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× and 12.9×, 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 $\mathbf{x}$, but she considers $\mathbf{x}$ 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 $F(\mathbf{x})$ but nothing else beyond what can be derived from $F(\mathbf{x})$. 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*[1], 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× and 12.9×, 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-

*Wen-jie and Zhicong contribute equally in this work.

[1]Our implementation is available from https://github.com/Alibaba-Gemini-Lab/OpenCheetah.

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 $\mathbb{Z}_{2^\ell}$? As shown in [50], for the non-linear functions of DNNs, OT-based protocols on the ring $\mathbb{Z}_{2^\ell}$ can perform $40\% - 60\%$ better than on the prime field $\mathbb{Z}_p$ in terms of bandwidth consumption. Another reason to use $\mathbb{Z}_{2^\ell}$ instead of $\mathbb{Z}_p$ is that modulo reduction in $\mathbb{Z}_{2^\ell}$ 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 $\mathbb{Z}_{2^\ell}$. 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 $\mathbb{Z}_{2^\ell}$ using a prime modulus $p \approx 2^\ell$ at the cost of increasing the overhead on the HE-side by many (e.g., 3–5) times. But this would ruin the gains of the non-linear part. In fact, most of the current HE-hybrid systems work over a prime field and tolerate the less efficient non-linear protocols. This raises the question: *Could we find a way to achieve the best of both worlds? That is to enjoy amortized homomorphic operations while keeping the efficient non-linear protocols on the yard without extra overheads.* As we will show, the answer is yes with our new design of HE-based and SIMD-free protocols for the linear functions of DNNs.

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., $30\times$ 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 $(\pi^i_{\mathcal{F}}, \pi^w_{\mathcal{F}})$ that enable us to evaluate the linear layers $\mathcal{F} \in \{\text{CONV}, \text{BN}, \text{FC}\}$ of DNNs via polynomial arithmetic circuits. These encoding functions map the values of the input (e.g., tensor or vector) to the proper coefficients of the output polynomial(s). *By careful design of the coefficient mappings, we not only eliminate the expensive rotations but are also able to accept secret shares from $\mathbb{Z}_{2^\ell}$ for free.* For example, the secure convolution $\mathsf{HomCONV}(\mathbf{T}, \mathbf{K})$ (given later in Figure 4) in *Cheetah* could be computed via just a single homomorphic multiplication between two polynomials $\hat{t}$ and $\hat{k}$ where $\hat{t} = \pi^i_{\text{CONV}}(\mathbf{T})$ is the encoded tensor and $\hat{k} = \pi^w_{\text{CONV}}(\mathbf{K})$ is the encoded kernel, respectively.

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

designed to eliminate two probability errors e_0 and e_1 where $Pr(|e_0| = 1) = 0.5$ and $\Pr(0 < |e_1| < 2^{\ell}) < \varepsilon$ (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 $\{0, \cdots, n-1\}$ for $n \in \mathbb{N}$. We use $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$ and $\lfloor \cdot \rceil$ to denote the ceiling, flooring, and rounding function, respectively. We denote $\mathbb{Z}_q = \mathbb{Z} \cap [-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$ 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. λ 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 $\mathbf{1}\{\mathcal{P}\}$ denote the indicator function that is 1 when $\mathcal{P}$ is true and 0 when $\mathcal{P}$ 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 $\mathbf{u}_i \in \mathbb{F}^{n_i}$ and outputs a vector $\mathbf{u}_o \in \mathbb{F}^{n_o}$. We write $n^* = \min(n_i, n_o)$ and $\bar{n} = \max(n_i, n_o)$. The non-linear functions include the comparison of two ℓ-bit private integers and the truncation of the low f-bit of ℓ-bit integers. λ is the security parameter (usually $\lambda \geq 128$).

Linear Function		Mult.	Rotations
CONV	[44, 50]	$O(MCHWh^2)$	$O(MCHWh^2)$
	Cheetah	$O(MCHW)$	0
BN	[50]	$O(CHW)$	0
	Cheetah	$O(CHW)$	0
FC	[27]	$O(n_o n_i)$	$O(\bar{n}(n^* + \log_2(\frac{n_o}{n^*})))$
	Cheetah	$O(n_o n_i)$	0

Non-linear Function		Communication (bits)
Compare	[50]	$< \lambda\ell + 14\ell$
	Cheetah	$< 11\ell$
Trunc	[50]	$\lambda(\ell + f + 2) + 19\ell + 14f$
	Cheetah	13ℓ

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

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

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

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

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 $\mathsf{HE.pp} = \{N, \sigma, q, p\}$

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 $\mathsf{sk} = \hat{s} \in \mathbb{A}_{N,q}$. The associated public key is computed as $\mathsf{pk} = (\hat{u}_0 \cdot \hat{s} + \hat{e}_0, \hat{u}_0)$ where $\hat{u}_0 \in \mathbb{A}_{N,q}$ is chosen uniformly at random, and the error $\hat{e}_0 \in \mathbb{A}_{N,q}$ is chosen by sampling its coefficients from χ_σ a discrete Gaussian distribution of standard deviation of σ. The RLWE encryption of a message $\hat{m} \in \mathbb{A}_{N,p}$ is given as a tuple of polynomials $\mathsf{RLWE}^{N,q,p}_{\mathsf{pk}}(\hat{m}) = (\lfloor \frac{q}{p}\hat{m} \rceil + \hat{e}, 0) - \hat{u} \cdot \mathsf{pk} \in \mathbb{A}^2_{N,p}$, where the coefficients of $\hat{e} \in \mathbb{A}_{N,q}$ is sampled from χ_σ and the coefficients of $\hat{u} \in \mathbb{A}_{N,q}$ is chosen from $\{0, \pm 1\}$ uniformly at random. A RLWE ciphertext $(\hat{b}, \hat{a})$ is decrypted as $\mathsf{RLWE}^{-1}_{\mathsf{sk}}(\hat{b}, \hat{a}) = \lfloor \frac{p}{q}(\hat{b} + \hat{a} \cdot \hat{s}) \rceil \equiv \hat{m} \bmod p$.

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

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

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

- **Extract.** Given $(\hat{b}, \hat{a}) = \mathsf{RLWE}^{N,q,p}_{\mathsf{pk}}(\hat{m})$ of $\hat{m}$ we can extract an LWE ciphertext of the k-th coefficient of $\hat{m}$, i.e., $(b, \mathbf{a}) = \mathsf{Extract}((\hat{b}, \hat{a}), k)$. The tuple $(b, \mathbf{a})$ 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 $\mathsf{Extract}$ 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 $\binom{2}{1}$-$\mathtt{OT}_\ell$, a sender inputs two messages m_0 and m_1 of length ℓ bits and a receiver inputs a choice bit $c \in \{0, 1\}$. At the end of the protocol, the receiver learns m_c, whereas the sender learns nothing. OT is usually realized by building a few base OT instances with public key cryptography and extending to a large amount of instances with efficient symmetric cryptographic operations (IKNP OT extension [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 $\binom{n}{1}$-$\mathtt{OT}_\ell$ can be implemented in an IKNP-style OT extension, or with $\log_2 n$ calls to $\binom{2}{1}$-$\mathtt{OT}_\lambda$ [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 ℓ-bit value x is shared additively in the ring $\mathbb{Z}_{2^\ell}$ as the sum of two values, say $\langle x \rangle^A_{2^\ell}$ and $\langle x \rangle^B_{2^\ell}$. To reconstruct the value x, we compute the modulo addition, i.e., $x \equiv \langle x \rangle^A_{2^\ell} + \langle x \rangle^B_{2^\ell} \bmod 2^\ell$. 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 $\langle x \rangle^A_{2^\ell}$ and letting Bob hold the share $\langle x \rangle^B_{2^\ell}$. Also, we will omit the subscript if the modulo 2^ℓ is clear from the context.

Fixed-point Values and Truncation. Most prior works on 2PC use fixed-point arithmetic, where a real value $\tilde{x} \in \mathbb{R}$ is encoded as a fixed-point value $x = \lfloor \tilde{x} 2^f \rfloor \in \mathbb{Z}$ 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 $\langle x \rangle^A_{2^\ell}$ and $\langle x \rangle^B_{2^\ell}$ are the shares of a double-precision value, i.e., $\lfloor \tilde{x} 2^{2f} \rfloor$. A faithful truncation protocol should take $\langle x \rangle^A_{2^\ell}$ and $\langle x \rangle^B_{2^\ell}$ as input and compute the shares of $\lfloor \tilde{x} 2^{2f} \rfloor \gg f$.

2.5 Local Truncation Trade-offs

The local truncation protocol of [46] introduces probability errors. For example, to perform the local truncation on $\langle x \rangle^A_{2^\ell}$ and $\langle x \rangle^B_{2^\ell}$, two share holders set $\langle x' \rangle^A_{2^\ell} = \lfloor \langle x \rangle^A_{2^\ell} / 2^f \rfloor$ and $\langle x' \rangle^B_{2^\ell} = 2^\ell - \lfloor 2^\ell - \langle x \rangle^B_{2^\ell} / 2^f \rfloor \bmod 2^\ell$, respectively. Then the

value x' equals to $\lfloor \tilde{x}2^f \rfloor + e_0 + e_1$ 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 ℓ to avoid the harsh error as much as possible. For example, DELPHI [44] sets $\ell = 41$ which can translate a probability $\varepsilon \approx 2^{-19}$ of the e_1 error on their datasets. However we consider a such ε is still not small enough for DNNs because the number of local truncations needed is vast for DNNs. For instance, the number of local truncations needed by the *the first* convolution layer in ResNet50 is about 8.0×10^5. In other words, the local truncation will introduce *at least one* e_1 error for this layer at the chance of $1-(1-\varepsilon)^{8.0\times 10^5}$ 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 decreases the error probability ε a larger bitlength ℓ 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 $\mathsf{CT} = (\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$ be an RLWE ciphertext that decrypts to $\hat{m} \in \mathbb{A}_{N,p}$. 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 $\mathsf{CT}' = (\hat{b} + \hat{r} \bmod q, \hat{a})$ 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 $\mathsf{RLWE}_{\mathsf{sk}}^{-1}(\mathsf{CT}')$ and denote $\hat{m}_A = -\lceil p \cdot \hat{r}/q \rfloor \bmod p$. 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., $|\hat{m} - (\hat{m}_A + \hat{m}_B \bmod p)| \leq 1$. 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 $\mathbf{x}$ (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., $\mathbf{z} = f_d(f_{d-1}(\cdots(f_1(\mathbf{x}, \mathbf{W}_1), \cdots), \mathbf{W}_{d-1}), \mathbf{W}_d)$ 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 targetting, 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 $\mathbf{v} \in \mathbb{F}^{n_i}$ of length n_i and its output is a vector $\mathbf{u} \in \mathbb{F}^{n_o}$ of length n_o. A fully connected layer is parameterized by the tuple $(\mathbf{W}, \mathbf{b})$ where $\mathbf{W} \in \mathbb{F}^{n_o \times n_i}$ is the weight matrix and $\mathbf{b}$ is an n_o-sized bias vector. The output is specified by the linear transformation $\mathbf{u} = \mathbf{W}\mathbf{v} + \mathbf{b}$.

Convolution Layer (CONV). A two dimensional strided convolution $\mathsf{Conv2D}(\mathbf{T}, \mathbf{K}; s)$ 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 $\mathbf{K} \in \mathbb{F}^{M \times C \times h \times h}$ to generate a 3-dimension output tensor $\mathbf{T}' \in \mathbb{F}^{M \times H' \times W'}$ where $H' = \lfloor (H - h + s)/s \rfloor$ and $W' = \lfloor (W - h + s)/s \rfloor$. 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

$$\mathbf{T}'[c', i', j'] = \sum_{\substack{c \in [[C]] \\ l, l' \in [[h]]}} \mathbf{T}[c, i's + l, j's + l'] \mathbf{K}[c', c, l, l']. \quad (2)$$

for each position (c', i', j') of the output tensor $\mathbf{T}'$.

Batch Normalization Layer (BN). In DNNs, a BN layer $\mathsf{BN}(\mathbf{T}; \alpha, \beta)$ 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 (μ, θ) where $\mu \in \mathbb{F}^C$ is the scaling vector and $\theta \in \mathbb{F}^C$ is the shift vector. For all $c \in [[C]], i \in [[H]]$ and $j \in [[W]]$, $\mathbf{T}'$ is computed via

$$\mathbf{T}'[c, i, j] = \mu[c]\mathbf{T}[c, i, j] + \theta[c]. \quad (3)$$

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

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

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

Parameters: Shape meta M, C, H, W, h and stride $s > 0$.
Computation: On input $\langle \mathbf{T} \rangle^A \in \mathbb{F}^{C \times H \times W}, \mathbf{K} \in \mathbb{F}^{M \times C \times h \times h}$ from Alice and input $\langle \mathbf{T} \rangle^B \in \mathbb{F}^{C \times H \times W}$ from Bob, compute $\mathbf{T}' = \mathsf{Conv2D}(\mathbf{T}, \mathbf{K}; s)$. Sample an uniform tensor $\mathbf{R}$ from $\mathbb{F}$ with the same shape of $\mathbf{T}'$.
Return: $\mathbf{R}$ to Alice and $\mathbf{T}' - \mathbf{R} \in \mathbb{F}^*$ to Bob.

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

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

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

Figure 1: Ideal Functionalities of Linear Layers

size. Typical non-linear functions can be one of several types: the most common ones in DNNs are ReLU functions (i.e., $\mathsf{ReLU}(x) = \max(0, x)$) and max-pooling functions. In the context of 2PC, truncation is also considered as a non-linear layer because truncation is beyond the ability of arithmetic circuit. Following the blueprint of CrypTFlow2, these non-linear layers can be evaluated securely via OT-based protocols.

2.7.3 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 Π, respectively. We write $O_A, O_B \leftarrow \Pi(I_A, I_B)$ to denote an execution of Π where O_A and O_B are the output to Alice and Bob, respectively.

We target the same threat model of DELPHI and CrypTFlow2. *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 $p \equiv 1 \bmod 2N$.

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 $(\pi^i_{\mathcal{F}}, \pi^w_{\mathcal{F}})$ to properly place the values in the input and weight to polynomial coefficients for each of the functionality $\mathcal{F}$ in Figure 1. With the help of $(\pi^i_{\mathcal{F}}, \pi^w_{\mathcal{F}})$, we then show how to evaluate these functionalities privately. Also, $\pi^i_{\mathcal{F}}$ and $\pi^w_{\mathcal{F}}$ are well-defined for any $p > 1$ such as $p = 2^\ell$, allowing our protocols to accept secretly shared input from the ring $\mathbb{Z}_{2^\ell}$ 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 $\mathbf{u} = \mathbf{W}\mathbf{v}$ which can be decomposed into inner products of vectors. Our mapping functions π^w_{fc} and π^i_{fc} 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 $\pi^w_{\text{fc}} : \mathbb{Z}_p^{n_{\text{o}} \times n_{\text{i}}} \mapsto \mathbb{A}_{N,p}$ and $\pi^i_{\text{fc}} : \mathbb{Z}_p^{n_{\text{i}}} \mapsto \mathbb{A}_{N,p}$. Here we first require $n_{\text{o}} n_{\text{i}} \leq N$ for simplicity.

$$\begin{aligned} \hat{v} &= \pi^i_{\text{fc}}(\mathbf{v}) \text{ where } \hat{v}[j] = \mathbf{v}[j] \\ \hat{w} &= \pi^w_{\text{fc}}(\mathbf{W}) \text{ where } \hat{w}[i \cdot n_{\text{i}} + n_{\text{i}} - 1 - j] = \mathbf{W}[i, j], \end{aligned} \tag{4}$$

where $i \in [[n_{\text{o}}]], j \in [[n_{\text{i}}]]$ and all other coefficients of $\hat{v}$ and $\hat{w}$ are set to 0. The multiplication of polynomials $\hat{u} = \hat{w} \cdot \hat{v} \in \mathbb{A}_{N,p}$ directly gives the matrix–vector multiplication $\mathbf{W}\mathbf{v} \equiv \mathbf{u} \bmod p$ *in some of its coefficients.*

Inputs & Outputs:

$$\langle \mathbf{u} \rangle^A, \langle \mathbf{u} \rangle^B \leftarrow \mathsf{HomFC}(\{\langle \mathbf{v} \rangle^A, \mathbf{W}, \mathbf{b}\}, \{\langle \mathbf{v} \rangle^B, \mathsf{sk}\})$$

$\mathbf{W} \in \mathbb{Z}_p^{n_\mathrm{o} \times n_\mathrm{i}}$, $\mathbf{b} \in \mathbb{Z}_p^{n_\mathrm{o}}$, $\langle \mathbf{v} \rangle^A, \langle \mathbf{v} \rangle^B \in \mathbb{Z}_p^{n_\mathrm{i}}$, and $\mathbf{u} = \mathbf{W}\mathbf{v} + \mathbf{b} \in \mathbb{Z}_p^{n_\mathrm{o}}$.
Public Parameters: $\mathsf{pp} = (\mathsf{HE.pp}, \mathsf{pk}, n_\mathrm{i}, n_\mathrm{o}, n_\mathrm{iw}, n_\mathrm{ow})$.

- Shape meta $n_\mathrm{i}, n_\mathrm{o}$ such that $n_\mathrm{i}, n_\mathrm{o} > 0$. The partition window size $0 < n_\mathrm{iw} \leq n_\mathrm{i}, 0 < n_\mathrm{ow} \leq n_\mathrm{i}$ such that $n_\mathrm{iw} n_\mathrm{ow} \leq N$.
- Set $n_\mathrm{i}' = \lceil n_\mathrm{i}/n_\mathrm{iw} \rceil$ and $n_\mathrm{o}' = \lceil n_\mathrm{o}/n_\mathrm{ow} \rceil$.

1: Bob first partitions its input shares $\langle \mathbf{v} \rangle^B$ into subvectors $\langle \mathbf{v}_\alpha \rangle^B \in \mathbb{Z}_p^{n_\mathrm{iw}}$ for $\alpha \in [[n_i']]$. Zero-padding them when $n_\mathrm{i} \nmid n_\mathrm{iw}$.
2: Bob encodes the vectors to polynomials $\langle \hat{v}_\alpha \rangle^B = \pi^i_\mathrm{fc}(\langle \mathbf{v}_\alpha \rangle^B)$ for $\alpha \in [[n_i']]$. Then Bob sends the RLWE ciphertexts $\{\mathsf{CT}'_\alpha = \mathsf{RLWE}^{N,q,p}_\mathsf{pk}(\langle \hat{v}_\alpha \rangle^B)\}$ to Alice.
3: Similarly, Alice first partitions its shares $\langle \mathbf{v} \rangle^A$ into $\langle \mathbf{v}_\alpha \rangle^A \in \mathbb{Z}_p^{n_\mathrm{iw}}$ following the same manner in Step 1. Also, Alice partitions the weight matrix $\mathbf{W}$ into block matris $\mathbf{W}_{\beta,\alpha} \in \mathbb{Z}_p^{n_\mathrm{ow} \times n_\mathrm{iw}}$. Zero-padding is used to the right-most (resp. bottom-most) blocks when $n_\mathrm{i} \nmid n_\mathrm{iw}$ (resp. $n_\mathrm{o} \nmid n_\mathrm{ow}$). Then Alice encodes the subvectors and block matrices to polynomials $\langle \hat{v}_\alpha \rangle^A = \pi^i_\mathrm{fc} \langle \mathbf{v}_\alpha \rangle^A$ and $\hat{w}_{\beta,\alpha} = \pi^w_\mathrm{fc}(\mathbf{W}_{\beta,\alpha})$ for $\alpha \in [[n_i']]$ and $\beta \in [[n_o']]$.
4: On receiving the ciphertexts $\{\mathsf{CT}'_\alpha\}$ from Bob, Alice operates $\mathsf{CT}_\beta = \boxplus_{\alpha \in [[n_i']]}((\mathsf{CT}'_\alpha \boxplus \langle \hat{v}_\alpha \rangle^A) \boxtimes \hat{w}_{\beta,\alpha})$ for $\beta \in [[n_o']]$.
5: Alice samples $\mathbf{r} \in \mathbb{Z}_q^{n_\mathrm{o}}$ uniformly at random to re-mask the computing ciphertexts. Specifically, for each $i \in [[n_\mathrm{o}]]$, Alice first extracts an LWE ciphertext $\mathsf{ct}_i = \mathsf{Extract}(\mathsf{CT}_{i \bmod n_o'}, (i \bmod n_o')n_\mathrm{i} + n_\mathrm{i} - 1)$ and then adds $\mathbf{r}[i]$ to the 1st component of ct_i and results at ct'_i. Alice then sends the LWE ciphertexts $\{\mathsf{ct}'_i\}$ back to Bob.
6: Alice outputs $\mathbf{b} - \lceil p \cdot \mathbf{r}/q \rfloor \bmod p$ as the share $\langle \mathbf{u} \rangle^A$.
7: On receiving the LWE ciphertexts $\{\mathsf{ct}'_i\}$ from Alice, Bob computes $\langle \mathbf{u} \rangle^B[i] = \mathsf{LWE}^{-1}_\mathsf{sk}(\mathsf{ct}'_i)$ for all $i \in [[n_\mathrm{o}]]$.

Figure 2: Proposed Secure Fully Connection Protocol

Proposition 1 *Given two polynomials* $\hat{v} = \pi^i_\mathrm{fc}(\mathbf{v}), \hat{w} = \pi^w_\mathrm{fc}(\mathbf{W}) \in \mathbb{A}_{N,p}$, *the multiplication* $\mathbf{W}\mathbf{v} \equiv \mathbf{u} \bmod p$ *can be evaluated via the product* $\hat{u} = \hat{v} \cdot \hat{w}$ *over the ring* $\mathbb{A}_{N,p}$. *That is* $\mathbf{u}[i]$ *is computed in* $\hat{u}[i \cdot n_\mathrm{i} + n_\mathrm{i} - 1]$ *for all* $i \in [[n_\mathrm{o}]]$.

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

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

When $n_\mathrm{i} n_\mathrm{o} > N$, we can first partition the weight matrix into sub-matrices of $0 < \bar{n}_i \times \bar{n}_o$ elements such that $\bar{n}_i \bar{n}_o \leq$N ($\bar{n}_i$ and $\bar{n}_o$ can be chosen freely as public parameters as long as they satisfy this constraint). Zero-padding is used when $n_\mathrm{i} \nmid \bar{n}_\mathrm{i}$ or $n_\mathrm{o} \nmid \bar{n}_\mathrm{o}$. Then we convert the task of matrix multiplication in shape $n_\mathrm{i} \times n_\mathrm{o}$ to subtasks of a smaller size $\bar{n}_i \times \bar{n}_o$.

Toy example over $\mathbb{Z}_{2^5}$.

$$\mathbf{W} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} \Rightarrow \mathbf{W}\mathbf{v} \equiv \begin{bmatrix} 18 & 26 \end{bmatrix} \bmod 2^5$$

Cheetah evaluates $\mathbf{W}\mathbf{v}$ using π^w_fc and π^i_fc.

$$\pi^w_\mathrm{fc}(\mathbf{W}) \to \hat{w} = 3X^0 + 2X^1 + 1X^2 + 6X^3 + 5X^4 + 4X^5 \in \mathbb{A}_{8,2^5}$$
$$\pi^i_\mathrm{fc}(\mathbf{v}) \to \hat{v} = 7X^0 + 8X^1 + 9X^2 \in \mathbb{A}_{8,2^5}$$
$$\Downarrow \hat{w} \cdot \hat{v} \bmod (X^8 + 1, 2^5)$$
$$\hat{w} \cdot \hat{v} \equiv 21X^0 + 6X^1 + 18X^2 + 4X^3 + 28^4 + 26X^5 + 13X^6 + 4X^7 \bmod (X^8 + 1, 2^5)$$
$$\Downarrow \text{Extract needed coefficients}$$
$$\mathsf{LWE}(18) = \mathsf{Extract}(\mathsf{RLWE}(\hat{w} \cdot \hat{v}), 2)$$
$$\mathsf{LWE}(26) = \mathsf{Extract}(\mathsf{RLWE}(\hat{w} \cdot \hat{v}), 5)$$

Figure 3: Toy example for π^w_fc and π^i_fc with $N = 8$ and $p = 2^5$.

Theorem 1 *The protocol* HomFC *in Figure 2 realizes the ideal functionality* $\mathcal{F}_\mathrm{FC}$ *of Figure 1a for* $\mathbb{F} = \mathbb{Z}_p$ *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 $n_o' n_i' = O(n_\mathrm{o} n_\mathrm{i}/N)$ homomorphic multiplications and additions. Alice sends n_o LWE ciphertexts to Bob for decryption which can be compressed to about $O((n_\mathrm{o} + n_o' N)\log_2 q)$ 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 $\mathbb{A}_{N,p}$. Consider the most simple case of (2) such that $H = W = h$ and $M = 1$, i.e., the input tensor has the same shape as the convolution kernel. Then the computation in (2) becomes an inner product of two flatten vectors by concatenating $\mathbf{T}$ and $\mathbf{K}$ row-by-row and channel-by-channel. For general cases, we can view the computation of 2-dimension convolution as *a batch of inner products of* h^2 *values*. We now give the definitions of $\pi^i_\mathrm{conv} : \mathbb{Z}_p^{C \times H \times W} \mapsto \mathbb{A}_{N,p}$ and $\pi^w_\mathrm{conv} : \mathbb{Z}_p^{M \times C \times h \times h} \mapsto \mathbb{A}_{N,p}$. Here we require $MCHW \leq N$ for simplicity.

$$\hat{t} = \pi^i_\mathrm{conv}(\mathbf{T}) \text{ st. } \hat{t}[cHW + iW + j] = \mathbf{T}[c, i, j] \tag{5}$$
$$\hat{k} = \pi^w_\mathrm{conv}(\mathbf{K}) \text{ st. } \hat{k}[O - c'CHW - cHW - lW - l'] = \mathbf{K}[c', c, l, l'],$$

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 $\hat{t} \cdot \hat{k} \in \mathbb{A}_{N,p}$ directly gives the 2-dimension convolution *in some of the coefficients* of the resulting polynomial.

Inputs and Outputs:

$$\langle \mathbf{T}' \rangle^A, \langle \mathbf{T}' \rangle^B \leftarrow \mathsf{HomCONV}(\{\langle \mathbf{T} \rangle^A, \mathbf{K}\}, \{\langle \mathbf{T} \rangle^B, \mathsf{sk}\})$$

such that $\langle \mathbf{T} \rangle^A, \langle \mathbf{T} \rangle^B \in \mathbb{Z}_p^{C \times H \times W}$, $\mathbf{K} \in \mathbb{Z}_p^{M \times C \times h \times h}$ and $\mathbf{T}' = \mathsf{Conv2D}(\mathbf{T}, \mathbf{K}; s) \in \mathbb{Z}_p^{M \times H' \times W'}$.
Public Parameters: $\mathsf{pp} = (\mathsf{HE.pp}, \mathsf{pk}, M, C, H, W, h, s)$.

- Shape meta M, C, H, W, h such that $MCHW \leq N$ and $h \leq H, W$, and stride $s > 0$.
- Set $H' = \lfloor \frac{H-h+s}{s} \rfloor$, $W' = \lfloor \frac{W-h+s}{s} \rfloor$, and $O = HW(MC-1) + W(h-1) + h - 1$.

1: Bob encodes its share as polynomial $\langle \hat{t} \rangle^B = \pi^i_{\mathsf{conv}}(\langle \mathbf{T} \rangle^B)$. Bob sends the ciphertext $\mathsf{CT}' = \mathsf{RLWE}^{N,q,p}_{\mathsf{pk}}(\langle \hat{t} \rangle^B)$ to Alice.
2: Alice encodes its share of input tensor and the filter as polynomials $\langle \hat{t} \rangle^A = \pi^i_{\mathsf{conv}}(\langle \mathbf{T} \rangle^A), \hat{k} = \pi^w_{\mathsf{conv}}(\mathbf{K})$.
3: Alice samples $\mathbf{R} \in \mathbb{Z}_q^{M \times H' \times W'}$ uniformly at random.
4: On receiving the RLWE ciphertext CT' from Bob, Alice operates to obtain $\mathsf{CT} = (\mathsf{CT}' \boxplus \langle \hat{t} \rangle^A) \boxtimes \hat{k}$.
5: For each $c' \in [[M]], i' \in [[H']]$ and $j' \in [[W']]$, Alice first extracts a LWE ciphertext $\mathsf{ct}_{c',i',j'} = \mathsf{Extract}(\mathsf{CT}, O - c'CHW + i'sW + j's)$ and then adds $\mathbf{R}[c', i', j']$ to the 1st component of $\mathsf{ct}_{c',i',j'}$, resulting $\mathsf{ct}'_{c',i',j'}$. Alice then sends the ciphertexts $\{\mathsf{ct}'_{c',i',j'}\}$ back to Bob.
6: Alice outputs $-\lceil p \cdot \mathbf{R}/q \rfloor \bmod p$ as the share $\langle \mathbf{T}' \rangle^A$.
7: On receiving the LWE ciphertexts from Alice, Bob computes $\langle \mathbf{T}' \rangle^B[c', i', j'] = \mathsf{LWE}^{-1}_{\mathsf{sk}}(\mathsf{ct}'_{c',i',j'})$ for all positions (c', i', j').

Figure 4: Proposed Secure Convolution Protocol (Basic Ver)

Proposition 2 *Given two polynomials* $\hat{t} = \pi^i_{\mathsf{conv}}(\mathbf{T}), \hat{k} = \pi^w_{\mathsf{conv}}(\mathbf{K}) \in \mathbb{A}_{N,p}$, *the convolution* $\mathbf{T}' = \mathsf{Conv2D}(\mathbf{T}, \mathbf{K}; s)$ (2) *can be evaluated by the polynomial multiplication* $\hat{t}' = \hat{t} \cdot \hat{k}$ *over the ring* $\mathbb{A}_{N,p}$. *For all positions* (c', i', j') *of* $\mathbf{T}'$, $\mathbf{T}'[c', i', j']$ *is computed in the coefficient* $\hat{t}'[O - c'CHW + i'sW + j's]$ *where* $O = HW(MC-1) + W(h-1) + h - 1$.

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

Theorem 2 *The protocol* $\mathsf{HomCONV}$ *in Figure 4 realizes the ideal functionality* $\mathcal{F}_{\mathrm{CONV}}$ *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 $M_{\mathrm{w}}, C_{\mathrm{w}}, H_{\mathrm{w}}$ and W_{w}, respectively. The size of the partition windows can be chosen freely as long as they satisfy the following constraints: $0 < M_{\mathrm{w}} \leq M$, $0 < C_{\mathrm{w}} \leq C$, $h \leq H_{\mathrm{w}} \leq H$, $h \leq W_{\mathrm{w}} \leq W$ and $M_{\mathrm{w}} C_{\mathrm{w}} H_{\mathrm{w}} W_{\mathrm{w}} \leq N$. 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 $\lceil \frac{C}{\lfloor N/(H_{\mathrm{w}} W_{\mathrm{w}}) \rfloor} \rceil \cdot \lceil \frac{H-h+1}{H_{\mathrm{w}}-h+1} \rceil \cdot \lceil \frac{W-h+1}{W_{\mathrm{w}}+h-1} \rceil$. When H_{w} and W_{w} are specified, the partition window sizes along the C-axis and M-axis is $C_{\mathrm{w}} = \min(C, \lfloor N/(H_{\mathrm{w}} W_{\mathrm{w}}) \rfloor)$ and $M_{\mathrm{w}} = \min(M, \lfloor N/(C_{\mathrm{w}} H_{\mathrm{w}} W_{\mathrm{w}}) \rfloor)$, respectively.

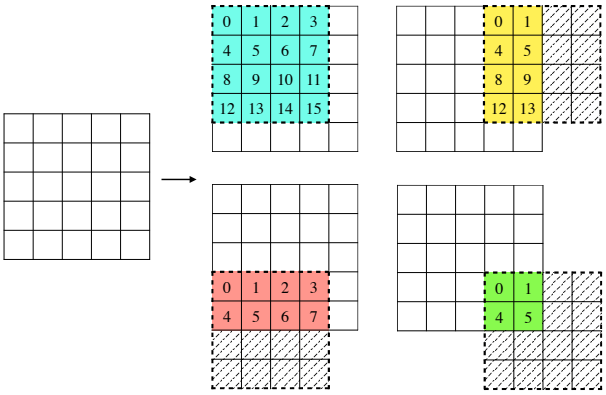

Figure 5: Partitioning the input tensor along the H-axis and W-axis where $N = 16, C = 1, H, W = 5, h = 2$, and $H_{\mathrm{w}}, W_{\mathrm{w}} = 4$. Digits represent the coefficient indices in an encoded polynomial. The dashed parts are zero padding.

By splitting the big tensor and kernel into smaller blocks and zero-padding the margin blocks, we can apply (5) on the corresponding pair of subtensor and subkernel. We demonstrate why such partitions can work correctly. From the definition in (2), the convolution along the M-axis is independent for each subkernel, and thus we can equally split the M-axis into $d_M = \lceil \frac{M}{M_{\mathrm{w}}} \rceil$ groups and each of them contains M_{w} subkernels. Similarly, the convolution along the C-axis requires extra additions which are supported by HEs. Thus we can safely partition along the C-axis without overlapping too.

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

Inputs and Outputs:

$$\langle \mathbf{T}' \rangle^A, \langle \mathbf{T}' \rangle^B \leftarrow \mathsf{HomBN}\left(\left\{\langle \mathbf{T} \rangle^A, \mu, \theta\right\}, \left\{\langle \mathbf{T} \rangle^B, \mathsf{sk}\right\}\right)$$

s.t. $\langle \mathbf{T} \rangle^A, \langle \mathbf{T} \rangle^B \in \mathbb{Z}_p^{C \times H \times W}$, $\mu, \theta \in \mathbb{Z}_p^C$ and $\mathbf{T}' = \mathsf{BN}(\mathbf{T}; \mu, \theta)$.
Public Parameters: $\mathsf{pp} = (\mathsf{HE.pp}, \mathsf{pk}, C, H, W, C_\mathrm{w}, H_\mathrm{w}, W_\mathrm{w})$.

- The partition window sizes $0 < C_\mathrm{w} \le C, 0 < H_\mathrm{w} \le H$ and $0 < W_\mathrm{w} \le W$ such that $C_\mathrm{w}^2 H_\mathrm{w} W_\mathrm{w} \le N$.
- Let $d_C = \lceil C/C_\mathrm{w} \rceil$, $d_H = \lceil H/H_\mathrm{w} \rceil$ and $d_W = \lceil W/W_\mathrm{w} \rceil$.

1: Bob partitions its share tensor into sub-blocks $\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^B \in \mathbb{Z}_p^{C_\mathrm{w} \times H_\mathrm{w} \times W_\mathrm{w}}$ (with zero-padding if necessary).
2: Bob then encodes the sub-blocks as polynomials $\langle \hat{t}_{\gamma,\alpha,\beta} \rangle^B = \pi_{\mathrm{bn}}^i(\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^B)$ for $\gamma \in [[d_C]]$, $\alpha \in [[d_H]]$ and $\beta \in [[d_W]]$. Bob then sends $\{\mathsf{CT}'_{\gamma,\alpha,\beta} = \mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(\langle \hat{t}_{\gamma,\alpha,\beta} \rangle^B)\}$ to Alice.
3: Alice partitions and encodes its input tensor and the scaling vector as polynomials $\langle \hat{t}_{\gamma,\alpha,\beta} \rangle^A = \pi_{\mathrm{bn}}^i(\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^A)$ and $\hat{a}_\gamma = \pi_{\mathrm{bn}}^w(\mu_\gamma)$, respectively.
4: Alice samples $\mathbf{R} \in \mathbb{Z}_p^{C \times H \times W}$ uniformly at random.
5: On receiving the ciphertexts $\{\mathsf{CT}'_{\gamma,\alpha,\beta}\}$ from Bob, Alice operates $\mathsf{CT}_{\gamma,\alpha,\beta} = (\mathsf{CT}'_{\gamma,\alpha,\beta} \boxplus \langle \hat{t}_{\gamma,\alpha,\beta} \rangle^A) \boxtimes \hat{a}_\gamma$ for $\gamma \in [[d_C]]$, $\alpha \in [[d_H]]$ and $\beta \in [[d_W]]$.
6: For each $c \in [[C]], i \in [[H]]$ and $j \in [[W]]$, Alice first extracts a LWE ciphertext $\mathsf{ct}_{c,i,j} = \mathsf{Extract}(\mathsf{CT}_{c',i',j'}, cCHW + cHW + iH + j)$, where $c' \equiv c \bmod d_C$, $i' \equiv i \bmod H$ and $j' \equiv j \bmod W$. Then Bob adds $\mathbf{R}[c,i,j]$ to the 1st component of $\mathsf{ct}_{c,i,j}$, resulting $\mathsf{ct}'_{c,i,j}$.
7: Alice sends the ciphertexts $\{\mathsf{ct}'_{c,i,j}\}$ back to Bob, and ouputs $\langle \mathbf{T}' \rangle^A$ as $\langle \mathbf{T}' \rangle^A[c,i,j] = \theta[c] - \lceil p \cdot \mathbf{R}[c,i,j]/q \rfloor \bmod p$.
8: On receiving the LWE ciphertexts from Alice, Bob ouputs $\langle \mathbf{T}' \rangle^B$ as $\langle \mathbf{T}' \rangle^B[c,i,j] = \mathsf{LWE}_{\mathsf{sk}}^{-1}(\mathsf{ct}'_{c,i,j})$.

Figure 6: Proposed Secure Batch Normalization Protocol

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 $O(\frac{M}{M_\mathrm{w}} H'W')$ 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 $O(C\lceil HW/N \rceil)$ ciphertexts. We can reduce this communication cost by "stacking" multiple channels of the tensor into a single polynomial. For example, when $C^2HW \le N$, we can even put all the channels into one polynomial. We now give the definitions of $\pi_{\mathrm{bn}}^i : \mathbb{Z}_p^{C \times H \times W} \mapsto \mathbb{A}_{N,p}$ and $\pi_{\mathrm{bn}}^w : \mathbb{Z}_p^C \mapsto \mathbb{A}_{N,p}$. Here we demand $C^2HW \le N$ for simplicity.

$$\hat{t} = \pi_{\mathrm{bn}}^i(\mathbf{T}) \text{ s.t. } \hat{t}[cCHW + iH + j] = \mathbf{T}[c,i,j]$$
$$\hat{a} = \pi_{\mathrm{bn}}^w(\alpha) \text{ s.t. } \hat{a}[cHW] = \alpha[c]$$

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 $\hat{t}' = \hat{t} \cdot \hat{a}$ gives the multiplication part of (3) in some of coefficients of $\hat{t}'$. That is $\hat{t}'[cCHW + cHW + iH + j]$ equals to $\mathbf{T}[c,i,j]\alpha[c]$ for all (c,i,j) position. When $C^2HW > N$, we can first partition $\mathbf{T}$ into sub-blocks in a shape of $C_\mathrm{w} \times H_\mathrm{w} \times W_\mathrm{w}$ such that $C_\mathrm{w}^2 H_\mathrm{w} W_\mathrm{w} \le N$. Since each channel is proceed independently in the BN computation, we can just apply the mapping functions $\pi_{\mathrm{bn}}^i, \pi_{\mathrm{bn}}^w$ to the sub-blocks of $\mathbf{T}$.

Theorem 3 *The protocol* HomBN *in Figure 6 realizes the ideal functionality $\mathcal{F}_{\mathrm{BN}}$ of Figure 1c for the field $\mathbb{F} = \mathbb{Z}_p$ 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.

Parameters: Bit width $\ell > 0$.
Computation: On input $x \in \mathbb{Z}_{2^\ell}$ from Alice and input $y \in \mathbb{Z}_{2^\ell}$ from Bob, compute $b = \mathbf{1}\{x > y\}$. Sample a bit r from $\mathbb{Z}_2$ uniformly at random.
Return: r to Alice and $b \oplus r$ to Bob.

(a) Ideal Functionality $\mathcal{F}_{\mathsf{Mill}}^\ell$

Parameters: Bit width $\ell > 0$ and fixed-point precision $0 < f < \ell$.
Computation: On input $\langle x \rangle_{2^\ell}^A$ from Alice and input $\langle x \rangle_\ell^B$ from Bob, compute $x' = \lfloor x/2^f \rfloor$. Sample a uniform value r from $\mathbb{Z}_{2^\ell}$.
Return: r to Alice and $x' - r \in \mathbb{Z}_{2^\ell}$ to Bob.

(b) Ideal Functionality $\mathcal{F}_{\mathrm{trunc}}^{\ell,f}$

Figure 7: Ideal Functionalities of Non-linear Functions

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 $\binom{2}{1}$-$\mathtt{OT}_\ell$, $\binom{2}{1}$-$\mathtt{COT}_\ell$ and $\binom{n}{1}$-$\mathtt{OT}_\ell$. We instantiate $\binom{n}{1}$-$\mathtt{OT}_\ell$ with the algorithm proposed by Naor and Pinkas [47] by using $\log_2 n$ calls to

Table 2: Communication complexity of various OT protocols used in CrypTFlow2 and *Cheetah*. *Cheetah* uses VOLE-style OT for the underlying calls to $\binom{2}{1}$-$\mathtt{ROT}_\lambda$.

Function	Communication (bits)	
	CrypTFlow2	*Cheetah*
$\binom{2}{1}$-$\mathtt{OT}_\ell$	$\lambda+2\ell$	$2\ell+1$
$\binom{2}{1}$-$\mathtt{COT}_\ell$	$\lambda+\ell$	$\ell+1$
$\binom{n}{1}$-$\mathtt{OT}_\ell$ $(n\geq 3)$	$2\lambda+n\ell$	$n\ell+\log_2 n$

Table 3: Communication complexity of millionaires' protocols of ℓ-bit integers.

Millionaires' Protocol	Communication (bits)
CrypTFlow2 ($m=4$, IKNP, $\binom{n}{1}$-$\mathtt{OT}$)	$<\lambda\ell+14\ell$
Naive ($m=1$, VOLE, $\binom{2}{1}$-$\mathtt{ROT}$)	$<16\ell$
Cheetah ($m=4$, VOLE, $\binom{2}{1}$-$\mathtt{ROT}$)	$<11\ell$

$\binom{2}{1}$-$\mathtt{ROT}_\lambda$, whereas CrypTFlow2 implements it with the IKNP-style OT extension proposed in [37]. Due to the usage of VOLE-style OT, our calls to $\binom{2}{1}$-$\mathtt{ROT}_\lambda$ enjoys almost 0 amortized communication cost. We provide a brief comparison of these two approaches in Table 2. When n and the message length ℓ are small, our approach demonstrates prominent advantage, which is indeed the case in all protocols used in neural network inference (usually, $n\leq 16$, $\ell\leq 2$).

4.1 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 $\mathcal{F}_{\text{AND}}$ 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 ℓ-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 $\binom{2^m}{1}$-$\mathtt{OT}_1$ to compute a boolean share of $\mathtt{lt}_j=\mathbf{1}\{x_j<y_j\}$ (similar invocation for $\mathtt{eq}_j=\mathbf{1}\{x_j=y_j\}$).

2. Two parties use $\mathcal{F}_{\text{AND}}$ to combine the former outputs in a tree evaluation (of depth $\log(\ell/m)$) with the observation: $\mathbf{1}\{x<y\}=\mathbf{1}\{x_1<y_1\}\oplus(\mathbf{1}\{x_1=y_1\}\wedge\mathbf{1}\{x_0<y_0\})$, where $x=x_1||x_0$ and $y=y_1||y_0$.

In stage 2, [50] presents various optimizations to realize $\mathcal{F}_{\text{AND}}$ efficiently in the context of IKNP-style OT extension, such as using $\binom{16}{1}$-$\mathtt{OT}_2$ to generate two beaver triples and $\binom{8}{1}$-$\mathtt{OT}_2$ to generate correlated triples. However, in *Cheetah*, these are less efficient than using $\binom{2}{1}$-$\mathtt{ROT}_1$ 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 $\mathcal{F}_{\text{AND}}$ implementation with our aforementioned solution. Let $(\langle m\rangle_2^A,\langle m\rangle_2^B)=\text{Mill}^\ell(x,y)$ denote the millionaires' protocol where Alice's input is x, Bob's input is y, and they receive boolean shares as output: $\langle m\rangle_2^A\oplus\langle m\rangle_2^B=\mathbf{1}\{x>y\}$.

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 $\mathcal{F}_{\text{Trunc}}^{\ell,f}$ (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 $\mathcal{F}_{\text{Trunc}}^{\ell,f}$, 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 ℓ-bit integer x and its arithmetic shares x_0 and x_1, define $w=\mathbf{1}\{x_0+x_1>2^\ell-1\}$. 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^ℓ. If so, there will be large error for local truncation and it should be corrected by subtracting 2^ℓ. c comes from a probabilistic last-bit error that is decided by whether the sum of least significant s bits of x_0 and x_1 wraps around 2^f.* ■

Let $\mathcal{F}_{\text{B2A}}^{2^f}$ denote the functionality that accepts boolean shares of $x\in\{0,1\}$ and returns arithmetic shares of x in $\mathbb{Z}_{2^f}$. We provide an implementation of $\mathcal{F}_{\text{B2A}}^{2^f}$ in Appendix G. The one-bit approximate truncation protocol is provided in Figure 8.

Inputs and Outputs: $\langle z\rangle_{2^\ell}^A, \langle z\rangle_{2^\ell}^B \leftarrow \mathsf{Trunc}(\langle x\rangle_{2^\ell}^A, \langle x\rangle_{2^\ell}^B, f)$ such that $x \in \mathbb{Z}_{2^\ell}$, and $z = x \gg f$ or $z = (x \gg f) - 1$.

1: Alice and Bob call the millionaires' protocol: $(\langle w\rangle_2^A, \langle w\rangle_2^B) = \mathsf{Mill}^\ell(\langle x\rangle^A, 2^\ell - 1 - \langle x\rangle^B)$.
2: Alice and Bob invoke an instance of $\mathcal{F}_{\mathrm{B2A}}^{2^f}$ with inputs $(\langle w\rangle_2^A, \langle w\rangle_2^B)$ and learn $(\langle w\rangle_{2^f}^A, \langle w\rangle_{2^f}^B)$.
3: Alice computes $\langle z\rangle^A = (\langle x\rangle^A \gg f) - \langle w\rangle_{2^f}^A \cdot 2^{\ell-f}$. Bob computes $\langle z\rangle^B = (\langle x\rangle^B \gg f) - \langle w\rangle_{2^f}^B \cdot 2^{\ell-f}$.

Figure 8: Proposed One-Bit Approximate Truncation Protocol in the $\mathcal{F}_{\mathrm{B2A}}^{2^f}$-hybrid model.

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 $\binom{2}{1}$-$\mathtt{OT}_1$ instead of the expensive Mill protocol. Indeed, when MSB is zero, $w = \mathtt{msb}(\langle x\rangle^A) \vee \mathtt{msb}(\langle x\rangle^B)$. A similar protocol can also be derived when MSB is one and $w = \mathtt{msb}(\langle x\rangle^A) \wedge \mathtt{msb}(\langle x\rangle^B)$.

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 $\mathcal{F}_{\mathsf{Mill}}^{\ell-1}$, $\binom{4}{1}$-$\mathtt{OT}_\ell$, $\mathcal{F}_{\mathrm{B2A}}^{2^\ell}$, and $\mathcal{F}_{\mathsf{Mill}}^{f}$. This leads to a communication upper bound of $\lambda(\ell+f+2)+19\ell+14f$ bits when sub-block length $m=4$ in the millionaires' protocol. A direct replacement of $\mathtt{OT}$ with the VOLE version gives us an upper bound of $16\ell+11f$ bits, according to our complexity analysis in Table 2 and 3 ($\mathcal{F}_{\mathrm{B2A}}^{2^\ell}$ uses a single call of $\binom{2}{1}$-$\mathtt{COT}_\ell$). One-bit approximate truncation involves a call to $\mathcal{F}_{\mathsf{Mill}}^{\ell}$ and $\mathcal{F}_{\mathrm{B2A}}^{2^f}$, which gives a communication of 13ℓ 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

Inputs and Outputs: $\langle z\rangle_{2^\ell}^A, \langle z\rangle_{2^\ell}^B \leftarrow \mathsf{Trunc}_{\mathsf{msb}}\left(\langle x\rangle_{2^\ell}^A, \langle x\rangle_{2^\ell}^B, f\right)$ such that $x \in \mathbb{Z}_{2^\ell}$, $\mathtt{msb}(x) = 0$, and $z = x \gg f$ or $z = (x \gg f) - 1$.

1: Alice samples $\langle w\rangle_2^A \leftarrow_R \{0,1\}$.
2: If $\mathtt{msb}(\langle x\rangle^A) = 0$, Alice sets $(s_0, s_1) = (\langle w\rangle_2^A, 1 \oplus \langle w\rangle_2^A)$; otherwise, Alice sets $(s_0, s_1) = (1 \oplus \langle w\rangle_2^A, 1 \oplus \langle w\rangle_2^A)$.
3: Alice and Bob invoke $\binom{2}{1}$-$\mathtt{OT}_1$, where Alice inputs messages (s_0, s_1) and Bob inputs a choice $\mathtt{msb}(\langle x\rangle^B)$. Bob learns $\langle w\rangle_2^B$ as its output.
4: Alice and Bob invoke an instance of $\mathcal{F}_{\mathrm{B2A}}^{2^f}$ with inputs $(\langle w\rangle_2^A, \langle w\rangle_2^B)$ and learn $(\langle w\rangle_{2^f}^A, \langle w\rangle_{2^f}^B)$.
5: Alice computes $\langle z\rangle^A = (\langle x\rangle^A \gg f) - \langle w\rangle_{2^f}^A \cdot 2^{\ell-f}$. Bob computes $\langle z\rangle^B = (\langle x\rangle^B \gg f) - \langle w\rangle_{2^f}^B \cdot 2^{\ell-f}$.

Figure 9: Proposed Truncation Protocol with Input MSB=0 in the $\mathcal{F}_{\mathrm{B2A}}^{2^f}$-hybrid model.

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., μ) to the kernel $\mathbf{K}$ of the CONV layer. Then Alice and Bob jointly invoke the HomCONV protocol on the scaled kernel. After HomCONV, Alice adds the shift vector θ 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 $\mathbf{a}$ 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 $(b, \mathbf{a}) \in \mathbb{Z}_q^{N+1}$ to Bob for decryption. Our optimization suggests Alice send just some of the high-end bits of b and the values in $\mathbf{a}$ to Bob and skip the remaining low-end parts. In brief, Alice can skip the low $\ell_b = \lfloor \log_2(q/p) \rfloor - 1$ bits of b and the low $\ell_a = \lfloor \log_2(q/(6.6\sqrt{N}p)) \rfloor$ bits of the values in $\mathbf{a}$ 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 ℓ_b and ℓ_a to to Appendix F.

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

Truncation Protocol	Communication (bits)
Faithful [50]	$\lambda(\ell+f+2)+19\ell+14f$
Faithful (VOLE)	$16\ell+11f$
One-bit approx. (VOLE)	13ℓ
One-bit approx. (VOLE, MSB)	$f+4$

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*[2] protocol in EMP to support various application-level OT types, such as $\binom{n}{1}$-$\mathtt{OT}_\ell$. 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 $\mathsf{HE.pp}_{Cheetah} = \{N = 4096, q \approx 2^{105}, p = 2^{37}, \sigma = 3.2\}$. 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 $\mathsf{HE.pp}_{\mathsf{SCI}} = \{N = 8192, q \approx 2^{180}, p \approx 2^{37}, \sigma = 3.2\}$. 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.

[2] Our protocol can use any VOLE-style OT extension such as [10, 15, 60].

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.

FC	n_i, n_o	End2End Time LAN	End2End Time WAN	Commu.
[50, 52]	2048, 1001	1,533ms	1,587ms	0.50MB
	512, 10	17ms	60ms	0.50MB
Ours	2048, 1001	111ms	171ms	1.83MB
	512, 10	4ms	51ms	0.11MB
CONV	HW, C, M, h, s	**End2End Time LAN**	**End2End Time WAN**	**Commu.**
[50, 52]	$224^2, 3, 64, 7, 2$	25.52s	27.28s	76.02MB
	$224^2, 3, 64, 3, 2$	7.06s	8.78s	76.02MB
	$56^2, 64, 256, 1, 1$	8.21s	8.74s	28.01MB
	$56^2, 256, 64, 1, 1$	7.41s	8.51s	52.02MB
Ours	$224^2, 3, 64, 7, 2$	1.30s	2.11s	49.62MB
	$224^2, 3, 64, 3, 2$	1.33s	2.16s	49.62MB
	$56^2, 64, 256, 1, 1$	0.83s	1.10s	15.30MB
	$56^2, 256, 64, 1, 1$	0.70s	0.99s	17.07MB
BN	HW, C	**End2End Time LAN**	**End2End Time WAN**	**Commu.**
[50, 52]	$56^2, 64$	0.28s	0.50s	12.51MB
	$56^2, 256$	1.14s	2.17s	49.01MB
Ours	$56^2, 64$	0.22s	0.34s	6.84MB
	$56^2, 256$	0.88s	1.36s	27.34MB

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.

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 $\mathbb{Z}_{2^\ell}$ 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 $\mathbb{Z}_p$. To accept shares from $\mathbb{Z}_{2^\ell}$, they need to

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

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.

Benchmark	Method	End2End Time LAN	End2End Time WAN	Commu.
Millionaire	[50]	572ms	1,680ms	31.56MB
	Ours	496ms	663ms	2.72MB
		1.1×	2.5×	11.6×
Truncation	[50]	544ms	1,914ms	34.57MB
	Ours	432ms	655ms	2.86MB
		1.2×	2.9×	12.0×
Trunc. with known MSB	[50]	213ms	716ms	14.09MB
	Ours	31ms	101ms	0.16MB
		6.8×	7.1×	88.0×

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 SCI_{HE} and *Cheetah* with a single thread under the LAN and WAN settings. The performance accounts for 2^{16} calls to the protocols, which resembles a scenario of batch execution in neural network inference. From the table, we observe that our solution significantly reduces the communication cost, i.e., by more than 10× 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.

Table 7: Performance comparison with DELPHI [44, Figure 13] in the LAN setting. Inputs to the benchmarks were RGB images of 32×32 pixels. Aligned parameters of sharing modulo to $\mathbb{Z}_{2^{41}}$ and fixed-point precision to $f = 11$ with DELPHI were used for *Cheetah* in this benchmark.

Benchmark	System	End2End Time	Commu.
MiniONN	DELPHI [44]	≈ 110s	≈ 3.6GB
	Cheetah	3.55s	0.03GB
		≈ 30×	≈ 117×
ResNet32	DELPHI [44]	≈ 200s	≈ 6.5GB
	Cheetah	15.95s	0.11GB
		≈ 12×	≈ 58×

1GB = 2^{30} bytes.

Table 8: Performance comparison with SCI_{HE} and *SecureQ8* (three-party) on large-scale DNNs. We used the precision $f = 12$ for fixed-point values and 4 threads for the benchmarks.

Benchmark	System	End2End Time LAN	End2End Time WAN	Commu.
SqNet	SCI_{HE} [50]	41.1s	147.2s	5.9GB
	SecureQ8 [16]	4.4s	134.1s	0.8GB
	Cheetah	16.0s	39.1s	0.5GB
RN50	SCI_{HE} [50]	295.7s	759.1s	29.2GB
	SecureQ8 [16]	32.6s	379.2s	3.8GB
	Cheetah	80.3s	134.7s	2.3GB
DNet	SCI_{HE} [50]	296.2s	929.0s	35.4GB
	SecureQ8 [16]	22.5s	342.6s	4.6GB
	Cheetah	79.3s	177.7s	2.4GB

SqNet = SqueezeNet; RN50 = ResNet50; DNet = DenseNet121

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× – 5× faster than SCI_{HE} within 8% bandwidth consumption of SCI_{HE}. To the best of our knowledge, no prior secure two-party inference system can evaluate ResNet50 and DenseNet121 within 10 minutes under the commodity hardware and network conditions as ours.

Compare with a Three-party Approach. In Table 8, we also provide the performance of *SecureQ8* [16], one of the most efficient **three-party** secure inference framework. These results are obtained by re-running their implementation [54] under ours environment. As is shown, *SecureQ8* was 3× - 4× faster than *Cheetah* on LAN, while *Cheetah* was 2× - 3×

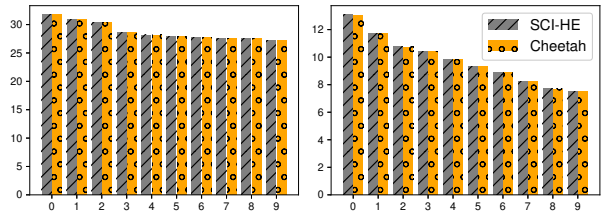

Figure 10: The top-10 values in the prediction vector of 1000 values from SqueezeNet (left) and ResNet50 (right).

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*$_{\mathsf{HE}}$. 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*$_{\mathsf{HE}}$ and *Cheetah* on ResNet50 and SqueezeNet. Here we can see that *Cheetah* gives almost the same prediction vectors as *SCI*$_{\mathsf{HE}}$ 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 furture 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.

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.

[4] Note that the implementation [54] uses fixed ring sizes i.e., $\ell \in \{64, 128\}$.

References

[1] Imagenet-sample-images. https://github.com/EliSchwartz/imagenet-sample-images, 2020. 1000 images, one random image per image-net class. For easy visualization/exploration of classes.

[2] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *CCS*, pages 308–318, 2016.

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *CCS*, pages 1231–1247, 2019.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 535–548, New York, NY, USA, 2013.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576, pages 420–432, 1991.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[7] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *WAHC*, pages 45–56. ACM, 2019.

[8] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Computing Frontiers*, pages 3–13, 2019.

[9] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[10] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS*, pages 291–308, 2019.

[11] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *ICML*, pages 812–821, 2019.

[12] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[13] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *CCS*, pages 395–412, 2019.

[14] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS*, volume 12726, pages 460–479, 2021.

[15] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO*, pages 502–534, 2021.

[16] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proc. Priv. Enhancing Technol.*, 2020(4):355–375, 2020.

[17] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In Michael Bailey and Rachel Greenstadt, editors, *USENIX*, pages 2183–2200, 2021.

[18] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *SIGPLAN*, pages 142–156, 2019.

[19] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society, 2015.

[20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *CRYPTO*, pages 823–852, 2020.

[21] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC 2009*, pages 169–178, 2009.

[22] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, pages 850–867, 2012.

[23] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, pages 201–210, 2016.

[24] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[25] Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML Confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC*, pages 1–21, 2012.

[26] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, R. Kim, R. Raman, P. C. Nelson, J. L. Mega, , and D. R. Webster. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316:2402–2410, 2016.

[27] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO*, pages 554–571, 2014.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[29] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. Privacy-preserving machine learning as a service. *PoPETs*, 2018(3):123–142, 2018.

[30] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, pages 2261–2269, 2017.

[31] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, 2016.

[32] Intel HEXL (release 1.2.2). https://arxiv.org/abs/2103.16400, October 2021.

[33] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, pages 145–161, 2003.

[34] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In Nadia Heninger and Patrick Traynor, editors, *USENIX*, pages 1895–1912, 2019.

[35] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *CCS*, pages 1209–1222, 2018.

[36] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX*, pages 1651–1669, 2018.

[37] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, pages 54–70, 2013.

[38] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: super-fast and robust privacy-preserving machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX*, pages 2651–2668, 2021.

[39] Alex Krizhevsky. Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.

[40] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure tensorflow inference. In *SP*, pages 336–353. IEEE, 2020.

[41] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. 2017.

[42] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, pages 619–631, 2017.

[43] Wenjie Lu and Jun Sakuma. More practical privacy-preserving machine learning as A service via efficient secure matrix multiplication. In Michael Brenner and Kurt Rohloff, editors, *WAHC*, pages 25–36, 2018.

[44] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. DELPHI: A cryptographic inference service for neural networks. In *USENIX*, pages 2505–2522, 2020.

[45] Payman Mohassel and Peter Rindal. ABY3: A Mixed Protocol Framework for Machine Learning. In *CCS*, pages 35–52, 2018.

[46] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *SP*, pages 19–38. IEEE Computer Society, 2017.

[47] Moni Naor and Benny Pinkas. Oblivious Transfer and Polynomial Evaluation. In *STOC*, pages 245–254, 1999.

[48] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX*, pages 2165–2182, 2021.

[49] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRnn: A math library for secure RNN inference. In *SP*, pages 1003–1020. IEEE, 2021.

[50] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In *CCS*, pages 325–342. ACM, 2020.

[51] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *USENIX*, pages 1501–1518, 2019.

[52] Secure and correct inference (SCI) library. https://github.com/mpc-msri/EzPC/tree/master/SCI, June 2021.

[53] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[54] Multi-protocol SPDZ. https://github.com/data61/MP-SPDZ/tree/master, February 2022.

[55] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *SP*, pages 3–18, 2017.

[56] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.

[57] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *ICLR*, 2019.

[58] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX*, pages 601–618, 2016.

[59] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2022.

[60] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS*, pages 1607–1626, 2020.

[61] Xiaoyong Zhu, George Iordanescu, Ilia Karmanovi, and Mazen Zawaideh. Using microsoft ai to build a lung-disease prediction model using chest x-ray images.

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 $\Pi_{\mathcal{F}}$ and follows the protocol specification honestly. Security is modeled by defining two interactions: a real interaction where Alice and Bob execute the protocol $\Pi_{\mathcal{F}}$ 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 $\mathcal{F}$ faithfully. Security requires that for every adversary $\mathcal{A}$ in the real interaction, there is an adversary Sim (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 $\mathbf{W}_1, \cdots, \mathbf{W}_d$. The client holds an input vector $\mathbf{x}$.

Definition 1 *A protocol Π between a server having as input model parameters $\mathcal{W} = (\mathbf{W}_1, \ldots, \mathbf{W}_d)$ and a client having as input a feature vector $\mathbf{x}$ 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 $\mathbf{x}$ of the client, the output of the client at the end of the protocol is the correct prediction $\mathcal{W}(\mathbf{x})$.*
- **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 Sim_C such that $\mathsf{View}_C^{\Pi} \approx_c \mathsf{Sim}_C(\mathsf{meta}, \mathsf{out})$ where View_C^{Π} is the view of the client in the execution of Π, meta includes the meta information (i.e., the public parameters $\mathsf{HE.pp}$, the public key pk, the number of layers, the size and type of each layer, and the activation) and out denotes the output of the inference.*

  We also require that a corrupted, semi-honest server does not learn anything about the private input $\mathbf{x}$ of the client. Formally, we require the existence of an efficient simulator Sim_S such that $\mathsf{View}_S^{\Pi} \approx_c \mathsf{Sim}_S(\mathsf{meta})$ where View_S^{Π} is the view of the server in the execution of Π.

B SIMD and Homomorphic Rotation

The SIMD technique [56] uses a discrete Fourier transform over the prime field $\mathbb{Z}_p$ to convert vectors $\mathbf{v}, \mathbf{u} \in \mathbb{Z}_p^N$ of N elements to ring elements $\hat{v} \in \mathbb{A}_{N,p}$ and $\hat{u} \in \mathbb{A}_{N,p}$, respectively. The product polynomial $\hat{v} \cdot \hat{u} \in \mathbb{A}_{N,p}$ decodes to the Hadamard product between the vectors $\mathbf{v}$ and $\mathbf{u}$. 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 $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(\rho_{5^k \bmod 2N}(\mathsf{sk}))$ where the automorphism $\rho_g : \mathbb{A}_{N,p} \mapsto \mathbb{A}_{N,p}$ is defined by $\rho_g(\hat{a}(X)) = \hat{a}(X^g) \bmod X^N + 1$.

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

Suppose $\mathsf{CT} = (\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$ be an RLWE ciphertext that decrypts to $\hat{m} \in \mathbb{A}_{N,p}$ and attaches with a noise polynomial $\hat{e}$. More specifically, to reveal the attached noise, the decryptor first computes the phase $\hat{b} + \hat{a} \cdot \mathsf{sk} \bmod q$ and then computes $p \cdot (\hat{b} + \hat{a} \cdot \mathsf{sk}) \bmod q$ which equals to $p \cdot \hat{e} \bmod q$ 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 $\mathbb{A}_{N,q}$ and then update the ciphertext as $\mathsf{CT}' = (\hat{b} + \hat{r} \bmod q, \hat{a})$. As a result, from the view of decryptor, the phase $\hat{b} + \hat{r} + \hat{a} \cdot \mathsf{sk} \bmod q$ distributes uniformly in $\mathbb{A}_{N,q}$. Thus no information of $\hat{e}$ is leaked from CT'.

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

$$-1 = \lceil -\frac{1}{2} - \frac{p}{2q} \rfloor < \lceil p \cdot (\hat{e} + \hat{r}_l + \hat{\mathfrak{e}})/q \rfloor < \lceil \frac{1}{2} + 1 + \frac{p}{2q} \rfloor = 2.$$
$$0 = \lceil 0 - \frac{p}{2q} \rfloor \leq \lceil p \cdot (\hat{r}_l + \hat{\mathfrak{e}})/q \rfloor \leq \lceil 1 + \frac{p}{2q} \rfloor = 1.$$

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

$$\Pr\left(\lceil \frac{p \cdot (\hat{r}[i] + \hat{e}[i] \bmod q)}{q} \rfloor - \lceil \frac{p \cdot \hat{r}[i]}{q} \rfloor \neq 0 \right) = \frac{4|\hat{e}[i]|}{q}$$

which is bounded by $\frac{2}{p}$, 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

Inputs and Outputs $\langle \mathbf{T}' \rangle^A, \langle \mathbf{T}' \rangle^B \leftarrow \mathsf{HomCONV}\left(\{\langle \mathbf{T} \rangle^A, \mathbf{K}\}, \{\langle \mathbf{T} \rangle^B, \mathsf{sk}\}\right)$ such that $\langle \mathbf{T} \rangle^A, \langle \mathbf{T} \rangle^B \in \mathbb{Z}_p^{C \times H \times W}$, $\mathbf{K} \in \mathbb{Z}_p^{M \times C \times h \times h}$ and $\mathbf{T}' = \mathsf{Conv2D}(\mathbf{T}, \mathbf{K}; s) \in \mathbb{Z}_p^{M \times H' \times W'}$.
Public Parameters: $\mathsf{pp} = (\mathsf{HE.pp}, \mathsf{pk}, M, C, H, W, h, s)$.

- Shape meta M, C, H, W, h such that $h^2 \leq N$ and stride $s > 0$. The optimal sizes of the partition winodws that minimize $\mathrm{argmin}_{H_\mathrm{w}, W_\mathrm{w}} \lceil \frac{C}{\lfloor N/(H_\mathrm{w} W_\mathrm{w}) \rfloor} \rceil \lceil \frac{H-h+1}{H_\mathrm{w}-h+1} \rceil \lceil \frac{W-h+1}{W_\mathrm{w}+h-1} \rceil$ such that $h \leq H_\mathrm{w} \leq H$, $h \leq W_\mathrm{w} \leq W$ and $H_\mathrm{w} W_\mathrm{w} \leq N$.
- The partition window size along the C-axis and M-axis is $C_w = \min(C, \lfloor \frac{N}{H_\mathrm{w} W_\mathrm{w}} \rfloor)$ and $M_\mathrm{w} = \min(M, \lfloor \frac{N}{C_\mathrm{w} H_\mathrm{w} W_\mathrm{w}} \rfloor)$.
- Set $d_M = \lceil \frac{M}{M_\mathrm{w}} \rceil, d_C = \lceil \frac{C}{C_\mathrm{w}} \rceil, d_H = \lceil \frac{H-h+1}{H_\mathrm{w}-h+1} \rceil$ and $d_W = \lceil \frac{W-h+1}{W_\mathrm{w}-h+1} \rceil$. $O_w = H_\mathrm{w} W_\mathrm{w} (M_\mathrm{w} C_\mathrm{w} - 1) + W_\mathrm{w}(h-1) + h - 1$.
- Set $H' = \lfloor \frac{H-h+s}{s} \rfloor$, $W' = \lfloor \frac{W-h+s}{s} \rfloor$, $H'_\mathrm{w} = \lfloor \frac{H_\mathrm{w}-h+s}{s} \rfloor$ and $W'_\mathrm{w} = \lfloor \frac{W_\mathrm{w}-h+s}{s} \rfloor$.

1: Bob first partitions $\langle \mathbf{T} \rangle^B$ into blocks along the H-axis and W-axis $\langle \mathbf{T}_{\alpha,\beta} \rangle^B \in \mathbb{Z}_q^{C \times H_\mathrm{w} \times W_\mathrm{w}}$ for $\alpha \in [[d_H]]$ and $\beta \in [[d_W]]$. Each block $\langle \mathbf{T}_{\alpha,\beta} \rangle^B$ consists of H_w continuous rows and W_w continuous columns of $\langle \mathbf{T} \rangle^B$. Indeed, $\langle \mathbf{T}_{\alpha,\beta} \rangle^B$ is taken from the $\alpha(H_\mathrm{w} - h + 1)$-th row and $\beta(W_\mathrm{w} - h + 1)$-th column of $\langle \mathbf{T} \rangle^B$. Zero-padding might be used to make sure all $\langle \mathbf{T}_{\alpha,\beta} \rangle^B$ blocks contain the same number of rows and columns.
2: Bob then splits the channels of each block tensor $\langle \mathbf{T}_{\alpha,\beta} \rangle^B$ into *non-overlapping* blocks $\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^B \in \mathbb{Z}_p^{C_\mathrm{w} \times H_\mathrm{w} \times W_\mathrm{w}}$ for $\gamma \in [[d_C]]$. Also, zero-padding might be used to make sure all $\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^B$ blocks contain the same number of channels.
3: Bob encrypts and then sends the RLWE ciphertexts $\{\mathsf{CT}_{\gamma,\alpha,\beta} = \mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(\pi^i_{\mathrm{conv}}(\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^B))\}$ to Alice.
4: Alice partitions $\langle \mathbf{T} \rangle^A$ into blocks $\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^A \in \mathbb{Z}_p^{C_\mathrm{w} \times H_\mathrm{w} \times W_\mathrm{w}}$ following the same manner in Step 1 and Step 2. Then Alice encodes each block to a polynomial via $\langle \hat{t}_{\gamma,\alpha,\beta} \rangle^A = \pi^i_{\mathrm{conv}}(\langle \mathbf{T}_{\gamma,\alpha,\beta} \rangle^A)$.
5: Alice then splits $\mathbf{K}$ into sub-kernels along the M-axis, i.e., $\mathbf{K}_\theta \in \mathbb{Z}_p^{M_\mathrm{w} \times C \times h \times h}$ for $\theta \in [[d_M]]$. Then Alice further split each $\mathbf{K}_\theta$ into smaller block along the C-axis, i.e., $\mathbf{K}_{\theta,\gamma} \in \mathbb{Z}_p^{M_\mathrm{w} \times C_\mathrm{w} \times h \times h}$ 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 $\hat{k}_{\theta,\gamma} = \pi^w_{\mathrm{conv}}(\mathbf{K}_{\theta,\gamma}, O_w)$.
6: Alice samples $\mathbf{R}$ from $\mathbb{Z}_q^{M \times H' \times W'}$ uniformly at random and outputs $\langle \mathbf{T}' \rangle^A = -\lceil p \cdot \mathbf{R}/q \rfloor \bmod p$
7: On receiving $\{\mathsf{CT}_{\gamma,\alpha,\beta}\}$, Alice computes $\mathsf{CT}'_{\theta,\alpha,\beta} = \boxplus_{\gamma \in [[d_C]]}(\mathsf{CT}_{\gamma,\alpha,\beta} \boxplus \langle \hat{t}_{\gamma,\alpha,\beta} \rangle^A) \boxtimes \hat{k}_{\theta,\gamma}$ 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 LWE ciphertext to Bob $\mathsf{ct}_{c',i',j'} = \mathsf{Extract}(\mathsf{CT}'_{\theta,\alpha,\beta}, O_w - cC_\mathrm{w}H_\mathrm{w}W_\mathrm{w} + isH_\mathrm{w} + js) + \mathbf{R}[c', i', j']$, where the index of $\mathsf{CT}'_{\theta,\alpha,\beta}$ is calculated as $\theta = \lfloor c'/M_\mathrm{w} \rfloor$, $\alpha = \lfloor i's/(H_\mathrm{w} - h + 1) \rfloor$, and $\beta = \lfloor j's/(W_\mathrm{w} - h + 1) \rfloor$. The position of the extracting coefficient is determined by $c \equiv c' \bmod M_\mathrm{w}$, $i \equiv i' \bmod H'_w$ and $j \equiv j' \bmod W'_w$.
9: On receiving the ciphertexts $\{\mathsf{ct}_{c',i',j'}\}$, Bob outputs $\langle \mathbf{T}' \rangle^B \in \mathbb{Z}_p^{M \times H' \times W'}$ where $\langle \mathbf{T}' \rangle^B[c', i', j'] = \mathsf{LWE}^{-1}_{\mathsf{sk}}(\mathsf{ct}_{c',i',j'})$.

Figure 11: Proposed Secure Convolution Protocol (Full Version)

field is error sensitive. That is 1-bit error in the polynomial coefficients will render a large error in the decoding vector.

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 $C_w \times W_w \times H_w$ along the three dimensions. Bob then sends to Alice $d_C d_H d_W$ RLWE ciphertexts that each of them encrypts one partition block of $\langle \mathbf{T} \rangle^B$ in Step 3. In the next two steps, Alice first partitions its share $\langle \mathbf{T} \rangle^A$ following the same partitioning manner in Step 1 and Step 2. After that Alice partitions the kernel $\mathbf{K}$ into $d_M d_C$ non-overlapping blocks Then Alice can encode each of them using π^w_{conv} with an identical parameter O_w that is because the tensor $\mathbf{T}$ is partitioned into blocks of the same shape $C_w \times H_w \times W_w$. On receiving the RLWE ciphertexts from Bob, Alice then computes the secure convolution using $d_M d_C d_H d_W$ homomorphic multiplications and homomorphic additions. Indeed, the RLWE ciphertext $\mathsf{CT}'_{\theta,\alpha,\beta}$ in Step 7 corresponds to the convolution of the block tensor $\mathbf{T}_{\alpha,\beta}$ and the sub-kernels $\mathbf{K}_\theta$. As a result, each RLWE ciphertext $\mathsf{CT}'_{\theta,\alpha,\beta}$ obtains at most $M_w H'_w W'_w$ values of $\mathbf{T}'$ in its encrypted coefficients. Finally, Alice extracts

$MH'W'$ LWE ciphertexts that each of them encrypts one entry of the output tensor $\mathbf{T}'$. 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*

$$\hat{t}'[x] = \sum_{0 \leq d \leq x} \hat{t}[d]\hat{k}[x-d] - \sum_{x < d < N} \hat{t}[d]\hat{k}[N+x-d] \qquad (6)$$

for all $x \in [[N]]$. Since $\hat{t}[x]$ is zero for all $x \geq CHW$ and the target position $O' + i'sW + j's > CHW$, we thus have

$$\hat{t}'[O' + i'sW + j's] = \sum_{d < CHW} \hat{t}[d]\hat{k}[O' + i'sW + j's - d]$$

$$= \sum_{c,i,j} \hat{t}[cHW + iW + j]\hat{k}[O' + i'sW + j's - (cHW + iW + j)]$$

$$= \sum_{c,i,j} \hat{t}[cHW + iW + j]\hat{k}[O' - cHW - (i - i's)W - (j - j's)]$$

$$= \sum_{c,i',j'} \hat{t}[cHW + (i's + l)W + j's + l']\hat{k}[O' - cHW - lW - l']$$

The last line replaces $i = i's + l$ and $j = j's + l'$. Also, according to the definition of π^w_{conv}, the value $\hat{k}[O' - cHW - lW - l']$ 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.

$$= \sum_{\substack{c \in [[C]] \\ l,l' \in [[h]]}} \hat{t}[cHW + (i's + l)W + j's + l']\hat{k}[O' - cHW - lW - l']$$

$$= \sum_{\substack{c \in [[C]] \\ l,l' \in [[h]]}} \mathbf{T}[c, i's + l, j's + l']\hat{k}[O - c'CHW - cHW - lW - l']$$

$$= \sum_{\substack{c \in [[C]] \\ l,l' \in [[h]]}} \mathbf{T}[c, i's + l, j's + l']\mathbf{K}[c', c, l, l']$$

The final line is exactly $\mathbf{T}'[c', i', j']$. ■

Proof 4 (Theorem 2.) *The correctness of Theorem 2 is directly derived from Proposition 2. We now show the the privacy part.*
*(**Corrupted Alice**.) Alice's view of $\mathsf{View}_A^{\mathsf{HomCONV}}$ consists of an RLWE ciphertext CT'. The simulator Sim_A for this view can be constructed as follows.*

1. *Given the access to* meta, Sim_A *outputs the ciphertext $\widetilde{\mathsf{CT}}' = \mathsf{RLWE}^{N,q}_{\mathsf{pk}}(0)$ to Alice.*

Table 9: DNN Architectures.

Networks	linear operations			non-linear operations	
	CONV	BN	FC	ReLU-then-Trunc	Trunc
MNet	7	0	1	7	0
RN32	34	34	1	31	37
SqNet	26	0	0	26	0
RN50	53	49	1	49	49
DNet	121	121	0	121	120

MNet = MiniONN; RN32 = ResNet32; RN50 = ResNet50
SqNet = SqueezeNet; DNet = DenseNet121

The security against a corrupted Alice (Server) is directly reduced to the semantic security of the underly encryption. Thus we have $\mathsf{View}_A^{\mathsf{HomCONV}} \approx_c \mathsf{Sim}_A(\mathsf{meta})$.
*(**Corrupted Bob**.) Bob's view of $\mathsf{View}_B^{\mathsf{HomCONV}}$ consists of LWE ciphertexts $\{\mathsf{ct}_{c',i',j'}\}$, and the decryption of these LWE ciphertexts, i.e., $\langle \mathbf{T}' \rangle^B$. The simulator Sim_C for this view can be constructed as follows.*

1. *On receiving the RLWE ciphertext* CT *from Bob and given the access to* meta, Sim_B *samples uniform random polynomial $\hat{r} \in \mathbb{A}_{N,p}$ and computes $\widetilde{\mathsf{CT}} = \mathsf{RLWE}^{N,q}_{\mathsf{pk}}(\hat{r})$.*
2. *For each $c' \in [[M]], i' \in [[H']]$ and $j' \in [[W']]$, Sim_B outputs an LWE ciphertext $\tilde{\mathsf{ct}}_{c',i',j'} = \mathsf{Extract}(\widetilde{\mathsf{CT}})$ to Bob.*
3. *Given the access to* out, Sim_B *outputs the tensor $\tilde{\mathbf{T}}$ such that $\tilde{\mathbf{T}}[c', i', j'] = \hat{r}[O - c'CHW + i'sW + j's]$ for each $c' \in [[M]], i' \in [[H']]$ and $j' \in [[W']]$.*

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

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

F On the Calculation of ℓ_a and ℓ_b

Remind that the LWE ciphertext $(b, \mathbf{a}) \in \mathbb{Z}_q^{N+1}$ of message $m \in \mathbb{Z}_p$ is decrypted by first computing $m_e = b + \mathbf{a}^\top \mathbf{s}$ which equals to $\lfloor q/p \rceil m + e$ for some error e. Then we obtain the message as $m = \lfloor m_e p/q \rceil$. In other words, the low $\ell' = \lfloor \log_2(q/p) \rfloor$ bits of m_e will be dropped during the divide-then-round and thus are useless for decryption. This motives us to skip some of the low-end bits of b and $\mathbf{a}$, 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 $\mathbf{a}$ as $b = b_H 2^{\ell_b} + b_L$ and $\mathbf{a} = \mathbf{a}_H 2^{\ell_a} + \mathbf{a}_L$ such that $0 \leq b_L < 2^{\ell_b}$ and the values in $\mathbf{a}_L$ are 2^{ℓ_a}-bounded. Then m_e can be written

Inputs & Outputs: $\langle z\rangle_2^A, \langle z\rangle_2^B \leftarrow \mathsf{AND}\left(\langle x\rangle_2^A, \langle x\rangle_2^B, \langle y\rangle_2^A, \langle y\rangle_2^B\right)$ s.t. $x, y, z \in \{0,1\}$ and $z = x \wedge y$.

1: Alice and Bob jointly generate a beaver triple: $\langle c\rangle_2^A \oplus \langle c\rangle_2^B = (\langle a\rangle_2^A \oplus \langle a\rangle_2^B) \wedge (\langle b\rangle_2^A \oplus \langle b\rangle_2^B)$.
2: Alice computes $\langle e\rangle_2^A = \langle x\rangle_2^A \oplus \langle a\rangle_2^A$ and $\langle f\rangle_2^A = \langle y\rangle_2^A \oplus \langle b\rangle_2^A$. Bob computes $\langle e\rangle_2^B = \langle x\rangle_2^B \oplus \langle a\rangle_2^B$ and $\langle f\rangle_2^B = \langle y\rangle_2^B \oplus \langle b\rangle_2^B$.
3: Alice and Bob open e and f.
4: Alice computes $\langle z\rangle_2^A = (\langle a\rangle_2^A \cdot f) \oplus (\langle b\rangle_2^A \cdot e) \oplus \langle c\rangle_2^A$, and Bob computes $\langle z\rangle_2^B = (e \cdot f) \oplus (\langle a\rangle_2^B \cdot f) \oplus (\langle b\rangle_2^B \cdot e) \oplus \langle c\rangle_2^B$

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

Inputs and Outputs: $\langle d\rangle_{2^f}^A, \langle d\rangle_{2^f}^B \leftarrow \mathsf{B2A}\left(\langle c\rangle_2^A, \langle c\rangle_2^B, f\right)$ such that $c \in \{0,1\}$ and $d = c$.

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

Figure 13: B2A Protocol.

as $m_e = b_L + \mathbf{a}_L^\top \mathbf{s} + b_H 2^{\ell_b} + \mathbf{a}_H^\top \mathbf{s} 2^{\ell_a}$. If the sum $b_L + \mathbf{a}_L^\top \mathbf{s} < 2^{\ell'}$ then we can just send the high-end parts b_H and $\mathbf{a}_H$ for decryption. The key here is to find out an upper bound for $\mathbf{a}_L^\top \mathbf{s}$. Remind that the secret vector $\mathbf{s}$ distributes uniformly in $\{0, \pm 1\}^N$. Since $\mathbf{a}$ is basically an uniform random vector, we can assume that the values in $\mathbf{a}_L$ distributes uniformly in $[0, 2^{\ell_a})$. Then the variance of $\mathbf{a}_L^\top \mathbf{s}$ is $\mathsf{Var} \approx \frac{2N}{9}(2^{\ell_a})^2$. We use $7\sqrt{\mathsf{Var}}$ (i.e., 7 times standard deviation) as a high-probability upper bound ($\approx 1 - 2^{-38.5}$) on the value $\mathbf{a}_L^\top \mathbf{s}$. Finally, to determine the concrete ℓ_b and ℓ_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 $7\sqrt{\frac{2N}{9}} 2^{\ell_a} + 2^{\ell_b} < 2^{\ell'}$, or just setting $\ell_b = \ell' - 1$ and $\ell_a = \ell' - 1 - \lfloor \log_2(7\sqrt{\frac{2N}{9}}) \rfloor$.

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

We describe the classic protocol for computing $\mathcal{F}_{\text{AND}}$ in Figure 12, where beaver triples are generated with $\binom{2}{1}$-$\mathtt{ROT}_1$ [4]. We describe a protocol for $\mathcal{F}_{\text{B2A}}^{2^f}$ in Figure 13, which is also used in CryptoFlow2 [50, 52], except that $\binom{2}{1}$-$\mathtt{COT}_f$ is instantiated with VOLE-style OT.