Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference (Full Version)

Zhicong 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 $5.6 \times$ and $12.9 \times$, respectively.

1 Introduction

To alleviate some of the privacy concerns associated with the ubiquitous deployment of deep learning technologies, many works [3, 7, 11, 18, 23, 36, 42, 44] in the past few years have introduced cryptographic frameworks based on secure two-party computation (2PC) [6] 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 [50] 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 [30], 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 $5.6 \times$ and $12.9 \times$, respectively.

Potential Real World Applications. In [61], the researchers train a DenseNet121 to predict lung diseases from chest X-ray images. Also, [26] uses DNNs for diagnosing Diabetic Retinopathy (one of the major causes of blindness) from reti-
nal 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 [44] and CrypTFLOW2 [50] 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 [42, 44, 46, 50] 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 [50], 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 [7, 8, 11, 23, 36] 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 [56] 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 implementations of lattice-based HEs apply the same prime technique \[22\] 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.

1.1.2 Fast and SIMD-free Linear Protocols

Due to the spatial property of the convolution and matrix-vector multiplication, it is inevitable for the prior HE-based and SIMD-tailored protocols [7, 8, 11, 23, 36] 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× more expensive cf. [7, 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 \((\pi^\pi_{\text{FC}}, \pi^\pi_{\text{CONV}})\) that enable us to evaluate the linear layers \( f \in \{\text{CONV}, \text{BN}, \text{FC}\} \) 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 \( \text{HomCONV}(T, K) \) (given later in Figure 4) in Cheetah could be computed via just a single homomorphic multiplication between two polynomials \( \hat{i} \) and \( \hat{k} \) where \( \hat{i} = \pi^\pi_{\text{CONV}}(T) \) is the encoded tensor and \( \hat{k} = \pi^\pi_{\text{CONV}}(K) \) 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 \[22\] 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 [10] built upon vector oblivious linear evaluation (VOLE), many communication-efficient OT extensions [15, 60] are proposed, and the landscape of non-linear function evaluation demands further in-depth adaptation. [10, 15, 60] 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
designated to eliminate two probability errors $e_0$ and $e_1$ where $Pr(e_0) = 0.5$ and $Pr(0 < |e_1| < 2^λ) < ε$ (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 [49], 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 CrypTFlow2 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 [9,25]. CryptoNets [23] 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 [11,36], matrix multiplications [35,43] and non-linear activation functions [20]. 2) Optimizations for a specific class of NN. For instance, NN uses linear activation functions only [8,18,29] and binarized NNs [3,51]. 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 [7,36,42,46,48,57]. Some other works consider the secure inference problem with more than two parties such as [16,17,38,38,40,45]. These HE-free approaches are usually more efficient than the two-party counterparts. For instance, [40] is more than 15× faster than CrypTFlow2 on ResNet50.

2 Preliminaries

2.1 Notations

We denote by $[n]$ the set $\{0, \ldots, n-1\}$ for $n \in \mathbb{N}$. We use $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$ and $\cdot$ to denote the ceiling, flooring, and rounding function, respectively. We denote $\mathbb{Z}_q = \mathbb{Z} \cap [0, q/2]$ 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 $\wedge$, $\vee$ and $\oplus$, respectively. Let $1\{\cdot\}$ denote the indicator function that is 1 when $\cdot$ is true and 0 when $\cdot$ is false.

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-dimension 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 $\mathbf{u} \in \mathbb{F}^n$ and outputs a vector $\mathbf{u} \in \mathbb{F}^m$. We write $n^* = \min(n_i, n_o)$ and $h = \max(n_i, n_o)$. The non-linear functions include the comparison of two $t$-bit private integers and the truncation of the low $j$-bit of $t$-bit integers. $\lambda$ is the security parameter (usually $\lambda \geq 128$).

<table>
<thead>
<tr>
<th>Linear Function</th>
<th>Mult.</th>
<th>Rotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV [44, 50]</td>
<td>$O(\mathcal{MCHW}h^2)$</td>
<td>$O(\mathcal{MCHW}h^2)$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(\mathcal{MCHW})$</td>
<td>0</td>
</tr>
<tr>
<td>BN [50]</td>
<td>$O(CHW)$</td>
<td>0</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(CHW)$</td>
<td>0</td>
</tr>
<tr>
<td>FC [27]</td>
<td>$O(n_o n_i)$</td>
<td>$O(n^* + \log_2(\frac{q}{n^*}))$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$O(n_o n_i)$</td>
<td>0</td>
</tr>
<tr>
<td>Non-linear Function</td>
<td>Communication (bits)</td>
<td></td>
</tr>
<tr>
<td>Compare [50]</td>
<td>$\lambda(\ell + f + 2) + 19\ell + 14f$</td>
<td>13$\ell$</td>
</tr>
<tr>
<td>Cheetah</td>
<td>$&lt; \lambda \ell + 14\ell$</td>
<td>$&lt; 11\ell$</td>
</tr>
</tbody>
</table>

* Assume MSB is already known.

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 · to denote the multiplication of polynomials. For a 2-power number $N$, and $q > 0$, we write $\hat{k}_{N,q}$ to denote the set of integer polynomials $\hat{k}_{N,q} = \mathbb{Z}[X]/(X^N + 1)$. 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 $\otimes$ to denote the inner product of vectors.

### Polynomial Arithmetic

Given polynomials $\hat{a}, \hat{b} \in \hat{k}_{N,q}$, the product $\hat{d} = \hat{a} \cdot \hat{b}$ over $\hat{k}_{N,q}$ is defined by

$$\hat{d}[i] = \sum_{0 \leq j \leq i} \hat{a}[j] \cdot \hat{b}[i-j] - \sum_{i \leq j < N} \hat{a}[j] \cdot \hat{b}[N+j-i] \mod q.$$  (1)

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

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

In our protocols, we use the notation $LWE_{\hat{m}}(\hat{p}, \hat{a})$ to denote the LWE encryption of a message $\hat{m} \in \mathbb{Z}_p$ under a secret vector $\hat{s} \in \mathbb{Z}_q^n$. The LWE ciphertext of $\hat{m}$ is given as a tuple $(\hat{b}, \hat{a}) \in \mathbb{Z}_p^n$ and it is decrypted by computing $LWE_{\hat{m}}^{-1}(\hat{b}, \hat{a}) = [\frac{\hat{b}}{q} (\hat{b} + \hat{a} \cdot \hat{s}) \bmod p] = \hat{m} \mod p$. To lighten the notation, we unify the secret and RLWE ciphertexts by identifying the LWE secret $s_i [j] = \delta [j] \bmod 2^n$. We write the LWE encryption of $\hat{m}$ as $LWE_{sk}^{-1}(\hat{b}, \hat{a}) = [\frac{\hat{b}}{q} (\hat{b} + \hat{a} \cdot \hat{s}) \bmod p] \equiv \hat{m} \mod p$.

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

- **Homomorphic addition** ($\oplus$) and **subtraction** ($\ominus$). Given RLWE ciphertexts $CT_0$ and $CT_1$, which respectively encrypts a polynomial $\hat{p}_0$ and $\hat{p}_1$, the operation $CT_0 \oplus CT_1$ results in an RLWE ciphertext $CT'$ that decrypts to $\hat{p}_0 + \hat{p}_1 \in \mathbb{A}_{N,p}$ (resp. $\hat{p}_0 - \hat{p}_1$).

- **Homomorphic multiplication** ($\times$). Given an RLWE ciphertext $CT$ that encrypts a polynomial $\hat{p}$, and given a plain element $\hat{c} \in \mathbb{A}_{N,p}$, the operation $\hat{c} \times CT$ results in an RLWE ciphertext $CT'$ that decrypts to $\hat{p} \cdot \hat{c} \in \mathbb{A}_{N,p}$.

- **Extract**. Given $(\hat{b}, \hat{a}) = RLWE_{pk}(\hat{m})$ of $\hat{m}$ we can extract an LWE ciphertext of the $k$-th coefficient of $\hat{m}$, i.e., $(\hat{b}, \hat{a}) = Extract((\hat{b}, \hat{a}), k)$. The tuple $(\hat{b}, \hat{a})$ 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 [14, §3.3].

By setting a prime $p \equiv 1 \bmod 2N$, it is possible to use the SIMD technique [56] to amortize the cost of homomorphic multiplications. For instance, [27] 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 $\langle \hat{a} \rangle$-$OT_i$, 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 extension [33]). 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 [4]. More general 1-out-of-N OT $\langle \hat{a} \rangle$-$OT_i$ can be implemented in an IKNP-style OT extension, or with log$_2 n$ calls to $\langle \hat{1} \rangle$-$OT_i$ [47].

Recently, [10] 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 [60] and Silver [15]. With little communication cost, these VOLE-style OTs 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 $(x)A_2$ and $(x)B_2$. To reconstruct the value $x$, we compute the modulo addition, i.e., $x \equiv (x)A_2 + (x)B_2 \mod 2^\ell$. In the two-party setting, value $x \in \mathbb{Z}_{2^\ell}$ is secretly shared between Alice and Bob, by letting Alice hold the share $(x)A_2$ and letting Bob hold the share $(x)B_2$. 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 $\bar{x} \in \mathbb{R}$ is encoded as a fixed-point value $x = \lfloor \bar{x}2^f \rfloor \in \mathbb{Z}$ 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 $(x)A_2$ and $(x)B_2$ are the shares of a double-precision value, i.e., $\lfloor x2^{2f} \rfloor$. A faithful truncation protocol should take $(x)A_2$ and $(x)B_2$ as input and compute the shares of $\lfloor x2^f \rfloor \gg f$.

### 2.5 Local Truncation Trade-offs

The local truncation protocol of [46] introduces probability errors. For example, to perform the local truncation on $(x)A_2$ and $(x)B_2$, two share holders set $(x)′A_2 = \lfloor (x)A_2/2^f \rfloor$ and $(x)′B_2 = 2^f - \lfloor 2^f - (x)B_2/2^f \rfloor \mod 2^f$, respectively. Then the
value \( x' \) equals to \( |x^{2\ell}| + e_0 + e_1 \) with two probability errors where the small error \( |e_0| \leq 1 \) occurs at the chance of \( 1/2 \) and the harsh error \( |e_1| < 2^\ell \) occurs with a chance of \( x/2^\ell \).

Many previous 2PC-NN systems that leverage the local truncation will set a proper bit length \( \ell \) to avoid the harsh error as much as possible. For example, DELPHI [44] sets \( \ell = 41 \) which can translate a probability \( \epsilon \approx 2^{-19} \) of the \( e_1 \) error on their datasets. However, we consider a such \( \epsilon \) is still not small enough for DNNs because the number of local truncations is needed for vast for DNNs. For instance, the number of local truncations needed by the first convolution layer in ResNet50 is about \( 8.0 \times 10^5 \). In other words, the local truncation will introduce at least one \( e_1 \) error for this layer at the chance of \( 1 - (1 - \epsilon)^{8.0 \times 10^5} \) which is about 78% for DELPHI’s parameters. What’s worse, the large error(s) will propagate to the following layers and finally ruin the prediction. To decrease the error probability \( \epsilon \) a larger bitlength \( \ell \) is needed. This also renders a larger overhead on both of the linear protocols and non-linear protocols. This also somehow demonstrates why prior systems that leverage the local truncation might be limited to small datasets and shadow neural networks or introduce a large overhead.

### 2.6 Leakage from Ciphertext Noise

The (R)LWE ciphertexts have noise associated with them whose distribution changes along with homomorphic operations. This can lead to potential leakage through the noise, and reveal information particularly in the case of plaintext-ciphertext multiplication. To solve this issue, we can apply the noise flooding technique [21]. This technique adds a statistically independent noise from an interval \( B' \) much larger than \( B \), assuming that the ciphertext noise is bounded by \( B \) at the end of the computation.

We present a different approach to prevent the leakage from (R)LWE ciphertext noise by taking the advantage of secret sharing and our coefficient packings. Let \( CT = (\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2 \) be an RLWE ciphertext that decrypts to \( \hat{m} \in \mathbb{A}_{N,q} \). In our 2PC-NN setting, CT is usually computed by the server and is sent to the client for decryption. To hide the noise in CT, we suggest to send the ciphertext CT’ = \((\hat{b} + \hat{r} \text{ mod } q, \hat{a})\) to the client where \( \hat{r} \) is taken from \( \mathbb{A}_{N,q} \) uniformly at random. The distribution of the attached noise in CT’ becomes independent with the origin noise in CT. Let \( \hat{m}_B \) be the output from RLWE^{-1}_{sk}(CT’), and denote \( \hat{m}_A = -[\hat{m} \cdot \hat{r} / q \text{ mod } p] \). We will show that \( \hat{m}_A \) and \( \hat{m}_B \) is the arithmetic sharing of \( \hat{m} \) with at most 1-bit error, i.e., \( |\hat{m} - (\hat{m}_A + \hat{m}_B \text{ mod } p)| \leq 1 \). We defer the analysis of this re-sharing technique to appendix C.

### 2.7 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., \( z = f_d(f_{d-1}(\cdots(f_1(x, \mathbf{W}_1), \cdots), \mathbf{W}_{d-1}), \mathbf{W}_d) \) where \( f_d \) is the function evaluated in the \( d \)-th layer and \( \mathbf{W}_d \) 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 targeting, and then present how to evaluate those functions privately.

#### 2.7.1 Linear Layers

**Fully Connected Layer (FC).** The input to a fully connected layer is a vector \( v \in \mathbb{F}^n \) of length \( n_1 \) and its output is a vector \( u \in \mathbb{F}^n \) of length \( n_2 \). A fully connected layer is parameterized by the tuple \((\mathbf{W}, \mathbf{b})\) where \( \mathbf{W} \in \mathbb{F}^{n_2 \times n_1} \) is the weight matrix and \( \mathbf{b} \) is an \( n_2 \)-sized bias vector. The output is specified by the linear transformation \( u = \mathbf{W}v + \mathbf{b} \).

**Convolution Layer (CONV).** A two dimensional strided convolution \( \text{Conv2D}(\mathbf{T}, \mathbf{K}; \mathbf{s}) \) over a field \( \mathbb{F} \) operates on a 3-dimension tensor \( \mathbf{T} \in \mathbb{F}^{C \times H \times W} \) with a stride \( s > 0 \) and a set of kernels (also called filters) represented by a 4-dimension tensor \( \mathbf{K} \in \mathbb{F}^{M \times C \times h \times w} \) to generate a 3-dimension output tensor \( \mathbf{T}' \in \mathbb{F}^{M \times h' \times w'} \) where \( h' = [(H - h + s)/s] \) and \( w' = [(W - h + s)/s] \). If \( \mathbf{T} \) 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

\[
\mathbf{T}'[c', l', f] = \sum_{c, l, f'} \mathbf{T}[c, l's + l, f's + f'] \mathbf{K}[c', c, l, l'],
\]

for each position \((c', l', f')\) of the output tensor \( \mathbf{T}' \).

**Batch Normalization Layer (BN).** In DNNs, a BN layer \( \text{BN}(\mathbf{T}; \alpha, \beta) \) takes as input of 3-dimension tensor \( \mathbf{T} \in \mathbb{F}^{C \times H \times W} \) and output a 3-dimension tensor \( \mathbf{T}' \) 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 \( c \in [\lceil C \rceil], i \in [\lceil H \rceil] \) and \( j \in [\lceil W \rceil] \), \( \mathbf{T}' \) is computed via

\[
\mathbf{T}'[c, i, j] = \mu[c] \mathbf{T}[c, i, j] + \theta[c].
\]

By viewing each channel of \( \mathbf{T} \) as a \( HW \)-sized vector, we can naturally rewrite the BN evaluation to a form that involves scalar-vector multiplications and vector additions only.

#### 2.7.2 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
while Bob holds an input to the network, typically an image. O
write Alice and Bob to a two-party protocol say (Server) and Bob (Client). Let
of threat model in Appendix A. All other information about Bob’s
ice to learn only two pieces of information: the architecture
By assuming semi-honest Alice and Bob, our system enables
setting in which both of parties follow the specification of the
Flow2.

Suppose the secure inference is executed jointly by Alice
and input \( \langle x \rangle \) from Alice and input \( \langle y \rangle \) from Bob, compute \( u = Wv + b \).
Sample an uniform vector \( r \) from \( \mathbb{F}^n \).
Return: \( r \) to Alice and \( v - r \in \mathbb{F}^* \) to Bob.

(a) Ideal Functionality \( f_{\text{fc}} \)

Parameters: Shape meta \( n_i, n_o > 0 \).
Computation: On input \( \langle x \rangle \in \mathbb{F}^n \), \( W \in \mathbb{F}^{n_o \times n_i} \) and \( b \in \mathbb{F}^{n_o} \) from Alice and input \( \langle y \rangle \in \mathbb{F}^n \) from Bob, compute \( u = Wv + b \).
Sample an uniform vector \( r \) from \( \mathbb{F}^n \).
Return: \( r \) to Alice and \( v - r \in \mathbb{F}^* \) to Bob.

(b) Ideal Functionality \( f_{\text{conv}} \)

Parameters: Shape meta \( C, H, W \).
Computation: On input \( \langle x \rangle \in \mathbb{F}^{C \times H \times W} \), \( K \in \mathbb{F}^{M \times C \times H \times W} \) from Alice and input \( \langle y \rangle \in \mathbb{F}^{C \times H \times W} \) from Bob, compute \( T = \text{Conv2D}(T, K, s) \).
Sample an uniform tensor \( R \) from \( \mathbb{F}^* \) with the same shape of \( T \).
Return: \( R \) to Alice and \( T' - R \in \mathbb{F}^* \) to Bob.

(c) Ideal Functionality \( f_{\text{bn}} \)

Parameters: Shape meta \( C, H, W \).
Computation: On input \( \langle x \rangle \in \mathbb{F}^{C \times H \times W} \), \( \theta \in \mathbb{F}^C \) from Alice and input \( \langle y \rangle \in \mathbb{F}^{C \times H \times W} \) from Bob, compute \( T' = \text{BN}(T, \mu, \theta) \).
Sample an uniform tensor \( R \) from \( \mathbb{F}^* \) with the same shape of \( T' \).
Return: \( R \) to Alice and \( T' - R \in \mathbb{F}^* \) to Bob.

Figure 1: Ideal Functionalities of Linear Layers

2.7.3 Two-Party Inference System & Threat Model

Suppose the secure inference is executed jointly by Alice (Server) and Bob (Client). Let \( I_A \) and \( I_B \) be the private input of Alice and Bob to a two-party protocol say \( \Pi \), respectively. We write \( O_A, O_B \leftarrow \Pi(I_A, I_B) \) 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 CryptTFLOW2. Cheetah 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 [55,58]). Indeed, we can integrate orthogonal techniques such as differential privacy [2,34] 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 \( \mathbb{Z}_p \) such that \( p \equiv 1 \text{ mod } 2N \).

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 for computing the inner product using polynomial arithmetic. We observe that the polynomial multiplication (1) itself can be viewed as a batch of inner products. We now give the definitions of \( \pi_{fc}^w \) and \( \pi_{fc}^\ell \) 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 \( \pi_{fc}^w, \pi_{fc}^\ell \), we then show how to evaluate these functionalities privately. Also, \( \pi_{fc}^w \) and \( \pi_{fc}^\ell \) are well-defined for any \( p > 1 \) such as \( p = 2^l \), allowing our protocols to accept secretly shared input from the ring \( \mathbb{Z}/2^l \) 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 \( \pi_{fc}^w \) and \( \pi_{fc}^\ell \) are specialized 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 \( \pi_{fc}^w : \mathbb{Z}_p^{n_o \times n_i} \rightarrow \mathbb{F}_{N,p} \) and \( \pi_{fc}^\ell : \mathbb{Z}_p^\ell \rightarrow \mathbb{F}_{N,p} \). Here we first require \( n_o n_i \leq N \) for simplicity.

\[
\hat{v} = \pi_{fc}^\ell(v) \quad \text{where} \quad \hat{v}[j] = v[j] \quad \text{for} \quad j \in [n_o]
\]

\[
\hat{w} = \pi_{fc}^w(W) \quad \text{where} \quad \hat{w}[i \cdot n_i + n_i - 1 - j] = W[i, j], \quad \text{for} \quad i \in [n_o], j \in [n_i]
\]

where all other coefficients of \( \hat{v} \) and \( \hat{w} \) are set to 0. The multiplication of polynomials \( \hat{u} = \hat{v} \cdot \hat{w} \in \mathbb{F}_{N,p} \) directly gives the matrix–vector multiplication \( Wv \equiv \text{u mod } p \) in some of its coefficients.
Inputs & Outputs:
\[
\langle u \rangle^A, \langle u \rangle^B \leftarrow \text{HomFC} (\langle v \rangle^A, W, b); \langle v \rangle^B, sk)
\]
\[W \in \mathbb{Z}_p^{n \times n}, b \in \mathbb{Z}_p^n, \langle v \rangle^A \in \mathbb{Z}_p^n\] and \[u = Wv + b \in \mathbb{Z}_p^n\].

**Public Parameters:** \(pp = (\mathbb{HE}, pp, pk, n, n_o, n_w, n_{ow})\).

- Shape meta \(n_i, n_o\) such that \(n_i, n_o > 0\). The partition window size \(0 < n_i < n_o \leq n_i, 0 < n_o \leq n_i\) such that \(n_w n_{ow} \leq N\).

- Set \(n_i' = \lfloor n_i / n_w \rfloor\) and \(n_o' = \lfloor n_o / n_w \rfloor\).

1. Bob first partitions its input shares \(\langle v \rangle_A\) into subvectors \(\langle v_\alpha \rangle_A \in \mathbb{Z}_p^{n_w}\) for \(\alpha \in [n_i']\). Zero-padding them when \(n_i \nmid n_w\).

2. Bob encodes the vectors to polynomials \(\langle v_\alpha \rangle_A = \pi_{\text{fc}}^n (\langle v_\alpha \rangle_A)\) for \(\alpha \in [n_i']\). Then Bob sends the RLWE ciphertexts \((CT_{\alpha} = \text{RLWE}_{pk}^{N, d} (\langle v_\alpha \rangle_A))\) to Alice.

3. Similarly, Alice first partitions its shares \(\langle v \rangle_A^A\) into \(\langle v_\alpha \rangle_A^A \in \mathbb{Z}_p^{n_w}\) following the same manner in Step 1. Also, Alice partitions the weight matrix \(W\) into block matrices \(W_{\alpha, \beta} \in \mathbb{Z}_p^{n_w \times n_w}\). Zero-padding is used to the right-most (resp. bottom-most) blocks when \(n_i \nmid n_w\) (resp. \(n_o \nmid n_w\)). Then Alice encodes the subvectors to polynomials \(\langle v_\alpha \rangle_A = \pi_{\text{fc}}^n (\langle v_\alpha \rangle_A)\) for \(\alpha \in [n_i']\) and \(\beta \in [n_o']\).

4. On receiving the ciphertexts \((CT_{\alpha}^B)\) from Bob, Alice operates \(CT_{\beta} = \oplus_{\alpha \in [n_i']} (CT_{\alpha}^B \oplus (\langle v_\alpha \rangle_A^A) \oplus \langle v_\beta \rangle_{\beta, \alpha} = \pi_{\text{fc}}^n (W_{\alpha, \beta}^A))\).

5. Alice samples \(r \in \mathbb{Z}_p^n\) uniformly at random to remask the computing ciphertexts. Specifically, for each \(i \in [n_i]\), Alice first extracts an LWE ciphertext \(ct_i = \text{Extract}(CT_{\text{mod} n_i} \cdot (\text{mod} n_i) n_i + n_i - 1)\) and then adds \(r_i\) to the 1st component of \(ct_i\) and results at \(ct_i^r\). Alice then sends the LWE ciphertexts \((ct_i^r)\) back to Bob.

6. Alice outputs \(b - p \cdot r / q \mod p\) as the share \(u_i\).

7. On receiving the LWE ciphertexts \((ct_i^r)\) from Alice, Bob computes \(\langle u \rangle_B = \text{LWE}_{sk}^{n_w} (ct_i^r)\) for all \(i \in [n_i]\).

---

**Proposition 1** Given two polynomials \(\hat{v} = \pi_{\text{fc}}^n (v), \hat{w} = \pi_{\text{fc}}^n (W) \in \mathbb{Z}_p^{n \times n},\) the multiplication \((Wv + b)\) can be evaluated via the product \(\hat{u} = \hat{v} \cdot \hat{w}\) over the ring \(\mathbb{Z}_p^{n \times n}\). That is \(u[i]\) is computed in \(\hat{u}[i \cdot n_i + n_i - 1]\) for all \(i \in [n_i]\). \(\square\)

**Proof** For each \(i \in [n_i]\), we write \(n_i = i \cdot n_i + n_i - 1\) for simplicity. By the definition of (1) and because \(\hat{v}[j] = 0\) for \(j \geq n_i\), we have \(\hat{u}[n_i] = \sum_{0 \leq j < n_i} \hat{v}[j] \hat{w}[n_i - j] = \sum_{0 \leq j < n_i} v[j] W[i][j]\) which is exactly \(u[i]\). \(\square\)

In Proposition 1, other coefficients besides \(\hat{u}[n_i]\) 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 \(n_i n_o > N\), we can first partition the weight matrix into sub-matrices of \(0 < n_i \times n_o \) elements such that \(n_i n_o \leq N\). \(n_i\) and \(n_o\) can be chosen freely as public parameters as long as they satisfy this constraint. Zero-padding is used when \(n_i \nmid n_i\)

---

**Toy example over \(\mathbb{Z}_2^5\):**

\[
W = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, v = \begin{bmatrix} 7 \end{bmatrix} \Rightarrow Wv \equiv \begin{bmatrix} 18 \end{bmatrix} \mod 2^5
\]

---

**Cheetah evaluates \(Wv\) using \(\pi_{\text{fc}}^n\) and \(\pi_{\text{fc}}^n\):**

\[
\pi_{\text{fc}}^n (W) \rightarrow \hat{w} = 3X^0 + 2X^1 + 1X^2 + 6X^3 + 5X^4 + 4X^5 \in \mathbb{Z}_2[X]
\]

\[
\pi_{\text{fc}}^n (v) \rightarrow \hat{v} = 7X^0 + 8X^1 + 9X^2 \in \mathbb{Z}_2[X]
\]

\[
\downarrow \hat{w} \cdot \hat{v} \mod (X^8 + 1.2^5)
\]

\[
\hat{v} \cdot \hat{v} \equiv 21X^0 + 6X^1 + 18X^2 + 4X^3 + 28X^4 + 26X^5 + 13X^6 + 4X^7 \mod (X^8 + 1.2^5)
\]

\[
\downarrow \text{Extract needed coefficients}
\]

**Figure 3:** Toy example for \(\pi_{\text{fc}}^n\) and \(\pi_{\text{fc}}^n\) with \(N = 8\) and \(p = 2^5\).

---

**Theorem 1** The protocol HomFC in Figure 2 realizes the ideal functionality \(f_{\text{fc}}\) of Figure 1a for \(p = \mathbb{Z}_p\) in presence of a semi-honest admissible adversary.

We defer the proof to Appendix E due to the space limit. For the complexity, Bob encrypts and sends \(n_i\) RLWE ciphertexts to Alice. Alice operates with \(n_i n_o = O(n_i n_o N)\) homomorphic multiplications and additions. Alice sends \(n_o\) RLWE ciphertexts to Bob for decryption which can be compressed to about \(O(n_i n_o N \log_2 q)\) bits using the optimizations in § 5.2.

---

### 3.2 Two Dimensional Convolution

We now present how to evaluate the two dimensional convolution (2) using polynomial arithmetic over the ring \(\mathbb{Z}_p^{n \times n}\). 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 vectors by concatenating \(T\) and \(K\) 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 \(\pi_{\text{conv}} : \mathbb{Z}_p^{C \times H \times W} \rightarrow \mathbb{Z}_p^{N_{i}}\) and \(\pi_{\text{conv}} : \mathbb{Z}_p^{M \times C \times H \times W} \rightarrow \mathbb{Z}_p^{N_{i}}\). Here we require \(MCHW \leq N\) for simplicity.

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

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

where \(O = HW(MC - 1) + W(h - 1) + h - 1\), and all other coefficients of \(\hat{t}\) and \(\hat{k}\) are set to 0. The multiplication \(\hat{t} \cdot \hat{k}\) directly gives the 2-dimension convolution in some of the coefficients of the resulting polynomial.
We need to partition large input tensors and kernels so that we defer the correctness and security proofs to Appendix E.

Theorem 2 The protocol HomCONV in Figure 4 realizes the ideal functionality $f_{\text{CONV}}$ 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 E.

3.2.1 On the Cases of Large Tensors

We need to partition large input tensors and kernels so that each of the smaller blocks can be fit into one polynomial in $\mathbb{A}_{N,p}$. Particularly, we define the size of partition window for the $(M,C,H,W)$-axis as $M_w, C_w, H_w$ and $W_w$, respectively. The size of the partition windows can be chosen freely as long as they satisfy the following constraints:

\[
0 < M_w \leq M, \quad 0 < C_w \leq C, \quad h \leq H_w \leq H, \quad h \leq W_w \leq W.\]

For instance, in our experiments, to minimize the number of ciphertexts sent by Bob, we choose the window sizes $H_w$ and $W_w$ that minimize the product $\lfloor C/H_w \rfloor \cdot \lfloor H/H_w \rfloor \cdot \lfloor W/H_w \rfloor$. When $H_w$ and $W_w$ are specified, the partition window sizes along the $C$-axis and $M$-axis is $C_w = \min(C, \lfloor N/(H_w W_w) \rfloor)$ and $M_w = \min(M, \lfloor N/(C_w H_w W_w) \rfloor)$, respectively.

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 convolution into $d_M = \lceil M \rceil$ groups and each of them contains $M_w$ sub-kernels. 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 over-padding 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 the $H$-axis and $W$-axis into $d_H = \lceil H/H_w + 1 \rceil$ and $d_W = \lceil W/H_w + 1 \rceil$ blocks, respectively. For the sub-block indexed by $\alpha(H_w - h + 1)$-th row, and exact $W_w$ continuous columns that starts from the $\beta(W_w - h + 1)$-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.
3.2.2 HomCONV (Full Protocol)

The full version of HomCONV is given in Appendix D 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 $O(\frac{CHW}{h^2}\cdot HW)$ LWE ciphertexts to Bob for decryption. The computation complexity of HomCONV is independent with the kernel size $h$ where the previous secure convolution protocols [11, 36, 42] 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 $O(C[HW/N])$ ciphertexts. We can reduce this communication cost by “stacking” multiple channels of the tensor into a single polynomial. For example, when $C^2HW \leq N$, we can even put all the channels into one polynomial. We now give the definitions of $\pi_{\text{bn}}^w: \mathbb{Z}_p^{CHW} \mapsto \mathbb{A}_{N,p}$ and $\pi_{\text{bn}}^w: \mathbb{Z}_p^C \mapsto \mathbb{A}_{N,p}$. Here we demand $C^2HW \leq N$ for simplicity.

$$\hat{t} = \pi_{\text{bn}}(t) \text{ s.t. } \hat{t}[cCHW + iH + j] = T[c,i,j]$$

$$\hat{a} = \pi_{\text{bn}}^w(\alpha) \text{ s.t. } \hat{a}[CHW] = \alpha[c]$$

where $c \in [C]$, $i \in [H]$ and $j \in [W]$. All other coefficients of $\hat{t}$ and $\hat{a}$ are set to zero. The polynomial product $\hat{t} \cdot \hat{a}$ gives the multiplication part of (3) in some of coefficients of $\hat{p}$. That is $\hat{p}[cCHW + cCHW + iH + j]$ equals to $T[c,i,j] \alpha[c]$ for all $(c,i,j)$ position. When $C^2HW > N$, we can first partition $T$ into sub-blocks in a shape of $C_w \times H_w \times W_w$ such that $C_w^2H_wW_w \leq N$. Since each channel is proceed independently in the BN computation, we can just apply the mapping functions $\pi_{\text{bn}}^w, \pi_{\text{bn}}^w$ to the sub-blocks of $T$.

Theorem 3 The protocol HomBN in Figure 6 realizes the ideal functionality $f_{\text{BN}}$ of Figure 1c for the field $F = \mathbb{Z}_p$ in presence of a semi-honest admissible adversary.

We defer the proof to Appendix D due to the space limit. For the complexity, Bob encrypts and sends $O(CHW/N)$ RLWE ciphertexts to Alice. Alice operates with $O(CHW/N)$ homomorphic operations. Finally, Alice sends $O(CHW)$ LWE ciphertexts to Bob for decryption.

4 Optimized 2PC Protocols of Non-Linear Functions

For non-linear computation, [50] 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 $(\cdot)\cdot\text{OT}_t$, $(\cdot)\cdot\text{OT}_r$ and $(\cdot)\cdot\text{OT}_\ell$. We instantiate $(\cdot)\cdot\text{OT}_t$ with the algorithm proposed by Naor and Pinkas [47] by using $\log_2 n$ calls to
We briefly recall the high-level intuition of the millionaires’ protocol in stage 2, [50] presents various optimizations to realize the functionality that accepts boolean shares of two layers, such as

\[
\{\text{Millionaires’ protocol for comparing two integers (}\lt\text{)}\}
\]

\[
\{\text{neural network inference (usually, } n = 16, \ell = 64, \text{ which is the case in this work). Nonetheless, as shown by [16], the last-bit small error does not affect the quality of prediction models in machine learning which we also experimentally verify in a concrete neural network (see § 6.4). We observe that by removing the constraint of correcting the last-bit error, the truncation protocol can be made far more lightweight, providing significant improvement in communication and computation.}
\]

**4.1 Leaner and Better Millionaires’ Protocol**

Millionaires’ protocol for comparing two integers \((x, y \in \{0, 1\}^\ell)\) 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 \(x\) and \(y\) and returns boolean shares of \(x \wedge y\) (protocol details in Appendix G).

We briefly recall the high-level intuition of the millionaires’ protocol in [50]:

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 \(\binom{m}{1}\cdot\text{OT}_1\) to compute a boolean share of \(1\) if \(y_j > x_j\) \{similar invocation for \(\text{eq}_j = 1\{x_j = y_j\}\} \{\text{invariant for } f_{\text{AND}}\} \).

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, [50] presents various optimizations to realize \(f_{\text{AND}}\) efficiently in the context of IKNP-style OT extension, such as using \(\binom{n}{1}\cdot\text{OT}_1\) to generate two beaver triples and \(\binom{n}{1}\cdot\text{OT}_2\) to generate correlated triples. However, in Cheetah, these are less efficient than using \(\binom{\ell}{1}\cdot\text{ROT}\) to generate the beaver triples [4]. 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 \((m^A_2, m^B_2) = \text{Mill}^\ell(x, y)\) denote the millionaires’ protocol where Alice’s input is \(x\), Bob’s input is \(y\), and they receive boolean shares as output: \(m^A_2 \oplus m^B_2 = 1\{x > y\}\).

### Table 3: Communication complexity of millionaires’ protocols of \(\ell\)-bit integers.

<table>
<thead>
<tr>
<th>Millionaires’ Protocol</th>
<th>Communication (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrypTFlow2 ((m = 4, \text{IKNP}, \binom{n}{1}\cdot\text{OT}))</td>
<td>(&lt; \ell \lambda + 14\ell)</td>
</tr>
<tr>
<td>Naive ((m = 1, \text{VOLE}, \binom{n}{1}\cdot\text{ROT}))</td>
<td>(&lt; 16\ell)</td>
</tr>
<tr>
<td>Cheetah ((m = 4, \text{VOLE}, \binom{n}{1}\cdot\text{ROT}))</td>
<td>(&lt; 11\ell)</td>
</tr>
</tbody>
</table>

\((\binom{n}{1}\cdot\text{ROT})\), whereas CrypTFlow2 implements it with the IKNP-style OT extension proposed in [37]. Due to the usage of VOLE-style OT, our calls to \((\binom{n}{1}\cdot\text{ROT})\) 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, \(n \leq 16, \ell \leq 2\)).

### 4.2 Approximate Truncation

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

#### 4.2.1 One-Bit Approximate Truncation

In prior works [44–46], local truncation is used to approximate \(f_{\text{Trunc}}^{\ell, f}\), 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 [16], the last-bit small error does not affect the quality of prediction models in machine learning which we also experimentally 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 communication and computation.

**Proposition 3** Given an unsigned \(\ell\)-bit integer \(x\) and its arithmetic shares \(x_0\) and \(x_1\), define \(w = 1\{x_0 + x_1 > 2^f - 1\}\). Let \(c\) be one-bit integer \(0\) or \(1\). We have: \((x_0 \gg f) + (x_1 \gg f) - w \cdot 2^{\ell-f} = (x \gg f) - c\).

**Proof 2** The proposition follows from Corollary 4.2 in [50]. First, \(w\) denotes whether the sum of two shares wrap around \(2^\ell\). If so, there will be large error for local truncation and it should be corrected by subtracting \(2^\ell\). \(c\) 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 \(2^\ell\).

Let \(f_{\text{Trunc}}^{\ell, f}\) denote the functionality that accepts boolean shares of \(x \in \{0, 1\}\) and returns arithmetic shares of \(x\) in \(Z_{2^\ell}\). We provide an implementation of \(f_{\text{Trunc}}^{\ell, f}\) in Appendix G. The one-bit approximate truncation protocol is provided in Figure 8.
Inputs and Outputs: \((z)_2^A, (z)_2^B \leftarrow \text{Trunc}(\langle x \rangle_2^A, \langle x \rangle_2^B, f)\) such that \(x \in \mathbb{Z}_2\), and \(z = x \gg f\) or \(z = (x \gg f) - 1\).

1. Alice and Bob call the millionaires’ protocol: \(((w)_2^A, (w)_2^B) = \text{Mill}(\langle x \rangle_2^A, 2^f - 1 - \langle x \rangle_2^B)\).
2. Alice and Bob invoke an instance of \(f_{\text{B2A}}^{2f}\) with inputs \(((w)_2^A, (w)_2^B)\) and learn \(((w)_2^A, (w)_2^B)\).
3. Alice computes \((z)_2^A = (\langle x \rangle_2^A \gg f) - (w)_2^B \cdot 2^{-f-f}\). Bob computes \((z)_2^B = (\langle x \rangle_2^B \gg f) - (w)_2^B \cdot 2^{-f-f}\).

### 4.2.2 Approximate Truncation with Known MSB

As pointed out by [49], 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 \(\left(\frac{1}{2}\right)\)-OT\(_1\) instead of the expensive Mill protocol. Indeed, when MSB is zero, \(w = \text{mab}(\langle x \rangle^A) \lor \text{mab}(\langle x \rangle^B)\). A similar protocol can also be derived when MSB is one and \(w = \text{mab}(\langle x \rangle^A) \land \text{mab}(\langle x \rangle^B)\).

### 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 \(f_{\text{Mill}^{1}}^{1}\), \(\left(\frac{1}{2}\right)\)-OT\(_1\), \(f_{\text{B2A}}^{2}\), and \(f_{\text{Mill}}^{2}\). This leads to a communication upper bound of \(k(\ell + f + 2) + 19\ell + 14f\) 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 \(16\ell + 11f\) bits, according to our complexity analysis in Table 2 and 3 (\(f_{\text{B2A}}^{2}\) uses a single call of \(\left(\frac{1}{2}\right)\)-CDT\(_1\)). One-bit approximate truncation involves a call to \(f_{\text{Mill}}^{1}\) and \(f_{\text{B2A}}^{2}\), 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 [7, 36, 44, 50]. 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 \((b, a) \in \mathbb{Z}_N^{N+1}\) 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 \(\ell_a = \lfloor \log_2(q/\sqrt{Np}) \rfloor\) 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 Appendix F.
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. We measure the total communication including all the messages sent by Alice and Bob but excluding the one-time setup (e.g., keys generation and base-OT). We measure the performance of DNN Architectures.

### 6 Evaluations

#### 6.1 Experiment Setup

Our implementation is built on top of the SEAL library [53] with the HEXL acceleration [32] and the EMP toolkit [59]. We also extend the Ferret protocol in EMP to support various application-level OT types, such as \( \binom{N}{1} \)-OT\(_{1}\). 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 [36,44], and the matrix–vector multiplication from [13,27] 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.

**Testbed Environment.** All the following experiments were performed on Alibaba Cloud ecs.c7.2xlarge instances with a 2.70 GHz processor and 16 gigabytes of RAM. All our programs are implemented in C++ and compiled by gcc-8.4.0. 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 \( HE.pp_{cheetah} = \{N = 4096, q \approx 2^{105}, p = 2^{37}, \sigma = 3.2\} \). Recall that the current secure convolution protocol in \( SCI_{HE} \) [52] demands \( HW/s^2 \leq N/2 \). For most of the time, it can be instantiated using the SEAL parameters \( HE.pp_{pSCI} = \{N = 8192, q \approx 2^{180}, p \approx 2^{37}, \sigma = 3.2\} \). 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. For instance, the ResNet50 is trained to classify RGB images of 224 × 224 pixels into 1001 classes. The ResNet50 has over 23 million trainable parameters. It consists of 53 CONV layers, 49 BN layers and 1 FC layer. For the non-linear operations, ResNet50 consists of 49 ReLU, 98 truncation, 1 max-pooling, 1 average-pooling and 1 argmax.

#### 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\(^3\), depending on the input size.

Moreover, our linear protocols can accept additive shares from the ring \( Z_{p^k} \) that is more friendly for subsequent non-linear layers, while prior approaches \([27,36,44,50]\) that leverage the homomorphic rotation could only accept shares from a prime filed \( Z_p \). To accept shares from \( Z_{p^k} \), they need to

---

\(^3\)The only exception is the FC layer, where our communication cost may be 1.33MB higher than theirs.

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 [50]</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>

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></th>
<th>FC ( n_i, n_o )</th>
<th>( \text{End2End Time} )</th>
<th>( \text{Comm.} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>[50,52]</td>
<td>2048,1001</td>
<td>1.533ms</td>
<td>1.587ms</td>
</tr>
<tr>
<td></td>
<td>512,10</td>
<td>17ms</td>
<td>60ms</td>
</tr>
<tr>
<td>Ours</td>
<td>2048,1001</td>
<td>111ms</td>
<td>171ms</td>
</tr>
<tr>
<td></td>
<td>512,10</td>
<td>4ms</td>
<td>51ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CONV ( HW , C, M, h,s )</th>
<th>( \text{End2End Time} )</th>
<th>( \text{Comm.} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>[50,52]</td>
<td>224(^2),3,64,7,2</td>
<td>25.52s</td>
<td>27.28s</td>
</tr>
<tr>
<td></td>
<td>56(^2),64,256,1,1</td>
<td>8.21s</td>
<td>8.74s</td>
</tr>
<tr>
<td>Ours</td>
<td>224(^2),3,64,7,2</td>
<td>1.30s</td>
<td>2.11s</td>
</tr>
<tr>
<td></td>
<td>56(^2),64,256,1,1</td>
<td>0.83s</td>
<td>1.10s</td>
</tr>
<tr>
<td></td>
<td>56(^2),256,64,1,1</td>
<td>0.70s</td>
<td>0.99s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BN ( HW , C )</th>
<th>( \text{End2End Time} )</th>
<th>( \text{Comm.} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>[50,52]</td>
<td>56(^2),64</td>
<td>0.28s</td>
<td>0.50s</td>
</tr>
<tr>
<td></td>
<td>56(^2),256</td>
<td>1.14s</td>
<td>2.17s</td>
</tr>
<tr>
<td>Ours</td>
<td>56(^2),64</td>
<td>0.22s</td>
<td>0.34s</td>
</tr>
<tr>
<td></td>
<td>56(^2),256</td>
<td>0.88s</td>
<td>1.36s</td>
</tr>
</tbody>
</table>

**Metrics.** We measure the total communication including all the messages sent by Alice and Bob but excluding the one-time setup (e.g., keys generation and base-OT). We measure the end-to-end running time including the time of transferring messages through the LAN/WAN but we rule out the time for HE key generation and base-OT.
apply the Chinese Remainder Theorem to expand the plaintext modulus to above \((2\ell + 1 + 40)\) bits for achieving 40-bit statistical security [19]. 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 SCIHE 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× 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 [44], one of the state-of-the-art 2PC-NN systems. Similar to DELPHI, we perform these computations on the CIFAR dataset [39] 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 [5] 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.

### 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 [31], ResNet50 [28], and DenseNet121 [30] within 3 minutes even under the WAN setting. The end-to-end running time of Cheetah was about \(2 \times 5\times\) faster than SCIHE within 8% bandwidth consumption of SCIHE. 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.

### Table 6: Comparing the running time and communication costs of our non-linear protocols with the state-of-the-art. All runs were executed using single thread. The communication and timing are accumulated for \(2^{16}\) runs of the protocols.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Method</th>
<th>End2End Time</th>
<th>Commu.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>Millionaire</td>
<td>Ours [50]</td>
<td>572ms</td>
<td>1,680ms</td>
</tr>
<tr>
<td>Truncation</td>
<td>Ours [50]</td>
<td>544ms</td>
<td>1,914ms</td>
</tr>
<tr>
<td>Trunc. with</td>
<td>Ours [50]</td>
<td>213ms</td>
<td>716ms</td>
</tr>
<tr>
<td>known MSB</td>
<td></td>
<td>31ms</td>
<td>101ms</td>
</tr>
</tbody>
</table>

### Table 7: Performance comparison with DELPHI [44, Figure 13] in the LAN setting. Inputs to the benchmarks were RGB images of \(32 \times 32\) pixels. Aligned parameters of sharing modulus 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></td>
<td>DELPHI [44]</td>
<td>110s</td>
<td>3.6GB</td>
</tr>
<tr>
<td></td>
<td>Cheetah</td>
<td>3.55s</td>
<td>0.03GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30×</td>
<td>117×</td>
</tr>
<tr>
<td>ResNet32</td>
<td>Cheetah</td>
<td>200s</td>
<td>6.5GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12×</td>
<td>58×</td>
</tr>
</tbody>
</table>

### Table 8: Performance comparison with SCIHE and SecureQ8 (three-party) on large-scale DNNs. We used the precision \(f = 12\) for fixed-point values and 4 threads for the benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>System</th>
<th>End2End Time</th>
<th>Commu.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCIHE [50]</td>
<td>147.2s</td>
<td>5.9GB</td>
</tr>
<tr>
<td></td>
<td>SecureQ8 [16]</td>
<td>134.1s</td>
<td>0.8GB</td>
</tr>
<tr>
<td>SqNet</td>
<td>Cheetah</td>
<td>16.0s</td>
<td>0.5GB</td>
</tr>
<tr>
<td></td>
<td>SCIHE [50]</td>
<td>759.1s</td>
<td>29.2GB</td>
</tr>
<tr>
<td></td>
<td>SecureQ8 [16]</td>
<td>379.2s</td>
<td>3.8GB</td>
</tr>
<tr>
<td></td>
<td>Cheetah</td>
<td>80.3s</td>
<td>2.3GB</td>
</tr>
<tr>
<td>NW50</td>
<td>SecureQ8 [16]</td>
<td>342.6s</td>
<td>4.6GB</td>
</tr>
<tr>
<td></td>
<td>Cheetah</td>
<td>79.3s</td>
<td>2.4GB</td>
</tr>
</tbody>
</table>

\(\text{SqNet} = \text{SqueezeNet}; \text{RN50} = \text{ResNet50}; \text{DNet} = \text{DenseNet121}\)
faster on WAN by the virtue of less communication \(^4\).

### 6.5 Effective Approximate Truncation

To demonstrate the effectiveness of our approximate truncation protocols, we performed more predictions on SqueezeNet using an image set \([1]\) that includes about 1,000 images from ImageNet. For all these images, Cheetah outputs the same classification label as SCI\(\text{HE}\). 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\(\text{HE}\) and Cheetah on ResNet50 and SqueezeNet. Here we can see that Cheetah gives almost the same prediction vectors as SCI\(\text{HE}\) which are bit-wise equivalent to the plaintext fixed-point computation. Also, according to the extensive experiments in CrypTFlow2 \([50, Table 10]\), 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.

**Availability.** Cheetah is available from [https://github.com/Alibaba-Gemini-Lab/OpenCheetah](https://github.com/Alibaba-Gemini-Lab/OpenCheetah).

**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 [12, 24, 41]. 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 \( \mathcal{E} \) 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 \( \tilde{\mathcal{A}} \) (called the simulator) in the ideal interaction, such that no environment \( \mathcal{E} \) can distinguish between real and ideal interactions.

We recap the definition of a cryptographic inference protocol in DELPHI [44]. The server holds a model \( \mathcal{W} \) 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 \( \mathcal{W} = (W_1, \ldots, W_d) \) 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 \( \mathcal{W} \) 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 output \( \mathcal{W}(x) \).

- **Privacy.** We require that a corrupted, semi-honest client does not learn anything about the server’s network parameters \( \mathcal{W} \). Formally, we require the existence of an efficient simulator \( \text{Sim}_{\mathcal{C}} \) such that \( \text{View}_C^\Pi \approx_e \text{Sim}_C(\text{meta, out}) \) where \( \text{View}_C^\Pi \) is the view of the client in the execution of \( \Pi \), meta includes the meta information (i.e., the public parameters \( \text{HE.p, pp, the public key pk, the number of layers, the size and type of each layer, and the activation} \) 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 \( \text{Sim}_{\mathcal{S}} \) such that \( \text{View}_S^\Pi \approx_e \text{Sim}_S(\text{meta}) \) where \( \text{View}_S^\Pi \) is the view of the server in the execution of \( \Pi \).

B SIMD and Homomorphic Rotation

The SIMD technique [56] 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 \( \hat{v} \in \mathbb{Z}_p \) and \( \hat{u} \in \mathbb{Z}_p \), respectively. The product polynomial \( \hat{v} \cdot \hat{u} \in \mathbb{Z}_p \) 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 \( \text{RLWE}^N_{sk}(\mathcal{P}^k \mod 2N(\mathcal{S})) \) where the automorphism \( \mathcal{P}^k \colon \mathbb{Z}_{N,p} \mapsto \mathbb{Z}_{N,p} \) is defined by \( \mathcal{P}^k(\hat{a}(X)) = \hat{a}(X^k) \mod X^N + 1 \).

C Solving Leakage from Ciphertext Noise by One-bit Approximate Re-sharing

Suppose \( \text{CT} = (\hat{b}, \hat{a}) \in \mathbb{Z}_p^2 \) be an RLWE ciphertext that decrypts to \( \hat{m} \in \mathbb{Z}_p \) and attaches with a noise polynomial \( \hat{e} \). More specifically, to reveal the attached noise, the decryptor first computes the phase \( \hat{b} + \hat{a} \cdot \text{sk} \mod q \) and then computes \( p \cdot (\hat{b} + \hat{a} \cdot \text{sk}) \mod q \) which equals to \( p \cdot \hat{e} \mod q \) where \( \hat{e} \) is the attached noise in CT. When CT is generated by homomorphic multiplication, the polynomial \( \hat{e} \) contains information about the multiplier which should be hidden from the view of decryptor. To prevent such leakage, we suggest to use a re-sharing method. We first sample a uniform polynomial \( \hat{r} \) from \( \mathbb{Z}_{N,q}^p \) and then update the ciphertext as \( \text{CT}' = (\hat{b} + \hat{r} \mod q, \hat{a}) \). As a result, from the view of decryptor, the phase \( \hat{b} + \hat{r} + \hat{a} \cdot \text{sk} \mod q \) distributes uniformly in \( \mathbb{Z}_{N,q}^p \). Thus no information of \( \hat{e} \) is leaked from \( \text{CT}' \).

Our re-sharing method will introduce 1-bit error in the arithmetic sharing. Let \( \hat{m}_B = \lceil p \cdot (\hat{b} + \hat{r} + \hat{a} \cdot \text{sk} \mod q) \rceil \mod p \), that is the output from \( \text{RLWE}_{sk}^{-1} \) (CT') and \( \hat{m}_A = -\lfloor p \cdot \hat{r} / q \rfloor \mod p \). We have \( \hat{m}_A + \hat{m}_B \equiv \hat{m} + \lfloor p \cdot (\hat{e} + \hat{r}) / q \rfloor \mod p \). In other words, our re-sharing approach will introduce an error when \( \lfloor p \cdot (\hat{e} + \hat{r}) / q \rfloor \neq \lfloor p \cdot \hat{r} / q \rfloor \). To analyze the magnitude of this error, we first rewrite \( \hat{r} \) into three parts that is \( \hat{r} = q \cdot \hat{r}_p / p + \hat{r}_l + \hat{e} \) such that \( \hat{r}_p[i] \in \mathbb{Z}_{p^2}, \hat{r}_l[i] \in \mathbb{Z}_{q[p]} \) and \( \hat{e} \in \{-1/2, 0, 1/2\} \) for all positions. Then the polynomial \( \lfloor p \cdot (\hat{e} + \hat{r}) / q \rfloor = r_h + \lfloor p \cdot (\hat{e} + \hat{r}_l + \hat{e}) / q \rfloor \) and the polynomial \( \lfloor p \cdot \hat{r} / q \rfloor = r_h + \lfloor p \cdot (\hat{r}_l + \hat{e}) / q \rfloor \). By assuming \( q \gg p \) and \( |\hat{e}| < q/2^p \), we have

\[
-1 = \left\lfloor \frac{1}{2} - \frac{p}{2q} \right\rfloor \leq \frac{p \cdot (\hat{e} + \hat{r}_l + \hat{e}) / q}{q} \leq \frac{1}{2} + 1 + \frac{p}{2q} = 2.
\]

\[
0 = \left\lfloor \frac{p}{2q} \right\rfloor \leq \frac{p \cdot (\hat{r}_l + \hat{e}) / q}{q} \leq \frac{1 + p}{2q} = 1.
\]

In other words, our re-sharing approach will introduce at most 1-bit error, i.e., \( |\hat{m}_A + \hat{m}_B \mod p - \hat{m}| \leq 1 \). Also the chance of such 1-bit error is a function of the noise magnitude:

\[
\Pr \left( \left\lfloor \frac{p \cdot (\hat{r}_l + \hat{e}) / q}{q} \right\rfloor \neq \frac{p \cdot (\hat{r}_l + \hat{e}) / q}{q} \right) = \frac{4}{q} |\hat{e}| / q
\]

which is bounded by \( 2/p \), e.g., less than \( 2^{-36} \) in our experiments.

We note that our re-sharing approach can not be used with the SIMD [56] since the Fourier transform over the prime
We now present the full version of encrypts one partition block of tensor (with zero-padding) into smaller blocks of the same size.

In Step 1 and Step 2 of Figure 11, Bob first partitions its share into blocks along the \( C \)-axis and \( M \)-axis is \( C_w = \min(C, \lfloor N \div H_w \rfloor) \) and \( M_w = \min(M, \lfloor \frac{N}{C_w H_w} \rfloor) \).

1. Bob first partitions \( \langle T \rangle^B \) into blocks along the \( H \)-axis and \( W \)-axis \( \langle T_{\alpha, \beta} \rangle^B \in \mathbb{Z}_p^{C \times H_w \times W_w} \) for \( \alpha \in [[d_H]] \) and \( \beta \in [[d_W]] \). Each block \( \langle T_{\alpha, \beta} \rangle^B \) consists of \( H_w \) continuous row blocks 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 \( \langle T_{\gamma, \alpha, \beta} \rangle^B \in \mathbb{Z}_p^{C_w \times H_w \times W_w} \) for \( \gamma \in [[d_C]] \). 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 \( \langle C_{T_{\gamma, \alpha, \beta}} \rangle^B \) to Alice.

4. Alice partitions \( \langle T \rangle^A \) into blocks \( \langle T_{\gamma, \alpha, \beta} \rangle^A \in \mathbb{Z}_p^{C_w \times H_w \times W_w} \) following the same manner in Step 1 and Step 2. Then Alice encodes each block to a polynomial via \( \langle \hat{t}_{\gamma, \alpha, \beta} \rangle^A = \pi_{\text{enc}}^w(\langle T_{\gamma, \alpha, \beta} \rangle^A) \).

5. Alice then splits \( K \) into sub-kernels along the \( M \)-axis, i.e., \( K_0 \in \mathbb{Z}_p^{M_w \times C \times h \times h} \) for \( \theta \in [[d_M]] \). Then Alice further split each \( K_0 \) into smaller block along the \( C \)-axis, i.e., \( K_{\alpha, \gamma} \in \mathbb{Z}_p^{M_w \times C_w \times h \times h} \) for \( \gamma \in [[d_C]] \). Zero-padding might be used to make sure all blocks contains the same number of elements. Then Alice encodes these block tensors into polynomials \( \hat{k}_{\theta, \gamma} = \pi_{\text{enc}}^w(K_{\theta, \alpha, \gamma}, O_w) \).

6. Alice samples \( R \) from \( \mathbb{Z}_q^{M_w \times H' \times W'} \) uniformly at random and outputs \( \langle T \rangle^A = -[p \cdot R \mod q] \).

7. On receiving \( \{ C_{T_{\theta, \alpha, \beta}} \} \), Alice computes \( C_{T_{\theta, \alpha, \beta}} = \oplus_{\theta \in [[d_M]]}(C_{T_{\theta, \gamma, \alpha, \beta}} \oplus \langle \hat{t}_{\gamma, \alpha, \beta} \rangle^A \otimes \hat{k}_{\theta, \gamma} \) for \( \theta \in [[d_M]], \alpha \in [[d_H]] \) and \( \beta \in [[d_W]] \).

8. For each \( c' \in [[M]] \), \( i' \in [[H']] \), and \( j' \in [[W']] \), Alice sends an RLWE ciphertext to Bob \( c_{c', i', j'} = \text{Extract}(C_{T_{c', i', j'}}, O_w - c_{C_w H_w W_w + i'S_h W_w + j'S_h W_w} + R[c', i', j'], [c', i', j'], \theta, \alpha, \beta) \) where the index of \( C_{T_{c', i', j'}} \) is calculated as \( \theta = [c' \div M_w], \alpha = [i' \div (H_w - h + 1) + 1], \) and \( \beta = [j' \div (W_w - h + 1)] \). The position of the extracting coefficient is determined by \( c \equiv c' \mod M_w, i \equiv i' \mod H'_w \) and \( j \equiv j' \mod W'_w \).

9. On receiving the ciphertexts \( \{ c_{c', i', j'} \} \), Bob outputs \( \langle T \rangle^B \in \mathbb{Z}_p^{M_w \times H'_w \times W'} \) where \( \langle T \rangle^B \hat{c}_{c', i', j'} = \text{RLWE}_{sl}^w(\langle T \rangle^B \langle c_{c', i', j'} \rangle) \).

Figure 11: Proposed Secure Convolution Protocol (Full Version)

**D Full Version of HomConv**

We now present the full version of 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. Then Alice obtains at most \( M_w H'_w W'_w \) values of \( \langle T \rangle^B \) in its encrypted coefficients. Finally, Alice extracts the same partitioning manner in Step 1 and Step 2. After that Alice partitions the kernel \( K \) into \( d_M d_C d_W \) non-overlapping blocks of the same shape. Then Alice extracts each of them using \( \pi_{\text{dec}}^w \) with an identical parameter \( O_w \) that is because the tensor \( T \) is partitioned into blocks of the same shape. Then Alice computes the convolution of the block tensor \( T_{\alpha, \beta} \) and the sub-kernels \( K_0 \). As a result, each RLWE ciphertext \( C_{T_{\theta, \alpha, \beta}} \) obtains at most \( M_h \) values of \( \langle T \rangle^B \) in its encrypted coefficients.
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 \( \mathcal{F}_{\text{CONV}} \) 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.

### E Proofs

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

\[
i'[x] = \sum_{0 \leq d < x} i[d] \hat{k}[x - d] - \sum_{x < d < N} i[d] \hat{k}[N + x - d]
\]

for all \( x \in \mathbb{Z}_n \). Since \( i[x] \) is zero for all \( x \geq CHW \) and the target position \( O' + i'sW + j's > CHW \), we thus have

\[
i'[O' + i'sW + j's] = \sum_{x < d < CHW} i[d] \hat{k}[O' + i'sW + j's - d]
\]

The last line replaces \( i = i's + l \) and \( j = j's + l' \). Also, according to the definition of \( \pi_{\text{conv}}^w \), the value \( \hat{k}[O' - cHW - IW - l'] \) is zero when \( l, l' \notin [h] \). Thus, we only need to take care of the positions that \( l, l' \in [h] \). The above equation continues.

\[
\sum_{\substack{c \in \mathbb{Z}_n \cap \mathbb{Z}_q, \cr l, l' \in [h]}} T[c, i's + l, j's + l'] \hat{k}[O' - cCHW - CHW - IW - l']
\]

The final line is exactly \( T'[c', i', j', f] \).

**Proof 4 (Theorem 2.)** The correctness of Theorem 2 is directly derived from Proposition 2. We now show the privacy part.

(\textbf{Corrupted Alice}) Alice’s view of \( \text{View}_{A}^{\text{HomCONV}} \) consists of an LWE ciphertext \( \tilde{C}^T \). 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 \( \tilde{C}^T = \text{RLWE}_{pk}^{N,q}(0) \) to Alice.

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

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

1. On receiving the RLWE ciphertext \( \tilde{C} \) from Bob and given the access to meta, \( \text{Sim}_B \) samples uniform random polynomial \( r \in \mathbb{Z}_p \) and computes \( \tilde{C} = \text{RLWE}_{pk}^{N,q}(r) \).

2. For each \( c' \in [M], j' \in [H'] \) and \( j' \in [W'] \), \( \text{Sim}_B \) outputs an LWE ciphertext \( \tilde{c}_t, c', f' \approx \text{Extract} (\tilde{C}) \) to Bob.

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

Similarly, the LWE ciphertexts \( \{ct, c', f, j, f' \} \approx_c \{\tilde{c}_t, c', f' \} \) due to the semantic security. Also, the values in the output tensor \( \langle T' \rangle^{B} \) of Bob in HomCONV distribute uniformly in \( \mathbb{Z}_p \), which is exact the same distribution \( \text{Sim}_B \) samples \( \tilde{T} \). Thus we have \( \text{View}_B^{\text{HomCONV}} \approx_c \text{Sim}_B(\text{meta}, \text{out}) \).

The security proofs for the Theorems 1, 3 and 4 can be given in a similar manner.

### F On the Calculation of \( \ell_a \) and \( \ell_b \)

Remind that the LWE ciphertext \( \langle b, a \rangle \in \mathbb{Z}_q^{N+1} \) of message \( m \in \mathbb{Z}_p \) is decrypted by first computing \( \tilde{m} = b + a^{-1}s \) which equals to \( \lceil q/p \rceil m + e \) for some error \( e \). Then we obtain the message as \( m = \lceil q/p \rceil m + e \). In other words, the low \( \ell = \lceil \log_2(q/p) \rceil \) bits of \( m \) will be dropped during the divide-then-round and thus are useless for decryption. This motivates us to skip some of the low-end bits of \( b \) and \( a \), and to only transfer the remains high-end bits to reduce the communication cost if the LWE ciphertext is sent for decryption.

We explicitly write the high-end and low-end parts of \( b \) and \( a \) as \( b = b_H 2^\ell + b_L \) and \( a = a_H 2^\ell + a_L \) such that \( 0 \leq b_L < 2^\ell \) and the values in \( a_L \) are \( 2^\ell \)-bounded. Then \( m \) can be written
Inputs & Outputs: \( \langle z_2^A \rangle_2 \cdot \langle z_2^B \rangle_2 \leftarrow \text{AND} \left( \langle x_2^A \rangle_2, \langle x_2^B \rangle_2, \langle y_2^A \rangle_2, \langle y_2^B \rangle_2 \right) \)

s.t. \( x, y, z \in \{0, 1\} \) and \( z = x \land y \).

1: Alice and Bob jointly generate a beaver triple: \( \langle c_2^A \rangle_2 \oplus \langle c_2^B \rangle_2 = (\langle a_2^A \rangle_2 \oplus \langle a_2^B \rangle_2) \land (\langle b_2^A \rangle_2 \oplus \langle b_2^B \rangle_2) \).
2: Alice computes \( c_2^A = \langle x_2^A \rangle_2 \oplus \langle a_2^A \rangle_2 \) and \( \langle f_2^A \rangle_2 = \langle y_2^A \rangle_2 \oplus \langle b_2^A \rangle_2 \).
   Bob computes \( \langle f_2^B \rangle_2 = \langle y_2^B \rangle_2 \oplus \langle b_2^B \rangle_2 \).
3: Alice and Bob open \( e \) and \( f \).
4: Alice computes \( \langle z_2^B \rangle_2 = (\langle a_2^A \rangle_2 \cdot f) \oplus (\langle b_2^A \rangle_2 \cdot e) \oplus \langle c_2^A \rangle_2 \) and Bob computes \( \langle z_2^B \rangle_2 = (e \cdot f) \oplus (\langle a_2^B \rangle_2 \cdot f) \oplus (\langle b_2^B \rangle_2 \cdot e) \oplus \langle c_2^B \rangle_2 \).

Figure 12: Protocol for \( \mathcal{F}_{\text{AND}} \).

Inputs and Outputs: \( \langle d_2^A \rangle_2, \langle d_2^B \rangle_2 \leftarrow \text{B2A} \left( \langle c_2^A \rangle_2, \langle c_2^B \rangle_2, f \right) \) such that \( c \in \{0, 1\} \) and \( d = c \).

1: Alice and Bob invoke an instance of \( \text{COT}_f \), where Alice is the sender with correlation function \( g(x) = x - 2 \langle c_2^A \rangle_2 \) and Bob is the receiver with input choice \( \langle c_2^B \rangle_2 \). Alice learns \( x \) and Bob learns \( y \).
2: Alice computes \( \langle d_2^A \rangle_2 = \langle c_2^A \rangle_2 - x \) and Bob computes \( \langle d_2^B \rangle_2 = \langle c_2^B \rangle_2 + y \).

Figure 13: B2A Protocol.

as \( m_e = b_L + a_L s + b_H 2^{\ell_b} + a_H s 2^{\ell_a} \). If the sum \( b_L + a_L s < 2^\ell \) then we can just send the high-end parts \( b_H \) and \( a_H \) for decryption. The key here is to find out an upper bound for \( a_L s \). Remind that the secret vector \( s \) distributes uniformly in \( \{0, 1\}^N \). Since \( a \) is basically an uniform random vector, we can assume that the values in \( a_L \) distributes uniformly in \([0, 2^\ell]\). Then the variance of \( a_L s \) is \( \text{Var} = \frac{2^N}{9} (2^\ell)^2 \). We use \( 7 \sqrt{\text{Var}} \) (i.e., 7 times standard deviation) as a high-probability upper bound (\( \approx 1 - 2^{-38.5} \)) on the value \( a_L s \).

Finally, to determine the concrete \( \ell_b \) and \( \ell_a \), we can either maximize the number of bits that can be saved that is \( \ell_b + N \ell_a \) under the constraints \( \ell_a, \ell_b \in [0, \lceil \log_2(q) \rceil] \) and \( 7 \sqrt{\text{Var}} 2^{\ell_a} + 2^{\ell_b} < 2^{\ell} \), or just setting \( \ell_b = \ell' - 1 \) and \( \ell_a = \ell' - 1 - \lceil \log_2(7 \sqrt{\text{Var}}) \rceil \).

G Protocols for \( \mathcal{F}_{\text{AND}} \) and \( \mathcal{F}^{2f}_{\text{B2A}} \)

We describe the classic protocol for computing \( \mathcal{F}_{\text{AND}} \) in Figure 12, where beaver triples are generated with \((\frac{\ell}{2})\)-ROT \([4]\).

We describe a protocol for \( \mathcal{F}^{2f}_{\text{B2A}} \) in Figure 13, which is also used in CryptoFlow2 \([50, 52]\), except that \((\frac{\ell}{2})\)-ROT is instantiated with VOLE-style OT.