Antelope: Fast and Secure Neural Network Inference

Xiaoyuan Liu, Hongwei Li, *Fellow, IEEE*, Guowen Xu, Shengmin Xu, Xinyi Huang, Tianwei Zhang, Yijing Lin, and Jianying Zhou

***Abstract*—In this paper, we present Antelope, a semi-honest large-scale secure inference system without revealing either clients' data or model parameters. The main contributions of Antelope are new two-party computation (2PC) protocols over a ring $\mathbb{Z}_{2^\ell}$ for non-linear layers, which optimize the online computation and communication overhead thus outperforming the state-of-the-art 2PC systems. Specifically, we reformulate the comparison function as an Equality-to-Zero test followed by multiplication, decoupling the bit-wise rounding dependency in traditional secret sharing-based bit extraction. With this technique, the evaluation of the ReLU non-linear activation function is $1.7\times$-$84.5\times$ faster than existing solutions in online communication cost. We also develop a suite of optimizations that improve the efficiency of secure division protocols, which are tailored to different divisor settings in the neural networks. We extend our protocols to construct efficient implementations for several building blocks such as ReLU, Max-pool, truncation, and Softmax. End-to-end evaluation on realistic ImageNet-scale networks demonstrates that Antelope achieves over $22.3\times$ and $23.0\times$ online runtime speedups in LAN and WAN settings, respectively, without accuracy loss, compared to the state-of-the-art works.**

***Index Terms*—Neural network inference, secure two-party protocol.**

Received 24 December 2023; revised 21 May 2025; accepted 11 June 2025. Date of publication 7 August 2025; date of current version 4 November 2025. This work was supported in part by Beijing Natural Science Foundation under Grant L251038 and Grant QY24203, in part by CCF-Huawei Populus Grove Fund under Grant TC202418, in part by the Fellowship of China National Postdoctoral Program for Innovative Talents under Grant BX20240045, and in part by China Postdoctoral Science Foundation General Program under Grant 2025M773481. *(Corresponding author: Guowen Xu.)*

Xiaoyuan Liu is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, also with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China, and also with DBAPP Security Co., Ltd., Hangzhou 310000, China (e-mail: xiaoyuan.l@foxmail.com).

Hongwei Li and Guowen Xu are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: hongweili@uestc.edu.cn; guowen.xu@foxmail.com).

Shengmin Xu is with the College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350109, China (e-mail: smxu1989@gmail.com).

Xinyi Huang is with Artificial Intelligence thrust, Information Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511458, China (e-mail: xinyi@ust.hk).

Tianwei Zhang is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: tianwei.zhang@ntu.edu.sg).

Yijing Lin is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: yjlin@bupt.edu.cn).

Jianying Zhou is with the Singapore University of Technology and Design, Singapore 487372 (e-mail: jianying_zhou@sutd.edu.sg).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2025.3596652, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2025.3596652

I. Introduction

MACHINE learning has emerged as a promising branch of applied science due to the increasing volume of data. Its potential is available in critical areas such as medical diagnosis, face and speech recognition, object classification, question answering, and threat analysis. Many well-known enterprises, such as Amazon, Google, and Microsoft, have already pioneered pay-per-use cloud-based machine learning platforms to provide model inference services, also referred to as Machine Learning as a Service (MLaaS) [27]. However, the deployment of MLaaS for commercial applications raises significant privacy concerns: existing services either require clients to upload potentially sensitive query data to the cloud or deploy the proprietary model on their devices. Neither approach is desirable as the former compromises the clients' privacy, while the latter could reveal important information such as model parameters and training data, seriously infringing the commercial intellectual property of the service provider's model.

To alleviate the concerns mentioned above, the machine learning community has dedicated itself to tailoring specialized cryptographic protocols to evaluate neural network inference without revealing either the client's data or the service provider's model [8], [20], [22], [31], [39], [40], [50], [52]. Recently, the hybrid methods that combine secret sharing-based linear calculations with Yao's garbled circuit (GC) [59] based non-linear operations have shown superior performance [36], [41], [43]. By relocating heavy cryptographic computations to a data-independence phase, secret sharing enables fast addition and multiplication. This makes the evaluation speed of the linear layers close to that of plaintext. Currently, the computation of non-linear functions such as ReLU is a major hurdle for both 2PC constructions and realistic neural network inference. Privately evaluating such functions with GC requires that the function is decomposed as a circuit of binary gates that process the input bit-wise, where each gate is expressed as encrypted two-input truth tables [21]. Consequently, evaluating any function with a ℓ-bit input takes the communication complexity of at least $O(\kappa\ell)$, where κ is the cryptographic security parameter. As such, the high computation and communication overheads caused by cryptographic tools limit the above works to shallow models on small datasets [51]. For instance, these methods may perform satisfactorily on the MNIST classification task using the LeNet-5 model, which processes only approximately 60 K parameters. They are not efficient in privately performing the practical

TABLE I
COMPARISON WITH PRIOR WORKS FOR SECURE COMPARISON PROTOCOLS, WHERE THE BEST RESULTS ARE IN BOLD

Protocol	Work	Setup	Online	
		Comm	Comm	Rounds
Secure Comparison over ring $\mathbb{Z}_{2^\ell}$	GC [18], [36], [41], [60]	–	$16\ell\kappa - 6\kappa$	2
	CrypTFlow2 [49]	–	$\frac{3}{2}\kappa(\ell+1) + 31\ell - 13$	$\log\ell + 2$
	Cheetah [33]	–	$4(13\kappa + 712) + 4\ell + 2$	$\log\ell + 2$
	ABY2.0 [47]	$x_1(\kappa+1) + \ell$	$\ell + 2x_2$	$\log_4\ell + 1$
	Our work	$6\ell^2 + 9\ell + 44n + 1$	$\mathbf{2\ell + 2n}$	$\log_4\ell + 2$
An instance of Secure Comparison $\ell = 64$, $\kappa = 128$, $n = 21$	GC [18], [36], [41], [60]	–	130,304	2
	CrypTFlow2 [49]	–	14,451	8
	Cheetah [33]	–	9,762	8
	ABY2.0 [47]	171,892	294	4
	Our work	**26,077**	**170**	5

κ is the cryptographic security parameter associated with standard cryptographic primitives, typically 128. n refers to the number of four-input AND gates. For ABY2.0 [47], we have $x_1 = 2n_2+8n_3+22n_4$ and $x_2 = n_2+n_3+n_4$, where n_2, n_3, n_4 denote the number of AND gates in the bit extraction circuit with 2, 3, 4 inputs, respectively. For example, when $\ell = 64$, the circuit needs $n_2 = 41$, $n_3 = 27$ and $n_4 = 47$.

ImageNet inference task, e.g., using the ResNet50 model with nearly 23 M parameters.

CrypTFlow2 [49] is the first work to show the possibility of secure inference over large-scale models. It proposes a new comparison protocol using the Oblivious Transfer (OT) technique [5] and applies it to non-linear layers of neural networks, which achieves significant improvements in computation efficiency. Meanwhile, it also gives a secure division protocol with public divisor by representing the division of ring element to signed integer division, which achieves 54× less communication over GC. However, this work is still far from practical. Taking a home surveillance system that uses ResNet50 for activity monitoring as an example, CrypTFlow2 takes about 60 minutes to recognize malicious activities in an Intra Picture with the resolution of 256× 256pixels and then issue an alert (see Table 7 in [49]). More recent research, Cheetah [33], makes improvements over CrypTFlow2 by 5× in runtime and 13× in communication costs, respectively. It designs a homomorphic encryption-based protocol to evaluate linear layers without expensive rotation operations and replaces the OT protocols [38] used in CrypTFlow2 with more efficient ones [7], [54], [58]. Despite these advanced solutions, Cheetah still can not meet the real-time requirements in practice, especially for efficiency-critical scenarios such as aviation and finance. This motivates us to design a more efficient and secure protocol for real-world neural network inference.

A. Our Contributions

This paper proposes Antelope, a fast and secure two-party computation (2PC) scheme over a ℓ-bit ring (i.e., $\mathbb{Z}_{2^\ell}$) for neural network inference, which provides security against a static semi-honest adversary. Antelope inherits the efficient implementation for linear layers from prior works [18], [47], and aims to construct novel building blocks for non-linear functions to achieve high online efficiency. By pushing heavy cryptographic operations into an input-independent setup phase, the proposed protocols enable a lightweight "non-cryptographic" online phase once the inputs are known, empowering us to perform secure inferences on more realistic network architectures than those considered in prior works [18], [41], [47]. Our gains largely stem from simple protocol constructions, reduced round complexity and communication cost, as well as improvements in silent OT generators [7], [58]. As a result, our scheme enjoys a lower amortized communication cost for preprocessing materials and competitive online efficiency in secure inference. To sum up, we make the following contributions:

- *New secure comparison protocol:* We first refactor the comparison operation as a sign extraction computation. Then, we design a novel protocol for sign extraction using an Equality-to-Zero circuit, which decouples the bit-wise rounding dependency in traditional secret sharing based bit extraction. This design reduces the online communication cost by 762.0× over GC [41], [47], 84.5× over CrypTFlow2 [49], 57.1× over Cheetah [33], and 1.7× over ABY2.0 [47] (see Table I for detailed comparisons). We theoretically and experimentally verify that this design results in a direct improvement in online and setup communication overhead. With this end-to-end solution, we can securely and efficiently evaluate the non-linear operations (e.g., ReLU) in neural networks.
- *New optimizations for secure division protocols:* By carefully co-designing the model architecture and cryptographic components, we propose novel optimizations for two division protocols with different divisor settings to evaluate neural networks: a) the divisor is secretly shared between two parties, which can be used to perform Softmax; b) the divisor is a public constant for both parties, such as truncation after multiplication. The secure division protocol with a public divisor in Antelope improves the performance by up to 1.7× over the state-of-the-art [33] in online communication cost with less than half of the communication rounds. For more challenging secret-shared divisor setting, our protocol reduces the online communication overhead from $O(\kappa\ell^2)$ to $O((Deg + \log\ell)\cdot\ell)$, where $Deg + \log\ell \ll \kappa\ell$.
- *Concrete efficiency:* We make a black-box use of the silent OT technique [7], [58] to prepare pre-processing materials in secure computation such as multiplication triples with low communication overhead. With this, we can compute 10^7 triples in less than 3 seconds, resulting in significant reductions in communication and computation costs of the setup phase. We conduct extensive experiments on realistic ImageNet-scale models: SqueezeNet, ResNet50, and DenseNet121. Experimental results indicate that Antelope is an order of magnitude more efficient than state-of-the-art works. For example, in the

WAN setting, using Antelope for the secure inference on ResNet50 yields about 29.0× faster online runtime than Cheetah [33].

B. Paper Organization

We briefly review the threat model and primitives used in this paper in Section II. We elaborate on the details of the secure comparison protocol in Section III, followed by the secure division protocols in Section IV. We describe several building blocks for secure neural network inference and formal security proof in Section V. We provide detailed implementation and experimental results in Section VI. In Section VII, we discuss some related works. Section VIII concludes the paper and discusses future work.

II. Preliminaries

A. Threat Model

Our secure inference protocol Π_{NN} is in the two-party setting, where the service provider P_0 has a proprietary neural network X_S, while the client P_1 holds private query data X_C. Both the service provider and the client are assumed to be aware of the neural network architecture. Consistent with previous semi-honest 2PC inference works [18], [36], [41], [43], [47], [49], [60], we aim to defend against a static semi-honest adversary $\mathcal{A}$ corrupting one of the two parties. $\mathcal{A}$ may endeavor to gather additional information from what he masters. With this threat model, Antelope aims to allow the client P_1 to obtain the most inference result $\mathcal{F}_{\text{NN}}(X_S, X_C)$, where $\mathcal{F}_{\text{NN}}$ is the inference functionality, while the service provider P_0 is required to perform the prediction securely without obtaining any information on the private query data of the client. The formal security definition is given in Section A of Appendices, available online.

Since our scheme is built on a semi-honest adversary model, deliberately submitting a crafted query or arbitrary intermediate values to compromise privacy is beyond the scope of this paper [10]. We consider it the strongest security model and will discuss it in future work. Additionally, defending against indirect privacy attacks, such as membership inference [55] and model inversion [45], are orthogonal directions and are not considered in this paper. Our scheme can be combined with their efforts to construct an all-encompassing system for secure and private inference.

B. Notations

In this work, we consider a set of two parties $\mathcal{P} = \{P_0, P_1\}$ who interactively perform neural network inference protocol. For machine learning where the real values $\tilde{\text{v}} \in \mathbb{R}$ are floating-point numbers, we use fixed-point arithmetic representation to encode the value into fixed-point integers v over the ring $\mathbb{Z}_{2^\ell}$, where the most significant bit (MSB) represents the sign while the least significant s bits represent the fractional part. We use $\ell = 64$ and $s = 13$ in our implementation. Besides, $\mathbf{1}\{b\}$ denotes the indicator function that outputs 1 if and only if b is true. For a finite set $\mathcal{W}$, the notation $w \xleftarrow{\$} \mathcal{W}$ refers to sample w independently and uniformly at random from the set $\mathcal{W}$. To boost the efficiency of secure inference, we perform operations over the arithmetic or Boolean world. So we use superscripts A and B to distinguish the protocols or sharing in Arithmetic ring $\mathbb{Z}_{2^\ell}$ or in the Boolean world $\mathbb{Z}_2$. Also, we place the lowercase letter a in the superscript to denote the value over an arithmetic ring $\mathbb{Z}_{2^\ell}$ corresponding to its boolean representation over $\mathbb{Z}_2$. In some cases, superscripts are omitted for simplicity of description, which refers to the arithmetic world by default.

C. Private Neural Network Inference

Throughout this paper, we revolve around a classical machine learning algorithm, i.e., convolutional neural network. Typically, modern neural networks consist of one input layer, many hidden layers, and one output layer. The hidden layers stitch the linear layer and non-linear alternately and are also the most computationally heavy zone. The linear layers involve Convolution, Matrix Multiplication, BatchNorm, and Avgpool, while the non-linear layers include activation functions such as ReLU, truncation, Maxpool, and Softmax. To achieve good performance on realistic inference, it is essential to design efficient sub-protocols for each function. The idea of pushing expensive cryptographic computation into an input-independent setup phase has seen tremendous success in the setting of multiparty computation based neural network training and inference [13], [17], [41], [47]. Our scheme achieves the same goal for private neural network inference.

D. Cryptographic Primitives

1) Secret Sharing: In this work, we use two variants of secret sharing. Both variants are performed over the arithmetic world $\mathbb{Z}_{2^\ell}$ and the boolean world $\mathbb{Z}_2$. As is known, boolean sharing is efficient for boolean circuits consisting of XOR and AND gates, which are commonly used to construct non-linear operations. Although arithmetic sharing is beneficial to fast linear computations like addition and multiplication. To achieve better efficiency, we use a mix of two types of sharing and will describe the conversion between them when necessary. In particular, all operations on the Boolean ring are regarded as specific cases of the arithmetic world when $\ell = 1$, which can be achieved by substituting addition/subtraction operations for XORs ($\oplus$) and multiplication for ANDs ($\otimes$).

- $[\cdot]$-sharing: This is a typical 2-out-of-2 additive secret-sharing primitive over ring $\mathbb{Z}_{2^\ell}$. The value $\text{v} \in \mathbb{Z}_{2^\ell}$ is $[\cdot]$-shared between two parties P_0 and P_1, if the party P_i for $i \in \{0, 1\}$ holds the values $[\text{v}]_i \in \mathbb{Z}_{2^\ell}$ such that $\text{v} = [\text{v}]_0 + [\text{v}]_1$.
- $\langle\cdot\rangle$-sharing: The value $\text{v} \in \mathbb{Z}_{2^\ell}$ is said to be $\langle\cdot\rangle$-shared between two parties P_0 and P_1, if the party P_i for $i \in \{0, 1\}$ holds the values $(\Delta_\text{v}, [\delta_\text{v}]_i) \in \mathbb{Z}_{2^\ell}^2$, and there exist $\delta_\text{v} = [\delta_\text{v}]_0 + [\delta_\text{v}]_1$ and $\Delta_\text{v} = \text{v} + \delta_\text{v}$.

To enable P_i for $i \in \{0, 1\}$ to $\langle\cdot\rangle$-share his secret $\text{v} \in \mathbb{Z}_{2^\ell}$ with P_{1-i}, the protocol $\Pi_{\text{Sh}}(P_i, \text{v})$ is performed as follows: during the setup phase, P_i samples a random ring element $[\delta_\text{v}]_i \in \mathbb{Z}_{2^\ell}$ and both parties together sample $[\delta_\text{v}]_{1-i} \in \mathbb{Z}_{2^\ell}$. With the fact that $\delta_\text{v} = [\delta_\text{v}]_0 + [\delta_\text{v}]_1$, P_i calculates $\Delta_\text{v} = \text{v} + \delta_\text{v}$ and sends

TABLE II
RUNTIME OF NON-LINEAR FUNCTIONS

Block	Method	LAN (ms)		WAN (ms)	
		Setup	Online	Setup	Online
ReLU	CrypTFlow2	-	1.01×10^3	-	1.46×10^4
	Cheetah	-	411.54	-	949.82
	ABY2.0	4.36×10^3	28.11	5.06×10^4	108.49
	Ours	226.49	24.71	276.82	108.83
Trun	CrypTFlow2	-	3.14×10^3	-	4.53×10^4
	Cheetah	-	473.99	-	1.01×10^4
	Ours	155.71	3.57	190.32	352.51
Softmax	GC	-	6.22×10^4	-	1.35×10^5
	Ours	435.16	134.46	1.21×10^3	1.31×10^4

TABLE III
END-TO-END EVALUATION

NN	Method	LAN (s)		WAN (s)	
		Setup	Online	Setup	Online
SqNet	CrypTFlow2	-	145.11	-	1.65×10^3
	Cheetah	-	37.58	-	143.83
	ABY2.0	265.38	1.32	1.57×10^3	4.69
	Ours	41.05	1.63	138.59	5.37
RN50	CrypTFlow2	-	820.59	-	6.97×10^3
	Cheetah	-	145.93	-	735.97
	ABY2.0	687.19	5.11	7.14×10^3	21.38
	Ours	135.13	5.42	268.04	25.42
DNet	CrypTFlow2	-	306.65	-	8.44×10^3
	Cheetah	-	154.19	-	745.85
	ABY2.0	2.02×10^3	6.78	1.27×10^4	25.68
	Ours	104.47	6.93	228.14	32.47

SqNet=SqueezeNet; RN50=ResNet50; DNet=DenseNet121

it to P_{1-i} in the online phase. We let $\Pi_{\text{Rec}}(P_i, \mathrm{v})$ denote the reconstruction protocol, in which P_i receives the missing $[\delta_\mathrm{v}]_{1-i}$ from P_{1-i}, and then locally reconstructs the secret v.

2) Oblivious Transfer: Oblivious Transfer (OT) [37] is a fundamental cryptographic protocol, especially for secure two-party computation. The basic idea of a general 1-out-of-n OT (i.e., $\binom{n}{1}$-OT$_\ell$) is that one party as sender inputs n ℓ-bit messages $m_0, \ldots, m_{n-1}$ and gets nothing, while the other party (ie, receiver) obtains m_b after submitting a choice b, where $b \in \{0, \ldots, n-1\}$.

When the messages to be transferred are random or correlated, such a random OT protocol (ROT) or correlated OT (COT) can be implemented more efficiently [2], [58]. Recent optimizations to construct Vector Oblivious Linear Evaluation (VOLE) correlations can be used to generate a bulk of random COTs with a low-communication setup phase followed by local ("silent") online computation [7], [14], [58].

In this paper, we utilize the aforementioned functionality in a black-box manner as the main ingredient of our setup phase to generate multiplication triples in batches, which incurs a minimal amortized communication cost, since both inputs and outputs are random values. For example, Ferret [58] takes 788 ms to compute 10^7 random cOTs with a network bandwidth of 50Mbps (see Tables II and III). Then we convert it to chosen-input OT for the efficient implementation of online truncation computations. According to [33], [58], the amortized communication costs for $\binom{n}{1}$-OT$_\ell$, $\binom{2}{1}$-OT$_\ell$, and $\binom{2}{1}$-COT$_\ell$ are $n\ell + log_2 n$ bits, $2\ell + 1$ bits and $\ell + 1$ bits, respectively.

3) Multiplication With $[\cdot]$-Sharing: Gilboa's multiplication technique [23] (also called protocol Π_{GM} in the following) is commonly used to calculate the product of two $[\cdot]$-shared values. Observing that $\mathrm{y} = \mathrm{a} \cdot \mathrm{b} = ([\mathrm{a}]_0 + [\mathrm{a}]_1) \cdot ([\mathrm{b}]_0 + [\mathrm{b}]_1) = [\mathrm{a}]_0[\mathrm{b}]_0 + [\mathrm{a}]_1[\mathrm{b}]_1 + [\mathrm{a}]_0[\mathrm{b}]_1 + [\mathrm{a}]_1[\mathrm{b}]_0$, the first two terms $[\mathrm{a}]_0[\mathrm{b}]_0$ and $[\mathrm{a}]_1[\mathrm{b}]_1$ can be locally computed without interaction. Here, we provide Gilboa's multiplication procedure instantiated with the COT technique for evaluating the mixed terms $[\mathrm{a}]_0[\mathrm{b}]_1$ and $[\mathrm{a}]_1[\mathrm{b}]_0$, which is also applied in [18]. We take $[\mathrm{a}]_0[\mathrm{b}]_1$ as an example: both parties engage in ℓ COTs on ℓ-bit strings, that is, COT $^{\ell}_{\ell}$. In detail, P_0 is assumed as sender with the input $f(r_j) = [\mathrm{a}]_0 \cdot 2^j + r_j$ to j th COT and gets $(r_j, [\mathrm{a}]_0 \cdot 2^j + r_j)$, where $r_j \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. P_1 is the receiver taking as input the choice bit $[\mathrm{b}_j]_1$, where b_j refers to the jth bit of b. The execution of COT ensures that the receiver P_1 receives $[\mathrm{b}_j]_1 \cdot \mathrm{a}_0 \cdot 2^j + r_j$. That is followed by the step where party P_0 sets $[[\mathrm{a}]_0[\mathrm{b}]_1]_0 = -\sum_{j=1}^{\ell} r_j$ and party P_1 sets $[[\mathrm{a}]_0[\mathrm{b}]_1]_1 = \sum_{j=1}^{\ell}([\mathrm{b}_j]_1 \cdot \mathrm{a}_0 \cdot 2^j + r_j)$. Similarly, both parties compute $[[\mathrm{a}]_1[\mathrm{b}]_0]$ using another instance of COT$^{\ell}_{\ell}$. Finally, P_i sets $[\mathrm{y}]_i = [\mathrm{a}]_i[\mathrm{b}]_i + [[\mathrm{a}]_1[\mathrm{b}]_0]_i + [[\mathrm{a}]_0[\mathrm{b}]_1]_i$. The above procedure performs two instances of COT$^{\ell}_{\ell}$, resulting in the communication of $2\ell(\ell+1)$ bits.

4) Multiplication With $\langle\cdot\rangle$-Sharing: To facilitate understanding, we take the multiplication of two $\langle\cdot\rangle$-shared values a and b as an example. Let $\mathrm{y} = \mathrm{a} \cdot \mathrm{b}$, we have:

$$\mathrm{y} = (\Delta_\mathrm{a} - \delta_\mathrm{a}) \cdot (\Delta_\mathrm{b} - \delta_\mathrm{b}) = \Delta_\mathrm{a} \cdot \Delta_\mathrm{b} - \Delta_\mathrm{a} \cdot \delta_\mathrm{b} - \delta_\mathrm{a} \cdot \Delta_\mathrm{b} + \delta_\mathrm{a} \cdot \delta_\mathrm{b}$$

During the setup phase, both parties prepare respective random values $[\delta_\mathrm{y}]_i$ for $i \in \{0, 1\}$. Instead of using protocol Π_{GM}, parties can calculate the $[\cdot]$-shares of $\delta_{\mathrm{ab}} = \delta_\mathrm{a} \cdot \delta_\mathrm{b}$ by drawing two multiplication triples from the triples pool so that parties hold the $[\cdot]$-shares of $[\delta_\mathrm{a}]_0 \cdot [\delta_\mathrm{b}]_1$ and $[\delta_\mathrm{a}]_1 \cdot [\delta_\mathrm{b}]_0$. It can be easily achieved since these $[\delta_\mathrm{a}]_i$ and $[\delta_\mathrm{b}]_i$ for $i \in \{0, 1\}$ are randomly sampled. The parties then locally compute $[\delta_{\mathrm{ab}}]_i = [\delta_\mathrm{a}]_i[\delta_\mathrm{b}]_i + [[\delta_\mathrm{a}]_0[\delta_\mathrm{b}]_1]_i + [[\delta_\mathrm{a}]_1[\delta_\mathrm{b}]_0]_i$. By the way, P_i for $i \in \{0, 1\}$ can learn $[\mathrm{y}]_i$ by locally computing: $[\mathrm{y}]_i = i \cdot \Delta_\mathrm{a}\Delta_\mathrm{b} - \Delta_\mathrm{a}[\delta_\mathrm{b}]_i - [\delta_\mathrm{a}]_i\Delta_\mathrm{b} + [\delta_{\mathrm{ab}}]_i$. Furthermore, the protocol $\Pi_{\text{Mult}}(\mathrm{a}, \mathrm{b})$ gives the result in the form of $\langle\cdot\rangle$ by an additional interaction to reconstruct Δ_y in the online phase, where $[\Delta_\mathrm{y}]_i = [\mathrm{y}]_i + [\delta_\mathrm{y}]_i$ can be computed locally. It can be easily observed that the online phase only requires one communication round with the cost of two ring elements to exchange the missing $[\Delta_\mathrm{y}]_i$. In this way, these techniques can also be extended to multi-input multiplication without inflating the online communication, which remains the same online overhead with the multiplication of the two inputs. For ease of exposition, we provide the four-input multiplication protocol (also called the four-input AND gate in the Boolean world) in Protocol A.1 of Appendices, available online.

III. SECURE COMPARISON

Essentially, the secure comparison protocol is an important building block for securely evaluating ReLU (ReLU$(x) = \max\{0, x\}$), which is one of the main components of a neural network. In this section, we provide new insights for secure

comparison and show the performance gained from the core of Antelope.

A. Bit Extraction

To securely compare two ℓ-bit values a and b (i.e., $\mathbf{1}\{a < b\}$), an equivalent method in the Fixed-point Arithmetic Representation is to extract the MSB of $v = a - b$. To do that, some prior works [33], [47], [49] recursively extract the sign of a shared ℓ-bit boolean value from the least significant bit (LSB) to MSB, by using a large number of AND gates and XOR gates. Since the XOR gates are evaluated "for free" in the circuit, the evaluation of AND gates dominates the computation and communication overhead. For a 64-bit input, both CrypTFlow2 [49] and Cheetah [33] require 56 AND gates, while ABY2.0 [47] offered a depth-optimized Parallel-prefix Adder that consists of 41 two-input, 27 three-input, and 47 four-input AND gates. The number of AND gates directly correlates with the latency of secure inference, incurring prohibitive setup and online runtime in these solutions.

1) Comparison in Antelope: Similar to previous work [21], [47], we translate the comparison as a sign extraction computation. The bit extraction protocol in Antelope is built on two new insights that work together to reduce the number of AND gates in the circuit. Along with our design, for a 64-bit input, only 21 four-input AND gates are required in total, achieving a significant reduction in the online communication and computation complexity.

Intuition: Our first insight comes from Astra [11]: in the two's complement representation when two non-zero values are multiplied, the sign of the product is equivalent to XOR the signs of the two multipliers. Based on this, it holds: $\mathrm{MSB}(v \cdot r_{(\times)}) = \mathrm{MSB}(v) \oplus \mathrm{MSB}(r_{(\times)})$, where $r_{(\times)}$ is a non-zero multiplicative mask. So the MSB of a non-zero value can be easily derived as:

$$\mathrm{MSB}(v) = \mathrm{MSB}(v \cdot r_{(\times)}) \oplus \mathrm{MSB}(r_{(\times)}) \tag{1}$$

Chaudhari et al. [11] and Wei et al. [57] developed 3PC protocols for secure bit extraction based on the above observation. However, their methods reveal the product $v \cdot r_{(\times)}$ to a semi-honest third party to extract its MSB, thus overlooking certain types of data leakage: (1) Data leakage at $v = 0$. Given that $r_{(\times)}$ is a non-zero random value, the third party can deduce that $x = 0$ whenever the product is zero. (2) Parity leakage. If the product is odd, it inadvertently reveals that both v and $r_{(\times)}$ are odd. Moreover, these protocols introduce unique challenges in a 2PC setting. The multiplicative mask $r_{(\times)}$ and product $v \cdot r_{(\times)}$ must be known to different parties, providing a straightforward method to determine the corresponding MSB. Otherwise, extracting the MSB of $r_{(\times)}$ or $v \cdot r_{(\times)}$ has the same complexity as extracting the MSB of v.

To address the data leakage issue at $v = 0$, our second insight is that the MSB of zero is equal to that of positive values according to the definition of Fixed-point Arithmetic Representation. Therefore, when v is zero, we replace it with an additive mask $r_{(+)}$ randomly sampled from $\mathbb{Z}_{2^\ell} \cap (0, 2^{\ell-1} - 1]$. This substitution is represented as follows: $y = u^A \cdot r_{(+)} + v$ where u records the result of $\mathbf{1}\{v = 0\}$. For the parity leakage issue, our third insight is that the last significant bit (LSB) of the product is adequate for determining its parity. Truncating the LSB prior to reconstruction effectively mitigates the data distribution leakage. As analyzed in [33], [43], [46], local truncation [41], [43], [47] may cause a sign-flipping error with a probability of $\frac{1}{2^\ell}$. Our experimental results indicate that such an error minimally impacts accuracy due to the substantial size of the ring.

At a high level, the first step of Antelope's Bit Extraction protocol is to learn if $v = 0$ or not. Subsequently, both parties employ additive masking for zero values and then multiplicative masking for all values. Finally, they perform local truncation and reveal the product.

2) Protocol for Equality-to-Zero: Given the secret shared value v, the Equality-to-Zero protocol Π^B_{EQ0} is used to check if the input v is 0 or not. We start with the observation that $v \stackrel{?}{=} 0$ (described as $\Delta_v - [\delta_v]_0 - [\delta_v]_1 \stackrel{?}{=} 0$ using our secret sharing primitive), so it equals to check if $\Delta_v - [\delta_v]_0 = [\delta_v]_1$ or not. Specifically, both parties first generate $\langle [\delta_v]'_1 \rangle^B$ and $\langle [\delta_v]_1 \rangle^B$ by executing Π^B_{Sh}, where $[\delta_v]'_1 = \Delta_v - [\delta_v]_0$. Then, parties locally set:

$$z^B = NOT(\langle [\delta_v]'_1 \rangle^B \oplus \langle [\delta_v]_1 \rangle^B) \tag{2}$$

If the two values are equal, all bits of z^B should be 1. With a quad-tree structure, both parties get the output by utilizing four-input AND gates as follows: $u^B = AND_{j=0}^{j=\ell-1}(z_j)$, where z_j is the jth bit of z. We formally describe our protocol for Equality-to-Zero in Protocol A,2 in Appendices, available online.

3) Protocol for Bit Extraction: Given the secret shared value v, the Bit Extraction protocol Π_{BitExt} proceeds as follows: during the online phase, both parties first execute an instance of protocol $\Pi^B_{\mathrm{EQ0}}(v)$ to learn $\langle u \rangle^B$. Then, P_i for $i \in \{0, 1\}$ locally computes $[y]_i = [y_1]_i + [y_2]_i$, where $y_1 = u^A \cdot r_{(+)} \cdot r_{(\times)}$ and $y_2 = v \cdot r_{(\times)}$. We have the following observation: given a secret-shared bit u, there exists:

$$u^A = (\Delta^B_u \oplus \delta^B_u)^A = \Delta^a_u + \delta^A_u - 2\Delta^a_u \delta^A_u \tag{3}$$

where $[u]^a_i$ for $i \in \{0, 1\}$ denotes the value over the arithmetic ring $\mathbb{Z}_{2^\ell}$ corresponding to $[u]^B_i$ over $\mathbb{Z}_2$. Similarly, we have: $\delta^A_u = [\delta_u]^a_0 + [\delta_u]^a_1 - 2[\delta_u]^a_0[\delta_u]^a_1$. Along with this, P_i for $i \in \{0, 1\}$ computes the follows:

$$\begin{aligned}[y_1]_i = {} & i \cdot \Delta^a_u \Delta_r - \Delta^a_u [\delta_r]_i + \Delta_r (1 - 2\Delta^a_u)[\delta_u]^A_i \\ & - (1 - 2\Delta^a_u)[\delta^A_u \delta_r]_i \end{aligned} \tag{4}$$

After that, P_0 sends $[y]_0 = [y_1]_0 + [y_2]_0$ to P_1 so that P_1 can reconstruct y and get MSB(y). In the last communication round, P_1 $\langle \cdot \rangle^B$-shares MSB(y). Finally, both parties get the boolean sharing of the MSB(v) by locally computing $\langle \mathrm{MSB}(v) \rangle^B = \langle \mathrm{MSB}(y) \rangle^B \oplus \langle \mathrm{MSB}(r_{(\times)}) \rangle^B$.

During the setup phase, the parties execute computations that are independent of the input v, including the preparation of random values (steps 1 and 2) and the multiplication of random values (step 3). To enhance efficiency, we make an optimization: instead of using protocol Π_{GM} for all random values, we distinguish independent random values from non-independent ones and implement their multiplication with more

Protocol III.1: $\langle \text{MSB(v)} \rangle^B \leftarrow \Pi_{\text{BitExt}}(\text{v})$.

Setup:

1. P_0 samples a random $r_{(\times)} \xleftarrow{\$} \mathbb{Z}_{2^\ell} \setminus \{0\}$ and invokes protocol Π_{Sh} to get $\langle r_{(\times)} \rangle$ and $\langle \text{MSB}(r_{(\times)}) \rangle^B$.
2. P_i for $i \in \{0,1\}$ samples $[r_{(\times)}]_i \xleftarrow{\$} \mathbb{Z}_{2^\ell} \setminus \{0\}$ and generates $\langle r_{(+)} \rangle$ by calling Π_{Sh}.
3. Parties invoke protocol Π_{GM} to generate $[\delta_u^A \delta_{r_{(\times)}}]$, $[\delta_u]^A$ and $[\delta_{\text{v}r_{(\times)}}]$, where $r = r_{(+)} r_{(\times)}$. Note that $[\delta_r]$ can be achieved with one beaver triple.

Online:

1. Parties learn $\langle u \rangle^B$ by calling protocol $\Pi_{\text{EQ0}}(\text{v})$.
2. P_i for $i \in \{0,1\}$ locally computes $[\text{y}_1]_i = i \cdot \Delta_u^a \Delta_r - \Delta_u^a [\delta_r]_i + \Delta_r (1 - 2\Delta_u^a)[\delta_u]_i^A - (1 - 2\Delta_u^a)[\delta_u^A \delta_r]_i$ and $[y_2]_i = i \cdot \Delta_{\text{v}}(r_{(\times)}) - \Delta_{\text{v}} [\delta_{r_{(\times)}}]_i - \Delta_{r_{(\times)}} [\delta_{\text{v}}]_i + [\delta_{\text{v}r_{(\times)}}]_i$.
3. P_0 computes $[\text{y}^T]_0 = \lfloor ([\text{y}_1]_0 + [\text{y}_2]_0)/2 \rfloor$ and sends it to P_1.
4. P_1 sets $[\text{y}^T]_1 = 2^\ell - \lfloor 2^\ell - ([\text{y}^T]_0 + [\text{y}^T]_1)/2 \rfloor \mod 2^\ell$ and gets $\text{y}^T = [\text{y}^T]_0 + [\text{y}^T]_1$, and $\text{MSB}(\text{y}^T)$.
5. P_1 calls protocol $\Pi_{\text{Sh}}^B(P_1, \text{MSB}(\text{y}^T))$ to get $\langle \text{MSB}(\text{y}^T) \rangle^B$.
6. P_i for $i \in \{0,1\}$ locally sets $\langle \text{MSB(v)} \rangle^B = \langle \text{MSB}(\text{y}^T) \rangle^B \oplus \langle \text{MSB}(r_{(\times)}) \rangle^B$.

efficient VOLE [58]. Similar to prior works [33], [43], [49], our system allows the client to learn the model architecture such as the type of layers. This makes it possible to prepare materials for secure computation such as multiplication triples in a data-independent phase. For instance, to compute δ_r in (4), we first decompose it into $\delta_r = [\delta_{r_{(+)}}]_0 \cdot \delta_{r_{(\times)}} + [\delta_{r_{(+)}}]_1 \cdot \delta_{r_{(\times)}}$. Subsequently, we replace $[\delta_{r_{(+)}}]_1 \cdot \delta_{r_{(\times)}}$ with a triple. Given a triple: $w_0 + w_1 = v \cdot \Theta$, P_0 sets $\delta_{r_{(\times)}} = \Theta$ and computes $[\delta_r]_0 = [\delta_{r_{(+)}}]_0 \cdot \delta_{r_{(\times)}} + w_0$, while P_1 sets $[\delta_{r_{(+)}}]_1 = v$ and computes $[\delta_r]_1 = w_1$. More detailed descriptions are provided in Protocol III.1.

Remark III.1: Our comparison protocol requires both parties to evaluate the designed Equality-to-Zero circuit, rather than the Adder circuit. Intuitively, the evaluation of Equality-to-Zero does not depend on the bit-wise coupling relationships in the Adder circuit such as additive carry. That makes parallel computation (i.e., multi-input AND gate protocol) suitable to be exploited for efficiently evaluating the Equality-to-Zero circuit. Despite the significant reduction in the number of online communication rounds, an N-input AND gate requires at least a communication of $4(2^N - \sum_{j=1}^{2} \binom{N}{j} - 1)$ bits in the setup phase. For example, using a 64-input AND gate for the Equality-to-Zero circuit with 64-bit inputs, $u^B = AND_{j=0}^{j=63}(\text{z}_j)$ can be achieved with the online communication of 2 bits in one communication round, but it requires up to 64EB setup overhead, which would be much too large to materialize in practice. Thus, we consider $N = 4$ for a balance between the setup and online communication overhead.

Theorem III.1: (Communication of BitExt). The Bit Extraction protocol Π_{BitExt} requires the communication of $6\ell^2 + 9\ell + 44n + 1$ bits in the setup phase and $\log_4(\ell) + 2$ rounds and communication of $2\ell + 2n$ bits in the online phase, where κ is the cryptographic security parameter and n represents the number of four-input AND gates.

Proof: Please refer to Appendices for details, available online. □

B. Secure Max/Min

Given $\langle \text{a} \rangle$, $\langle \text{b} \rangle$, protocol $\Pi_{\text{Max}}(\text{a}, \text{b})$ enables the parties to compute the maximum between two secret values a and b in a privacy-preserving manner, which is one of the most commonly-used applications for secure comparison. For this, the key insight of the secure Max protocol is $\text{Max}(\text{a}, \text{b}) = \text{v}^A \cdot (\text{b} - \text{a}) + \text{a}$, where $\text{v} = \text{MSB}(\text{a} - \text{b})$. Although the initial approach requires transforming the bit into an arithmetic value due to Π_{BitExt}'s output format of $\langle \cdot \rangle^B$-shares, our analysis suggests that this step can be excluded to reduce communication overhead. As an alternative approach, the specific multiplication process is conducted using (3):

$$\begin{aligned} \text{y} = \text{v}^A(\text{b} - \text{a}) &= (\Delta_{\text{v}}^a + \delta_{\text{v}}^A(1 - 2\Delta_{\text{v}}^a))((\Delta_{\text{a}} - \Delta_{\text{b}}) - \delta_{\text{a}} + \delta_{\text{b}}) \\ &= \Delta_{\text{v}}^a(\Delta_{\text{a}} - \Delta_{\text{b}}) + \delta_{\text{v}}^A(1 - 2\Delta_{\text{v}}^a)(\Delta_{\text{a}} - \Delta_{\text{b}}) - \Delta_{\text{v}}^a(\delta_{\text{a}} - \delta_{\text{b}}) \\ &\quad - \delta_{\text{v}}^A(\delta_{\text{a}} - \delta_{\text{b}})(1 - 2\Delta_{\text{v}}^a) \end{aligned} \tag{5}$$

Similarly, we can compute the minimum as follows: $\text{Min}(\text{a}, \text{b}) = \text{v}^A \cdot (\text{a} - \text{b}) + \text{b}$.

IV. Secure Division

In this section, we present new optimizations for secure division protocols with different divisor settings by leveraging numerical analysis. Given a shared dividend $\langle \text{a} \rangle$, our protocols discuss two settings of divisor b in detail, which cover all the cases for division operations encountered in neural network inference.

A. Public Divisor

In the Fixed-point Arithmetic Representation, the multiplication of a and b with at least s bits in the fractional part results in an expansion to the fractional part by a factor of 2, i.e., $\text{y} = \text{a} \cdot \text{b}$ has at most $2s$ bits to represent the fractional part. It is essential to scale down the product to the s-bit precision, so the same scale s is maintained for all intermediate results. Since both parties always negotiate the way the data is represented before evaluating the neural network, truncation is a special case of the division with the public divisor. Some prior works [41], [43], [47] used the local truncation on additive shares of the product for the approximate truncation based on the result of [43]. Specifically, the two parties set $[\text{y}']_0 = \lfloor [\text{y}]_0 / 2^s \rfloor$ and $[\text{y}']_1 = 2^\ell - \lfloor 2^\ell - [\text{y}]_1 / 2^s \rfloor \mod 2^\ell$, respectively. Let $\text{y} \in [0, 2^{\ell_{\text{y}}}] \cup [2^\ell - 2^{\ell_{\text{y}}}, 2^\ell]$, where $\ell_{\text{y}} < \ell - 1$. However, the truncated product $\text{y}' = \lfloor \tilde{\text{y}} 2^s \rfloor + \dot{e} + \ddot{e}$ introduces two probability errors: a small error $|\dot{e}| \leq 1$ occurs with the probability of $\frac{2^{\ell-1} - |\text{y}|}{2^{\ell-1}}$ and an arbitrary error $|\ddot{e}| < 2^\ell$ with the probability of $\frac{|\text{y}|}{2^{\ell-1}}$ (See Appendix B in [43] for more details), available online. As demonstrated in [33], the local truncation will introduce at least one large error $\ddot{e}$ for the first

layer of ResNet50 with the probability of 78% using Delphi's parameters [41]. This somehow demonstrates that the local truncation is problematic for large neural network architectures or suffers from more communication and computation overhead incurred by mapping data to a larger field. To correct both of these errors, CrypTFlow2 [49] reduced the truncation to a secure division with a known power of two. By representing the division of ring element to signed integer division, their design implements faithful truncation but requires up to 15 communication rounds with the communication of 12 KB for each truncation after multiplication. Such a huge overhead and the limitation of local truncation motivate us to propose new solutions to achieve a better trade-off between correctness and efficiency. In the following, we introduce the idea that enables us to obtain better performance from both protocols.

1) Truncation in Antelope: As experimentally verified on real neural networks in [16], [33], [43], the small errors $\dot{e}$ have a negligible impact on the classification accuracy even if they occur with a high probability. With the observation that the truncation protocol can be lightweight in communication and computation overhead by removing the constraint of correcting small errors, we focus on how to avoid large errors from the faithful truncation. Cheetah [33] optimizes the truncation protocol based on the same observation. However, they simply attribute the large error to the overflow of the sum of additive shares and do not provide sound theoretical proof. In contrast to this, we evaluate the coefficient term for the large error from a comprehensive perspective of data representation in the ring and data overflow. Essentially, the large error in the context of the ring occurs primarily in the following two situations. The first case occurs when the product y that needs to be truncated is negative and both additive shares of y are positive, it holds $([\mathrm{y}]_0 \gg s) + ([\mathrm{y}]_1 \gg s) = (\mathrm{y} \gg s) - 2^{\ell-s} + c^A$, where c is from the error of the last bit due to truncation. The other case corresponds to that the product y is positive but both shares are negative. If this happens, there will become: $([\mathrm{y}]_0 \gg s) + ([\mathrm{y}]_1 \gg s) = (\mathrm{y} \gg s) + 2^{\ell-s} + c^A$. With that, our protocol requires one instance of comparison protocol for computing MSB(y) and a joint evaluation for the coefficient term of the large error $2^{\ell-s}$.

Proposition IV.1: Given a ring element y and its secret sharings $[\mathrm{y}]_i \in \mathbb{Z}_{2^\ell}$ for $i \in \{0,1\}$, let their unsigned representation be y_{int} and y_i, respectively. Let $c = \mathbf{1}\{\mathrm{y}_0^0 + \mathrm{y}_1^0 \geq 2^s\}$, where $\mathrm{y}_i^0 < 2^s$ is defined as $\mathrm{y}_i^0 = \mathrm{y}_i - \mathrm{y}_i^1 \cdot 2^s$. Then, we have:

$$(\mathrm{y} \gg s) + c^A = ([\mathrm{y}]_0 \gg s) + ([\mathrm{y}]_1 \gg s) + cor \cdot 2^{\ell-s}$$

where

$$cor = \begin{cases} -1 & \mathbf{1}\{\mathrm{y}_{int} \geq 2^{\ell-1}\} \wedge \mathbf{1}\{\mathrm{y}_0 < 2^{\ell-1}\} \wedge \mathbf{1}\{\mathrm{y}_1 < 2^{\ell-1}\} \\ 1 & \mathbf{1}\{\mathrm{y}_{int} < 2^{\ell-1}\} \wedge \mathbf{1}\{\mathrm{y}_0 \geq 2^{\ell-1}\} \wedge \mathbf{1}\{\mathrm{y}_1 \geq 2^{\ell-1}\} \\ 0 & \text{otherwise} \end{cases}$$

as defined in [49].

Proof: Please refer to Appendices for details, available online. □

2) Protocol for Truncation: Our protocol Π_{Trun} for truncation after multiplication works as follows: during the online phase, parties begin by learning the additive shares of $v = \mathrm{MSB}(\mathrm{y})$ from protocol Π_{BitExt}. In parallel, P_i for $i \in \{0,1\}$ locally truncates the additive shares of y and gets the MSB of the shares $u_i = \mathrm{MSB}([\mathrm{y}]_i)$. Next, parties jointly generate the additive shares of the coefficient term cor. More specifically, P_0 enumerates all the choices of $[v]_1^B$ and u_1, and generates a matrix $\mathbf{M}$ of dimension 2×2. Let $j = j_0 \| j_1 \in \mathbb{Z}_2^2$ be the set of all permutations of $[v]_1^B$ and u_1. It holds: $t_{[v]_1^B \| u_1} = ([v]_0^B \oplus [v]_1^B \oplus u_0) \wedge ([v]_0^B \oplus [v]_1^B \oplus u_1) = (\mathrm{MSB}(\mathrm{y}) \oplus \mathrm{MSB}([\mathrm{y}]_0)) \wedge (\mathrm{MSB}(\mathrm{y}) \oplus \mathrm{MSB}([\mathrm{y}]_1))$. Based on this, P_0 sets $\mathbf{M}_j = [cor]_1 = cor - [cor]_0$, where $[cor]_0$ is sampled independently and uniformly at random from $\mathbb{Z}_{2^\ell}$ at the setup phase. As the definition of the coefficient term cor, $cor = -1$ if $t_j \wedge \mathbf{1}\{[\mathrm{y}]_0 = 0\}$ and $cor = 1$ when $t_j \wedge \mathbf{1}\{[\mathrm{y}]_0 = 1\}$. Otherwise, $cor = 0$. From this setting, the elements of matrix $\mathbf{M}$ are all $-[cor]_0$ except with $-[cor]_0 \pm 1$ in coordinate j^*, where $t_{j^*} = 1$ implies that the first two cases of cor can be determined by $[\mathrm{y}]_0$. It is easy to get $[cor]_1 = \mathbf{M}_{[v]_1^B \| u_1}$ for P_1 with $\binom{4}{1}$-OT_ℓ protocol. The formal details are given in Protocol IV.1 for the truncation.

3) Optimizations: To enable P_0 to download a particular element with index j from a private matrix without revealing to P_1 which record is being requested, CrypTFlow2 exploits IKNP-style OT protocol [35]. For the matrix with the dimension of 2×2, IKNP-style OT requires the communication of $2\kappa + 4\ell$ bits within two online communication rounds. Recent optimizations to vector oblivious linear evaluation (VOLE) correlations can be used to generate multiple random OT correlations with a low-communication setup phase followed by a lightweight non-cryptographic online computation [7], [14], [58]. Such VOLE-style random COTs enjoy negligible amortized setup costs when a huge number of COTs are required [58]. When the above random OT correlations generation protocol with

Protocol IV.1: $\langle \mathrm{y}^T \rangle \leftarrow \Pi_{\text{Trun}}(\mathrm{y})$.

Online:

1. P_i for $i \in \{0,1\}$ obtains $[v]$ by executing part of protocol $\Pi_{\text{BitExt}}(\mathrm{y})$, where $v = \mathrm{MSB}(\mathrm{y})$.
2. P_i for $i \in \{0,1\}$ locally sets $[\mathrm{y}]_i = i \cdot \Delta_\mathrm{y} - [\delta_\mathrm{y}]_i$ and truncates $[\mathrm{y}]_i$ by performing $[\mathrm{y}]_i' = [\mathrm{y}]_i \gg s$. Party P_i sets $u_i = \mathrm{MSB}([\mathrm{y}]_i)$.
3. P_0 locally generates a matrix $\mathbf{M} \in \mathbb{Z}_{2^\ell}^{2\times 2}$. For each coordinate $j \in \{00, 01, 10, 11\}$ of matrix $\mathbf{M}$, P_0 computes $t_j = ([v]_0^B \oplus j_0 \oplus u_0) \wedge ([v]_0^B \oplus j_0 \oplus j_1)$, where $j = j_0 \| j_1$. If $t_j \wedge \mathbf{1}\{u_0 = 0\} = 1$, then $\mathbf{M}_j = -[cor]_0 - 1$. Else if $t_j \wedge \mathbf{1}\{u_0 = 1\} = 1$, then $\mathbf{M}_j = -[cor]_0 + 1$. Otherwise, $\mathbf{M}_j = -[cor]_0$.
4. P_1 obtain $[cor]_1 = \mathbf{M}_{[v]_1^B \| u_1}$ using VOLE-style $\binom{4}{1}$-OT_ℓ.
5. P_i for $i \in \{0,1\}$ locally computes $[\mathrm{y}^T]_i = [\mathrm{y}]_i' + [cor]_i \cdot 2^{\ell-s}$, and generates $\langle [\mathrm{y}^T]_i \rangle$ by calling protocol Π_{Sh}.
6. P_i for $i \in \{0,1\}$ locally computes $\langle \mathrm{y}^T \rangle = \langle [\mathrm{y}^T]_0 \rangle + \langle [\mathrm{y}^T]_1 \rangle$.

uniform choice bits is extended to generate COTs with chosen choice bits, we use the pre-computed reduction technique [4], at the price of only one bit of communication cost per random OT correlation. In this case, all messages that need to be sent are also chosen by one party (i.e., the sender in OT), rather than random ones. To achieve this, we use tweakable correlation robust hash functions (CRHFs) [26], [35] to transform the random COTs into general chosen-input OT protocol. As shown in prior work [26], the tweakable CRHF can be efficiently constructed by a random permutation such as AES-128 with a fixed key. With all optimizations built upon the VOLE-style OT protocol in the semi-honest setting proposed by Yang et al. [58], $\binom{4}{1}$-OT$_\ell$ has communication $4\ell + 2$ bits [33]. To put it into our truncation protocol, the online communication of only 493 bits per truncation is required for 64-bit ring elements, which enables 35.8× improvement in comparison with CrypTFlow2.

Theorem IV.1: (Communication of Truncation after Multiplication). The protocol Π_{Trun} requires the communication of $6\ell^2 + 9\ell + 44n$ bits in the setup phase, and $\log_4(\ell) + 4$ online rounds and communication of $7\ell + 2n + 1$ bits in the online phase.

Proof: Please refer to Appendices for details, available online. □

B. Secret-Shared Divisor

Let $\mathcal{F}_{\text{SS-Div}}$ be the functionality for the division that takes arithmetic sharing $\langle \mathrm{a} \rangle$, $\langle \mathrm{b} \rangle$ as inputs and returns the arithmetic sharing of a/b as output. Our protocol $\Pi_{\text{SS-Div}}$ builds on the basis of Goldschmidt's method [24] and is optimized with some subprotocols including the above secure comparison protocol.

Intuition: A crucial component of our numerical method is to convert the absolute value of divisor b to the form of $2^{\alpha_{|\mathrm{b}|}+1} \cdot x_{|\mathrm{b}|}$, where $2^{\alpha_{|\mathrm{b}|}} \leq |\mathrm{b}| < 2^{\alpha_{|\mathrm{b}|}+1}$, and $\alpha_{|\mathrm{b}|}$ is defined as the bounding power of absolute value $|\mathrm{b}|$. Note that $x_{|\mathrm{b}|}$ is within the range of [0.5,1) in the above expression. Then, $1/x$ for any $x \in [0.5, 1)$ can be approximatively computed by utilizing the designated choices of initializations given in [1], [9]. We use the suggested initial approximation $\omega_0 = 2.9142 - 2x$ with relative error $\varrho_0 < 0.08578$. For higher order approximations, it can be achieved by multiplying the initial approximate result by $(1 + \varepsilon_j)$ with a higher degree j, where $\varepsilon_0 = 1 - |\mathrm{b}| \cdot \omega_0$ and $\varepsilon_j = \varepsilon_{j-1}^2$ for $j > 0$. Besides, we observe that division in the neural network is always greater than 0, i.e., $|\mathrm{b}| = \mathrm{b}$. For example, Softmax, often as the last activation function of a neural network, its denominator is in the form of e^x. With that, a/b in neural network inference can be computed as follows:

$$\text{AppDiv}^{deg}(\mathrm{a}/\mathrm{b}) = \mathrm{a} \cdot \omega_0 \cdot \prod_{j=0}^{deg}(1 + \varepsilon_0^{2^j}) \tag{6}$$

where deg represents the approximation degree. With this, the related error with approximation degree deg is $\varrho_{deg} = \varrho_0^{2^{deg}}$. Given the initial approximation error which occupies 4 bits, the degree of approximation satisfies $deg = \lceil \log \frac{\ell}{4} \rceil$ to achieve ℓ-bits accuracy.

Protocol IV.2: $\langle \mathrm{a}/\mathrm{b} \rangle \leftarrow \Pi_{\text{SS\_Div}}(\mathrm{a}, \mathrm{b})$.

Setup:

1. Parties call protocol Π_{GM} to get $[\delta_b \delta_{\omega_0}]$.
2. P_0 samples a random value c_0' and invokes protocol $\Pi_{\text{Sh}}(P_i, c_0')$ to generate $\langle c_0' \rangle$.

Online:

1. Parties run protocol $\Pi_{\text{MSNB}}(\mathrm{b})$ to get the shares of encoding of MSNB(b): $\langle \text{One-hot}_{\alpha_\mathrm{b}} \rangle$.
2. Parties locally performs $\langle 2^{\ell-\alpha_b-1} \rangle = \sum_{j=1}^{\ell} \langle \text{One-hot}_{\alpha_\mathrm{b}}[j] \rangle * 2^{\ell-j-1}$.
3. Parties call protocol $\Pi_{\text{Mult}}(\mathrm{b}, 2^{\ell-\alpha_b-1})$ to get $\langle x_b \rangle$.
4. P_i for $i \in \{0, 1\}$ locally computes $\langle \omega_0 \rangle_i = 2.9142 - 2\langle x_b \rangle_i$ and $[\varepsilon_0]_i = i \cdot (1 - \Delta_b \Delta_{\omega_0}) + \Delta_b [\delta_{\omega_0}]_i + \Delta_{\omega_0} [\delta_b]_i - [\delta_b \delta_{\omega_0}]_i$.
5. Parties together evaluate the polynomial $Q(\varepsilon_0) = \prod_{j=0}^{deg}(1 + \varepsilon_0^{2^j})$ by running protocol Π_{SPE} so that P_1 obtains $[Q(\varepsilon_0)]_1$, where $Q(\varepsilon_0) = \prod_{j=0}^{deg}(1 + \varepsilon_0^{2^j})$ and $[Q(\varepsilon_0)]_0 = -c_0'$.
6. P_1 executes protocol $\Pi_{\text{Sh}}(P_1, [Q(\varepsilon_0)]_1)$.
7. Parties locally set $\langle Q(\varepsilon_0) \rangle = \langle [Q(\varepsilon_0)]_0 \rangle + \langle [Q(\varepsilon_0)]_1 \rangle$.
8. Parties call protocol Π_{Mult} to get $\langle \mathrm{y} \rangle$, where $\mathrm{y} = \mathrm{a} \cdot \omega_0$.
9. Parties call protocol $\Pi_{\text{Mult}}(\mathrm{y}, Q(\varepsilon_0))$ to get $\langle \mathrm{a}/\mathrm{b} \rangle$.

1) Secret-Shared Division Protocol: Protocol $\Pi_{\text{SS-Div}}$ is designed to evaluate the division with the secret-shared numerator a and denominator b during the neural network inference. The details are described in Protocol IV.2.

According to (6), the initial step requires a range reduction to map arbitrary b to $x_\mathrm{b} \in [0.5, 1)$ in a privacy-preserving way. This step involves calculating the most significant non-zero bit (MSNB) of b (Step 1), which is defined as $\text{MSNB}(\mathrm{b}) = \alpha_\mathrm{b} \in [\ell]$ if $\mathrm{b}_{\alpha_\mathrm{b}} = 1$ and all $\mathrm{b}_j = 0$ for all $j > \alpha_\mathrm{b}$. We consider the functionality $\mathcal{F}_{\text{MSNB}}$ that produces the shares of one-hot encoding of MSNB(b) and use the protocol for the same in [48]. Subsequently, x_b can be efficiently computed using dot products between this one-hot vector and publicly accessible vectors, as described in Steps 2 and 3.

Following that, Step 4 prepares the shares of ε_0. For the efficient evaluation of $Q(\varepsilon_0) = \prod_{j=0}^{\text{deg}}(1 + \varepsilon_0^{2^j})$, Step 5 employs the secure polynomial evaluation protocol Π_{SPE}, which offers a higher degree of parallelization compared to the iterative use of the multiplication protocol Π_{Mult}. As described in Section IV-B2, Π_{SPE} generates shares $[Q(\varepsilon_0)]$ for both parties within a single round of communication. These shares are then converted to $\langle Q(\varepsilon_0) \rangle$ through the protocol $\Pi_{\text{Sh}}(P_i, [Q(\varepsilon_0)]_i)$. Importantly, the share $[Q(\varepsilon_0)]_0 = -c_0'$ can be sampled independently and randomly, allowing its generation during the setup phase.

Finally, two successive invocations of Π_{Mult} for $\mathrm{y} = \mathrm{a} \cdot \omega_0$ and $\mathrm{y} \cdot Q(\varepsilon_0)$ are used to complete the secure evaluation of (6).

2) Optimizations: Secure Polynomial Evaluation (SPE) [44] enables one party (we assume P_1) with private input v to learn $Q(\mathrm{v})$ without learning anything more about the polynomial $Q(x) = \sum_j^{Deg} c_j x^j$ and without revealing the input to another party P_0 who knows all coefficients of the polynomial. Essentially, the evaluation of (6) can be viewed as a generalization of

SPE with degree $Deg = \sum_{j=0}^{deg} 2^j$. Let $\varepsilon_0 = [\varepsilon_0]_0 + [\varepsilon_0]_1$, the polynomial in (6) can be represented as follows:

$$Q(\varepsilon_0) = \prod_{j=0}^{deg}(1+\varepsilon_0^{2^j}) \rightarrow Q([\varepsilon_0]_1) = \sum_{j}^{Deg} c_j \cdot ([\varepsilon_0]_1)^j \quad (7)$$

where the coefficients c_j for $j \in [1, Deg]$ involving $[\varepsilon_0]_0$ are only known to P_0. To enforce the result to be secret-shared, we require that c_0 consists of a constant term $\sum_{j=0}^{Deg}([\varepsilon_0]_0)^j$ and a random value c_0' sampled by P_0. To evaluate the polynomial, we use the SPE algorithm introduced by [44] and optimized by [54], [58]. The core idea of these works is to reduce the evaluation of a degree Deg polynomial to Deg evaluations of linear polynomials of the forms $Q_t([\varepsilon_0]_1) = u_t \cdot [\varepsilon_0]_1 + v_t$ for $t \in [1, Deg]$ that can be executed in parallel, where P_0 holds the value (u_t, v_t) for $t \in [1, Deg]$, while P_1 has $[\varepsilon_0]_1$ and obtains $Q_t([\varepsilon_0]_1)$ for all $t \in [1, Deg]$. A natural approach to these Deg linear computations is Gilboa's multiplication protocol as discussed in Section II-D3. That inevitably requires the communication overhead of $2Deg\ell(\ell+1)$ bits. For efficiency gain in the online phase, we use Beaver's triple multiplication protocol [3] to evaluate Deg linear polynomials. Specifically, in the setup phase, parties use silent OT generator [7], [58] to generate Deg multiplication triples $z_t = [z_t]_0 + [z_t]_1 = x_t \cdot y_t$, where P_0 has x_t and obtains $[z_t]_0$ while P_1 holds y_t and receives $[z_t]_1$. In the online phase, parties share $e_t = u_t - x_t$ and $f_t = [\varepsilon_0]_1 - y_t$ with each other, respectively. Following that, P_0 computes $[Q_t([\varepsilon_0]_1)]_0 = e_t \cdot f_t + x_t \cdot f_t + [z_c]_0 + v_t$ and sends the results to P_1. Then, P_1 obtains $Q_t([\varepsilon_0]_1)$ by locally computing $[Q_t([\varepsilon_0]_1)]_1 = e_t \cdot y_t + [z_t]_1$ and adding it with $[Q_t([\varepsilon_0]_1)]_0$. Finally, P_0 sets the random item $-c_0'$ as the share of $Q(\varepsilon_0)$, i.e., $[Q(\varepsilon_0)]_0 = -c_0'$. P_1 computes the share of $Q(\varepsilon_0)$ as follows:

$$[Q(\varepsilon_0)]_1 = \sum_{t=1}^{Deg} Q_t([\varepsilon_0]_1) \cdot ([\varepsilon_0]_1)^{t-1} \quad (8)$$

As a result, the online phase of $Q(\varepsilon_0)$ results in two communication rounds with the communication cost of $3Deg \cdot \ell$ bits. Reader please refer to [54] for more details on the Secure Polynomial Evaluation algorithm used in this paper.

Theorem IV.2: (Communication of Secret-shared Division). Let $c = \ell | d$, the protocol $\Pi_{\text{SS-Div}}$ requires the communication of $8\ell^2 + 9\ell + (c-1)(7\,d^2 + 11d + 204) + d + 96$ bits in the setup phase, and $(c+1)(2\lambda + \ell \log \ell) + (3Deg + 6)\ell + 5dc + 16c - 4d - 10)$ online rounds and communication of $2\log_4 \ell + 3c + 5$ bits in the online phase.

Proof: Please refer to Appendices for details, available online. □

V. Antelope

Relying on the secure comparison protocol and the secure division protocols presented in the above two sections, our framework `Antelope` results in fast and secure inference.

A. Building Blocks for Inference

In the setting of secure inference, the first step is that the model owner (say P_0) and query user (i.e., P_1) generate the bulk of multiplication triples by running a silent OT generator [58] and execute the instance of protocol Π_{Sh} such that model parameters and users' query data are secret-shared between both parties. Following that, both parties stitch the protocols for arbitrary layers in a sequential fashion to obtain a secure computation protocol for any neural network architecture. Specifically, for linear computation on 2PC inference such as convolution and matrix multiplication, we use the efficient multiplication protocol proposed by ABY2.0 [47] and optimized by the silent OT generator. Along with our optimization for comparison and division protocols, we provide more efficient implementations for non-linear functions. Finally, the query users obtain the inference result by calling the reconstruction protocol. In the following, we present the details of specialized privacy-preserving non-linear building blocks for neural network inference.

ReLU: ReLU is one of the most popular nonlinear activation functions in neural networks. Accelerating the calculation of ReLU will greatly improve the efficiency of neural network inference. Given the arithmetic sharing $\langle \mathrm{v} \rangle$, the goal of the secure ReLU protocol is to compute the arithmetic sharing of the maximum value between 0 and v. In general, the protocol is a special case of the secure max protocol with 0 as one of the variables. Intuitively, secure ReLU protocol can be achieved by directly calling to protocol Π_{Max}.

Maxpool: Given the arithmetic sharing of v_j for $j \in [0, \mathrm{d}-1]$, protocol $\Pi_{\text{Maxpool}_\mathrm{d}}$ returns the arithmetic sharing of the maximum value among v_j for $j \in [0, \mathrm{d}-1]$ as follows: $\text{Maxpool}_\mathrm{d}(\mathrm{v}) = \max\{\mathrm{v}_0, \ldots, \mathrm{v}_{\mathrm{d}-1}\}$. Given $\langle \mathrm{a} \rangle$, $\langle \mathrm{b} \rangle$ and $\langle \mathrm{c} \rangle$, we let $\mathrm{u}_1^B = 1$ if and only if $\mathrm{a} < \mathrm{b}$, and define u_2^B and u_3 in the same way, that is, $\mathrm{u}_2^B = \mathbf{1}\{\mathrm{b} < \mathrm{c}\}$ and $\mathrm{u}_3^B = \mathbf{1}\{\mathrm{c} < \mathrm{a}\}$. To compute the maximum among $\langle \mathrm{a} \rangle$, $\langle \mathrm{b} \rangle$ and $\langle \mathrm{c} \rangle$, we can obtain the result by computing $\mathrm{y} = \bar{\mathrm{u}}_1^B \cdot \mathrm{u}_3^B \cdot \mathrm{a} + \mathrm{u}_1^B \cdot \bar{\mathrm{u}}_2^B \cdot \mathrm{b} + \mathrm{u}_1^B \cdot \mathrm{u}_2^B \cdot \mathrm{c}$, where $\bar{\mathrm{u}}^B = \mathrm{u}^B \oplus 1$. Observe that $\mathrm{u}^B = \Delta_\mathrm{u} + \delta_\mathrm{u}(1 - 2\Delta_\mathrm{u})$, we utilize the protocol Π_{Mult3} to learn $\mathrm{u}_i^B \cdot \mathrm{u}_j^B \cdot \mathrm{v}$ for $i, j \in \{1, 2, 3\}$. For the simplicity of exposition, the method is called Max3. Based on this, we arrange d values in each patch to be the leaf of a ternary tree. The ternary tree proceeds in a bottom-up fashion. Each block is composed of three values and evaluated by utilizing Max3. If the ternary tree cannot form a complete ternary tree, the Max protocol mentioned before is called to evaluate the last block when there are two values, or else the result is output directly when there is only one element.

Softmax: For the sake of numerical stability [25], the softmax function is commonly computed as: $\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i - \overline{x}}}{\sum_i e^{x_i - \overline{x}}}$, where $\overline{x} = \text{Maxpool}_{|\mathbf{x}|}(\mathbf{x})$. It is compatible with the conditions of application of the secure division protocol with the secret-shared divisor, i.e., the division $\sum_i e^{x_i - \overline{x}} > 0$. Before running protocol $\Pi_{\text{SS-Div}}$, parties first calculate $\overline{x}$ with protocol $\Pi_{\text{Maxpool}_{|\mathbf{x}|}}$. Then we use the Taylor series to approximate the exponentiation e^{-x}: $e^{-x} = (1 - \frac{x}{2^n})^{2^n}$. Suppose we use $s = 13$, then we set $n = 5$ to achieve an average error within 2^{-10}. Similar with (7), the approximation can be computed by running

one instance of protocol Π_{SPE} with the degree of 32. Following that, the server and client run an instance of protocol $\Pi_{\text{SS-Div}}$ for the result of Softmax.

B. Security Analysis

Theorem V.1: (Security of Antelope). Antelope provides a secure inference protocol to realize the ideal functionality of neural network inference with the service provider's model parameters X_S and the client's query data X_C.

Proof: Please refer to Appendices for details, available online. □

VI. Evaluation

In this section, we perform comprehensive experiments to evaluate our 2PC and secure inference protocols. Particularly, we show that Antelope's protocols for non-linear functions give better performance than the state-of-the-art implementations. We also validate the effectiveness of our truncation for large-scale models. Considering three ImageNet-scale neural network architectures, Antelope achieves more efficient secure inference than previous works (CrypTFlow2 [49], ABY2.0 [47], Cheetah [33]) in terms of runtime and communication overhead for the online phase.

A. Experiment Setup

All experiments were carried out on Cloud servers with the configuration of Intel(R) Xeon(R) E5-2680 v4 (2.4 GHz) and 48 GB RAM. We ran our benchmarks in two network settings: (1) LAN: the bandwidth is 580MBps and the echo latency is about 0.17 ms. (2) WAN: the bandwidth is 38MBps and the echo latency is about 72 ms. The secure multi-party cryptography operations were implemented in C++, where OT is built on top of the EMP toolkit [33], [58]. Each experiment was repeated five times, and the average results were displayed. The client and server executions used 4 threads each. In all our experiments, the data length was set to 64, i.e., $\mathbb{Z}_{2^{64}}$, and the last $s = 13$ bits represent the fractional part.

Following CrypTFlow2 [49], we evaluate the performance of Antelope on three neural network architectures over ImageNet [19]: SqueezeNet [34], ResNet50 [30] and DenseNet121 [32]. More details about these network architectures can be found in [49].

B. Microbenchmarks

We provide the microbenchmarks of Antelope's performance, compared with its closest competitors. *Improvement of Non-linear Functions:* Fig. 1 presents the improvement of our comparison protocol over ABY2.0 in terms of communication cost. The x-axis indicates the number of comparison protocols performed, ranging from 2^{10} (1K) to 2^{15} (32K). ABY2.0 introduces improved mixed protocols for a secure two-party computation framework by efficiently combining Arithmetic sharing, Boolean sharing, and Yao's sharing. Moreover, it proposes round efficient Parallel Prefix Adder constructions to

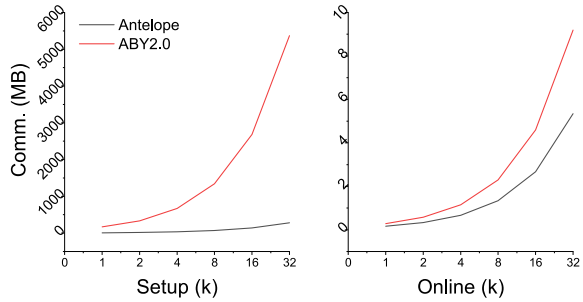


Fig. 1. Communication cost comparison of the comparison protocol between Antelope and ABY2.0.

reduce the online communication cost of secure comparison. In Antelope, we design a new Equality-to-Zero circuit for comparison operation. By decoupling the bit-wise rounding dependency in the Adder circuit and utilizing the silent OT generator, Antelope achieves a significant reduction in the communication overhead in both setup and online phases. In addition, neither ABY2.0 nor Antelope requires heavy cryptographic computations in the online phase. The communication rounds are amortized over a large number of ReLU operations in evaluating realistic neural networks. This implies that communication dominates the online runtime. From Fig. 1, we observe that as the number of evaluation comparison protocols increases, the benefits of Antelope become larger over ABY2.0, especially in the setup phase. The gains in communication overhead would map to the evaluation performance.

Next, we compare Antelope with the state-of-the-art works (CrypTFlow2, ABY2.0 and Cheetah) in large-scale inference. CrypTFlow2 has made considerable headway toward scalable secure neural network inference. For the first time, it demonstrates the ability to perform secure inference at the scale of ImageNet. Cheetah subsequently optimizes CrypTFlow2 from the following two aspects: new homomorphic encryption-based multiplication for evaluating linear layers without rotation operations and a more efficient OT technique for ReLU and truncation. Both CrypTFlow2 and Cheetah are designed to improve overall runtime. In other words, their protocols do not distinguish between the setup and online phases. So both methods do not exchange data in the setup phase. As we mentioned earlier, neither of them is practical enough for critical tasks with high real-time requirements. Antelope provides a feasible solution to well address such an issue in this scenario. ABY2.0 utilizes the offline-online paradigm, but uses GC for division operations. Table II shows the improvement of ReLU, truncation, and Softmax in our Antelope (corresponding to our comparison and division protocols in practical application) over three previous works, in both LAN and WAN settings. The runtime and communication are reported for 2^{16} instances of the protocols. The evaluation of ReLU using Antelope is up to $41.2\times$ and $135.1\times$ faster than CrypTFlow2, and $16.7\times$ and $8.7\times$ faster than Cheetah, in the two network conditions, respectively. Compared to ABY2.0, our evaluation reveals performance improvement in the setup phase by factors of 19.3 and

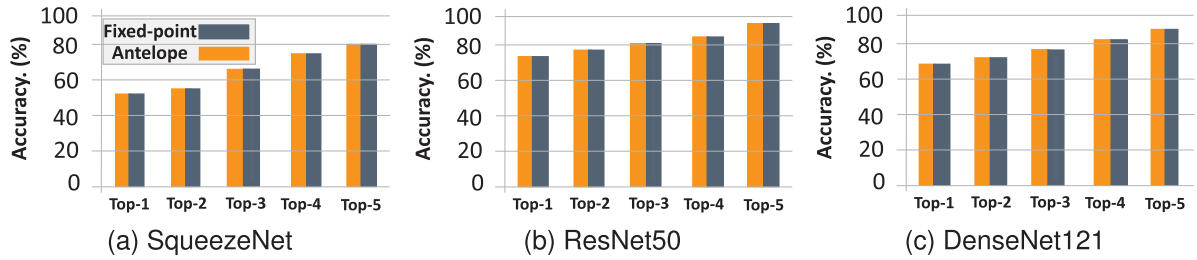


Fig. 2. The classification accuracy from top-1 to top-5.

183.1 in LAN and WAN, although maintaining little competitive online runtimes. We also perform the truncation protocol as it is necessary to maintain the accuracy of ImageNet-scale networks. As expected, our protocol shows its superiority in the online phase over CrypTFlow2 and Cheetah in two settings. Besides, we observe that our runtimes for the evaluation of Softmax are about $46.26\times$ lower than GC [18] in the LAN setting while giving slightly better performance in the WAN setting. These improvements come from our careful designs: decoupling the bit-wise carry dependency in the Bit Extraction protocol and novel optimizations for secure division protocols.

Fixed-point accuracy of our benchmarks: Here, we show the impact of our truncation protocol on classification accuracy. In Fig. 2, we present the accuracy of the non-private inference and the accuracy achieved by `Antelope` on three ImageNet-scale networks. The gray area in Fig. 2 represents the accuracy of the non-private inference achieved by the 64-bit fixed-point representation. We observed that the above experiments demonstrate that `Antelope` gives almost the same classification results with fixed-point computation, even in the context of ImageNet scale inference. This is mainly due to our truncation leading to the reliable implementation of fixed-point arithmetic. Besides, CrypTFlow2 demonstrated that the classification accuracy with fixed-point representation is the same as the float-point accuracy. This somehow demonstrates that `Antelope` is effective for ImageNet-scale inference.

C. End-to-End Evaluation

With all our protocols and optimizations, we demonstrate that `Antelope` is efficient and effective enough to perform large-scale secure inference. Table III compares the end-to-end inference latency of CrypTFlow2, Cheetah, ABY2.0, and `Antelope` over three models in different network settings. We observe that `Antelope` is about $22.25 \sim 26.92\times$ faster than Cheetah in the LAN setting, and $22.97 \sim 28.95\times$ faster in the WAN setting. Similar results can be obtained in the comparison with CrypTFlow2, demonstrating `Antelope`'s superior performance advantages. Although the online phase is slightly less efficient compared to ABY2.0 due to truncation operations, the performance improvement of the setup phase of `Antelope` provides a lot of energy savings. To sum up, despite the slightly greater intensity of offline computation, the improved online performance of `Antelope` delivers greater benefits for real-world applications.

VII. Related Work

A. Secure Neural Network Inference

To ease the privacy concerns in neural network inference, there are a large number of works with different technologies being proposed. These technologies can be simply categorized into three types. Each type of method has its merits and demerits depending on the applied scenario.

Secure Multi-party Computation (MPC) [36], [41], [43], [56], [60]. A secure MPC protocol can be used in place of a trusted third party to implement the same function by exploiting the secret sharing primitive and multiple rounds of interactions between the parties [15], [29].

Existing works usually rely on a strong assumption, which is challenging to realize in practice, i.e., the existence of two [18], [47], [49] or three [11], [42], even four parties [12] that do not collide with each other. On this basis, efficient protocols for the honest majority or the dishonest majority are proposed. Note that the computation and communication costs of the latter are several orders of magnitude larger than those of the former. For secure 2PC inference protocols, existing schemes either require multiple rounds of interactions between the client and server or assume that the server provides two non-colluding computing entities.

Homomorphic Encryption (HE) [20], [22], [31], [53]. The most attractive feature of HE is the ability to perform linear computations, i.e., addition and multiplication, on the encrypted data without decryption operations. HE can be directly used to achieve linear operations in the neural network, such as convolution. Nevertheless, a series of permutation operations in HE-based dot product and convolution still dominate the major computation time, even applying "batched" parallel computations over ciphertext (i.e., SIMD technique) exemplified by the CryptoNets framework [22] and the row-wise weight matrix encoding designed by Gazelle [36] and GALA [60]. For non-linear computations such as ReLU, polynomial approximations, and piece-wise functions are generally preferred. However, in addition to the high communication overhead associated with ciphertext expansion, both approximation methods require a trade-off between efficiency and classification accuracy.

Trusted Execution Environment (TEE) [28]. TEE allows untrusted parties to perform secure computation by isolating the code and data in hardware such as Intel Software Guard Extension (Intel SGX). However, hardware-assisted security limits

TEE's scalability and is very expensive to implement. Furthermore, many side-channel attacks on TEE-based systems have been reported [6]. In summary, secure MPC is a more efficient interactive technique but requires a strong assumption for non-colluded parties; HE provides an outsourced approach but exhibits a trade-off between accuracy and efficiency for non-linear computation; and TEE provides hardware-assisted security but lacks scalability. In this paper, we consider secure 2PC to achieve secure inference for efficiency reasons. To speed up the inference process even further, we develop special protocols that are implemented with fewer communication rounds and lower communication overhead.

B. 2PC Based Cryptographic Protocols

Here, we discuss separately how existing secure 2PC protocols implement linear and non-linear computation.

There are two main directions for existing secure 2PC schemes to implement linear functions, one is concerned with the whole runtime and reduces the computation time of matrix multiplication by designing new methods for permutation operations, such as Gazelle [36], GALA [60], and Cheetah [33]. The other works focus on the online execution time. For example, ABY2.0 benefits from a new secret-sharing primitive that performs the multiplication of two values, requiring only one online round, and the communication of two ring elements [47]. Our solution follows ABY2.0 and shares the high efficiency of linear computation.

On the other hand, it is still a tricky problem to reduce the communication cost incurred by secure protocols for nonlinear functions such as comparison and truncation. In 2PC, almost all of the prior works [18], [36], [41], [43] for securely and faithfully performing non-linear functions rely on GC. However, GC based protocols require a communication complexity of $O(\kappa\ell)$. To further obtain performance gains for the neural network prediction tasks, several recent efforts have been dedicated to designing a more efficient and secure comparison protocol. Using a popular cryptographic tool in 2PC, i.e., 1-out-of-n Oblivious Transfer functionality [5], CrypTFlow2 provides an order of magnitude fewer communications [49]. By using a more efficient oblivious technique, Cheetah achieves $110\times$ improvement over GC based works along with a reduction in online communication complexity to $O(\ell)$ [47]. Despite the significant improvement, CrypTFlow2 and Cheetah require online communication rounds of $O(\log(\ell))$. Furthermore, CrypTFlow2 utilizes OT to achieve secure truncation, which speeds up $20\times$-$30\times$ more than GC-based schemes in runtime. However, the OT-based secure truncation requires $O(\log(\ell))$ communication rounds, and neither work discusses how to achieve secure division with a secret-shared divisor.

Compared to the above works, the main innovation of this paper is to propose secure protocols for comparison and three types of divisions with $O(\ell)$ online communication. We extend them to implement the main primitives for neural network inference so that the communication overhead of each primitive in the online phase is $O(\ell)$, greatly improving the efficiency of neural network inference.

VIII. Conclusion

In this paper, we present Antelope, an efficient and secure implementation for practical neural network inference. By designing new protocols for comparison and division, Antelope improves upon the state-of-the-art in online communication overhead. We then use them to efficiently implement the main primitives for neural network inference, which can outperform previous works by several orders of magnitude in latency. Finally, we conducted experiments to evaluate the practical performance of our protocols, and the results demonstrated the superiority of Antelope compared with existing works. In the future, we would like to further exploit new ways to hide more information, such as model architectures, while optimizing efficiency. We are also interested in developing practical, secure protocols against a malicious adversary.

References

[1] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, "Secure computation on floating point numbers," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2013, pp. 1–19.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 535–548.

[3] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1991, pp. 420–432.

[4] D. Beaver, "Precomputing oblivious transfer," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1995, pp. 97–109.

[5] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1989, pp. 547–557.

[6] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A. R. Sadeghi, "The guard's dilemma: Efficient code-reuse attacks against Intel SGX," in *Proc. USENIX Secur. Symp.*, 2018, pp. 1213–1227.

[7] E. Boyle et al., "Efficient two-round OT extension and silent non-interactive secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 291–308.

[8] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 812–821.

[9] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Springer, 2010, pp. 35–50.

[10] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "SIMC: ML inference secure against malicious clients at semi-honest cost," in *Proc. USENIX Secur. Symp.*, 2022, pp. 1361–1378.

[11] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High throughput 3PC over rings with application to secure prediction," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, 2019, pp. 81–92.

[12] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.

[13] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, 2020, pp. 44–75.

[14] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2021, pp. 502–534.

[15] R. Cramer et al., *Secure Multiparty Computation*. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[16] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," in *Proc. Privacy Enhancing Technol.*, 2020, pp. 355–375.

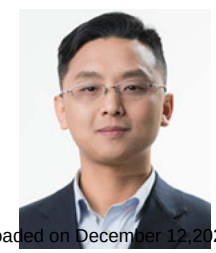
[17] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2012, pp. 643–662.

[18] D. Demmler, T. Schneider, and M. Zohner, "ABY–A framework for efficient mixed-protocol secure two-party computation," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[19] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[20] C. Edward, B. Josh, L. Daniel, Y. Serena, H. Albert, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," 2018, *arXiv: 1811.09953*.
[21] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic ReLUs for private deep learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 2241–2252.
[22] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.
[23] N. Gilboa, "Two party RSA key generation," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1999, pp. 116–129.
[24] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. thesis, Massachusetts Inst. Technol., 1964.
[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[26] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and secure multiparty computation from fixed-key block ciphers," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 825–841.
[27] L. Hanzlik et al., "MLCapsule: Guarded offline deployment of machine learning as a service," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 3300–3309.
[28] H. Hashemi, Y. Wang, and M. Annavaram, "DarKnight: A data privacy scheme for training and inference of deep neural networks," 2020, *arXiv: 2006.01300*.
[29] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Berlin, Germany: Springer Science & Business Media, 2010.
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[31] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv: 1711.05189*.
[32] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.
[33] Z. Huang, W. J. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Secur. Symp.*, 2022, pp. 809–826.
[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size," 2016, *arXiv:1602.07360*.
[35] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2003, pp. 145–161.
[36] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. USENIX Secur. Symp.*, 2018, pp. 1651–1669.
[37] J. Kilian, "Founding crytpography on oblivious transfer," in *Proc. Annu. ACM Symp. Theory Comput.*, 1988, pp. 20–31.
[38] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2013, pp. 54–70.
[39] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 336–353.
[40] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 619–631.
[41] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *Proc. USENIX Secur. Symp.*, 2020, pp. 2505–2522.
[42] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.
[43] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.
[44] M. Naor and B. Pinkas, "Oblivious polynomial evaluation," *SIAM J. Comput.*, vol. 35, no. 5, pp. 1254–1281, 2006.
[45] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 739–753.
[46] L. K. Ng and S. S. Chow, "GForce: GPU-Friendly oblivious and rapid neural network inference," in *Proc. USENIX Secur. Symp.*, 2021, pp. 2147–2164.
[47] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *Proc. USENIX Secur. Symp.*, 2021, pp. 2165–2182.
[48] D. Rathee et al., "SIRNN: A math library for secure RNN inference," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1003–1020.
[49] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 325–342.
[50] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. USENIX Secur. Symp.*, 2019, pp. 1501–1518.
[51] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proc. Annu. Des. Autom. Conf.*, 2018, pp. 1–6.
[52] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "TAPAS: Tricks to accelerate (encrypted) prediction as a service," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4490–4499.
[53] S. Sav et al., "POSEIDON: Privacy-preserving federated neural network learning," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–24.
[54] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, "Distributed vector-OLE: Improved constructions and implementation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1055–1072.
[55] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 3–18.
[56] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-majority maliciously secure framework for private deep learning," 2020, *arXiv: 2004.02229*.
[57] M. Wei, W. Zhu, L. Cui, X. Li, and Q. Li, "Privacy leakage in privacy-preserving neural network inference," in *Proc. Eur. Symp. Res. Comput. Secur.*, Springer, 2022, pp. 133–152.
[58] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1607–1626.
[59] A. C. C. Yao, "How to generate and exchange secrets," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, 1986, pp. 162–167.
[60] Q. Zhang, C. Xin, and H. Wu, "GALA: Greedy computation for linear algebra in privacy-preserved neural networks," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–16.

Xiaoyuan Liu received the PhD degree in cyberspace security from the University of Electronic Science and Technology of China, in 2024. She is currently a postdoc researcher with the College of Computer Science and Technology, Zhejiang University. She has published more than 10 papers including *IEEE Transactions on Information Forensics and Security*, *IEEE Internet of Things Journal*, WWW etc. Her research interests include applied cryptography, IoT security, and AI security and privacy.



Hongwei Li (Fellow, IEEE) is currently the head and a professor with the Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include network security and applied cryptography. He is distinguished lecturer of IEEE Vehicular Technology Society.

Guowen Xu received the PhD degree in cyberspace security from UESTC. He is a full professor with the University of Electronic Science and Technology of China. His research focuses on cybersecurity, AI security and privacy, and applied cryptography. Before joining UESTC, he has held positions as a research fellow with Nanyang Technological University, Singapore, and a senior research fellow with the City University of Hong Kong. He has published more than 100 papers in top-tier IEEE journals and conferences, including *IEEE Transactions on Dependable and Secure Computing (TDSC)*, *IEEE Transactions on Information Forensics and Security (TIFS)*, IEEE INFOCOM, IEEE S&P, ICML, and NeurIPS. His work has been recognized with multiple awards, including the IEEE BigDataSecurity Best Paper Award (2023), IEEE ICPADS Best Paper Award (2020), Wu Wenjun First Prize of Artificial Intelligence Science and Technology Progress (2021), IEEE Early Career Speaker, IEEE Computer Society (2025), and Computing's Top 30 Early Career Professionals, IEEE Computer Society (2025). He has extensive editorial experience, currently serving as an associate editor for *IEEE Transactions on Dependable and Secure Computing (TDSC)*, *IEEE Transactions on Information Forensics and Security (TIFS)*, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *IEEE Transactions on Circuits and Systems for Video Technology*, and *IEEE Transactions on Network and Service Management*. He is also a lead guest editor for *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*. Beyond his editorial roles, he has been an area chair or Senior Program Committee Member for prestigious international conferences, including ICML, ICLR, KDD, AAAI, NeurIPS, and CSCW.

Shengmin Xu is currently an associate professor with Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, Fuzhou, China. Previously, he was a senior research engineer with the School of Computing and Information Systems, Singapore Management University. His research interests include cryptography and information security.

Xinyi Huang is currently an associate professor with the Thrust of Artificial Intelligence, Information Hub, Hong Kong University of Science and Technology (Guangzhou), China. His research interests include cryptography and information security. He is in the editorial board of *International Journal of Information Security* and *SCIENCE CHINA Information Sciences*. He has served as the program/general chair or program committee member in more than 120 international conferences.

Tianwei Zhang received the bachelor's degree from Peking University, in 2011, and the PhD degree from Princeton University, in 2017. He is an assistant professor with the School of Computer Science and Engineering, Nanyang Technological University. His research focuses on computer system security. He is particularly interested in security threats and defenses in machine learning systems, autonomous systems, computer architecture and distributed systems. He is serving on the editorial boards of *IEEE Transactions on Circuits and Systems for Video Technology*, *ACM Transactions on Sensor Networks*.

Yijing Lin received the PhD degree from the Beijing University of Posts and Telecommunications (BUPT), in 2024. He is a postdoc researcher with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT). She has published more than 20 papers including WWW, IJCAI, *IEEE Transactions on Services Computing*, *IEEE Transactions on Communications*, *IEEE Transactions on Network Science and Engineering*, *IEEE Transactions on Vehicular Technology* etc. Her publications include ESI highly cited papers, IEEE ComSoc Best Readings and received four Best Paper Awards including IEEE OJCS, IEEE IWCMC, IEEE-CCF Service Computing Technical Committee. Her current research interests include blockchain and data unlearning.

Jianying Zhou is a professor and center director for iTrust with the Singapore University of Technology and Design (SUTD). His research interests are in applied cryptography and network security, cyber-physical system security, mobile and wireless security. He is a co-founder & steering committee co-chairofACNS. He is also steering committee chair of ACMAsia CCS, and steering committee member of Asiacrypt. He has served more than 200 times in international cyber security conference committees (ACM CCS & AsiaCCS, IEEE CSF, ESORICS, RAID, ACNS, Asiacrypt, FC, PKC etc.) as general chair, program chair, and PC member. He is associate editor-in-chief of IEEE Security & Privacy. He has also been in the editorial board of top cyber security journals including *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, *Computers & Security*. He received the ESORICS Outstanding Contribution Award, in 2020, in recognition of his contributions to the community. He is an ACM distinguished member.