METEOR: Improved Secure 3-Party Neural Network Inference with Reducing Online Communication Costs

Ye Dong
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
dongye@iie.ac.cn

Xiaojun Chen*
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
chenxiaojun@iie.ac.cn

Weizhan Jing
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
jingweizhan@iie.ac.cn

Kaiyun Li
Beijing Baidu Netcom Science and Technology Co.,Ltd.
Beijing, China
likaiyun@baidu.com

Weiping Wang
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
wangweiping@iie.ac.cn

ABSTRACT

Secure neural network inference has been a promising solution to private Deep-Learning-as-a-Service, which enables the service provider and user to execute neural network inference without revealing their private inputs. However, the expensive overhead of current schemes is still an obstacle when applied in real applications. In this work, we present METEOR, an online communication-efficient and fast secure 3-party computation neural network inference system aginst semi-honest adversary in honest-majority. The main contributions of METEOR are two-fold: i) We propose a new and improved 3-party secret sharing scheme stemming from the *linearity* of replicated secret sharing, and design efficient protocols for the basic cryptographic primitives, including linear operations, multiplication, most significant bit extraction, and multiplexer. ii) Furthermore, we build efficient and secure blocks for the widely used neural network operators such as Matrix Multiplication, ReLU, and Maxpool, along with exploiting several specific optimizations for better efficiency. Our total communication with the setup phase is a little larger than SecureNN (PoPETs'19) and FALCON (PoPETs'21), two state-of-the-art solutions, but the gap is not significant when the online phase must be optimized as a priority. Using METEOR, we perform extensive evaluations on various neural networks. Compared to SecureNN and FALCON, we reduce the online communication costs by up to 25.6× and 1.5×, and improve the running-time by at most 9.8× (resp. 8.1×) and 1.5× (resp. 2.1×) in LAN (resp. WAN) for the online inference.

*Corresponding author

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583272

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**.

KEYWORDS

Privacy, Security, Secret Sharing, Neural Network

ACM Reference Format:
Ye Dong, Xiaojun Chen, Weizhan Jing, Kaiyun Li, and Weiping Wang. 2023. METEOR: Improved Secure 3-Party Neural Network Inference with Reducing Online Communication Costs. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3543507.3583272

1 INTRODUCTION

In the Deep-Learning-as-a-Service (DLaaS) paradigm, the service provider offers a trained neural network (NN), and a user calls a well-defined API for data analysis. Aiming to alleviate the privacy concerns associated with DLaaS [1, 3], existing works have introduced secure computation to enable *Secure Inference*. Secure inference exploits cryptographic primitives to ensure that the only information available for the user is the inference result, and nothing more is revealed to either party.

Secure inference protocols can provide high privacy protection, but the key concern is how to obtain privacy with satisfying efficiency. Note that different cryptographic tools offer their characteristics and trade-offs. In particular, fully homomorphic encryption (FHE)-based methods are efficient in communication but still limited by expensive computation burdens [29, 30, 34]. Garbled circuits [73] (GC)-based schemes only require a constant round of interactions but have a high communication overhead and are expensive for arithmetic operations [5, 59]. Secret sharing [64]-based approaches provide efficient arithmetic operations and support non-linear functions [46, 49, 50, 60, 70, 71] using much less communication, yet usually require interactions in proportion to the depth of Multiplication (MULT) gates. Among the secret sharing-based works, 2-out-of-3 replicated secret sharing-based secure 3-party

computation (3PC) approaches [49, 71] have achieved significant improvements and gained much attention.

However, the online communication of replicated secret sharing is still the efficiency bottleneck even in *semi-honest* model. The costly online communication limits the users' query throughput, especially in WAN. Therefore, improving online communication (and running-time) is challenging and a priority in real applications.

The online communication mainly stems from MULT, and we analyze the detailed costs in the following aspects: **i) Costs of Resharing:** Multiplying two ℓ-bit integers (2-MULT) generates 3-out-of-3 secret shared intermediate results. This requires interactive communication of ℓ bits per party for *resharing* 3-out-of-3 shares into 2-out-of-3 shares in 1 round for maintaining correctness and consistency; **ii) Costs of** 2-MULT **with Faithful Truncation:** When evaluating 2-MULT on two ℓ-bit fixed-point inputs, the parties need to truncate the product to avoid overflow. However, the best known protocol for 2-MULT equipped with faithful truncation needs an online communication of $\frac{4}{3}\ell$ bits per party in 1 round (incur $\frac{1}{3}\ell$ more bits than 2-MULT on integers); **iii) Costs of** N-MULT**:** N-MULT takes N ℓ-bit integers as inputs and multiplies them to produce the product. N-MULT plays an important role in extracting the most significant bit, but existing works achieve N-MULT by utilizing 2-MULT in a tree-manner. This incurs an online communication of $(N-1)\ell$ bits in $\lceil \log_2 N \rceil$ rounds.

In this paper, we focus on improving the online communication costs of the latter two kinds of MULT gates. Although our method requires more costs for the setup phase (*i.e.*, communicating $(2^N - 1 - N)\ell$ bits in $\log_2 N$ rounds per party for N-MULT), our significant improvements in the online phase are beneficial in real applications and might be of independent interest. Formally, our techniques and contributions are as follows:

Our Techniques We propose Meteor, an online communication-efficient and fast secure neural network inference system. Meteor achieves its performance improvements via our improved 3PC protocols and specific optimizations for secure NN operators. Following previous works [70, 71], our 3PC protocols are secure against a *semi-honest* adversary in honest-majority. We build several primitives with a focus on online efficiency by exploiting a *function-dependent* but *input-independent* setup.

Our construction is similar to the sharing semantics of ABY2.0 [54], but exploits a different perspective from the *linearity* of replicated secret sharing [6, 49, 71]: For the MULT, we only need linear operations of replicated secret sharing to generate 2-out-of-3, instead of 3-out-of-3, secret shares in the online phase. This new perspective can accelerate the 2-MULT for fixed-point inputs and N-MULT for integer inputs, and bring several further optimizations for more complex primitives, such as most significant bit extraction (MSB Ext.) and multiplexer (MUX). Besides, our *linearity* perspective is more straightforward to be generalized to any kind of linear secret sharing. Detailed comparison is shown in § 3.1.

Contributions Formally, we have the following contributions:

- **Improved Secure 3-Party Computation:** We propose an improved 3PC secret sharing scheme ($[\![\cdot]\!]$-sharing) and construct a set of basic cryptographic primitives, including linear operations (Lops), MULT, MSB Ext., MUX, and etc. Our primitives are more online communication- and round-efficient than that of SecureNN [70] and Falcon [71]. The detailed theoretical improvements are shown in Table 1.
- **Optimized Secure NN Operators:** Furthermore, we construct fast protocols for Matrix Multiplication (MatMul), ReLU function, and Maxpool (MP) based on our basic primitives with specific optimizations. Compared to SecureNN [70] and Falcon [71], we achieve 1.2-6× and 1.3-1.6× improvements in terms of communication costs. Meanwhile, we are approximately 1.8-20× and 1.5 faster for the secure online evaluation of NN operators, respectively.
- **Efficient Secure Inference:** In the end, we perform extensive secure inference experiments on various neural networks and datasets in both LAN and WAN settings: i) For single inference, we reduce the online communication by upto 25.6× and 1.5×, and improve the online running-time by at most 9.8× (resp. 8.1×) and 1.5× (resp. 2.1×) in LAN (resp. WAN) compared to SecureNN and Falcon, respectively. ii) For batch inference, Meteor is more scalable than Falcon. Specially, we improve the communication and running-time by both around 1.5× in WAN. Our source code is available: https://github.com/Ye-D/Meteor.

Organization We present the background and preliminaries in § 2, and give a high-level overview of Meteor in § 3. We propose efficient protocols for the basic primitives in § 4 and justify their security in § 5. And in § 6, we construct the optimized secure NN operators. The experimental results are illustrated in § 7. We discuss related works in § 8 and conclude this work in § 9.

2 BACKGROUND & PRELIMINARIES

We introduce the background and preliminaries about neural network and 3PC replicated secret sharing in this section.

2.1 Notations

The main notations are summarized in Table 2.

2.2 Neural Network

The computational flow of a neural network is composed of multiple linear and non-linear layers. Each layer receives input and processes it to produce an output that serves as input to the next layer.

Linear Layers Typical linear layers in NN inference include Fully-Connected (FC), Convolution (CONV), and Batch Normalization (BatchNorm, only being linear layer in NN inference):

- **FC:** Given input vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$, a FC layer generates the output $\mathbf{y} \in \mathbb{R}^{m\times 1}$ as $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{m\times n}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^{m\times 1}$ is the bias term. More generally, neural networks often take a batch of images as inputs $\mathbf{X}^{n\times|B|}$ ($|B|$ is the batchsize), thus the FC layer can be computed with matrix multiplication as $\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{B}$.
- **CONV:** The CONV layer computes the dot product of a small weight matrix (*filter*) and the neighborhood of an element of the input. The process is sliding each filter with a certain *stride*, and the size of filter is called *filter size*. For a generalized exposition on CONV, please refer to [70].
- **BatchNorm:** A BatchNorm layer is typically applied to shift its input x to amenable ranges. During the inference, the

Table 1: Online communication (Comm., in bits) and round complexity of SecureNN, Falcon, and Meteor. q is the smallest prime with $q \geq \ell$, and $k = \lceil \log_2 q \rceil$. For MSB Ext., we set $N = 3, 4$ for N-MULT. †: In SecureNN, MSB Ext. is equal to ShareConvert+Compute MSB (except Select Share), we summarize the corresponding communication and round costs here.

Framework		SecureNN		Falcon		Meteor	
Complexity		Comm.	Round	Comm.	Round	Comm.	Round
2-MULT	Integers	$\frac{4}{3}\ell$	1	ℓ	1	ℓ	1
	Fixed-Point	$\frac{4}{3}\ell$	1	$\frac{4}{3}\ell$	1	ℓ	1
N-MULT		$(N-1)\frac{4}{3}\ell$	$\lceil \log_2 N \rceil + 1$	$(N-1)\ell$	$\lceil \log_2 N \rceil$	ℓ	1
MSB Ext.		$\frac{8}{3}\ell k + \frac{14}{3}\ell^{\dagger}$	$7^{\dagger}$	$(2\ell+1)k + \ell$	$\lceil \log_2(\ell+1) \rceil + 3$	$(\frac{4}{3}\ell+1)k$	$\lceil \log_4(\ell+1) \rceil + 2$
MUX		$\frac{4}{3}\ell$	1	$1+\ell$	2	ℓ	1

Table 2: Notation table.

P_i	party i in 3PC
$\mathbf{X}$	uppercase bold letter denotes matrix
$\mathbf{x}$	lowercase bold letter denotes vector
x	lowercase letter denotes scalar
$\mathsf{x}[i]$	the ith bit of x
$[\cdot]$	3-out-of-3 sharing
$\langle\cdot\rangle$	2-out-of-3 replicated secret sharing
$[\![\cdot]\!]$	our 3PC secret sharing
$\mathbb{Z}_{2^\ell}$	discrete ring modulo 2^ℓ
$\mathbb{F}_q$	field modulo prime q
$\mathcal{F}_f$	the ideal functionality for $f(\cdot)$
$\in_R$	random sample

BatchNorm parameters γ and β are fixed, BatchNorm normalizes x as $\gamma \cdot x + \beta$.

Non-Linear Layers NN uses activation functions to model non-linear relationships between input and output. And Pool functions sometimes are applied.

- **Activation:** The activation functions are applied in element-wise. One of the most popular activation functions is ReLU function: $\mathsf{ReLU}(x) = \max(0, x)$. Other activation functions inlcude Sigmoid, Tanh, and etc [52];
- **Pool:** Pooling arranges inputs into several windows and aggregates elements of each window. Maxpool (resp. Avgpool) calculates the maximum (resp. average) for each window.

2.3 3PC Replicated Secret Sharing

Secret value $\mathsf{x} \in \mathbb{Z}_{2^\ell}$ is shared by three random values $\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2 \in \mathbb{Z}_{2^\ell}$ with $\mathsf{x} = \mathsf{x}_0 + \mathsf{x}_1 + \mathsf{x}_2 \pmod{2^\ell}$ [6, 49, 71]. In 3-out-of-3 sharing ($[\cdot]$-sharing), P_i has $[\mathsf{x}]_i = \mathsf{x}_i$. In replicated secret sharing (2-out-of-3, $\langle\cdot\rangle$-sharing), P_i gets $\langle\mathsf{x}\rangle_i = (\mathsf{x}_i, \mathsf{x}_{i+1})$. Without special declaration, we compute in $\mathbb{Z}_{2^\ell}$ and omit $\pmod{2^\ell}$ for brevity.

Sharing and Reconstruction To achieve functionality $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{SHARE}}$, secret owner samples random $\mathsf{x}_1, \mathsf{x}_2 \in_R \mathbb{Z}_{2^\ell}$, sets $\mathsf{x}_0 = \mathsf{x} - \mathsf{x}_1 - \mathsf{x}_2$, and sends $\langle\mathsf{x}\rangle_i = (\mathsf{x}_i, \mathsf{x}_{i+1})$ to P_i. And to implement $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{REC}}$, P_i sends x_{i+1} to P_{i-1} such that P_{i-1} reconstructs $\mathsf{x} = \mathsf{x}_0 + \mathsf{x}_1 + \mathsf{x}_2$ for $i \in \{0, 1, 2\}$.

Linear Operations Let (c_1, c_2, c_3) be public constants, and $(\langle\mathsf{x}\rangle, \langle\mathsf{y}\rangle)$ be two secret-shared values. Then, $\langle c_1\mathsf{x} + c_2\mathsf{y} + c_3\rangle$ can be computed as $(c_1\mathsf{x}_0 + c_2\mathsf{y}_0 + c_3, c_1\mathsf{x}_1 + c_2\mathsf{y}_1, c_1\mathsf{x}_2 + c_2\mathsf{y}_2)$ where P_i can compute its share locally. When $(c_1 = 1, c_2 = 1, c_3 = 0)$, we get $\langle\mathsf{x} + \mathsf{y}\rangle$.

Multiplication Functionality $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{MULT}}$ multiplies two shared values $\langle\mathsf{x}\rangle$ and $\langle\mathsf{y}\rangle$, existing protocol achieves this as follows: i) First, P_i computes $\mathsf{z}_i = \mathsf{x}_i\mathsf{y}_i + \mathsf{x}_{i+1}\mathsf{y}_i + \mathsf{x}_i\mathsf{y}_{i+1}$ locally such that z_i is $[\cdot]$-shared. ii) Parties then perform *re-sharing* by letting P_i sends $\mathsf{z}'_i = \alpha_i + \mathsf{z}_i$ to P_{i-1}, where $\alpha_0 + \alpha_1 + \alpha_2 = 0$ (P_i can generate α_i in the setup phase as [6, 49, 71]). iii) Finally, $\{(\mathsf{z}'_0, \mathsf{z}'_1), (\mathsf{z}'_1, \mathsf{z}'_2), (\mathsf{z}'_2, \mathsf{z}'_0)\}$ form $\langle\mathsf{x} \cdot \mathsf{y}\rangle$.

In the case of $\ell > 1$ (*e.g.*, $\ell = 64$) which support arithmetic operations (*e.g.*, $+$, $-$, and $\cdot$), we refer to this type as *Arithmetic Sharing* and use notation $\langle\cdot\rangle$. *Boolean Sharing* ($\langle\cdot\rangle^2$) refers to $\ell = 1$ where $+$, $-$ and $\cdot$ are respectively replaced by bit-wise $\oplus$ and $\wedge$.

MSB Extraction The key step of comparing $\langle\mathsf{x}\rangle \geq \langle\mathsf{y}\rangle$ in two's complement representation is extracting the most significant bit of $\langle\mathsf{z}\rangle = \langle\mathsf{x}\rangle - \langle\mathsf{y}\rangle$. General methods either re-interpret the arithmetic sharing as boolean sharing and evaluate an addition circuit on boolean shares to compute $\langle\mathsf{msb}(\mathsf{z})\rangle^2$, or employ garbled circuits to extract the most significant bit. Recently, Wagh *et al.* proposed an efficient MSB Ext. method based on wrap function and bit decomposition in replicated secret sharing [71]. We follow their approach in Meteor but optimize the online efficiency with our improved secret sharing scheme. And we plan to improve other MSB Ext. methods [28, 47, 49] with our novel secret sharing for future work.

Fixed-Point Representation In secure NN inference, we need to encode floating-point numbers as integers in rings [49, 50, 71]. Given floating-point $x \in \mathbb{R}$, its encoding is as: $\mathsf{x} = \lfloor 2^d \cdot x \rfloor \pmod{2^\ell}$, where it is usually $\ell = 64$ and $d = 13$ as [71]. In this way, we use $[0, 2^{\ell-1})$ to represent $x \in \mathbb{R}^+$, and $[2^{\ell-1}, 2^\ell)$ for negative values.

3 A HIGH-LEVEL OVERVIEW OF METEOR

We first present an overview of our $[\![\cdot]\!]$-sharing semantics in § 3.1. Then, we show the design and threat model of Meteor in § 3.2.

3.1 Overview of $[\![\cdot]\!]$-Sharing Semantics

Costs Analysis of MULT Existing $\langle\cdot\rangle$-sharing based 3PC approaches need ℓ bits per party in 1 round for evaluating 2-MULT with 2 integer inputs (*resharing*), and require more costs for 2-MULT with fixed-point inputs and N-MULT with integer inputs: i) When multiplying 2 fixed-point inputs x and y, the parties need to reveal $[\mathsf{z} + \mathsf{r}']$ and compute $(\mathsf{z} + \mathsf{r}')/2^d - \langle\mathsf{r}\rangle$ for faithful truncation, where $\mathsf{z} = \mathsf{x}\mathsf{y}$, $\mathsf{r} = \mathsf{r}'/2^d$, and $(\mathsf{r}, \mathsf{r}')$ are in secret. As $[\mathsf{z} + \mathsf{r}']$ is of $[\cdot]$-sharing, it needs $\frac{4}{3}\ell$ bits communication per party for mask-and-reveal in the online phase [49, 71]. ii) When multiplying N integers, parties need to perform *resharing* for each 2-MULT. This requires an online communication of $(N-1)\ell$ bits per party in $\lceil \log_2 N \rceil$ rounds [71].

The main costs of MULT stem from **Resharing**-related operations. Inspired by this conclusion, we are wondering: *Will the efficiency (e.g., communication and running-time) be improved if we can maintain the $\langle\cdot\rangle$-sharing format during the whole computation?*

Linearity of $\langle\cdot\rangle$-Sharing From § 2.3, we notice the linear operations of $\langle\cdot\rangle$-values lead to $\langle\cdot\rangle$-shared results locally (no communication). This is true for two $\langle\cdot\rangle$-shared inputs ($c_1\langle \mathsf{x}\rangle + c_2\langle \mathsf{y}\rangle + c_3$), and can be easily generalized to three or more $\langle\cdot\rangle$-shared inputs.

$[\![\cdot]\!]$-Sharing Inspired by the sharing semantics of [9, 10, 67] and with the *linearity* of $\langle\cdot\rangle$-sharing in mind, we propose an improved 3PC secret sharing ($[\![\cdot]\!]$-sharing) as follows:

Definition 1 ($[\![\cdot]\!]$-sharing). *A value $\mathsf{x} \in \mathbb{Z}_{2^\ell}$ is said to be $[\![\cdot]\!]$-shared among $\{P_0, P_1, P_2\}$ if there exists random ψ_x and m_x such that: i) ψ_x is $\langle\cdot\rangle$-shared among $\{P_0, P_1, P_2\}$; ii) $\mathsf{m}_\mathsf{x} = \mathsf{x} - \psi_\mathsf{x}$ is known to all parties in clear. The share of P_i is $[\![\mathsf{x}]\!]_i = (\mathsf{m}_\mathsf{x}, \langle\psi_\mathsf{x}\rangle_i)$ for $i \in \{0, 1, 2\}$.*

For brevity, we use notations $\psi_{\mathsf{x}_1 \ldots \mathsf{x}_n} = \psi_{\mathsf{x}_1}\psi_{\mathsf{x}_2}\ldots\psi_{\mathsf{x}_n}$ and $\mathsf{m}_{\mathsf{x}_1\ldots\mathsf{x}_n} = \mathsf{m}_{\mathsf{x}_1}\mathsf{m}_{\mathsf{x}_2}\ldots\mathsf{m}_{\mathsf{x}_n}$. Similarly, $[\![\cdot]\!]^q$-sharing is for $\mathsf{x} \in \mathbb{F}_q$ and $[\![\cdot]\!]^2$-sharing is for $\mathsf{x} \in \mathbb{Z}_2$, where we use modulo q in $[\![\cdot]\!]^q$-sharing and replace $+, -$ by $\oplus$ and $\cdot$ by $\wedge$ in $[\![\cdot]\!]^2$-sharing.

With $[\![\cdot]\!]$-sharing, we can evaluate MULT by computing the relatively expensive multiplication of secret random $\langle\psi\rangle$s in the setup phase, such that *the online phase only involves the linear operations of $\langle\cdot\rangle$-sharing*. Taking 2-MULT(x, y) with integer inputs as an example, the parties compute $\langle\psi_\mathsf{z}\rangle = \langle\psi_\mathsf{x}\rangle\langle\psi_\mathsf{y}\rangle$ in the setup phase, and compute m_z with linear operations and 1 round of revealing. $[\![\cdot]\!]$-sharing also needs ℓ bits per party, but gives the following benefits:

- For 2-MULT with fixed-point inputs, we only need ℓ bits per party in 1 round for mask-and-reveal in the online phase. This is because the intermediate results are in $\langle\cdot\rangle$-shared fashion. Hence, we improve the communication by 1.3×;
- For N-MULT with integer inputs, we only need linear operations of $\langle\cdot\rangle$-sharing to generate $\langle\cdot\rangle$-shared product of N integers in the online phase, since all multiplications among $\langle\psi\rangle$s can be evaluated in the setup phase. Therefore, our approach needs an online communication of ℓ bits per party in 1 round, which is independent of N and first achieved in the regime of 3PC. Compared to prior methods, we improve the online communication by $N\times$ and rounds by $\lceil\log_2 N\rceil\times$.

What's more, we propose efficient $[\![\cdot]\!]$-sharing based protocols for other primitives and NN operators in respective § 4 and § 6.

Comparison to ABY2.0 [54]. Patra *et al.* has proposed similar sharing semantics to improve the online efficiency of 2PC [54] inspired by ASTRA [18] and [53], but our $[\![\cdot]\!]$-sharing is different in the following aspects:

- **Beaver-Friendly v.s. Linearity:** ABY2.0 is inspired by Beaver triples [7] and reduces the communication by sharing the inputs in a Beaver-friendly format. However, $[\![\cdot]\!]$-sharing stems from the *linearity* of replicated secret sharing. They might be equivalent in some settings (*e.g.*, 2PC), but our *linearity* perspective is more straightforward to be generalized to other linear secret sharing.
- **2PC v.s. 3PC:** For the setup phase, ABY2.0 exploits Oblivious Transfer (OT) [32] or HE [25] to generate correlated randomness, but we utilize the multiplication protocol of $\langle\cdot\rangle$-sharing (free of OT or HE). Therefore, $[\![\cdot]\!]$-sharing is more efficient in setup when *honest-majority in 3PC* is available.

3.2 Design of Meteor

Our Meteor, as depicted in Figure 1, consists of three layers:

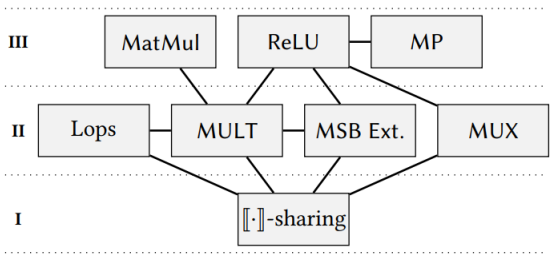


Figure 1: Dependency of protocols in Meteor.

- **I:** We first propose an improved 3-party secret sharing scheme ($[\![\cdot]\!]$-sharing) inspired by the *linearity* of replicated secret sharing in 3PC.
- **II:** Secondly, we design efficient protocols for the most common basic cryptographic primitives, *i.e.*, Linear Operations (Lops), MULT, MSB Ext., and MUX.
- **III:** Thirdly, we build secure blocks for the widely used NN operators, such as MatMul, ReLU, and MP, together with specific optimizations to support fast secure NN inference.

Function-dependent* but *Input-independent Setup Following [70, 71], we also focus on the online efficiency. Meteor is cast into a *function-dependent* but *input-independent* setup phase, and an *input-dependent* online phase as [19, 54, 55]. In the setup phase, we generate the *function-dependent* but *input-independent* correlated randomness for a given function to improve the online efficiency. This setup is available and widely utilized in many applications.

Threat Model Following works [70, 71], Meteor resists semi-honest adversaries in honest-majority [45]. Namely, each party follows the protocol, but may *individually* try to learn information about other inputs:

Definition 2 (Semi-Honest Security). *Let Π be a three-party protocol running in real-world and $\mathcal{F} : (\{0,1\}^n)^3 \to (\{0,1\}^m)^3$ be the ideal randomized functionality. We say Π securely computes $\mathcal{F}$ in presence of a single semi-honest adversary if for every corrupted party P_i ($i \in \{0,1,2\}$) and every input $\mathbf{x} \in (\{0,1\}^n)^3$, there exists an efficient simulator $\mathcal{S}$ such that:*

$$\{\mathsf{view}_{i,\Pi}(\mathbf{x}), \mathsf{output}_\Pi(\mathbf{x})\} \overset{c}{\approx} \{\mathcal{S}(I, \mathsf{x}_i, \mathcal{F}_i(\mathbf{x})), \mathcal{F}(\mathbf{x})\},$$

where $\mathsf{view}_{i,\Pi}(\mathbf{x})$ is the view of P_i in the execution of Π on $\mathbf{x}$, $\mathsf{output}_\Pi(\mathbf{x})$ is the output of all parties, and $\mathcal{F}_i(\mathbf{x})$ denotes the ith output of $\mathcal{F}(\mathbf{x})$.

4 IMPROVED SECURE 3-PARTY COMPUTATION

In this section, we present the detailed constructions of sharing and reconstruction (§ 4.1), linear operations (§ 4.2), MULT(§ 4.3), MSB Ext.(§ 4.4), and MUX (§ 4.5).

4.1 Sharing and Reconstruction

Sharing $\Pi_{\mathsf{SHARE}}(\mathsf{x})$ achieves $\mathcal{F}^{[\![\cdot]\!]}_{\mathsf{SHARE}}$ by enabling P_i (secret owner) to generate a $[\![\cdot]\!]$-sharing of its x. In the setup phase, all parties together sample random $\langle\psi_\mathsf{x}\rangle$ using existing $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{RAND}}$ with P_i gets ψ_x in clear (c.f. Appendix A). In the online phase, P_i reveals $\mathsf{m}_\mathsf{x} = \mathsf{x} - \psi_\mathsf{x}$.

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared $[\![\mathsf{x}]\!]$ and $[\![\mathsf{y}]\!]$.
Output: $[\![\mathsf{z}]\!] = [\![\mathsf{x}]\!] \cdot [\![\mathsf{y}]\!]$.
- **Setup:**
  (1) Parties generate $\langle\psi_\mathsf{z}\rangle$ using functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
  (2) Parties execute $\langle\psi_\mathsf{xy}\rangle = \mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}(\langle\psi_\mathsf{x}\rangle, \langle\psi_\mathsf{y}\rangle)$.
- **Online:**
  (1) P_i locally computes $\langle \mathsf{m_z}\rangle_i = \mathsf{m_{xy}} + \mathsf{m_x}\langle\psi_\mathsf{y}\rangle_i + \mathsf{m_y}\langle\psi_\mathsf{x}\rangle_i + \langle\psi_\mathsf{xy}\rangle_i - \langle\psi_\mathsf{z}\rangle_i$.
  (2) Parties exchange the shares of $\langle \mathsf{m_z}\rangle$ to reconstruct $\mathsf{m_z}$.
  (3) P_i outputs $[\![\mathsf{z}]\!]_i = (\mathsf{m_z}, \langle\psi_\mathsf{z}\rangle_i)$.

Figure 2: 2-Input Multiplication Protocol $\Pi_{\text{2-MULT}}$.

Reconstruction We describe our protocol $\Pi_{\mathsf{REC}}([\![\mathsf{x}]\!])$ for $\mathcal{F}_{\mathsf{REC}}^{[\![\cdot]\!]}$ that reconstructs x as follows: Given $[\![\mathsf{x}]\!]$, parties invoke $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$ to reconstruct ψ_x and locally compute $\mathsf{x} = \mathsf{m_x} + \psi_\mathsf{x}$.

4.2 Linear Operations

$[\![\cdot]\!]$-sharing is linear in the sense that given $[\![\mathsf{x}]\!]$, $[\![\mathsf{y}]\!]$ and public constants c_1, c_2, and c_3, parties can compute $[\![\mathsf{z}]\!] = c_1\cdot[\![\mathsf{x}]\!]+c_2\cdot[\![\mathsf{y}]\!]+c_3$ by locally setting $(\mathsf{m_z} = c_1\cdot\mathsf{m_x}+c_2\cdot\mathsf{m_y}+c_3, \langle\psi_\mathsf{z}\rangle = c_1\cdot\langle\psi_\mathsf{x}\rangle+c_2\cdot\langle\psi_\mathsf{y}\rangle)$.

4.3 Multiplication

We consider 2-input multiplication (2-MULT) and N-input multiplication (N-MULT). The former is employed in secure FC/CONV, while the latter plays an important role in secure MSB Ext.

4.3.1 2-Input Multiplication. We first consider the multiplication of two integers. Given the $[\![\cdot]\!]$-shares of integers x and y, functionality $\mathcal{F}_{\text{2-MULT}}^{[\![\cdot]\!]}$ generates $[\![\mathsf{z}]\!]$ with $\mathsf{z} = \mathsf{xy}$. For z, we will need:

$$\begin{aligned}\mathsf{m_z} &= \mathsf{z} - \psi_\mathsf{z} = \mathsf{xy} - \psi_\mathsf{z} \\ &= (\mathsf{m_x} + \psi_\mathsf{x})(\mathsf{m_y} + \psi_\mathsf{y}) - \psi_\mathsf{z} \\ &= \mathsf{m_x m_y} + \mathsf{m_x}\psi_\mathsf{y} + \mathsf{m_y}\psi_\mathsf{x} + \psi_\mathsf{xy} - \psi_\mathsf{z}\end{aligned} \tag{1}$$

In the setup phase, parties compute the input-independent $\langle\psi_\mathsf{xy}\rangle = \langle\psi_\mathsf{x}\rangle\langle\psi_\mathsf{y}\rangle$. And in the online phase, parties compute $\langle \mathsf{m_z}\rangle$ locally and collaboratively reveal it. So the challenge is reduced to generate $\langle\psi_\mathsf{xy}\rangle$ given $\langle\psi_\mathsf{x}\rangle$ and $\langle\psi_\mathsf{y}\rangle$. We leverage $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ to accomplish this task as § 2.3. The protocol is in Figure 2. $\Pi_{\text{2-MULT}}$ needs an online communication of ℓ bits per party in 1 round.

Fixed-Point Multiplication Extension As analyzed in § 3, we truncate the product (*i.e.*, $\mathsf{xy}/2^d$ where x and y are in fixed-point) after each multiplication in secure NN inference. Existing faithful truncation method [49, 71] needs $\frac{4}{3}\ell$ bits per party in online phase.

To reduce the costs, we propose online free faithful truncation at the same online communication as 2-MULT for integers: i) In the setup phase, parties generate $(\langle\psi_\mathsf{z}\rangle, \langle\psi_\mathsf{z}'\rangle)$ with $\psi_\mathsf{z} = \psi_\mathsf{z}'/2^d$ using the optimized binary circuits [49]. ii) In the online phase, parties compute and reveal $\langle \mathsf{m_z}'\rangle = \mathsf{xy} - \langle\psi_\mathsf{z}'\rangle$ as equation (1) and reveal $\mathsf{m_z}'$. iii) Parties set $[\![\mathsf{z}]\!] = (\mathsf{m_z}'/2^d, \langle\psi_\mathsf{z}\rangle)$ with $\mathsf{m_z}'/2^d + \psi_\mathsf{z} = \mathsf{xy}/2^d$ holds.

Online Communication The correctness and precision guarantees of our method are similar to prior works [49, 71], our main contributions here lie in the online communication improvements. As $\langle \mathsf{m_z}'\rangle$ is in $\langle\cdot\rangle$-sharing, our $\Pi_{\text{2-MULT}}$ with truncation for fixed-point inputs needs ℓ bits per party in 1 round for revealing it during the online phase, achieving 1.3× improvements.

Input: P_0, P_1, and P_2 hold $[\![\mathsf{x}]\!] = (\mathsf{m_x}, \langle\psi_\mathsf{x}\rangle)$.
Output: $[\![\mathsf{msb(x)}]\!]^2$.
- **Setup:**
  (1) The parties call functionality $\mathcal{F}_{\mathsf{PreMSB}}^{[\![\cdot]\!]}$ to generate $[\![\mathsf{msb}(\psi_\mathsf{x})]\!]^2$, $\{[\![\mathsf{s}[i]]\!]^q\}_{i=1}^{\ell}$ with $\mathsf{s} = 2\psi_\mathsf{x}$, $([\![\lambda]\!]^2, [\![\lambda]\!]^q)$ with $\lambda \in_R \mathbb{Z}_2$, and $[\![\zeta]\!]^q$ with $\zeta \in_R \mathbb{F}_q^*$.
  (2) The parties run the setup phase of $\mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]}$.
- **Online:**
  (1) Compute $\mathsf{b} = L - 2\mathsf{m_x}$.
  (2) **for** $i \in [\ell, \ell-1, \ldots, 1]$, all parties compute in $\mathbb{F}_q$:
  (3) $[\![\mathsf{u}[i]]\!]^q = (1 - 2[\![\lambda]\!]^q)\cdot([\![\mathsf{s}[i]]\!]^q - \mathsf{b}[i])$.
  (4) $[\![\mathsf{w}[i]]\!]^q = [\![\mathsf{s}[i]]\!]^q + \mathsf{b}[i] - 2[\![\mathsf{s}[i]]\!]^q\mathsf{b}[i]$.
  (5) $[\![\mathsf{e}[i]]\!]^q = [\![\mathsf{u}[i]]\!]^q + 1 + \sum_{k=i+1}^{\ell}[\![\mathsf{w}[k]]\!]^q$.
  (6) **end for**
  (7) Compute $[\![\mathsf{d}]\!]^q = [\![\zeta]\!]^q \cdot \Pi_{i=1}^{\ell}[\![\mathsf{e}[i]]\!]^q \pmod q$ using $\mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]}$, and reveal d.
  (8) Set $[\![\mathsf{c}]\!]^2 = \lambda' \oplus [\![\lambda]\!]^2$, where $\lambda' = (\mathsf{d} \neq 0)$.
  (9) Output $[\![\mathsf{msb(x)}]\!]^2 = \mathsf{msb}(\mathsf{m_x}) \oplus [\![\mathsf{msb}(\psi_\mathsf{x})]\!]^2 \oplus [\![c]\!]^2$.

Figure 3: Secure MSB Extraction Protocol Π_{SecMSB}.

4.3.2 N-Input Multiplication. Functionality $\mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]}$ multiplies N integers for any positive constant N. From the fact that secret random $\langle\psi\rangle$s are input-independent, we can multiply them in the setup phase. Therefore, we only need linear operations of $\langle\cdot\rangle$-shared values to get the $\langle\cdot\rangle$-shared product of N integers in the online phase. Taking $(\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_N)$ as inputs, we have $\mathsf{m_z} = \Pi_{t=1}^{N}\mathsf{x}_t - \psi_\mathsf{z} = \Pi_{t=1}^{N}(\mathsf{m}_{\mathsf{x}_t} + \psi_{\mathsf{x}_t}) - \psi_\mathsf{z} = \sum_{\mathcal{T}\subseteq\{1,\ldots,N\}}(\Pi_{j\notin\mathcal{T}}\mathsf{m}_{\mathsf{x}_j}\cdot\Pi_{k\in\mathcal{T}}\psi_{\mathsf{x}_k}) - \psi_\mathsf{z}$.

In the setup phase, parties can compute the input-independent $\langle\cdot\rangle$-shares of $\{\Pi_{k\in\mathcal{T}}\psi_{\mathsf{x}_k}\}_{\mathcal{T}\subseteq\{1,\ldots,N\}}$ exploiting $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$. In the online phase, the parties only need to reveal $\mathsf{m_z}$. The details are in Figure 5.

Online Communication The online communication remains just ℓ bits per party in 1 round independent of the fan-in. In contrast, previous $\langle\cdot\rangle$-sharing based methods require $(N-1)\ell$ bits per party in $\lceil\log_2 N\rceil$ rounds. In the setup phase, the above method requires $(2^N - 1 - N)\ell$ bits per party in $\lceil\log_2 N\rceil$ rounds. To balance the burden in the setup and online phases, we set $N = 3$ and 4 as [54].

4.4 Secure MSB Extraction

Given $[\![\mathsf{x}]\!]$, functionality $\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}$ extracts $[\![\mathsf{msb(x)}]\!]^2$ securely. From $\mathsf{x} = \mathsf{m_x} + \psi_\mathsf{x}$ for $[\![\mathsf{x}]\!]$, we can write

$$\mathsf{msb(x)} = \mathsf{msb}(\mathsf{m_x}) \oplus \mathsf{msb}(\psi_\mathsf{x}) \oplus \mathsf{c}, \tag{2}$$

where c is the carry bit of $\mathsf{m_x}$ and ψ_x modulo $\frac{L}{2}$ (ignoring their msb), which is formalized as $\mathsf{c} = (2\mathsf{m_x} + 2\psi_\mathsf{x} \geq L) = (2\psi_\mathsf{x} \geq L - 2\mathsf{m_x})$. Let $\mathsf{s} = 2\psi_\mathsf{x}$ and $\mathsf{b} = L - 2\mathsf{m_x}$, our key insights are as follows:

i) ψ_x is independent of inputs, we can compute $[\![\mathsf{msb}(\psi_\mathsf{x})]\!]^2$ and $[\![\cdot]\!]^q$-shares of bits of s in the setup phase.
ii) m_x and L are public in the online phase.

With our key insights in mind, we propose protocol Π_{SecMSB} as Figure 3. In the setup phase, we manage to generate $[\![\mathsf{msb}(\psi_\mathsf{x})]\!]^2$, $\{[\![\mathsf{s}[i]]\!]^q\}_{i=1}^{\ell}$, $[\![\lambda]\!]^2$, $[\![\lambda]\!]^q$, and $[\![\zeta]\!]^q$ using $\mathcal{F}_{\mathsf{PreMSB}}^{[\![\cdot]\!]}$. We construct protocol Π_{PreMSB} (c.f., Appendix § C) for $\mathcal{F}_{\mathsf{PreMSB}}^{[\![\cdot]\!]}$ based on [49, 71].

In the online phase, the challenge is computing c. Inspired by [71], we propose an optimized method as Figure 3. The key point is

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared $[\![\mathsf{x}]\!]$, $[\![\mathsf{y}]\!]$, and $[\![\mathsf{v}]\!]^2$.
Output: $[\![\mathsf{z}]\!]$ with $\mathsf{z} = \mathsf{x}$ if $\mathsf{v} = 1$ and $\mathsf{z} = \mathsf{y}$ otherwise.
• **Setup:**
(1) Parties generate $\langle\psi\rangle$ using functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
(2) Parties convert $\langle\psi_{\mathsf{v}}^{\ell}\rangle = \mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}(\langle\psi_{\mathsf{v}}\rangle^2)$ for $[\![\mathsf{v}]\!]^2$.
(3) Parties compute $\langle\psi_{\mathsf{u}}\rangle = \langle\psi_{\mathsf{x}}\rangle - \langle\psi_{\mathsf{y}}\rangle$.
(4) Parties invoke $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ to compute $\langle\psi_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell}\rangle = \langle\psi_{\mathsf{u}}\rangle \cdot \langle\psi_{\mathsf{v}}^{\ell}\rangle$.
• **Online:**
(1) P_i locally computes $\mathsf{m}_{\mathsf{u}} = \mathsf{m}_{\mathsf{x}} - \mathsf{m}_{\mathsf{y}}$.
(2) P_i locally computes $\langle\mathsf{m}_{\mathsf{uv}^{\ell}}\rangle_i = (1 - 2\langle\psi_{\mathsf{v}}^{\ell}\rangle_i)\mathsf{m}_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell} + \mathsf{m}_{\mathsf{u}}\langle\psi_{\mathsf{v}}^{\ell}\rangle_i + \langle\psi_{\mathsf{u}}\rangle_i\mathsf{m}_{\mathsf{v}}^{\ell} + (1 - 2\mathsf{m}_{\mathsf{v}}^{\ell})\langle\psi_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell}\rangle_i - \langle\psi\rangle_i$.
(3) Parties reconstruct $\mathsf{m}_{\mathsf{uv}^{\ell}}$ and set $[\![\mathsf{u}\cdot\mathsf{v}^{\ell}]\!] = (\mathsf{m}_{\mathsf{uv}^{\ell}}, \langle\psi\rangle)$.
(4) Parties locally compute and output $[\![\mathsf{z}]\!] = [\![\mathsf{u}\cdot\mathsf{v}^{\ell}]\!] + [\![\mathsf{y}]\!]$.

Figure 4: Secure Multiplexer Protocol Π_{MUX}.

that $\mathsf{s} < \mathsf{b}$ if and only if there $\exists i \in [\ell, \ldots, 1]$ subjected to $\mathsf{e}[i] = 0$, which means $\mathsf{s}[k] = \mathsf{b}[k]$ for $\forall k > i$ and $(\mathsf{s}[i] = 0, \mathsf{b}[i] = 1)$. Otherwise, we have $s \geq \mathsf{b} \Leftrightarrow \mathsf{e}[i] \neq 0$ for $\forall i \in [\ell, \ldots, 1]$.

Online Communication We use $\Pi_{N\text{-MULT}}$ with $N = 3, 4$. For the online phase: i) Steps 2-6 need $\ell\lceil\log_2 q\rceil$ bits per party in 1 round. ii) Step 7 needs $\approx (\frac{1}{3}\ell + 1)\lceil\log_2 q\rceil$ bits per party in $\lceil\log_4(\ell+1)\rceil + 1$ rounds. iii) Steps 1, 8, 9 are locally. Therefore, the online phase needs $\approx (\frac{4}{3}\ell + 1)\lceil\log_2 q\rceil$ bits per party in $\lceil\log_4(\ell+1)\rceil + 2$ rounds.

4.5 Multiplexer

Given $([\![\mathsf{x}]\!], [\![\mathsf{y}]\!], [\![\mathsf{v}]\!]^2)$, functionality $\mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]}$ outputs $[\![\mathsf{z}]\!] = [\![\mathsf{x}]\!]$ if $\mathsf{v} = 1$, and $[\![\mathsf{z}]\!] = [\![y]\!]$ otherwise. This is $[\![\mathsf{z}]\!] = ([\![\mathsf{x}]\!] - [\![\mathsf{y}]\!]) \cdot [\![\mathsf{v}]\!]^2 + [\![\mathsf{y}]\!]$.

Let $[\![\mathsf{u}]\!] = [\![\mathsf{x}]\!] - [\![\mathsf{y}]\!]$, the challenge is computing $[\![\mathsf{u}]\!] \cdot [\![\mathsf{v}]\!]^2$. A trivial solution is converting $[\![\mathsf{v}]\!]^2$ to $[\![\mathsf{v}]\!]$ by [28, 61] and computing $[\![\mathsf{u}]\!] \cdot [\![\mathsf{v}]\!]$ with $1 + \ell$ bits per party in 2 rounds. To reduce costs, we propose protocol Π_{MUX}. Denote the value of bit v in $\mathbb{Z}_{2^{\ell}}$ as v^{ℓ}. For $[\![\mathsf{v}]\!]^2 = (\mathsf{m}_{\mathsf{v}}, \langle\psi_{\mathsf{v}}\rangle^2)$, $\mathsf{v}^{\ell} = (\mathsf{m}_{\mathsf{v}} \oplus \psi_{\mathsf{v}})^{\ell} = \mathsf{m}_{\mathsf{v}}^{\ell} + \psi_{\mathsf{v}}^{\ell} - 2\mathsf{m}_{\mathsf{v}}^{\ell}\psi_{\mathsf{v}}^{\ell}$. Thus, for $[\![\mathsf{u}]\!] \cdot [\![\mathsf{v}]\!]^2$ we have

$$\begin{aligned}\mathsf{u}\cdot\mathsf{v}^{\ell} &= (\mathsf{m}_{\mathsf{u}} + \psi_{\mathsf{u}}) \cdot (\mathsf{m}_{\mathsf{v}}^{\ell} + \psi_{\mathsf{v}}^{\ell} - 2\mathsf{m}_{\mathsf{v}}^{\ell}\psi_{\mathsf{v}}^{\ell}) \\ &= \mathsf{m}_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell} + \mathsf{m}_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell} - 2\mathsf{m}_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell}\psi_{\mathsf{v}}^{\ell} + \psi_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell} + \psi_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell} - 2\psi_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell}\psi_{\mathsf{v}}^{\ell}. \\ &= (1 - 2\psi_{\mathsf{v}}^{\ell})\mathsf{m}_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell} + \mathsf{m}_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell} + \psi_{\mathsf{u}}\mathsf{m}_{\mathsf{v}}^{\ell} + (1 - 2\mathsf{m}_{\mathsf{v}}^{\ell})\psi_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell}\end{aligned} \tag{3}$$

In the setup phase, we compute $\langle\psi_{\mathsf{v}}^{\ell}\rangle$ from $\langle\psi_{\mathsf{v}}\rangle^2$ via $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ (c.f., Appendix § D) and compute $\langle\psi_{\mathsf{u}}\psi_{\mathsf{v}}^{\ell}\rangle$ using $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$. In online phase, parties compute and reveal $\mathsf{m}_{\mathsf{uv}^{\ell}}$, set $[\![\mathsf{u}\cdot\mathsf{v}^{\ell}]\!] = (\mathsf{m}_{\mathsf{uv}^{\ell}}, \langle\psi\rangle)$, and output $[\![\mathsf{z}]\!] = [\![\mathsf{u}\cdot\mathsf{v}^{\ell}]\!] + [\![\mathsf{y}]\!]$ as shown in Figure 4.

Online Communication Π_{MUX} needs ℓ bits per party in 1 round.

5 SECURITY ANALYSIS

Theorem 1 captures the security of our protocols, and the full proof is given in Appendix G.

Theorem 1. *In the hybrid model, our protocols securely realize the functionalities $\mathcal{F}_{\mathsf{SHARE}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\mathsf{REC}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\mathsf{Lops}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\text{2-MULT}}^{[\![\cdot]\!]}$, $\mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}$ and $\mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]}$ against a semi-honest adversary $\mathcal{A}$, who corrupts no more than one party.*

6 OPTIMIZED SECURE NN OPERATORS

In § 6.1, we show the secure FC, CONV, BatchNorm. In § 6.2, we construct secure ReLU. We give private MP and ReLU-MP equivalent switching in § 6.3. The full protocols are shown in Appendix E.

6.1 Secure Matrix Multiplication

Protocol $\Pi_{\text{2-MULT}}$ can be easily vectorized to MatMul . Given $[\![\mathbf{X}]\!] = (\mathsf{m}_{\mathbf{X}}, \langle\psi_{\mathbf{X}}\rangle)$ with dimension $m\times n$ and $[\![\mathbf{Y}]\!] = (\mathsf{m}_{\mathbf{Y}}, \langle\psi_{\mathbf{Y}}\rangle)$ with dimension $n \times o$: i) In the setup phase, parties execute $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ to compute $\langle\psi_{\mathbf{XY}}\rangle = \langle\psi_{\mathbf{X}}\rangle \cdot \langle\psi_{\mathbf{Y}}\rangle$. ii) In the online phase, parties locally compute $\langle\mathsf{m}_{\mathbf{Z}}\rangle = \mathsf{m}_{\mathbf{X}} \cdot \mathsf{m}_{\mathbf{Y}} + \mathsf{m}_{\mathbf{X}} \cdot \langle\psi_{\mathbf{Y}}\rangle + \langle\psi_{\mathbf{X}}\rangle \cdot \mathsf{m}_{\mathbf{Y}} + \langle\psi_{\mathbf{XY}}\rangle - \langle\psi_{\mathbf{Z}}\rangle$, reconstruct $\mathsf{m}_{\mathbf{Z}}$, and set $[\![\mathbf{Z}]\!] = (\mathsf{m}_{\mathbf{Z}}, \langle\psi_{\mathbf{Z}}\rangle)$. We need $mo\ell$ bits (independent of n) in 1 round for the online phase. The full protocol is shown in Figure 8. The security of MatMul follows in $\mathcal{F}_{\text{2-MULT}}^{[\![\cdot]\!]}$-hybrid model.

Secure FC **&** CONV We can leverage MatMul to achieve secure FC. For CONV, we reshape the input and filter to express convolution as MatMul for subsequent secure evaluation. To support fixed-point truncation, the parties can generate $(\langle\psi_{\mathbf{Z}}\rangle, \langle\psi'_{\mathbf{Z}}\rangle)$, and perform faithful truncation similarly as § 4.3.1 but in vectorization.

Fusing CONV & BatchNorm BatchNorm often goes after CONV, we can fuse them into one for better efficiency [4]. Suppose the trained parameters for BatchNorm and CONV are $(\mathbf{W}_{\mathsf{BN}}, \mathbf{b}_{\mathsf{BN}})$ and $(\mathbf{W}_{\mathsf{CONV}}, \mathbf{b}_{\mathsf{CONV}})$, we can replace them by a single CONV with $(\mathbf{W} = \mathbf{W}_{\mathsf{BN}} \cdot \mathbf{W}_{\mathsf{CONV}}, \mathbf{b} = \mathbf{W}_{\mathsf{BN}} \cdot \mathbf{b}_{\mathsf{CONV}} + \mathbf{b}_{\mathsf{BN}})$. Therefore, we can compute both layers together at the same costs as secure CONV.

6.2 Secure ReLU

The activation function considered in this work is the rectified linear unit (ReLU). Taking x as input, $\mathcal{F}_{\mathsf{ReLU}}^{[\![\cdot]\!]}$ returns x if $\mathsf{x} \geq 0$, and 0 otherwise. To achieve $\mathcal{F}_{\mathsf{ReLU}}^{[\![\cdot]\!]}$ securely, it suffices to first extract $[\![\mathsf{msb}(\mathsf{x})]\!]^2$ using $\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}$, and then execute $\mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]}([\![\mathsf{x}]\!], [\![\mathsf{y}]\!], [\![\mathsf{msb}(\mathsf{x})]\!]^2 \oplus 1)$ with $\mathsf{y} = 0$. The details are shown in Figure 9.

Our method needs an online communication of $(\frac{4}{3}\ell+1)\lceil\log_2 q\rceil + \ell$ bits per party in $\lceil\log_4(\ell+1)\rceil + 3$ rounds. And the security of Π_{ReLU} is easily to see in $(\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}, \mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]})$-hybrid model.

6.3 Secure Maxpool

Given $[\![\cdot]\!]$-shared vector $\mathbf{x} = (\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n)$ of size-n, the goal of functionality $\mathcal{F}_{\mathsf{MP}}^{[\![\cdot]\!]}$ is to compute the maximum value among the n elements. $\mathcal{F}_{\mathsf{MP}}^{[\![\cdot]\!]}$ can be implemented on top of ReLU. The key point is that the parties update $[\![\mathsf{max}]\!] = [\![\mathsf{x}_i]\!]$ if and only if $\mathsf{ReLU}([\![\mathsf{max}]\!] - [\![\mathsf{x}_i]\!]) = 0$ ($\Leftrightarrow \mathsf{max} < \mathsf{x}_i$). The full protocol is shown in Figure 10.

Furthermore, benefiting from binary sort on the inputs and small amounts of bookkeeping [71], the online phase needs approximately $(n-1)((\frac{4}{3}\ell+1)\lceil\log_2 q\rceil+\ell)$ bits per party in $\lceil\log_2 n\rceil(\lceil\log_4(\ell+1)\rceil+3)$ rounds, and the security follows in the $\mathcal{F}_{\mathsf{ReLU}}^{[\![\cdot]\!]}$-hybrid model.

6.3.1 ReLU-MP *Equivalent Switching.* MP is usually applied after ReLU, but they are commutative operators in NN inference: $\mathsf{ReLU}(\mathsf{MP}(\cdot)) = \mathsf{MP}(\mathsf{ReLU}(\cdot))$.

There is no significant performance difference of the alternation in cleartext, but $\mathsf{ReLU}(\mathsf{MP}(\cdot))$ is much more efficient than $\mathsf{MP}(\mathsf{ReLU}(\cdot))$ in MPC since the former reduces the number of ReLU operations significantly [42]. We thus evaluate MP before ReLU.

Table 3: Online costs of NN operators of SecureNN, FALCON, and METEOR. For MatMul, the inputs are of size $m \times n$ and $n \times o$. And CONV is with input $m \times m$, c input channels, o output channels, and filter of $f \times f$. ReLU is computed in element-wise with size n. And MP is with $m \times m$ inputs, c input channels, and $f \times f$ window. Communication is in MB and Running-Time is in seconds.

Operator	Size	Comm.			Time (LAN)			Time (WAN)		
		SecureNN	FALCON	METEOR	SecureNN	FALCON	METEOR	SecureNN	FALCON	METEOR
$\text{MatMul}_{m,n,o}$	(784, 128, 10)	0.563	0.084	**0.063**	0.065	0.003	**0.003**	0.146	0.103	**0.073**
	(128, 500, 100)	0.642	0.137	**0.102**	0.258	**0.007**	0.012	0.346	0.118	**0.076**
$\text{CONV}_{m,c,o,f}$	(28, 1, 20, 5)	0.110	0.123	**0.092**	0.019	0.005	**0.004**	0.122	0.103	**0.099**
	(8, 16, 50, 5)	0.143	0.008	**0.006**	0.019	0.002	**0.002**	0.091	0.102	**0.066**
ReLU_n	128×128	3.845	2.376	**1.556**	0.205	**0.035**	0.040	1.076	0.623	**0.433**
	576×20	2.703	1.670	**1.094**	0.148	**0.025**	0.030	0.815	0.566	**0.457**
$\text{MP}_{m,c,f}$	(24, 20, 2)	2.143	1.287	**0.821**	0.123	0.038	**0.031**	1.412	1.355	**0.929**
	(8, 50, 4)	0.744	0.483	**0.285**	0.139	0.118	**0.079**	5.523	6.409	**4.323**

7 EVALUATIONS

We present the prototype and detailed experimental results.

Experimental Details We implement METEOR on top of FALCON in C++ and run our experiments on Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz with 500GB RAM in both LAN and WAN with a single thread. For LAN, our bandwidth is about 1GB/s and round trip time (rtt) is about 1ms. For WAN, our bandwidth is about 49MB/s and rtt is about 70ms. For fair comparisons, we re-run SecureNN and FALCON in our settings with semi-honest security.

Optimizations Following FALCON [71], we focus on online efficiency and do not take setup costs into accountWe use Eigen library [2] for fast MatMul, uint64 for $\mathbb{Z}_{2^\ell}$, $d = 13$, and $q = 67$. All experiments are executed 10 times, and we record the average[1].

Datasets & Neural Networks We select 2 standard benchmarking datasets: MNIST [43] and CIFAR-10 [41], and 6 standard network architectures: 3 from the secure ML community (Network-A [50], B [60], and C [46]) and 3 from the ML community (LeNet [44], AlexNet [40], and VGG16 [65]) for extensive experiments.

7.1 Online Costs of Micro Benchmarks

We present the online costs of the NN operators, including MatMul, CONV, ReLU, and MP, in Table 3. Our improvements are as follows:

Communication Improvements For the linear operators (*i.e.*, MatMul and CONV), we improve the communication costs by approximately 1.2-6× and 1.3× over SecureNN and FALCON, respectively. This improvement arises from our free truncation technique. For ReLU and MP, we achieve respective 2.5× and 1.6× communication improvements compared with SecureNN and FALCON due to our communication efficient Π_{SecMSB} and Π_{MUX}.

Running-Time Improvements In the LAN setting, we improve the running-time by approximately 2-20× for linear operators and 4× for non-linear operators compared with SecureNN. Meanwhile, we achieve comparable running-time in comparison to FALCON in LAN. In the WAN setting, we are approximately 1.8× and 1.5× faster than SecureNN and FALCON, respectively.

7.2 Online Costs of Single Inference

Evaluation on MNIST We perform experiments on NN with MNIST as SecureNN [70] and FALCON [71], and the results are illustrated in Table 4. Compared to SecureNN, we improve the communication by upto 25.6× (11.1× on average), and the running-time by upto 9.8× (6.8× on average) and 8.1× (5.2× on average) in respective LAN and WAN. Besides, we reduce the communication by 1.5× on average, and are upto 1.5× (1.3× on average) and 2.1× (1.7× on average) faster than FALCON in LAN and WAN.

Evaluation on CIFAR10 As illustrated in Table 5, we evaluate METEOR on NN with CIFAR10 to demonstrate our improvements. For communication, we reduce the costs by 1.6× on average. For inference time, we achieve comparable efficiency in LAN and ≈ 1.5× improvements in WAN over FALCON. Note in this case, our improvements in LAN are not as significant as that in WAN. The reason is that in LAN, the overall time is more restricted by the computation burden since there is enough bandwidth and a small rtt [71]. Therefore, the improvements in communication gain limited running-time benefits when the NN is computationally expensive. However, as MPC protocols are more likely to be executed in WAN, our improvements are meaningful in practical applications.

The improvements mainly stem from two aspects: i) our efficient MatMul improves the online communication of linear layers by 1.3×. ii) More importantly, our online communication- and round-efficient protocols Π_{SecMSB} and Π_{MUX} improve the online efficiency of secure ReLU and MP functions.

7.3 Online Costs of Batch Inference

In this section, we measure the amortized online running-time for batch inference and show our improvements in scalability.

Amortized Running-Time Table 6 shows the running-time of METEOR over a batch of 128 images on AlexNet and VGG16 in LAN and WAN settings. For AlexNet, the amortized time for per-image drops from 0.429s to 0.146s (2.9× improvements) in LAN and from 8.997s to 1.588s (5.7× improvements) in WAN. While for VGG16, we achieve 1.9× and 1.4× time reduction for single image inference by batch processing in respective LAN and WAN. The improvement mainly comes from batch processing amortizes the computation and latency costs for each image.

Scalability Evaluation To present our scalability improvements against FALCON, we further measure the communication and running-time of METEOR on AlexNet and VGG16 with different batchsize as Figure 11 in Appendix F. We have the following findings: i) Given the batchsize $|B|$, we improve the online communication costs by

[1]Communication report in FALCON is incorrect due to implementation bugs and we have checked it with the authors. We re-run FALCON. Please refer to falcon-public.

Table 4: Online costs of single inference for NN-A, -B, -C, and LeNet on MNIST of SecureNN, Falcon, and Meteor. Communication is in MB and Running-Time is in seconds.

	Framework	NN-A		NN-B		NN-C		LeNet	
		Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time
LAN	SecureNN	0.700	0.175	1.352	0.236	2.954	0.416	6.314	0.861
	Falcon	0.041	0.035	0.168	0.031	1.641	0.128	2.415	0.160
	Meteor	**0.027**	**0.024**	**0.111**	**0.024**	**1.066**	**0.104**	**1.568**	**0.143**
WAN	SecureNN	0.700	5.223	1.352	6.734	2.954	9.666	6.314	14.892
	Falcon	0.041	1.544	0.168	1.191	1.641	5.517	2.415	6.759
	Meteor	**0.027**	**1.056**	**0.111**	**0.836**	**1.066**	**3.049**	**1.568**	**3.176**

Table 5: Online costs of single inference on CIFAR10. Communication is in MB and Running-Time is in seconds.

	Framework	AlexNet		VGG16	
		Comm.	Time	Comm.	Time
LAN	Falcon	4.075	0.480	44.844	**3.428**
	Meteor	**2.562**	**0.429**	**29.424**	4.240
WAN	Falcon	4.075	13.522	44.844	38.559
	Meteor	**2.562**	**8.997**	**29.424**	**27.174**

Table 6: Online Running-Time in seconds of Meteor with a batch of $|B| = 1$ and 128 images of CIFAR-10.

	LAN		WAN	
$\|B\|$	1	128	1	128
AlexNet	0.429	18.649	8.997	203.304
VGG16	4.240	282.682	27.174	2404.509

$\approx$ 1.5× in comparison to Falcon, which is consistent with the analysis for single inference in § 7.2. ii) For the online inference running-time, we achieve comparable efficiency in LAN but $\approx$ 1.5× improvements in WAN compared to Falcon, which is not unexpected; after all, the communication improvements have mere gains to running-time in LAN as analyzed in § 7.2. Also, the scalability improvement is primarily due to our proposed efficient protocols.

8 RELATED WORK

Secure NN inference using MPC has gained much attention recently. In the earlier stage, privacy-preserving machine learning mainly focused on linear regression [12, 26, 27, 63], logistic regression [66], decision trees [36], k-means clustering [14, 33], and SVM [69, 74].

In the area of two-party computation (2PC), CryptoNets [29] was one of the earliest works to use homomorphic encryption for secure NN inference, CryptoDL [30] developed approximate and low-degree polynomials to implement non-linear functions for efficiency improvements over CryptoNets. Mohassel *et al.* proposed SecurML [50] in the two-server setting with secret sharing and GC. Meanwhile, Liu *et al.* designed fast matrix multiplications protocols in MiniONN [46]. DeepSecure [62] uses GC to develop a privacy-preserving deep learning prediction framework, and Gazelle [34] combines techniques from HE and MPC to achieve fast private inference. EzPC [17] is a ABY-based [25] framework, and its follow-up works [31, 56–58] focus on improving performance.

In order to solve the performance bottleneck of 2PC, recent works introduce a third party to assist computations. Chameleon [60] used the same technique as in [46] to complete the matrix multiplication operations but employed a semi-honest third-party to generate correlated randomness for multiplication triplets in offline. In order to solve the computational bottleneck incurred by the garbled circuits [6], both SecureNN [70] and Cryptflow [42] constructed novel protocols for non-linear functions such as ReLU and Maxpool that completely avoid the use of GC with the help of a third-party. What's more, schemes based on 3PC replicated secret sharing also provide better overall efficiency [49, 71], Falcon is one of the most efficient methods and can evaluate large NN such as VGG16 and AlexNet. ASTRA [18] proposed a similar sharing scheme, and it communicated 2 elements and only required 2 parties active in the online phase. And ASTRA acted as the basis for several other maliciously-secure frameworks [15, 19, 38, 39, 54, 55]. ScionFL [10] extended ABY2.0 to multi-party, and its Inner-Product protocol needs 2 elements per party in 2 rounds, which is 2× more expensive than ours. And our method is independent of ScionFL. And more works can be found [13, 21, 23, 24, 35].

There are some other works combining quantized NN with MPC [5, 22, 59]. Riazi *et al.* proposed [59], where the weights and activations are in ±1, and they used GC and Oblivious Transfer (OT) to provide constant round private inference. Quotient [5] was proposed to realize the secure computation of ternarized NN, where the weights are in {-1,0,1}. The author converts the ternarized multiplication into two binary multiplications and completes them based on OT. And other functions are all processed by GC. Therefore, prior private binary (ternarized) NN inference schemes suffer from the enormous communication costs introduced by GC, and they are even slower than secret sharing-based approaches for floating-point NN. Recently, some works proposed to utilize hardware, such as GPU, to accelerate the computation of MPC [20, 48, 51, 68, 72]. Specially, GForce [51] is a 2PC inference framework, it proposed stochastic rounding and truncation layers to fuse (de)quantization between non-linear/linear layers for better efficiency, and a suite of GPU-friendly protocols for common operations. CryptGPU [68] embedded cryptographic operations of discrete secret-shared values into floating-point operations to exploit existing CUDA kernels, and proposed several optimizations to softmax.

9 CONCLUSION & FUTURE WORK

In Meteor, we propose an improved 3PC secret sharing scheme from the *linearity* of replicated secret sharing and construct secure blocks for secure NN inference. Extensive evaluations also present our improvements. For future work, we are willing to improve other MSB Ext. methods [28, 47, 49] with our novel secret sharing and the setup communication costs for better efficiency.

ACKNOWLEDGMENTS

This work is supported by The National Key Research and Development Program of China No. 2020YFB1006100 and the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC02040400.

REFERENCES

[1] 1996. *The Health Insurance Portability and Accountability Act of 1996 (HIPAA).* https://www.hhs.gov/hipaa/index.html
[2] 2011. Eigen library. https://eigen.tuxfamily.org/
[3] 2016. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (GDPR).* https://gdpr-info.eu/
[4] 2022. Fusing Convolution and Batch Norm using Custom Function. https://pytorch.org/tutorials/intermediate/custom_function_conv_bn_tutorial.html
[5] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. 2019. QUOTIENT: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1231–1247.
[6] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.
[7] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*. Springer, 420–432.
[8] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1996. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 514–523.
[9] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. 2019. Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. In *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer, 530–549.
[10] Yaniv Ben-Itzhak, Helen Möllering, Benny Pinkas, Thomas Schneider, Ajith Suresh, Oleksandr Tkachenko, Shay Vargaftik, Christian Weinert, Hossein Yalame, and Avishay Yanai. 2022. ScionFL: Secure Quantized Aggregation for Federated Learning. *arXiv preprint arXiv:2210.07376* (2022).
[11] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*. Springer, 192–206.
[12] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine learning classification over encrypted data. In *Symposium on Network and Distributed System Security (NDSS)*.
[13] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. 2022. MOTION–A Framework for Mixed-Protocol Multi-Party Computation. *ACM Transactions on Privacy and Security* 25, 2 (2022), 1–35.
[14] Paul Bunn and Rafail Ostrovsky. 2007. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security*. 486–497.
[15] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 459–480.
[16] Ran Canetti. 1998. Security and Composition of Multi-party Cryptographic Protocols. Cryptology ePrint Archive, Paper 1998/018. https://eprint.iacr.org/1998/018 https://eprint.iacr.org/1998/018.
[17] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2019. EzPC: programmable and efficient secure two-party computation for machine learning. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 496–511.
[18] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. Astra: High throughput 3pc over rings with application to secure prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 81–92.
[19] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2019. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv preprint arXiv:1912.02631* (2019).
[20] Zheng Chen, Feng Zhang, Amelie Chi Zhou, Jidong Zhai, Chenyang Zhang, and Xiaoyong Du. 2020. ParSecureML: An efficient parallel secure machine learning framework on GPUs. In *49th International Conference on Parallel Processing*. 1–11.
[21] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD$\mathbb{Z}_2^k$: efficient MPC mod 2^k for dishonest majority. In *Annual International Cryptology Conference*. Springer, 769–798.
[22] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 355–375.
[23] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic four: Honest-majority four-party secure computation with malicious security. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.
[24] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. 2019. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1102–1120.
[25] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *NDSS*.
[26] Wenliang Du, Mikhail J Atallah, et al. 2001. Privacy-Preserving Cooperative Scientific Computations.. In *csfw*, Vol. 1. Citeseer, 273.
[27] Wenliang Du, Yunghsiang S Han, and Shigang Chen. 2004. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM international conference on data mining*. SIAM, 222–233.
[28] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. 2020. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Annual International Cryptology conference*. Springer, 823–852.
[29] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.
[30] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).
[31] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. Cryptology ePrint Archive, Report 2022/207. https://ia.cr/2022/207.
[32] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*. Springer, 145–161.
[33] Geetha Jagannathan and Rebecca N Wright. 2005. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 593–599.
[34] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1651–1669.
[35] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1575–1590.
[36] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider. 2019. Sok: Modular and efficient private decision tree evaluation. *Proceedings on Privacy Enhancing Technologies* 2019, 2 (2019), 187–208.
[37] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.
[38] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. {SWIFT}: Super-fast and Robust Privacy-Preserving Machine Learning. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.
[39] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2021. Tetrad: Actively Secure 4PC for Secure Training and Inference. *arXiv preprint arXiv:2106.02850* (2021).
[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.
[41] V. Nair Krizhevsky and G. Hinton. 2014. The CIFAR-10 dataset.
[42] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2019. Cryptflow: Secure tensorflow inference. *arXiv preprint arXiv:1909.07814* (2019).
[43] Yann LeCun. 2017. MNIST database. http://yann.lecun.com/exdb/mnist/
[44] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[45] Yehuda Lindell and Benny Pinkas. 2009. A proof of security of Yao's protocol for two-party computation. *Journal of cryptology* 22, 2 (2009), 161–188.
[46] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 619–631.
[47] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. 2021. Rabbit: Efficient Comparison for Secure Multi-Party Computation. In *International Conference on Financial Cryptography and Data Security*. Springer, 249–270.
[48] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2505–2522.
[49] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on*

Computer and Communications Security. 35–52.
[50] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.
[51] Lucien KL Ng and Sherman SM Chow. 2021. GForce:GPU-Friendly Oblivious and Rapid Neural Network Inference. In *30th USENIX Security Symposium (USENIX Security 21)*. 2147–2164.
[52] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378* (2018).
[53] Satsuya Ohata and Koji Nuida. 2020. Communication-efficient (client-aided) secure two-party protocols and its application. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*. Springer, 369–385.
[54] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. {ABY2.0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation. In *30th USENIX Security Symposium (USENIX Security 21)*. 2165–2182.
[55] Arpita Patra and Ajith Suresh. 2020. BLAZE: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042* (2020).
[56] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. 2022. SECFLOAT: Accurate Floating-Point meets Secure 2-Party Computation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 576–595.
[57] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. SIRNN: A Math Library for Secure RNN Inference. *arXiv preprint arXiv:2105.04236* (2021).
[58] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3372297.3417274
[59] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. {XONN}: XNOR-based Oblivious Deep Neural Network Inference. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1501–1518.
[60] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 707–721.
[61] Dragos Rotaru and Tim Wood. 2019. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *International Conference on Cryptology in India*. Springer, 227–249.
[62] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.
[63] Ashish P Sanil, Alan F Karr, Xiaodong Lin, and Jerome P Reiter. 2004. Privacy preserving regression modelling via distributed computation. In *Proceedings of 10th ACM SIGKDD international conference on Knowledge discovery and data mining*. 677–682.
[64] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[65] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[66] Aleksandra B Slavkovic, Yuval Nardi, and Matthew M Tibbits. 2007. " Secure" Logistic Regression of Horizontally and Vertically Partitioned Distributed Databases. In *Seventh IEEE International Conference on Data Mining Workshops*. IEEE, 723–728.
[67] Ajith Suresh. 2021. Mpcleague: robust MPC platform for privacy-preserving machine learning. *arXiv preprint arXiv:2112.13338* (2021).
[68] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CRYPTGPU: Fast Privacy-Preserving Machine Learning on the GPU. *arXiv preprint arXiv:2104.10949* (2021).
[69] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. 2008. Privacy-preserving SVM classification. *Knowledge and Information Systems* 14, 2 (2008), 161–178.
[70] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 26–49.
[71] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *arXiv preprint arXiv:2004.02229* (2020).
[72] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A GPU Platform for Secure Computation. *Cryptology ePrint Archive* (2022).
[73] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 162–167.
[74] Hwanjo Yu, Jaideep Vaidya, and Xiaoqian Jiang. 2006. Privacy-preserving svm classification on vertically partitioned data. In *Pacific-asia conference on knowledge discovery and data mining*. Springer, 647–656.

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared $\{[\![\mathsf{x}_t]\!]\}_{t=1}^{N}$.
Output: $[\![\mathsf{z}]\!] = \Pi_{t=1}^{N}[\![\mathsf{x}_t]\!]$.
- **Setup:**
  (1) Parties generate $\langle\psi_\mathsf{z}\rangle$ using functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
  (2) For $\mathcal{T} \subseteq \{1,\ldots,N\}$, parties securely compute $\langle\Pi_{k\in\mathcal{T}}\psi_{\mathsf{x}_k}\rangle = \Pi_{k\in\mathcal{T}}\langle\psi_{\mathsf{x}_k}\rangle$ using $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ in a tree-manner.
- **Online:**
  (1) P_i locally computes $\langle\mathsf{m}_\mathsf{z}\rangle_i = \sum_{\mathcal{T}\subseteq\{1,\ldots,N\}}(\Pi_{j\notin\mathcal{T}}\mathsf{m}_{\mathsf{x}_j} \cdot \langle\Pi_{k\in\mathcal{T}}\psi_{\mathsf{x}_k}\rangle_i) - \langle\psi_\mathsf{z}\rangle_i$.
  (2) Parties exchange the shares of $\langle\mathsf{m}_\mathsf{z}\rangle$ to reconstruct m_z.
  (3) P_i outputs $[\![\mathsf{z}]\!]_i = (\mathsf{m}_\mathsf{z}, \langle\psi_\mathsf{z}\rangle_i)$.

Figure 5: N-Input Multiplication Protocol $\Pi_{N\text{-MULT}}$.

A NON-INTERACTIVE $\langle\cdot\rangle$-RANDOMNESS GENERATION

Functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$ enables parties sample an secret random number in $\langle\cdot\rangle$-shared fashion using a pseudorandom function PRF [8]. Protocol Π_{RAND} achieves $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$ as follows: Each pair of parties (*i.e.*, P_i and P_j) maintain a pre-set random shared-key $k_{ij} \in_R \{0,1\}^\kappa$, *i.e.*, P_0 and P_1 share key k_{01}. Then, P_i and P_{i+1} generate $\mathsf{r}_i \leftarrow \mathrm{PRF}(k_{i,i+1}, \mathsf{cnt})$ where cnt is a counter incremented for $i \in \{0,1,2\}$. Finally, P_i sets $\langle\mathsf{r}\rangle_i = (\mathsf{r}_i, \mathsf{r}_{i+1})$ [6, 49]. Also, we can let P_i (secret owner) obtain r in clear by letting P_i keep $k_{i-1,i+1}$ and generate r_{i-1} locally. As the pre-set shared-keys are secure (*i.e.*, $\kappa = 128$), the security of Π_{RAND} is easy to see. Similarly, parties can generate shared random bit $\langle\mathsf{r}\rangle^2$ in $\mathbb{Z}_2$ and $\langle\mathsf{r}\rangle^q$ in $\mathbb{F}_q$.

B PROTOCOL $\Pi_{N\text{-MULT}}$

Figure 5 show the N-input multiplication protocol for integers.

C INPUT-INDEPENDENT RANDOMNESS GENERATION

Following Falcon [71], we generate the input-independent randomness for protocol Π_{SecMSB} in the setup phase as Figure 6: i) We first accomplish the bit decomposition of ψ_x in $\langle\cdot\rangle$-sharing using $\mathcal{F}_{\mathsf{BitDec}}^{\langle\cdot\rangle}$ [49], so that it is trivially to extract $\langle\mathsf{msb}(\psi_\mathsf{x})\rangle^2$ and $\{\langle\mathsf{s}[i]\rangle^2\}_{i=1}^{\ell}$ ($\mathsf{s} = 2\psi_\mathsf{x}$). ii) Then, we reshare $\langle\mathsf{msb}(\psi_\mathsf{x})\rangle^2$ as $[\![\mathsf{msb}(\psi_\mathsf{x})]\!]^2$. iii) Next, we convert $\{\langle\mathsf{s}[i]\rangle^2\}_{i=1}^{\ell}$ to $\langle\cdot\rangle^q$-shares in $\mathbb{F}_q$ leveraging $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$, and reshare $\{\langle\mathsf{s}[i]\rangle^q\}_{i=1}^{\ell}$ as $\{[\![\mathsf{s}[i]]\!]^q\}_{i=1}^{\ell}$. iv) Finally, we generate $[\![\zeta]\!]^q \in \mathbb{F}_q^*$ based on $\mathcal{F}_{\mathsf{Prep}}^{\langle\cdot\rangle}$ from [71].

Protocol Π_{PreMSB} is a little expensive but practical since it can be executed in the setup phase. The correctness is guaranteed and security follows the analysis in [49, 71], and we omit it for brevity.

D BIT TO ARITHMETIC CONVERSION FOR $\langle\cdot\rangle$-SHARING

We first introduce functionality $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$, which generates the arithmetic sharing of a bit $\mathsf{v} \in \{0,1\}$, given $\langle\mathsf{v}\rangle^2 = (\mathsf{v}_0, \mathsf{v}_1, \mathsf{v}_2)$. We utilize protocol $\Pi_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ to achieve $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ as follows. Denote the value in $\mathbb{Z}_{2^\ell}$ of bit v as v^ℓ. Then we have

$$\begin{aligned}\mathsf{v}^\ell &= \mathsf{v}_0 \oplus \mathsf{v}_1 \oplus \mathsf{v}_2 \\ &= \mathsf{v}_0^\ell + \mathsf{v}_1^\ell + \mathsf{v}_2^\ell - 2\mathsf{v}_0^\ell\mathsf{v}_1^\ell - 2\mathsf{v}_1^\ell\mathsf{v}_2^\ell - 2\mathsf{v}_2^\ell\mathsf{v}_0^\ell + 4\mathsf{v}_0^\ell\mathsf{v}_1^\ell\mathsf{v}_2^\ell.\end{aligned} \tag{4}$$

Input: P_0, P_1, and P_2 hold $\langle \psi_{\mathsf{x}} \rangle$.
Output: $[\![\mathsf{msb}(\psi_{\mathsf{x}})]\!]^2$, $\{[\![\mathsf{s}[i]]\!]^q\}_{i=1}^{\ell}$ with $\mathsf{s} = 2\psi_{\mathsf{x}}$, $([\![\lambda]\!]^2, [\![\lambda]\!]^q)$ with $\lambda \in_R \mathbb{Z}_2$, and $[\![\zeta]\!]^q$ with $\zeta \in_R \mathbb{F}_q^*$.

(1) Perform bit decomposition $\mathcal{F}_{\mathsf{BitDec}}^{\langle\cdot\rangle}$ from [49] to get $\langle \psi_{\mathsf{x}} \rangle \to \langle \psi_{\mathsf{x}} \rangle^2$.
(2) Sample random bits $\langle \psi_{\psi_{\mathsf{x}}} \rangle^2$, $\langle \lambda \rangle^2$, and $\langle \psi_{\lambda} \rangle^2$, random values $\langle \psi_{\lambda} \rangle^q$, $\langle \psi_{\zeta} \rangle^q$, and $\{\langle \psi_{\mathsf{s}[i]} \rangle^q\}_{i=1}^{\ell}$ using $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
(3) Reveal $\langle \mathsf{m}_{\psi_{\mathsf{x}}} \rangle^2 = \langle \psi_{\mathsf{x}} \rangle^2[\ell] \oplus \langle \psi_{\psi_{\mathsf{x}}} \rangle^2$ using $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$, where $\langle \psi_{\mathsf{x}} \rangle^2[\ell]$ is the $\langle\cdot\rangle$-shares of the msb of ψ_{x}. And set $[\![\mathsf{msb}(\psi_{\mathsf{x}})]\!]^2 = (\mathsf{m}_{\psi_{\mathsf{x}}}, \langle \psi_{\psi_{\mathsf{x}}} \rangle^2)$.
(4) Compute $\langle \mathsf{s} \rangle^2 = (\langle \psi_{\mathsf{x}} \rangle^2 \ll 1)$.
(5) Use $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ for each bit of s to get $\langle \mathsf{s}[i] \rangle^2 \to \langle \mathsf{s}[i] \rangle^q$ with $i \in \{1, 2, \ldots, \ell\}$ and $\langle \lambda \rangle^2 \to \langle \lambda \rangle^q$.
(6) Reveal $\langle \mathsf{m}_{\mathsf{s}[i]} \rangle^q = \langle \mathsf{s}[i] \rangle^q - \langle \psi_{\mathsf{s}[i]} \rangle^q \pmod q$ exploiting $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$, and set $[\![\mathsf{s}[i]]\!]^q = (\mathsf{m}_{\mathsf{s}[i]}, \langle \psi_{\mathsf{s}[i]} \rangle^q)$, $i \in \{1, 2, \ldots, \ell\}$.
(7) Reveal $\langle \mathsf{m}_{\lambda} \rangle^2 = \langle \lambda \rangle^2 \oplus \langle \psi_{\lambda} \rangle^2$ leveraging $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$, and set $[\![\lambda]\!]^2 = (\mathsf{m}_{\lambda}, \langle \psi_{\lambda} \rangle^2)$.
(8) Reveal $\langle \mathsf{m}'_{\lambda} \rangle^q = \langle \lambda \rangle^q - \langle \psi_{\lambda} \rangle^q \pmod q$ via $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$, and set $[\![\lambda]\!]^q = (\mathsf{m}'_{\lambda}, \langle \psi_{\lambda} \rangle^q)$.
(9) Generate $\langle \zeta \rangle^q \in \mathbb{F}_q^*$ using $\mathcal{F}_{\mathsf{Prep}}^{\langle\cdot\rangle}$ from [71].
(10) Reveal $\langle \mathsf{m}_{\zeta} \rangle^q = \langle \zeta \rangle^q - \langle \psi_{\zeta} \rangle^q \pmod q$ by $\mathcal{F}_{\mathsf{REC}}^{\langle\cdot\rangle}$, and set $[\![\zeta]\!]^q = (\mathsf{m}_{\zeta}, \langle \psi_{\zeta} \rangle^q)$.
(11) Outputs $[\![\mathsf{msb}(\psi_x)]\!]^2$, $\{[\![\mathsf{s}[i]]\!]^q\}_{i=1}^{\ell}$, $([\![\lambda]\!]^2, [\![\lambda]\!]^q)$, and $[\![\zeta]\!]^q$.

Figure 6: Secure MSB Extraction Pre-processing Protocol Π_{Pre}.

Input: P_0, P_1, and P_2 hold $\langle \mathsf{v} \rangle^2$ with $\mathsf{v} \in \{0, 1\}$.
Output: $\langle \mathsf{v}^{\ell} \rangle$.

(1) P_2 samples $a_0, a_1 \in_R \mathbb{Z}_{2^{\ell}}$ and sends a_i to P_i for $i \in \{0, 1\}$.
(2) P_0 computes $\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell} + a_0$ and sends it to P_1; P_1 computes $\mathsf{v}_2^{\ell} + a_1$ and sends it to P_0.
(3) P_0 computes $[\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}]_0 = -a_0(\mathsf{v}_2^{\ell} + a_1)$, P_1 computes $[\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}]_1 = \mathsf{v}_2^{\ell}(\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell} + a_0)$, and P_2 computes $[\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}]_2 = a_0a_1$.
(4) P_i locally computes $[\mathsf{v}^{\ell}]_i = \mathsf{v}_i^{\ell} - 2\mathsf{v}_i^{\ell}\mathsf{v}_{i+1}^{\ell} - [\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}]_i$.
(5) Parties reshare $[\mathsf{v}^{\ell}]$ as $\langle \mathsf{v}^{\ell} \rangle$.

Figure 7: Bit2A Conversion Protocol for $\langle\cdot\rangle$-Sharing $\Pi_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$.

For equation (4), only the last term $\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}$ requires communication and other terms $(\mathsf{v}_i^{\ell}, \mathsf{v}_i^{\ell}\mathsf{v}_{i+1}^{\ell})_{i=0}^2$ can be computed by P_i locally. For $\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}$, we firstly let P_0 compute $\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}$. Then, parties can invoke Du-Atallah protocol [11, 60] to compute $\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}$ securely. Finally, we get the $[\cdot]$-shared $[\mathsf{v}_0^{\ell}\mathsf{v}_1^{\ell}\mathsf{v}_2^{\ell}]$ (step 1)-3) in Figure 7. Also, the other terms form $[\cdot]$-sharing (*e.g.*, for $[\mathsf{v}_0^{\ell}]$, $[\mathsf{v}_0^{\ell}]_0 = \mathsf{v}_0^{\ell}$, $[\mathsf{v}_0^{\ell}]_1 = [\mathsf{v}_0^{\ell}]_2 = 0$), we hence can compute $[\mathsf{v}^{\ell}]$ locally. In the end, parties can reshare $[\mathsf{v}^{\ell}]$ as $\langle \mathsf{v}^{\ell} \rangle$. The details are in Figure 7. Protocol $\Pi_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ needs a communication of $\frac{7}{3}\ell$ bits per party in 3 rounds, and the security is fully analyzed in [11].

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared matrices $[\![\mathbf{X}]\!]$ and $[\![\mathbf{Y}]\!]$.
Output: $[\![\mathbf{Z}]\!] = [\![\mathbf{X}]\!] \cdot [\![\mathbf{Y}]\!]$.

- **Setup:**
  (1) Parties generate $\langle \psi_{\mathbf{Z}} \rangle$ using functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
  (2) Parties execute $\langle \psi_{\mathbf{XY}} \rangle = \mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}(\langle \psi_{\mathbf{X}} \rangle, \langle \psi_{\mathbf{Y}} \rangle)$.
- **Online:**
  (1) P_i locally computes $\langle \mathsf{m}_{\mathbf{Z}} \rangle_i = \mathsf{m}_{\mathbf{XY}} + \mathsf{m}_{\mathbf{X}}\langle \psi_{\mathbf{Y}} \rangle_i + \mathsf{m}_{\mathbf{Y}}\langle \psi_{\mathbf{X}} \rangle_i + \langle \psi_{\mathbf{XY}} \rangle_i - \langle \psi_{\mathbf{Z}} \rangle_i$.
  (2) Parties exchange the shares of $\langle \mathsf{m}_{\mathbf{Z}} \rangle$ to reconstruct $\mathsf{m}_{\mathbf{Z}}$.
  (3) P_i sets and outputs $[\![\mathbf{Z}]\!]_i = (\mathsf{m}_{\mathbf{Z}}, \langle \psi_{\mathbf{Z}} \rangle_i)$.

Figure 8: Secure Matrix Multiplication Protocol Π_{MM}.

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared $[\![\mathsf{x}]\!]$.
Output: $[\![\mathsf{z}]\!] = \mathsf{ReLU}([\![\mathsf{x}]\!])$.

- **Setup:**
  (1) Parties execute the setup phases of $\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}$ and $\mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]}$.
- **Online:**
  (1) Parties compute $[\![\mathsf{msb}(\mathsf{x})]\!]^2$ using $\mathcal{F}_{\mathsf{SecMSB}}^{[\![\cdot]\!]}$.
  (2) Output $[\![\mathsf{z}]\!] = \mathcal{F}_{\mathsf{MUX}}^{[\![\cdot]\!]}([\![\mathsf{x}]\!], [\![\mathsf{y}]\!], [\![\mathsf{msb}(\mathsf{x})]\!]^2 \oplus 1)$ with $\mathsf{y} = 0$.

Figure 9: Secure ReLU Protocol Π_{ReLU}.

Input: P_0, P_1, and P_2 hold $[\![\cdot]\!]$-shared vector $[\![\mathsf{x}]\!] = (\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n)$.
Output: $[\![\mathsf{max}]\!]$ with $\mathsf{max} = \mathsf{Max}(\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n)$.

- **Setup:**
  (1) Parties generate $\langle \psi_{\mathsf{z}} \rangle$ using functionality $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$.
- **Online:**
  (1) Parties set $[\![\mathsf{max}]\!] = [\![\mathsf{x}_1]\!]$.
  (2) **for** $i = 2, 3, \ldots, n$, **do**
  (3) Parties compute $[\![\mathsf{max}]\!] = \mathsf{ReLU}([\![\mathsf{max}]\!] - [\![\mathsf{x}_i]\!]) + [\![\mathsf{x}_i]\!]$.
  (4) **end for**
  (5) Parties output $[\![\mathsf{max}]\!]$.

Figure 10: Secure Maxpool Protocol Π_{MP}.

D.1 Bit to Arithmetic Conversion for $[\![\cdot]\!]$-Sharing

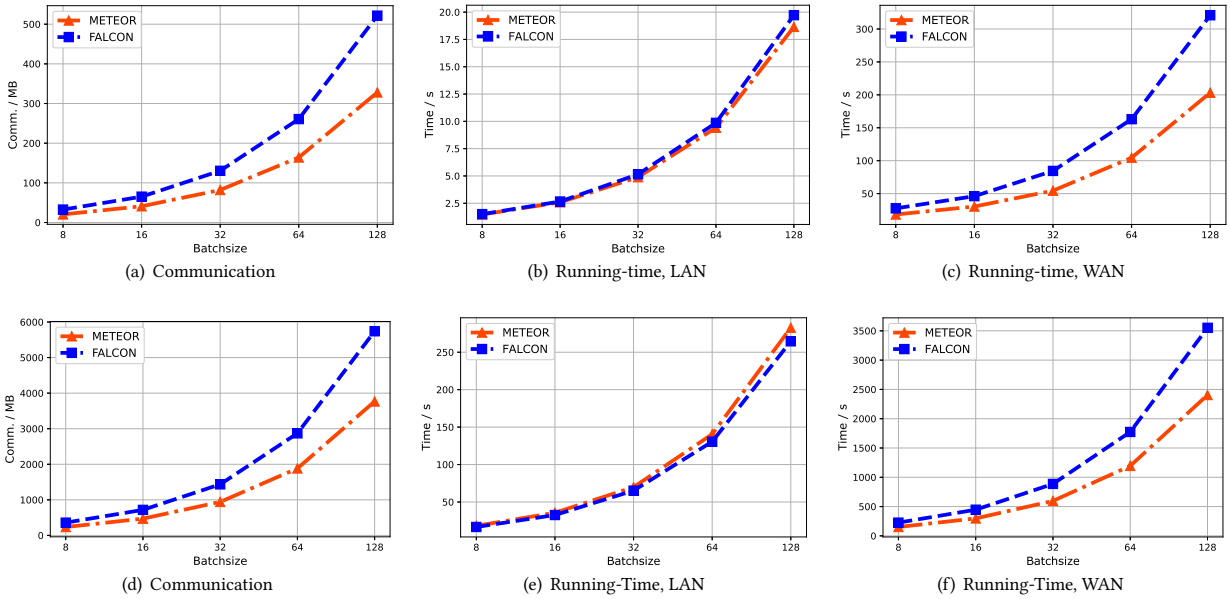
The goal of functionality $\mathcal{F}_{\mathsf{Bit2A}}^{[\![\cdot]\!]}$ is to generate the arithmetic sharing of a given secret bit $[\![\mathsf{v}]\!]^2 = (\mathsf{m}_{\mathsf{v}}, \langle \psi_{\mathsf{v}} \rangle^2)$. Given $[\![\mathsf{v}]\!]^2$, we have

$$\mathsf{v}^{\ell} = (\mathsf{m}_{\mathsf{v}} \oplus \psi_{\mathsf{v}})^{\ell} = \mathsf{m}_{\mathsf{v}}^{\ell} + \psi_{\mathsf{v}}^{\ell} - 2\mathsf{m}_{\mathsf{v}}^{\ell}\psi_{\mathsf{v}}^{\ell}. \tag{5}$$

In the setup phase, parties generate the $\langle\cdot\rangle$-shares of value ψ_{v}^{ℓ} using $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ and sample random values $\langle \psi'_{\mathsf{v}} \rangle$ in $\mathbb{Z}_{2^{\ell}}$ using $\mathcal{F}_{\mathsf{RAND}}^{\langle\cdot\rangle}$. In the online phase, parties locally compute $\langle \mathsf{v}^{\ell} \rangle = \mathsf{m}_{\mathsf{v}}^{\ell} + \langle \psi_{\mathsf{v}}^{\ell} \rangle - 2\mathsf{m}_{\mathsf{v}}^{\ell}\langle \psi_{\mathsf{v}}^{\ell} \rangle$ and reshare $\langle \mathsf{v}^{\ell} \rangle$ as $[\![\mathsf{v}^{\ell}]\!]$. Concretely, parties compute and reveal $\langle \mathsf{m}'_{\mathsf{v}} \rangle = \langle \mathsf{v}^{\ell} \rangle - \langle \psi'_{\mathsf{v}} \rangle$, and set $[\![\mathsf{v}^{\ell}]\!] = (\mathsf{m}'_{\mathsf{v}}, \langle \psi'_{\mathsf{v}} \rangle)$. $\mathcal{F}_{\mathsf{Bit2A}}^{[\![\cdot]\!]}$ might be of independent interest and be helpful in tasks such as secure e-voting.
Online Communication In the online phase, the parties reconstruct $\langle \mathsf{m}'_{\mathsf{v}} \rangle$ with a communication of ℓ bits per party in 1 round.

E SECURE BUILDINGS OF NN OPERATORS

Figure 8-10 show the secure protocols for MatMul, ReLU, and MP.
Secure Buildings of Other NN Operators: In addition, our basic primitives can support other secure NN operators: i) For other Activation functions (*i.e.*, Sigmoid and Tanh), we approximate them by piece-wise continuous polynomials following previous works [46,



(a) Communication (b) Running-time, LAN (c) Running-time, WAN

(d) Communication (e) Running-Time, LAN (f) Running-Time, WAN

Figure 11: Online costs of batch inference of Falcon and Meteor for AlexNet and VGG16 on CIFAR-10, where 11(a)-11(c) are for AlexNet and 11(d)-11(f) are for VGG16. Communication is in MB and Running-Time is in seconds.

50, 54]. The polynomials can be expressed as basic operations, and we can construct their secure protocols on top of the basic primitives. ii) Avgpool is much simpler than Maxpool. Note that the poolsize is in plaintext, parties thus can compute the sum of their respective shares and truncate the sum by the poolsize to get the approximate average [37, 68]. As we do not employ these operators in Meteor, we omit their detailed protocols for brevity.

F ONLINE COSTS OF BATCH INFERENCE

The online costs of batch inference are illustrated in Figure 11.

G SECURITY PROOF

Proof. Let the semi-honest adversary $\mathcal{A}$ corrupt no more than one party, we now present the steps of the ideal-world adversary (simulator) $\mathcal{S}$ for $\mathcal{A}$ in the stand-alone model with security under sequential composition [16]. Our simulator $\mathcal{S}$ for individual protocol is constructed as follows:

Security for Π_{SHARE}: For the instances where $\mathcal{A}$ is the owner of the secret value x, $\mathcal{S}$ has to do nothing since $\mathcal{A}$ is not receiving any messages. $\mathcal{S}$ receives $\mathsf{m_x}$ from $\mathcal{A}$ on behalf of honest parties. For the instances where one honest party is the owner, $\mathcal{S}$ sets x = 0 and follows protocol honestly.

Security for Π_{REC}: To reconstruct a value x, $\mathcal{S}$ is given the output x, which is the output of $\mathcal{A}$. Using x and shares corresponding to honest parties, $\mathcal{S}$ computes the shares corresponding to $\mathcal{A}$ and sends this to $\mathcal{A}$ on behalf of honest parties. $\mathcal{S}$ sends the shares of honest parties to $\mathcal{A}$ on behalf of honest parties.

Security for Π_{Lops}: There is nothing to simulate as the protocol Π_{Lops} is non-interactive.

Security for $\Pi_{2\text{-MULT}}$ & $\Pi_{N\text{-MULT}}$: For the setup phase, we consider the multiplication of $\langle\cdot\rangle$-sharing as an ideal functionality $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ which multiplies the randomness. Since we make only black-box access to $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$, the simulation for the same follows from the security of the underlying primitive used to instantiate $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$ [6]. During the online phase, $\mathcal{S}$ follows the step honestly using the data obtained from the corresponding setup phase.

Security for Π_{SecMSB}: For the setup phase, we invoke $\mathcal{F}_{\mathsf{PreMSB}}$ in a black-box manner as Falcon [71]. Therefore, the simulation for the same follows from the security analyzed in [71]. For the online phase, we make black-box access to $\mathcal{F}_{\mathsf{Lops}}^{[\![\cdot]\!]}$, $\mathcal{F}_{2\text{-MULT}}^{[\![\cdot]\!]}$, and $\mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]}$. To simulate the revealed d, $\mathcal{S}$ samples a random number $r \in_R \mathbb{F}_q$, shares r as $[\![r]\!]^q$, and sends $\mathcal{A}$'s share to it. The security of protocol Π_{SecMSB} follows in the $(\mathcal{F}_{\mathsf{PreMSB}}^{[\![\cdot]\!]}, \mathcal{F}_{2\text{-MULT}}^{[\![\cdot]\!]}, \mathcal{F}_{N\text{-MULT}}^{[\![\cdot]\!]})$-hybrid model.

Security for Π_{MUX}: The setup phase is constructed directly on $\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}$ and $\mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle}$, thus the security is easily to see in $(\mathcal{F}_{\mathsf{Bit2A}}^{\langle\cdot\rangle}, \mathcal{F}_{\mathsf{MULT}}^{\langle\cdot\rangle})$-hybrid model. For the online phase, $\mathcal{S}$ follows the steps honestly using the data obtained from the corresponding setup phase.

This concludes the proof. □