha \in [d_H]\) and \(β \in [d_W]\).
6. [Extract and Re-mask.] For each \(c \in [C], i \in [H]\) and \(j \in [W]\), Alice first extracts a LWE ciphertext \(c_{t, i, j} = \text{Extract}(\text{ct}_{c, i, j} = \text{RLWE}_{\text{pk}, p}(\langle T_{γ, β} \rangle B)\), where \(c = c \mod d_c, i = i \mod H\) and \(j = j \mod W\). Then Bob re-masks these ciphertexts via \(\text{ct}_{c, i, j} = \text{ct}_{c, i, j} \otimes R(c, i, j)\).
7. Alice sends the ciphertexts \(\langle \text{ct}_{c, i, j} \rangle\) back to Bob, and Alice outputs \(\langle T' \rangle A\) as \(\langle T' \rangle A[c, i, j] = R(c, i, j) + \theta(c) \mod p\).
8. On receiving the LWE ciphertexts from Alice, Bob outputs \(\langle T' \rangle B\) as \(\langle T' \rangle B[c, i, j] = \text{LWE}_{\text{sk}}(\text{ct}_{c, i, j})\).

3.2.2 HomCONV (Full Protocol)

The full version of HomCONV is given in Appendix C due to the space limit but we state the complexity of HomCONV. In HomCONV, Bob sends \(O(CHW/N)\) RLWE ciphertexts to Alice which are about \(O(2CHW \log_2 q)\) bits. Alice operates \(O(MCHW/N)\) homomorphic additions and multiplications. Alice sends back \(O(MCHW/N)\) LWE ciphertexts to Bob for decryption. The computation complexity of HomCONV is independent with the kernel size \(h\) where the previous secure convolution protocols [10, 34, 39] scale quadratically with \(h\).

3.3 Batch Normalization

The batch normalization (3) can be evaluated using scalar-polynomial multiplications by mapping each channel of the tensor to polynomials. This idea will lead to a secure BN protocol of a communication cost of \(O(CHW/N)\) ciphertexts. We can reduce this communication cost by “stacking”
We defer the proof to Appendix D due to the space limit. For protocols by Naor and Pinkas \([44]\) by using logs simplified and optimized on VOLE-style OT extension. For non-linear computation, \([46]\) presents various protocols that are tailored to be highly efficient on IKNP-style OT extension. In this section, we show that their protocols can be simplified and optimized on VOLE-style OT extension.

Throughout this section, we make use of \((\ell^2)\)-\text{OT}\(_\ell\), \((\ell^2)\)-\text{COT}\(_\ell\), and \((\ell^4)\)-\text{OT}\(_\ell\). We instantiate \((\ell^4)\)-\text{OT}\(_\ell\) with the algorithm proposed by Naor and Pinkas \([44]\) by using \(\log_2 n\) calls to \((\ell^4)\)-\text{ROT}_\lambda\), whereas CrypTFlow2 implements it with the IKNP-style OT extension proposed in \([35]\). Due to the usage of VOLE-style OT, our calls to \((\ell^4)\)-\text{ROT}_\lambda\) enjoys almost 0 amortized communication cost. We provide a brief comparison of these two approaches in Table 2. When \(n\) and the message length \(\ell\) are small, our approach demonstrates prominent advantage, which is indeed the case in all protocols used in neural network inference (usually, \(n \leq 16, \ell \leq 2\)).

### 4 Optimized 2PC Protocols of Non-Linear Functions

For non-linear computation, \([46]\) presents various protocols that are tailored to be highly efficient on IKNP-style OT extension. In this section, we show that their protocols can be simplified and optimized on VOLE-style OT extension.

#### 4.1 Leaner and Better Millionaires’ Protocol

Millionaires’ protocol for comparing two integers \((x, y) \in \{0, 1\}^s\) is the core building block of almost every non-linear layer, such as ReLU, truncation, and pooling. Let \(f_{\text{AND}}\) denote the functionality that accepts boolean shares of \(x, y\) and returns boolean shares of \(x \land y\) (protocol details in Appendix E). We briefly recall the high-level intuition of the millionaires’ protocol in \([46]\):

1. Each party parses its own \(\ell\)-bit input as smaller blocks of \(m\) bits. Let \(x_j\) and \(y_j\) denote the \(j\)-th block of \(P_0\) and \(P_1\), respectively. Two parties invoke \((\ell^m)\)-\text{OT}_1 to compute a boolean share of \(1 \land j = 1\{x_j < y_j\}\) (similar invocation for \(\text{eq}_j\)).

2. Two parties use \(f_{\text{AND}}\) to combine the former outputs in a tree evaluation (of depth \(\log (\ell/m)\)) with the observation:

\[
1\{x < y\} = 1\{x_1 < y_1\} \oplus 1\{x_1 = y_1\} \land 1\{x_0 < y_0\},
\]

where \(x = x_1|x_0\) and \(y = y_1|y_0\).

In stage 2, \([46]\) presents various optimizations to realize \(f_{\text{AND}}\) efficiently in the context of IKNP-style OT extension, such as using \((\ell^3)\)-\text{OT}_2 to generate two beaver triples and \((\ell^4)\)-\text{OT}_\ell to
We observe that by removing the constraint of correcting the cation and computation.

by \[15\], the last-bit small error does not affect the quality \(\ell\) in practice, the probability is actually non-negligible when
correction protocol. Indeed, the large error destroys the result, and
corrects both these errors, resulting in a heavy faithful truncation

\[4.2.1 \text{ One-Bit Approximate Truncation} \]

\[ \begin{align*}
1: & \text{ Alice and Bob call the millionaires’ protocol:} \\
& \left( (w_1^A, w_2^A) = \text{Mill}^f((x)^A, 2^f - 1 - (x)^B) \right) \\
2: & \text{ Alice and Bob invoke an instance of } F_{2B2A} \text{ with inputs} \\
& \left( (w_1^A, w_2^A) \text{ and learn } (w_1^B, w_2^B) \right) \\
3: & \text{ Alice computes } (z)^A = ((x)^A \gg f) - (w_2^B) \cdot 2^f. \text{ Bob computes} \\
& (z)^B = ((x)^B \gg f) - (w_2^B) \cdot 2^f. \\
\end{align*} \]

Figure 8: Proposed One-Bit Approximate Truncation Protocol in the \(F_{2B2A}\)-hybrid model.

generate correlated triples. However, in Cheetah, these are less
efficient than using \((\frac{\lambda}{2})\)-OT\(_1\) to generate the beaver triples [3].

Nevertheless, their insight in stage 1 to start working with
\(m\)-bit blocks remains important to greatly reduce the number of
AND gates compared to a naive approach of using an evaluation
tree of depth \(\log \ell\). We compare the communication
complexity of the three approaches in Table 3. We reuse the
millionaires’ protocol flow from CrypTFlow2, but replace the
underlying \(F_{\text{AND}}\) implementation with our aforementioned
solution. Let \(\langle m^A_1, m^B_2 \rangle = \text{Mill}\((x, y)\) denote the
millionaires’ protocol where Alice’s input is \(x\), Bob’s input is \(y\), and they receive boolean shares as output:
\(\langle m^A_1 \oplus m^B_2 \rangle = 1 \{x > y\}\).

\[4.2 \text{ Approximate Truncation} \]

In fixed-point computation, truncation is a necessary proce-
dure to maintain the precision after multiplication. CrypTFlow2
proposes faithful truncation that realizes the functionality
\(F_{\text{Trunc}}\) (Figure 7b). We observe that large overhead
in the protocol can be removed in neural network inference.

\[4.2.1 \text{ One-Bit Approximate Truncation} \]

In prior works [41-43], local truncation is used to approximate
\(F_{\text{Trunc}}\), which leads to both a large error with small probability
and a last-bit small error with 1/2 probability. CrypTFlow2
corrects both these errors, resulting in a heavy faithful truncation
protocol. Indeed, the large error destroys the result, and
in practice, the probability is actually non-negligible when
\(\ell \leq 64\), which is the case in this work. Nonetheless, as shown
by [15], the last-bit small error does not affect the quality of
prediction models in machine learning which we also experi-
mentally verify in a concrete neural network (see § 6.4).
We observe that by removing the constraint of correcting the
last-bit error, the truncation protocol can be made far more
lightweight, providing significant improvement in communi-
cation and computation.

\[\text{Proposition 3} \text{ Given an unsigned } \ell\text{-bit integer } x \text{ and its arith-
metic shares } x_0 \text{ and } x_1, \text{ define } w = 1\{x_0 + x_1 > 2^f - 1\}. \text{ Let} \]

\[\text{Inputs and Outputs: } (z)^A_2, (z)^B_2 \leftarrow \text{Trunc}_{\text{whit}}((x)^A_2, (x)^B_2, f) \text{ such that } x \in \mathbb{Z}_{2^f}, \text{ and } x = x \gg f \text{ or } z = (x \gg f) - 1. \]

\begin{align*}
1: & \text{ Alice samples } (w_1^A, w_2^A) \leftarrow k \{0, 1\}. \\
2: & \text{ If } \text{mb}(x)^A = 0, \text{ Alice sets } (s_0, s_1) = ((w_1^A), 1 \oplus (w_2^A)); \text{ other-
\newcommand{\text{wise, Alice sets } (s_0, s_1) = (1 \oplus (w_1^A), 1 \oplus (w_2^A)).} \\
3: & \text{ Alice and Bob invoke } (\frac{\lambda}{2})\text{-OT}_1, \text{ where Alice inputs messages} \\
& (s_0, s_1) \text{ and Bob inputs a choice } \text{mb}(x)^B). \text{ Bob learns } (w_2^B) \text{ as its output.} \\
4: & \text{ Alice and Bob invoke an instance of } F_{2B2A} \text{ with inputs} \\
& ((w_1^A), (w_2^A) \text{ and learn } ((w_1^B), (w_2^B))). \\
5: & \text{ Alice computes } (z)^A = ((x)^A \gg f) - (w_1^B) \cdot 2^f. \text{ Bob computes} \\
& (z)^B = ((x)^B \gg f) - (w_2^B) \cdot 2^f. \\
\end{align*} \]

Figure 9: Proposed Truncation Protocol with Input MSB=0 in
the \(F_{2B2A}\)-hybrid model.

c be one-bit integer 0 or 1. We have: \((x_0 \gg f) + (x_1 \gg f) - w \cdot 2^f = (x \gg f) - c. \]

\[\text{Proof 2 The proposition follows from Corollary 4.2 in [46].} \]

First, \(w\) denotes whether the sum of two shares wrap around
\(2^f\). If so, there will be large error for local truncation and
it should be corrected by subtracting \(2^f\). \(c\) comes from a
probabilistic last-bit error that is decided by whether the sum
of least significant \(s\) bits of \(x_0\) and \(x_1\) wraps around \(2^f\). \]

Let \(F_{2B2A}^f\) denote the functionality that accepts boolean shares
of \(x \in \{0, 1\}\) and returns arithmetic shares of \(x \in \mathbb{Z}_{2^f}\).

We provide an implementation of \(F_{2B2A}^f\) in Appendix E. The one-
bit approximate truncation protocol is provided in Figure 8.

\[4.2.2 \text{ Approximate Truncation with Known MSB} \]

As pointed out by [45], 2PC protocols could be designed in
a far more efficient way when the MSB of the inputs are
known. In these cases, truncation can be further optimized
to be a cheap operation. We consider the case of our one-bit
approximate truncation when MSB is zero (e.g., after ReLU),
and give the corresponding protocol details in Figure 9. When
MSB is known, computing boolean shares of the wrap-around
bit \(w\) can be accomplished with a single call to \((\frac{\lambda}{2})\text{-OT}_1\) instead
of the expensive Mill protocol. Indeed, when MSB is zero,
\(w = \text{mb}(x)^A \lor \text{mb}(x)^B)\). A similar protocol can also be
derived when MSB is one and \(w = \text{mb}(x)^A \land \text{mb}(x)^B)\).

\[\text{4.2.3 Communication Complexity} \]

We provide a comparison of communication complexity
among the discussed truncation protocols in Table 4. CrypTFlow2’s
faithful truncation involves a single call to \(F_{\text{Mill}}^f\),
\((\frac{\lambda}{2})\text{-OT}_1, F_{2B2A}^f\) and \(F_{\text{Mill}}^f\). This leads to a communication upper
bound of \(\lambda(\ell + f + 2) + 19\ell + 14f\) bits when sub-block
length $m = 4$ in the millionaires’ protocol. A direct replacement of OT with the VOLE version gives us an upper bound of $16f + 11f$ bits, according to our complexity analysis in Table 2 and 3 ($f_{B2A}$ uses a single call of $\binom{1}{i}$-OT$_i$). One-bit approximate truncation involves a call to $f_{\text{Mill}}^\ell$, which gives a communication of $13\ell$ bits. When MSB is known, the cost is reduced to $f + 4$ bits.

5 Further Optimizations

We present some optimizations that have not been considered in the previous inference systems [6, 34, 41, 46]. Some of our optimizations can also be applied to these systems.

5.1 Reducing Computation Overhead

The BN layer can be placed right after a CONV or FC layer. For instance, 58 out of the 121 BN layers in DenseNet121 are placed right after a CONV layer. We observe that we can apply the BN fusion technique to save the computation of HomBN since the weights of the BN layer and the CONV layer are already known by Alice. Specifically, for each BN-then-CONV structure in the DNN, Alice first multiplies the weights of the BN layer (i.e., $\mu$) to the kernel $K$ of the CONV layer. Then Alice and Bob jointly invoke the HomCONV protocol on the scaled kernel. After HomCONV, Alice adds the shift vector $\theta$ to her additive share. Interestingly, by using BN fusion, we also save the computation of truncation since we have saved one depth of fixed-point multiplication.

5.2 Reducing Communication Overhead

We present two orthogonal optimizations for reducing the volume of ciphertexts sent by Alice in our protocols.

In the proposed linear protocols, Alice sends many LWE ciphertexts back to Bob for decryption. Indeed, some of the LWE ciphertexts will share the same vector $a$ in their second component if they are extracted from the same RLWE ciphertext. To reduce the communication overhead, Alice can only send one copy of this vector to Bob.

Our second optimization comes from an insightful observation to the (R)LWE decryption formula. Suppose Alice needs to send an LWE ciphertext $(b, a) \in \mathbb{Z}_q^{N+1}$ to Bob for decryption. Our optimization suggests Alice send just some of the high-end bits of $b$ and the values in $a$ to Bob and skip the remaining low-end parts. In brief, Alice can skip the low $\ell_b = \lfloor \log_2(q/p) \rfloor - 1$ bits of $b$ and the low $\ell_a = \lfloor \log_2(q/(6.6\sqrt{N}p)) \rfloor$ bits of the values in $a$ when transferring the ciphertext to Bob. Bob might fail to decrypt the “deform” ciphertext at a negligible chance (i.e., $< 2^{-38}$). Indeed, this optimization saves about 16% – 25% of the communication sent by Alice in our HE-based protocols. We defer the calculation of $\ell_b$ and $\ell_a$ to the full version of this manuscript.

6 Evaluations

6.1 Experiment Setup

Our implementation is built on top of the SEAL library [49] with the HEXL acceleration [30] and the EMP toolkit [55]. We also extend the Ferrer$^+$ protocol in EMP to support various application-level OT types, such as $\binom{1}{i}$-OT$_i$. To compare, we use the open-source library SCI$_{HE}$ that provided by the authors of CrypTFlow2. Besides the OT-based non-linear protocols in CrypTFlow2, SCI$_{HE}$ also offers many optimized implementation of state-of-the-arts including the secure convolution [34, 41], and the matrix–vector multiplication from [12, 25] using an old version of SEAL. For a fair comparison, we modify the code in SCI$_{HE}$ to adopt the latest version of SEAL and to apply HEXL acceleration. Both Cheetah and SCI$_{HE}$ are implemented in C++ and compiled by gcc (version 8.4.0) on Ubuntu 18.04.

Testbed Environment. All the following experiments were performed on Alibaba Cloud with a 2.70 GHz processor and 16 gigabytes of RAM. We ran our benchmarks in two network settings. The bandwidth between the cloud instances were about 384 MBps (LAN) and 44 MBps (WAN), respectively. The round-trip time were about 0.3ms (LAN) and 40ms (WAN), respectively.

Concrete SEAL’s Parameters. We instantiate our protocols using the SEAL parameters HE$_{ppCheetah} = \{N = 4096, q \approx 2^{105}, p = 2^{37}, \sigma = 3.2\}$. Recall that the current secure convolution protocol in SCI$_{HE}$ [48] demands $HW/s^2 \leq N/2$. For most of the time, it can be instantiated using the SEAL parameters HE$_{ppSCI} = \{N = 8192, q \approx 2^{180}, p \approx 2^{37}, \sigma = 3.2\}$. On the other hand, to handle the convolution on large tensors we have to instantiate SCI$_{HE}$ using a larger lattice dimension e.g., $N = 32768$ in their implementation. The security levels under these parameters are not aligned but all of them can provide at least $\lambda = 128$ bits of security according to SEAL.

DNN Architectures. We measure the performance of Cheetah on 6 DNNs (Table 9). For instance, the ResNet50 is trained to classify RGB images of $224 \times 224$ pixels into 1001 classes. The ResNet50 has over 23 million trainable parameters. It

---

Table 4: Communication complexity of truncation protocols of $\ell$-bit integers truncated by $f$ bits.

<table>
<thead>
<tr>
<th>Truncation Protocol</th>
<th>Communication (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faithful [46]</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
</tr>
<tr>
<td>Faithful (VOLE)</td>
<td>$16\ell + 11f$</td>
</tr>
<tr>
<td>One-bit approx. (VOLE)</td>
<td>$13\ell$</td>
</tr>
<tr>
<td>One-bit approx. (VOLE, MSB)</td>
<td>$f + 4$</td>
</tr>
</tbody>
</table>

---

2Our protocol can use any VOLE-style OT extension such as [9, 14, 56].
Table 5: Comparing the running time and communication costs of our linear protocols with the state-of-the-arts. All runs were executed using single thread. 1MB = 2^{20} bytes.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Method</th>
<th>End2End Time</th>
<th>Commm.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>Millionaire</td>
<td>[46]</td>
<td>572ms</td>
<td>1,680ms</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>496ms</td>
<td>663ms</td>
</tr>
<tr>
<td>Truncation</td>
<td>[46]</td>
<td>544ms</td>
<td>1,914ms</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>432ms</td>
<td>655ms</td>
</tr>
<tr>
<td>Trunc. with known MSB</td>
<td>[45, 46]</td>
<td>213ms</td>
<td>716ms</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>31ms</td>
<td>101ms</td>
</tr>
</tbody>
</table>

The only exception is the FC layer, where our communication cost may be 1.33MB higher than theirs.

We compare the performance of the proposed linear protocols with the counterparts implemented in SCI_{HE} in Table 5. The key takeaway is that our computation time is about 1.3 \times - 20\times faster than SCI_{HE}’s, and our communication cost is about 1.5 \times - 2\times lower, depending on the input size.

Moreover, our linear protocols can accept additive shares from the ring \(\mathbb{Z}_2\) that is more friendly for subsequent non-linear layers, while prior approaches [25, 34, 41, 46] that leverage the homomorphic rotation could only accept shares from a prime filed \(\mathbb{Z}_p\). To accept shares from \(\mathbb{Z}_2\), they need to apply the Chinese Remainder Theorem to expand the plaintext modulus to above \((2\ell + 1 + 40)\) bits for achieving 40-bit statistical security [18]. The downside is that it will increase the computation and communication costs by a factor of \(O((2\ell + 1 + 40)/\ell)\) which is about 4 for our parameters.

6.2 Microbenchmarks

6.2.1 Linear Functions

We compare the performance of the proposed linear protocols with the counterparts implemented in SCI_{HE} in Table 5. The key takeaway is that our computation time is about 1.3 \times - 20\times faster than SCI_{HE}’s, and our communication cost is about 1.5 \times - 2\times lower, depending on the input size.

Moreover, our linear protocols can accept additive shares from the ring \(\mathbb{Z}_2\) that is more friendly for subsequent non-linear layers, while prior approaches [25, 34, 41, 46] that leverage the homomorphic rotation could only accept shares from a prime filed \(\mathbb{Z}_p\). To accept shares from \(\mathbb{Z}_2\), they need to apply the Chinese Remainder Theorem to expand the plaintext modulus to above \((2\ell + 1 + 40)\) bits for achieving 40-bit statistical security [18]. The downside is that it will increase the computation and communication costs by a factor of \(O((2\ell + 1 + 40)/\ell)\) which is about 4 for our parameters.

6.2.2 Non-linear Functions

We also benchmark the non-linear functions in Table 6, by running SCI_{HE} and Cheetah with a single thread under the LAN and WAN settings. The performance accounts for 2^{16} calls to the protocols, which resembles a scenario of batch execution in neural network inference. From the table, we observe that our solution significantly reduces the communication cost, i.e., by more than 10\times in all non-linear protocols. The improvements come from both the VOLE-style OT upgrades, and our more concise protocol designs.

6.3 Comparison with DELPHI

In Table 7, we compare Cheetah with DELPHI [41], one of the state-of-the-art 2PC-NN systems. Similar to DELPHI, we perform these computations on the CIFAR dataset [36] with a larger bit-width of \(\ell = 41\) in the WAN setting using 4 threads. Note that Cheetah’s HE-based protocols are also compatible with the online/offline optimizations [4] used by DELPHI. Thus, in Table 7, we present the sum of the offline and online costs of DELPHI reported in their paper (updated version). Here, Cheetah is about 1 order of magnitude faster than DELPHI and about 2 orders of magnitude efficient than DELPHI in terms of communication. Cheetah can give the same output as DELPHI since both frameworks will introduce 1-bit error in each truncation if we assume the harsh truncation error in DELPHI does not occur on shallow NNs.
Table 7: Performance comparison with DELPHI [41, Figure 13] in the LAN setting. Inputs to the benchmarks were RGB images of 32 x 32 pixels. Aligned parameters of sharing modulo to $Z_{2^{41}}$ and fixed-point precision to $f = 11$ with DELPHI were used for Cheetah in this benchmark.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>System</th>
<th>End2End Time</th>
<th>Commu.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELPHI [41]</td>
<td>Cheetah</td>
<td>$\approx 110s$</td>
<td>$\approx 3.6GB$</td>
</tr>
<tr>
<td>SecureQ8 [15]</td>
<td>Cheetah</td>
<td>3.55s</td>
<td>0.03GB</td>
</tr>
</tbody>
</table>

$1GB = 2^{30}$ bytes.

Figure 10: The top-10 values in the prediction vector of 1001 values from SqueezeNet (left) and ResNet50 (right).

### 6.4 Comparison with CrypTFlow2

With all our protocols and optimizations in place, we demonstrate the performance of Cheetah by running cryptographic inferences on large DNNs efficiently. Table 8 shows that Cheetah is efficient enough to evaluate SqueezeNet [29], ResNet50 [26], and DenseNet121 [28] within 3 minutes even under the WAN setting. The end-to-end running time of Cheetah was about $2 \times - 5 \times$ faster than $SCI_{HE}$ within 8% bandwidth consumption of $SCI_{HE}$. To the best of our knowledge, no prior secure two-party inference system can evaluate ResNet50 and DenseNet121 within 10 minutes under the commodity hardware and network conditions as ours.

**Compare with a Three-party Approach.** In Table 8, we also provide the performance of SecureQ8 [15], one of the most efficient three-party secure inference framework. These results are obtained by re-running their implementation [50] under ours environment. As is shown, SecureQ8 was $3 \times - 4 \times$ faster than Cheetah on LAN, while Cheetah was $2 \times - 3 \times$ faster on WAN by the virtue of less communication.

### 6.5 Effective Approximate Truncation

To demonstrate the effectiveness of our approximate truncation protocols, we performed more predictions on SqueezeNet using an image set that includes about 1,000 images from ImageNet. For all these images, Cheetah outputs the same classification label as $SCI_{HE}$. Let’s zoom in a bit. In Figure 10, we present the top-10 values in the prediction vectors (i.e., the input to the final ArgMax layer) computed by $SCI_{HE}$ and Cheetah on ResNet50 and SqueezeNet. Here we can see that Cheetah gives almost the same prediction vectors as $SCI_{HE}$ which are bit-wise equivalent to the plaintext fixed-point computation. Also, according to the extensive experiments in CrypTFlow2 [46, Table 10], the prediction accuracy achieved by fixed-point computation can match the accuracy of the floating-point counterpart. We conclude that our approximate truncation protocols are effective for the 2PC-NN setting.

### Conclusion

Secure two-party deep neural network inference is getting closer to practicality. Cheetah presents a highly optimized architecture that runs the complex ResNet50 model in less than 2.5 minutes even under the WAN setting, consuming 2.3 GB of communication. The evidential improvement over stage-of-the-art comes from a set of novel cryptographic protocols that are built upon a more profound exploration of the problem-setting. Cheetah also provide better alternatives to a wide spectrum of functional evaluation problems in secure two-party computation. One of our future work is to apply orthogonal optimizations commonly adopted in DNN such as quantization and hardware acceleration. We believe the day is not that far off when some applications such as privacy-preserving medical diagnosis could be done in seconds.

**Acknowledgment.** The authors would like to thank the anonymous reviewers for their insightful comments and Dr. Marina Blanton for the shepherding. The authors would also like to thank ChenKai Weng from Northwestern University for helpful discussions on Ferret.
References


A Threat Model and Security

We provide security against a static semi-honest probabilistic polynomial time adversary $\mathcal{A}$ following the simulation paradigm [11, 22, 38]. That is, a computationally bounded adversary $\mathcal{A}$ corrupts either Alice (Server) or Bob (Client) at the beginning of the protocol $\Pi_f$ and follows the protocol specification honestly. Security is modeled by defining two interactions: a real interaction where Alice and Bob execute the protocol $\Pi_f$ in the presence of $\mathcal{A}$ and the environment $E$ and an ideal interaction where the parties send their inputs to a trusted party that computes the functionality $f$ faithfully. Security requires that for every adversary $\mathcal{A}$ in the real interaction, there is an adversary $\text{Sim}$ (called the simulator) in the ideal interaction, such that no environment $E$ can distinguish between real and ideal interactions.

We recap the definition of a cryptographic inference protocol in DELPHI [41]. The server holds a model $W$ consisting of $d$ layers $W_1, \ldots, W_d$. The client holds an input vector $x$.

Definition 1 A protocol $\Pi$ between a server having as input model parameters $W = (W_1, \ldots, W_d)$ and a client having as input a feature vector $x$ is a cryptographic inference protocol if it satisfies the following guarantees.

- **Correctness.** On every set of model parameters $W$ that the server holds and every input vector $x$ of the client, the output of the client at the end of the protocol is the correct prediction $W(x)$.

- **Privacy.** We require that a corrupted, semi-honest server does not learn anything about the server’s network parameters $W$. Formally, we require the existence of an efficient simulator $\text{Sim}_C$ such that $\text{View}_C \approx_c \text{Sim}_C(\text{meta}, \text{out})$ where $\text{View}_C$ is the view of the client in the execution of $\Pi$, meta includes the meta information (i.e., the public parameters $HE_{pp}$, the public key pk, the number of layers, the size and type of each layer and the activation) and out denotes the output of the inference. We also require that a corrupted, semi-honest server does not learn anything about the private input $x$ of the client. Formally, we require the existence of an efficient simulator $\text{Sim}_S$ such that $\text{View}_S \approx_c \text{Sim}_S(\text{meta})$ where $\text{View}_S$ is the view of the server in the execution of $\Pi$.

B SIMD and Homomorphic Rotation

The SIMD technique [52] uses a discrete fourier transform over the prime field $\mathbb{Z}_p$ to convert vectors $v, u \in \mathbb{Z}_p^N$ of $N$ elements to ring elements $\tilde{v}, \tilde{u} \in \mathbb{A}_{N,p}$, respectively. The product polynomial $\tilde{v} \cdot \tilde{u} \in \mathbb{A}_{N,p}$ decodes to the Hadamard product between the vectors $v$ and $u$. In the context of encryption, the SIMD technique can amortize the cost of homomorphic multiplication by a factor of $1/N$. However, once the SIMD-encoded vector is encrypted, it is not straightforward to manipulate the positions of the encoded values. For example, to homomorphically right-hand-side rotate the encrypted vector by $k \in [1, N]$ unit, one needs to multiply the ciphertext with a rotation key given as $\text{RLWE}^{N;q;p} \left( \psi \cdot \text{mod} \ 2N(\text{sk}) \right)$ where the automorphism $\rho_g : \mathbb{A}_{N,p} \mapsto \mathbb{A}_{N,p}$ is defined by $\rho_g(\tilde{a}(X)) = \tilde{a}(X^g) \mod X^N + 1$.

C Full Version of HomCONV

We now present the full version of $\text{HomCONV}$ in Figure 11. In Step 1 and Step 2 of Figure 11, Bob first partitions its share of tensor (with zero-padding) into smaller blocks of the same shape $C_w \times W_w \times H_w$ along the three dimensions. Bob then sends to Alice $d_c d_m d_w$ RLWE ciphertexts that each of them encrypts one partition block of $T^b$ in Step 3. In the next two steps, Alice first partitions its share $T^a$ following the same partitioning manner in Step 1 and Step 2. After that Alice partitions the kernel $K$ into $d_k d_c$ non-overlapping blocks. Then Alice can encode each of them using $\pi_{\text{con}}$, with an identical parameter $O_w$ that is because the tensor $T$ is partitioned into blocks of the same shape $C_w \times H_w \times W_w$. On receiving the RLWE ciphertexts from Bob, Alice then computes the...
Inputs and Outputs \( \langle T' \rangle^A, \langle T' \rangle^B \leftarrow \text{HomCONV} \left( \{ \langle T \rangle^A, K \}, \{ \langle T \rangle^B, sk \} \right) \) such that \( \langle T \rangle^A, \langle T \rangle^B \in \mathbb{Z}_p^{C \times H \times W}, K \in \mathbb{Z}_p^{M \times C \times h \times h} \) and \( T' = \text{Conv2D}(T, K, s) \in \mathbb{Z}_p^{M' \times H' \times W'} \).

Public Parameters: \( pp = (\text{HE}_p, pk, M, C, H, W, h, s) \).

- Shape meta \( M, C, H, W, h, s \) such that \( h^2 \leq N \) and stride \( s > 0 \). The optimal sizes of the partition window that minimize \( \arg\min_{h_w, w} \left\lceil \frac{C}{N/(H_w \times W_w)} \right\rceil \left\lceil \frac{h + 1}{H_w} \right\lceil \left\lfloor \frac{W - h + 1}{W_w} \right\rfloor \) such that \( h \leq H_w \leq H, h \leq W_w \leq W \) and \( H_w W_w \leq N \).

- The partition window size along the \( C \)-axis and \( M \)-axis is \( C_w = \min(C, \left\lceil \frac{N}{H_w W_w} \right\rceil) \) and \( M_w = \min(M, \left\lceil \frac{N}{C_w H_w} \right\rceil) \).

- Set \( d_M = \left\lceil \frac{M}{M_w} \right\rceil, d_C = \left\lceil \frac{C}{C_w} \right\rceil, d_H = \left\lceil \frac{H}{H_w} \right\rceil \) and \( d_W = \left\lceil \frac{W}{W_w} \right\rceil \). \( O_w = H_w W_w (M_w C_w - 1) + W_w (h - 1) + h - 1 \).

- Set \( H' = \left\lceil \frac{H - h + 1}{s} \right\rceil, W' = \left\lceil \frac{W - h + 1}{s} \right\rceil, H'_w = \left\lfloor \frac{H_w - h + 1}{s} \right\rfloor \) and \( W'_w = \left\lfloor \frac{W_w - h + 1}{s} \right\rfloor \).

1: Bob first partitions \( \langle T \rangle^B \) into blocks along the \( H \)-axis and \( W \)-axis \( \langle T_{\alpha, \beta} \rangle^B \in \mathbb{Z}_p^{C \times H \times W} \) for \( \alpha \in \left[ d_H \right] \) and \( \beta \in \left[ d_W \right] \). Each block \( \langle T_{\alpha, \beta} \rangle^B \) consists of \( H_w \) continuous rows and \( W_w \) continuous columns of \( \langle T \rangle^B \). Indeed, \( \langle T_{\alpha, \beta} \rangle^B \) is taken from the \( \alpha(h_w - h + 1) \)-th row and \( \beta(W_w - h + 1) \)-th column of \( \langle T \rangle^B \). Zero-padding might be used to make sure all \( \langle T_{\alpha, \beta} \rangle^B \) blocks contain the same number of rows and columns.

2: Bob then splits the channels of each block tensor \( \langle T_{\alpha, \beta} \rangle^B \) into non-overlapping blocks \( \langle T_{\gamma, \alpha, \beta} \rangle^B \in \mathbb{Z}_p^{M \times C \times h \times h} \) for \( \gamma \in \left[ d_C \right] \). Also, zero-padding might be used to make sure all \( \langle T_{\gamma, \alpha, \beta} \rangle^B \) blocks contain the same number of channels.

3: Bob encrypts and then sends the RLWE ciphertexts \( \langle CT_{\gamma, \alpha, \beta} \rangle_{pk}^w = \text{RLWE}_{pk}^{\{i\}_{\gamma, \alpha, \beta}}(\pi_{\text{conv}}(\langle T_{\gamma, \alpha, \beta} \rangle^B)) \) to Alice.

4: Alice partitions \( \langle T \rangle^A \) into blocks \( \langle T_{\alpha, \beta} \rangle^A \in \mathbb{Z}_p^{C \times H \times W} \) following the same manner in Step 1 and Step 2. Then Alice encodes each block to a polynomial via \( \langle \tilde{t}_{\alpha, \beta} \rangle^A = \pi_{\text{conv}}(\langle T_{\alpha, \beta} \rangle^A) \).

5: Alice then splits \( K \) into sub-kernels along the \( M \)-axis, i.e., \( K_{\theta} \in \mathbb{Z}_p^{M \times C \times h \times h} \) for \( \theta \in \left[ d_M \right] \). Then Alice further split each \( K_{\theta} \) into smaller block along the \( C \)-axis, i.e., \( K_{\alpha, \gamma} \in \mathbb{Z}_p^{M \times C_w \times h \times h} \) for \( \gamma \in \left[ d_C \right] \). Zero-padding might be used to make sure all blocks contains the same number of elements. Then Alice encodes these block tensors into polynomials \( \hat{k}_{\theta, \gamma} = \pi_{\text{conv}}(K_{\theta, \gamma}) \).

6: Alice samples \( \langle T' \rangle^A \in \mathbb{Z}_p^{M' \times H' \times W'} \) uniformly at random and outputs it.

7: On receiving \( \{ CT_{\gamma, \alpha, \beta} \} \), Alice computes \( CT_{\theta, \alpha, \beta} = \text{Hom}_{\text{HE}}(\langle T_{\alpha, \beta} \rangle^A) \otimes \hat{k}_{\theta, \gamma} \) for \( \theta \in \left[ d_M \right], \alpha \in \left[ d_H \right] \) and \( \beta \in \left[ d_W \right] \).

8: [Extract and Re-mask.] For each \( c' \in \left[ d_C \right], i' \in \left[ H' \right] \), and \( j' \in \left[ W' \right] \), and Alice sends an LWE ciphertext to Bob \( ct_{c', i', j'} = \text{Extract}(CT_{\theta, \alpha, \beta}, Ow = c_w H_w W_w + is H_w + js) \otimes \langle T \rangle^A \) \( [c', i', j'] \), where the index of \( CT_{\theta, \alpha, \beta} \) is calculated as \( \theta = \left\lfloor \frac{c'}{M_w} \right\rfloor \), \( \alpha = \left\lfloor \frac{i'}{H_w} \right\rfloor \), and \( \beta = \left\lfloor \frac{j'}{W_w} \right\rfloor \). The position of the extracting coefficient is determined by \( c = c' \mod M_w, i = i' \mod H'_w \) and \( j = j' \mod W'_w \).

9: On receiving the ciphertexts \( \{ ct_{c', i', j'} \} \), Bob outputs \( \langle T' \rangle^B \in \mathbb{Z}_p^{M' \times H' \times W'} \) where \( \langle T' \rangle^B [c', i', j'] = \text{LWE}_{pk}^{-1}(ct_{c', i', j'}) \).

Figure 11: Proposed Secure Convolution Protocol (Full Version)

secure convolution using \( d_M d_C d_H d_W \) homomorphic multiplications and homomorphic additions. Indeed, the RLWE ciphertext \( CT'_{\theta, \alpha, \beta} \) in Step 7 corresponds to the convolution of the block tensor \( T_{\alpha, \beta} \) and the sub-kernels \( K_{\theta} \). As a result, each RLWE ciphertext \( CT'_{\theta, \alpha, \beta} \) obtains at most \( M_w H'_w W'_w \) values of \( T' \) in its encrypted coefficients. Finally, Alice extracts \( MH'W' \) LWE ciphertexts that each of them encrypts one entry of the output tensor \( T' \). Similar to the basic version, in Step 8 Alice randomizes the encrypted values in the LWE ciphertexts by homomorphically adding uniform random values before sending back the LWE ciphertexts to Bob for decryption.

**Theorem 4** The protocol HomCONV in Figure 11 realizes the ideal functionality \( f_{\text{CONV}} \) of Figure 1b for the field \( \mathbb{F} = \mathbb{Z}_p \) and for \( h^2 \leq N \) in presence of a semi-honest admissible adversary.

**D. Proofs**

**Proof 3** (Proposition 2) We write \( O' = O - c'CHW \) for simplicity. By the definition (1) we have

\[
\hat{p}'[x] = \sum_{0 \leq d \leq x} \hat{f}[d] \hat{k}[x - d] - \sum_{x < d < N} \hat{f}[d] \hat{k}[N + x - d]
\]
for all $x \in \{0, 1\}$. Since $i[x]$ is zero for all $x \geq CHW$ and the target position $O' + i'sW + j's > CHW$, we thus have

$$
i'(O' + i'sW + j's) = \sum_{d < CHW} i[d] \tilde{k}[O' + i'sW + j's - d]
$$

$$= \sum_{c \in [C]} \sum_{l \in \mathbb{Z}} \sum_{\hat{i} \in \mathbb{Z}} \tilde{k}[cHW + iW + j]\tilde{k}[O' + i'sW + j's - (cHW + iW + j)]
$$

$$= \sum_{c \in [C]} \sum_{l \in \mathbb{Z}} \sum_{\hat{i} \in \mathbb{Z}} \tilde{k}[cHW + iW + j]\tilde{k}[O' - cHW - iW - l']
$$

The last line replaces $i = l + s + l'$ and $j = j' + s + l$. Also, according to the definition of $\pi$, the value $\tilde{k}[O' - cHW - iW - l']$ is zero when $l', l' \not\in \{\hat{h}\}$. Thus, we only need to take care of the positions that $l', l' \leq \{\hat{h}\}$. The above equation continues.

$$= \sum_{c \in [C]} \sum_{l \in \mathbb{Z}} \sum_{\hat{i} \in \mathbb{Z}} \tilde{k}[cHW + (l + s + l')W + j's + l']\tilde{k}[O' - cHW - iW - l']
$$

$$= \sum_{c \in [C]} \sum_{l \in \mathbb{Z}} \sum_{\hat{i} \in \mathbb{Z}} \tilde{k}[cHW + (l + s + l')W + j's + l']\tilde{k}[O' - cHW - cHW - iW - l']
$$

$$= \sum_{c \in [C]} \sum_{l \in \mathbb{Z}} \sum_{\hat{i} \in \mathbb{Z}} \tilde{k}[cHW + (l + s + l')W + j's + l']\tilde{k}[cHW + (l + s + l')W + j's + l']
$$

The final line is exactly $T'[c', i', j']$. □

**Proof (Theorem 2.)** The correctness of Theorem 2 is directly derived from Proposition 2. We now show the the privacy part.

( Corrupted Alice.) Alice’s view of $\text{View}_A^{\text{HomCONV}}$ consists of an RLWE ciphertext $CT'$. The simulator $\text{Sim}_A$ for this view can be constructed as follows.

1. Given the access to meta, $\text{Sim}_A$ outputs the ciphertext $CT' = \text{RLWE}_{pk}^{N,q}(0)$ to Alice.

The security against a corrupted Alice (Server) is directly reduced to the semantic security of the underlying encryption. Thus we have $\text{View}_A^{\text{HomCONV}} \approx c \text{Sim}_A(\text{meta})$.

( Corrupted Bob.) Bob’s view of $\text{View}_B^{\text{HomCONV}}$ consists of LWE ciphertexts $\{ct_{c', i', j'}\}$, and the decryption of these LWE ciphertexts, i.e., $(T)^B$. The simulator $\text{Sim}_B$ for this view can be constructed as follows.

1. On receiving the RLWE ciphertext $CT$ from Bob and given the access to meta, $\text{Sim}_B$ samples uniform random polynomial $\tilde{r} \in A_{n,p}$ and computes $CT' = \text{RLWE}_{pk}^{N,q}(\tilde{f})$.

2. For each $c' \in [M]$, $i' \in [H']$ and $j' \in [W']$, $\text{Sim}_B$ outputs an LWE ciphertext $ct_{c', i', j'} = \text{Extract}(CT)$ to Bob.

3. Given the access to out, $\text{Sim}_B$ outputs the tensor $T$ such that $T[c', i', j'] = \tilde{r}[O' + c'CHW + i'sW + j's]$ for each $c' \in [M]$, $i' \in [H']$ and $j' \in [W']$.


<table>
<thead>
<tr>
<th>Networks</th>
<th>linear operations</th>
<th>non-linear operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV</td>
<td>BN</td>
<td>FC</td>
</tr>
<tr>
<td>MNet</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>RN32</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>SQNet</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>RN50</td>
<td>53</td>
<td>49</td>
</tr>
<tr>
<td>DNet</td>
<td>121</td>
<td>121</td>
</tr>
</tbody>
</table>

MNet = MinIONN; RN32 = ResNet32; RN50 = ResNet50

SQNet = SqueezeNet; DNet = DenseNet121

Table 9: DNN Architectures.

<table>
<thead>
<tr>
<th>Inputs &amp; Outputs: $(a', b, c) \rightarrow \text{AND}(a', b, c)$</th>
<th>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs $x, y, z \in {0, 1}$ and $z = x \land y$, $a, b, c \in {0, 1}$</td>
<td>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</td>
</tr>
<tr>
<td>1. Alice and Bob jointly generate a beaver triple: $(c, a', b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$</td>
<td>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</td>
</tr>
<tr>
<td>2. Alice computes $(c, a', b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$, Bob computes $(a, b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$</td>
<td>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</td>
</tr>
<tr>
<td>3. Alice and Bob open $a$ and $b'$, Bob computes $(c, a', b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$</td>
<td>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</td>
</tr>
<tr>
<td>4. Alice computes $(c, a', b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$, Bob computes $(a, b') = ((a_0)^A + (a_1)^A, (b_0)^B + (b_1)^B)$</td>
<td>$(x, y, z) \rightarrow \text{AND}(x, y, z)$</td>
</tr>
</tbody>
</table>

Figure 12: Protocol for $\text{AND}$. 

<table>
<thead>
<tr>
<th>Inputs and Outputs: $(a' = A, b' = B)$</th>
<th>$(a', b, c) \rightarrow \text{B2A}(a', b, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs $a, b, c \in {0, 1}$ and $A \neq B$</td>
<td>$(a', b, c) \rightarrow \text{B2A}(a', b, c)$</td>
</tr>
<tr>
<td>1. Alice and Bob invoke an instance of $(\tilde{f})\text{-COT}_f$, where Alice is the sender with correlation function $g(x) = x - 2(c)^A$ and Bob is the receiver with input choice $(c)^A$. Alice learns $x$ and Bob learns $y$.</td>
<td>$(a', b, c) \rightarrow \text{B2A}(a', b, c)$</td>
</tr>
<tr>
<td>2. Alice computes $(a' = A, b' = c, b' = c$) and Bob computes $(a' = A, b' = c, b' = c$)</td>
<td>$(a', b, c) \rightarrow \text{B2A}(a', b, c)$</td>
</tr>
</tbody>
</table>

Figure 13: B2A Protocol.

Similarly, the LWE ciphertexts $\{ct_{c', i', j'}\}$ due to the semantic security. Also, the values in the output tensor $(T)^B$ of Bob in $\text{HomCONV}$ distribute uniformly in $Z_p$ which is exact the same distribution $\text{Sim}_B$ samples $T$. Thus we have $\text{View}_B^{\text{HomCONV}} \approx \text{Sim}_B(\text{meta}, \text{out})$.

The security proofs for Theorems 1, 2, and 3 can be given in a similar manner.

**E Protocols for $\text{AND}$ and $\text{B2A}$**

We describe the classic protocol for computing $\text{AND}$ in Figure 12, where beaver triples are generated with $(\tilde{f})\text{-ROT}_1$ [3]. We describe a protocol for $\text{B2A}$ in Figure 13, which is also used in CryptoFlow2 [46, 48], except that $(\tilde{f})\text{-COT}_f$ is instantiated with VOLE-style OT.