AQ2PNN: Enabling Two-party Privacy-Preserving Deep Neural Network Inference with Adaptive Quantization

Yukui Luo
Northeastern University
Boston, MA, USA
luo.yuk@northeastern.edu

Nuo Xu
Lehigh University
Bethlehem, PA, USA
nux219@lehigh.edu

Hongwu Peng
University of Connecticut
Storrs, CT, USA
hongwu.peng@uconn.edu

Chenghong Wang
Indiana University Bloomington
Bloomington, IN, USA
cw166@iu.edu

Shijin Duan
Northeastern University
Boston, MA, USA
duan.s@northeastern.edu

Kaleel Mahmood
University of Connecticut
Storrs, CT, USA
kaleel.mahmood@uconn.edu

Wujie Wen
North Carolina State University
Raleigh, NC, USA
wwen2@ncsu.edu

Caiwen Ding
University of Connecticut
Storrs, CT, USA
caiwen.ding@uconn.edu

Xiaolin Xu
Northeastern University
Boston, MA, USA
x.xu@northeastern.edu

ABSTRACT

The growing prevalence of Machine Learning as a Service (MLaaS) enables a wide range of applications but simultaneously raises numerous security and privacy concerns. A key issue involves the potential privacy exposure of involved parties, such as the customer's input data and the vendor's model. Consequently, two-party computing (2PC) has emerged as a promising solution to safeguard the privacy of different parties during deep neural network (DNN) inference. However, the state-of-the-art (SOTA) 2PC-DNN techniques are tailored explicitly to traditional instruction set architecture (ISA) systems like CPUs and CPU+GPU. This reliance on ISA systems significantly constrains their energy efficiency, as these architectures typically employ 32- or 64-bit instruction sets. In contrast, the possibilities of harnessing dynamic and adaptive quantization to build high-performance 2PC-DNNs remain largely unexplored due to the lack of compatible algorithms and hardware accelerators.

To mitigate the bottleneck of SOTA solutions and fill the existing research gaps, this work investigates the construction of 2PC-DNNs on field programmable gate arrays (FPGAs). We introduce AQ2PNN, an end-to-end framework that effectively employs adaptive quantization schemes to develop high-performance 2PC-DNNs on FPGAs. From an algorithmic perspective, AQ2PNN introduces an innovative 2PC-ReLU method to replace Yao's Garbled Circuits (GC). Regarding hardware, AQ2PNN employs an extensive set of building blocks for linear operators, non-linear operators, and a specialized Oblivious Transfer (OT) module for secure data exchange, respectively. These algorithm-hardware co-designed modules extremely utilize the fine-grained reconfigurability of FPGAs, to adapt the data bit-width of different DNN layers in the ciphertext domain, thereby reducing communication overhead between parties without compromising DNN performance, such as accuracy. We thoroughly assess AQ2PNN using widely adopted DNN architectures, including ResNet18, ResNet50, and VGG16, all trained on ImageNet and producing quantized models. Experimental results demonstrate that AQ2PNN outperforms SOTA solutions, achieving significantly reduced communication overhead by 25%, improved energy efficiency by 26.3×, and comparable or even superior throughput and accuracy.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**.

KEYWORDS

Privacy-Preserving machine learning, Deep learning, FPGA, Quantization, Two-party computing

ACM Reference Format:
Yukui Luo, Nuo Xu, Hongwu Peng, Chenghong Wang, Shijin Duan, Kaleel Mahmood, Wujie Wen, Caiwen Ding, and Xiaolin Xu. 2023. AQ2PNN: Enabling Two-party Privacy-Preserving Deep Neural Network Inference with Adaptive Quantization. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3613424.3614297

1 INTRODUCTION

Machine Learning as a Service (MLaaS) has become increasingly popular in recent years [46], where users, organizations, and enterprises can all utilize the centralized computing service and infrastructures (e.g., AWS SageMaker [25], Google AI Platform [7], and Azure Machine Learning [3]) to build and deploy their own deep

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10…$15.00
https://doi.org/10.1145/3613424.3614297

neural networks (DNN). Despite its popularity, the practical MLaaS is challenged by the the ever-increasing privacy and security concerns, especially for these applications processing privacy-sensitive data like health records and location-based information [49]. These concerns highlight the urgent need for privacy-preserving solutions in the context of deep learning. A number of solutions have been proposed, mainly including trusted execution environment (TEE) [38], homomorphic encryption (HE)[19], and multi-party computation (MPC)[15, 44].

However, the two common methods, HE and MPC, have their unique pros and cons. HE, especially the leveled HE like CKKS [11, 36], is mainly used to protect client's data with the advantage of linear operations directly on ciphertexts. It works well for small to medium-scale DNNs without involving costly bootstrapping and large communication overhead. However, it is not suitable for protecting large-scale networks. Also, HE relies on public/private key generation and distribution, which introduces considerable extra costs and requires either a trusted curator to securely sample the keys or secret-sharing of the private key among two non-colluding parties[17]. Furthermore, HE techniques often incur high-cost multiplicative gate operations[18] for complex applications like DNN inference, which results in significantly slow inference. For example, the inference speed for HE-evaluated deep networks is 164.7 seconds per image for a 10-layer SqueezeNet inference on the CIFAR10 dataset [14].

MPC protocols such as Yao's Garbled Circuits (GC), which protect both client's data and model IP, on the other hand, often can support large-scale networks by partitioning the inference between clients and MLaaS providers. Similar to HE, the performance of MPC is also a key concern that hinders its practical deployment. For example, among the state-of-the-art (SOTA) works, Gazelle [26] requires more than 200 seconds latency and 8 GB communication to perform ResNet-32 on the CIFAR-100, and Falcon [53] consumes 5.9×10^4 times latency in DNN inference, than its plaintext counterpart. In contrast, a recent work CryptGPU[51] achieves much higher performance, i.e., it takes only 2.7 seconds to process VGG-16 inference for an image from the CIFAR10 dataset. This substantial difference in performance highlights the advantages of MPC over HE for privacy-preserving DNN inference. However, CryptGPU [51] incurs ultra high power consumption, i.e., it employs two CryptGPU platforms [51] with a power budget of 315×2 Watts, while implementing VGG16 for ImageNet inference.

This work focuses on the two-party computation (2PC) setup, a special case of MPC, for securing DNN inference. Our proposed method targets the following aspects to improve the efficiency of 2PC-DNN inference: (1) Developing ciphertext-dedicated DNN model quantization techniques and (2) Exploring secure two-party comparison methods without using Yao's Gabled Circuits [21]. These advancements drastically reduce the communication overhead in a 2PC setup, facilitating faster secret exchanges. We develop a new FPGA-based hardware accelerator capable of exploiting the adaptively quantized DNN model. Such innovations would undoubtedly benefit the MLaaS ecosystem and pave the way for more secure and efficient privacy-preserving solutions.

To this end, this work addresses the following ***challenges***: (1) It is non-trivial to quantize DNNs to embrace the 2PC protocols, which otherwise introduces quantization errors and affects the quality of the ML service. For example, improper ring size design can lead to the collapse of the overall secure inference process. More details on this challenge can be found in Sec. 5. (2) Like other works, the basic 2PC-DNN protocols incur ultra-high computation and communication overhead [35, 40]. Specially, applying 2PC on large complex DNN models exacerbates the problem due to (i) a vast amount of data communication and (ii) limited resources (e.g., on-chip memory size) on devices. (3) Adding cryptographic operations could greatly limit the solution practicality of secure deep learning (DL) in mobile and IoT devices.

We summarize our contributions in this work as follows:

• We propose a 2PC-DNN framework, AQ2PNN, from an algorithm and hardware co-design perspective – a hardware-friendly adaptive quantization scheme with a high accuracy guarantee. Since the data width of the secret directly determines the communication overhead of 2PC-DNN shared input feature map, which is highly compatible with our proposed adaptive quantization scheme. Moreover, we propose a novel two-party ReLU operation method that eliminates the usage of Yao's Garbled Circuit [21]. This method constructively combines arithmetic-to-binary share conversion [15] and oblivious transfer [5] to build ReLU in ciphertext domain.

• We design, optimize, and implement various modules for AQ2PNN. These modules include an arithmetic share-based general matrix multiplication unit (AS-GEMM), an arithmetic share-based arithmetic logic unit (AS-ALU), an arithmetic-to-binary share conversion machine (A2BM), and a secure comparison machine (SCM). All these modules are tailored to efficiently support various 2PC-DNN operations, enabling efficient execution on the heterogeneous CPU+FPGA platform. Moreover, our proposed design is systematic and highly modular (Fig. 1), laying the foundation for future ASIC developments.

• We implement AQ2PNN on low-cost AMD-Xilinx multiprocessor system-on-chip devices, leveraging the reconfigurable hardware resources in a CPU+FPGA system. The experimental results demonstrate that AQ2PNN outperforms SOTA GPU implementations, in terms of energy efficiency while maintaining comparable or even higher accuracy for popular DNN models.

• We evaluate AQ2PNN across datasets of different scales, small, medium, and large, as well as on various architectures including LeNet5, AlexNet, ResNet18, VGG16, and ResNet50. These experimental results showcase AQ2PNN's excellent scalability in terms of model size. Furthermore, we investigate the accuracy loss introduced by our adaptive quantization scheme and identify the optimal data width. Lastly, we discover that modifying model structures can improve throughput without significantly affecting accuracy, such as replacing Max pooling with Average pooling, substantially increasing throughput.

2 RELATED WORK AND THREAT MODEL

2.1 Two-party Computation for DNN

A two-party computation (2PC) setup [15, 41, 54] enables two collaborators (e.g., customer and vendor) to jointly evaluate a function on the private input data from them each. Specifically, a 2PC setup masks and distributes privacy-sensitive data on separate computing devices in a secret-sharing manner to protect the privacy of these two parties [9].

Taking the 2PC-DNN inference as an example, the two parties will provide the input data and model, respectively. In such a setup, no private information is leaked besides what can be obtained from the expected function outputs. The state-of-the-art (SOTA) 2PC-DNNs are predominantly built on software-programmable platforms like CPU [32] and CPU+GPU [51]. For example, Cryptflow [32], a CPU-based MPC framework that employs multiple MPC methods, offers 2PC-DNNs based on ABY2 [44]. CryptGPU [51], another SOTA CPU+GPU MPC-based DNN framework, operates in a 3-party setting and uses a 2-out-of-3 replicated secret-sharing scheme [2, 23]. In practice, CryptGPU can also be configured to address 2PC-DNNs.

Previous works have all been set to full precision computation (e.g., 32- or 64-bit data width), which is enforced by the underlying instruction set architectures of CPU or GPU. For instance, CryptGPU [51] creates a new abstraction called `CUDALongTensor`, a 64-bit integer-valued cryptographic datatype that is designed to utilize the optimized 64-bit floating-point GPU kernel. This is referred to as "GPU-friendly cryptography". However, these solutions are not efficient in addressing the large overheads associated with the communication between two parties. This problem becomes aggravated as modern DNN size continues to grow.

2.2 2PC Acceleration Using FPGA

FPGAs have witness great use in accelerating 2PC protocols like Yao's Garbled Circuits (GC) [22] and secret-sharing [6]. For example, Huang *et al.* deployed GC on Amazon AWS FPGA [21], which achieves 15× speedup compared to its CPU counterpart. A recent GC-based accelerator HAAC [39], leverages hardware support like HBM2 and provides ReLU with a maximum speedup of 779×, compared to its CPU counterpart. However, the underlying mechanism of GC incurs a large communication and circuit overhead, e.g., ReLU requires 67.9K wires, which significantly limits its applicability in complex tasks like DNN inference. Differently, the secret-sharing-based 2PC employs more rounds of lightweight communication to mitigate the bulky communication in Yao's Garbled Circuits, which makes it more lightweight, e.g., Wolfe *et al.* [57] demonstrate that secret-sharing outperforms other 2PC methods in data centers. Nonetheless, such explorations of the secret-sharing 2PC are dominantly focused on various fundamental benchmarks like the Advanced Encryption Standard (AES) and random number generator. Still, it could not satisfy the need of practical complex applications like DNN inference, due to the large communication overhead. In this work, we make use of the bit-wise reconfigurability of FPGA to mitigate the 2PC-DNN performance bottleneck, caused by its prohibitively high communication overhead.

2.3 Relevant Terminologies

Secret-sharing: Secret-sharing is a cryptographic technique that divides a secret into multiple "shares", such that no single share can reveal the original secret. There are a number of secret-sharing methods, such as additive [13], Blakley's secret-sharing [8], and Shamir's secret-sharing [47]. Among these solutions, additive secret-sharing has been applied to numerous 2PC applications, such as Cryptflow2 [45], CryptGPU [51]. Without loss of generality, we adopt the additive secret-sharing as the foundation for our AQ2PNN design, see details in Definition 3 and Sec. 4.

Oblivious Transfer: Oblivious Transfer (OT) is a popular method for building secret-shared comparison operators. Specifically, in the context of 2PC, party i has a set of messages $M = m_1, m_2, ..., m_N$, and party j holds an index n ($n \in [1, N]$). By evaluating a $(1, N)$-OT (1-out-of-N OT) protocol (see Definition 2), party j learns only message m_n but nothing else, while party i learns nothing about index n. In this work, we implement OT following the general design rules proposed by Chou *et al.* [12], with necessary modifications and hardware-friendly optimizations, see the related details in Sec. 4.3.

2.4 Threat Model

We consider a threat model for Machine Learning as a Service (MLaaS) in a 2PC context, where the users seek inference services from model providers and they both contribute to the computation (i.e., inference) process. Without loss of generality, we follow the 2PC setup similar to previous works [15, 29, 51, 53], where the two parties involved in secure MLaaS have opposing roles, i.e., the user and model provider, who strive to ensure the privacy of their data, exhibiting the characteristics of semi-honest but non-colluding parties [15, 27]. Last, in our setup, the data from both parties are protected by the additive secret-sharing method.

3 AQ2PNN: OVERVIEW

Table 1: Definitions of terms in AQ2PNN.

A2BM	Arithmetic-to-Binary Sharing Conversion Machine
ABReLU	Arithmetic-to-Binary Sharing based two-party ReLU
AS-ALU	Arithmetic Sharing based Arithmetic Logic Unit
AS-CST Buffer	Arithmetic Sharing Pre-Compute Triple's Buffer
AS-GEMM	Arithmetic Sharing based General Matrix Multiplication
AS-INP Buffer	Arithmetic Sharing Input's Buffer
AS-INP-MSK Buffer	Arithmetic Sharing Input Mask's Buffer
AS-OUP Buffer	Arithmetic Sharing Computing Output's Buffer
AS-WGT Buffer	Arithmetic Sharing Weight's Buffer
AS-WGT-MSK Buffer	Arithmetic Sharing Weight Mask's Buffer
BS-INP Buffer	Binary Sharing Input's Buffer
BS-OUP Buffer	Binary Sharing Output's Buffer
INST Q	Instruction Queue
OUT-MSK Buffer	Comparision Result Mask's Buffer
SCM	Secure Comparison Machine
Sec-COMM. Module	Secure-communication Module
Sec-COMP. Module	Secure-computing Module

We provide a systematic overview of the proposed AQ2PNN framework in Fig. 1, which details its underlying hardware modules. This system implementation consists of a CPU, DRAM, an AQ2PNN accelerator, and a Network Inference Card (NIC) for the data exchange between the two parties, which is suitable for quantized 2PC-DNN models. Specifically, we take into consideration the different requirements for inference accuracy, model architecture, and

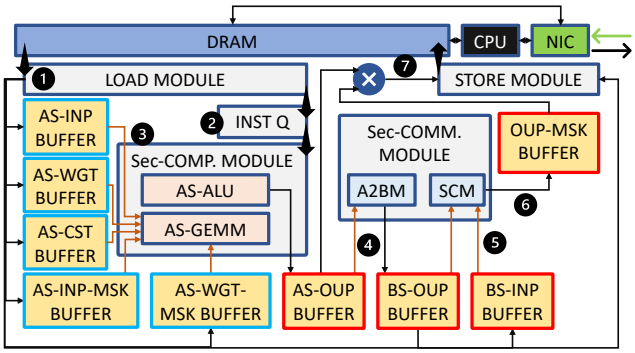

Figure 1: Overview of AQ2PNN framework, which is deployed on both party i and j in the 2PC-DNN setup.

latency from various applications, and we resort to the hardware-reconfigurable devices to develop a high-performance AQ2PNN accelerator. Our proposed adaptive quantization strategies are detailed in Sec. 5.1, 6.2, and 6.5. This method is implemented on both parties in a 2PC-DNN setup. Specifically, we focus on the design and optimization of the AQ2PNN accelerator part, which consists of four modules: LOAD Module, STORE Module, Secure-Computing (Sec-COMP.) Module, and Secure Communication (Sec-COMM.) Module. The Sec-COMP. Module contains a General Matrix Multiplication calculator for the secret arithmetic shares (AS-GEMM), which is detailed in Sec. 4.1.2, as well as an arithmetic share based arithmetic logic unit (*AS-ALU*). Sec. 4.1.3 provides the different computations supported by the AS-ALU, including addition, left shift, right shift, and clipping, all of them are based on 2PC.

We propose a novel 2PC-based rectified linear unit (ReLU), namely *ABReLU*, which realizes secret arithmetic to binary share conversion [15] and oblivious transfer (OT) [58], see details in Sec. 4.4. We employ the *Sec-COMM. Module* to implement ABReLU using the Arithmetic-to-Binary share conversion machine (A2BM) and the Secure Comparison Machine (SCM), detailed in Sec. 4.3. The workflow of AQ2PNN shown in Fig. 1 is as follows:

Step-1: The LOAD Module distributes the pre-compute secure constant (AS-CST), the secret arithmetic shares of the user's input (AS-INP) and input masks (AS-INP-MSK), as well as the model provider's weight (AS-WGT) and weight masks (AS-WGT-MSK) to their corresponding buffers.

Step-2: The instruction queue (INST Q) drives the Sec-COMP. Module to perform secure computation, see Sec. 4.1.

Step-3: The results from the Sec-COMP. Module are buffered in the Secret Arithmetic Share Output Buffer (AS-OUP Buffer).

Step-4: The data in the AS-OUP Buffer is used as inputs for A2BM, which converts the secret arithmetic shares to binary shares and stores them in the Secret Binary Share Output Buffer (BS-OUP Buffer), as described in Sec. 4.3.2.

Step-5: A data exchange occurs between two parties i and j, and the secret binary share from the other party is loaded into the Secret Binary Share Input Buffer (BS-INP Buffer).

Step-6: The BS-INP Buffer and BS-OUP Buffer carry the inputs for SCM to complete the 2PC secure comparison. The output mask is then stored in the OUP-MSK Buffer. Share Input Buffer (BS-INP Buffer).

Step-7: The results in the AS-OUP Buffer and OUP-MSK Buffer are combined and transferred back to DRAM. See an example in Sec. 5.1.

Note that steps 4-6 are performed using our developed ABReLU (see Sec. 4.4), in conjunction with the OT protocol described in Sec. 4.3, for secure 2PC-ReLU computation.

4 AQ2PNN: IMPLEMENTATION

Definition 1. ***Operation on a ring:*** *Given an unsigned integer ring $\mathbb{Z}_Q$, where the ring size is $Q = 2^\ell$ and ℓ denotes the bit-length. All operations performed on the ring $\mathbb{Z}_Q$ takes a modular (mod Q). A bit-length overflow in a hardware accelerator can easily replace this modular operator.*

Definition 2. ***(n,t)-secret-sharing:*** *A (n,t)-secret-sharing (i.e., t-out-of-n) over $\mathbb{Z}_Q$ refers to the scheme that splits a secret value $x \in \mathbb{Z}_Q$ among n parties. The security property ensures that any t' out of n parties can recover x, if and only if $t' \geq t$. Otherwise, no one knows any information about the secret value x.*

Definition 3. ***2PC additive secret-sharing:*** *We denote the two parties in the 2PC system as i and j, where i and j can get an index from set $\{0,1\}$ and $i \neq j$. This setup constructs a $(2,2)$-secret-sharing. For any value $x \in \mathbb{Z}_Q$, we use $[\![x]\!] \leftarrow (x_i, x_j)$ to denote its corresponding secret shares, where i and j denote the secret disseminated to party i and j separately.*

We introduce two basic operations over such additive secret-shared data.

- *Secret share $[\![x]\!]$ generation*: Given $x \in \mathbb{Z}_Q$, it samples a random value r in $\mathbb{Z}_Q$, and returns $[\![x]\!] \leftarrow (r, x - r)$.
- *Secret share recovering* $\text{rec}([\![x]\!])$: Given secret shares $[\![x]\!] \leftarrow (x_i, x_j)$, it computes $x \leftarrow (x_i + x_j) \bmod Q$, then returns x.

4.1 Secure-computing Module

The Secure-computing Modul (Sec-COMP. Module) is used in the linear operations of 2PC-DNNs, such as 2D convolution (2PC-Conv2D). In the 2PC setup, the 2PC-Conv2D operation should be conducted in the ciphertext domain, which can be accomplished as matrix multiplication accumulation based on two additive secret-shared data, i.e., **ciphertext-to-ciphertext (C-C) multiplication and accumulation**, which are implemented by AS-GEMM and the add logic in AS-ALU. The 2PC-BNReQ operator can be abstracted as multiplication and division of one additive secret-shared data and an unsigned integer constant, corresponding to the left and right shift logic in AS-ALU, and the requantization using the truncation logic in AS-ALU, i.e., **ciphertext-plaintext (C-P) multiplication, division, and truncation.** In the following subsections, we will discuss the microarchitecture of the Sec-COMP. Module, focusing on AS-GEMM and AS-ALU.

4.1.1 Instruction Queue. We develop an instruction queue module (INST Q) to drive the execution flow of AQ2PNN. These instructions are primarily for invoking the functions of AS-GEMM, AS-ALU, as well as allocating the workload based on the hyperparameter configuration of different DNN layers. This design method is compatible with the real-world deisgn practice, i.e., there are many DNN accelerator compilers for generating such an instruction queue, such as the open-source Apache TVM [10].

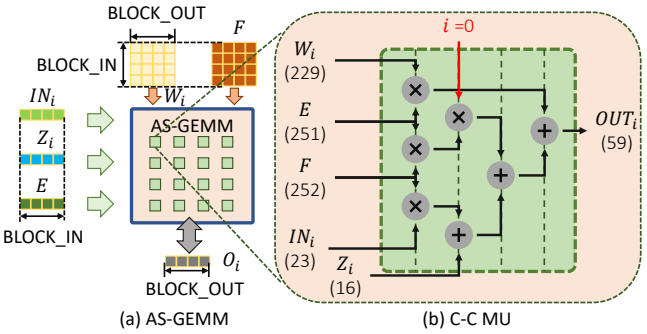

Figure 2: C-C matrix multiplication hardware. (a) Arithmetic share based general matrix multiplication array with (BLOCK_IN, BLOCK_OUT) = (4, 4) (b) C-C multiplication unit, its example input/output correspond to the OUT_i = 59 in Fig. 3.

4.1.2 Arithmetic Share-based GEMM. In 2PC-DNN context, the arithmetic-share-based general matrix multiplication (GEMM) is entirely different from that in the plaintext domain, i.e., due to the **C-C matrix multiplication** over secret-sharing information. According to the additive share and two-party setting, assuming a matrix multiplication over input ($[\![IN]\!]$) and weight ($[\![W]\!]$): $[\![OUT]\!] \leftarrow [\![IN]\!] \otimes [\![W]\!]$ that produces a new secret shared value $[\![OUT]\!] \leftarrow (OUT_i, OUT_j)$, such that $\text{rec}([\![OUT]\!]) = \text{rec}([\![IN]\!]) \otimes \text{rec}([\![W]\!])$. In this process, we need two masks $[\![E]\!]$ and $[\![F]\!]$ to assist each party's computation, i.e., hide the secret information of $[\![IN]\!]$ and $[\![W]\!]$ by using pre-computed multiplication triple $[\![A]\!]$, $[\![B]\!]$, and $[\![Z]\!]$, where $[\![Z]\!] = [\![A]\!] \cdot [\![B]\!]$. The three secret shared matrices of triple are $[\![A]\!] \leftarrow (A_i, A_j)$, $[\![B]\!] \leftarrow (B_i, B_j)$, and $[\![Z]\!] \leftarrow (Z_i, Z_j)$.

$$OUT_i = -i \cdot E \otimes F + IN_i \otimes F + E \otimes W_i + Z_i \ (i \in \{0, 1\}) \quad (1)$$

Initially, each party computes $[\![E]\!] = [\![IN]\!] - [\![A]\!]$ and $[\![F]\!] = [\![W]\!] - [\![B]\!]$. Then both parties jointly recover $E \leftarrow \text{rec}([\![E]\!])$, and $F \leftarrow \text{rec}([\![F]\!])$, and each party computes Eq. 1, the so-called *Beaver multiplication triples* [4] adopted by many other relevant works like Crypten [29], CryptGPU [51], ABY [15], and ABY2[44]. Taking party i as an example, OUT_i will then be treated as the secret share for $[\![OUT]\!]$. Typically, the multiplication triple can be generated using homomorphic encryption [60] or with oblivious transfer (OT)[28]. For a better understanding of Eq. 1, we provide an illustration in Fig. 3 to demonstrate its complete flow and compare the recovered results from ciphertext with the results in plaintext.

For facilitate the C-C maxtrix multiplication in the ciphertext domain, we design an arithmetic share based GEMM (AS-GEMM), for which we draw inspiration from the normal GEMM used in the open-source DNN accelerator VTA [42]. We build an AS-GEMM computation array as shown in Fig. 2 (a), to enable parallelism in the input and output channels with size BLOCK_IN and BLOCK_OUT. We apply a fully pipelined initiation interval of 1 in the output channel. This design can perform piplined matrix multiplication at a rate of one input-weight matrix multiplication per cycle.

In AS-GEMM, we replace the conventional multiplication with a **C-C multiplication unit** (C-C MU) shown in Fig. 2 (b). Using party i as an example, this unit complies with Eq. 1. The mask E is related to the user's input, which is variable, so the calculation of E must be performed for each inference. We store the value of E in

Algorithm 1: AS-GEMM implementation for party i

Data: Secret shared input matrix IN_i, weight matrix W_i, prepared C-C MU masked values E, F and Z_i for each output OUT_i
Result: Each parallel output channels OUT_i
$oc \leftarrow$ BLOCK_OUT;
while $oc \neq 0$ **do**
  $OUT_i(oc) = 0$; /* Initialization */
  $ic \leftarrow$ BLOCK_IN;
  while $ic \neq 0$ **do**
    $OUT_i(oc)$ += C-C MU$(IN_i, W_i, E, F, Z_i, i)_{ic}$;
    $ic \leftarrow ic - 1$;
  end
  $oc \leftarrow oc - 1$;
end

the AS-INP-MSK Buffer. The mask F is related to the weight from party j (i.e., model provider), which can be pre-computed to reduce the overhead for communication and computation, e.g., it can be pre-deployed in the memory of each party. When computing each layer, F is loaded into the mask buffer (AS-WGT-MSK). The user's input share IN_i and the model provider's weight share W_i will be loaded into the AS-INP Buffer and AS-WGT Buffer, respectively. Z_i is a pre-computed value stored in the AS-CST Buffer. In addition, as shown in Def. 2, we need to determine the index of party i, where $i \in \{0, 1\}$, which affects the method used by parties i and j when computing C-C matrix multiplication and ensures the correctness of the recovery process. Then, we construct the AS-GEMM hardware based on the pseudocode in Alg. 1, and the C-C multiplication unit.

4.1.3 Arithmetic Share-based Arithmetic Logic Unit. The diverse arithmetic operations supported by the arithmetic share based arithmetic logic unit (AS-ALU) in between the ciphertext (**C**) and Plaintext (**P**) are as follows.

C-C addition: For any secret-shared values $[\![x]\!]$ and $[\![y]\!]$, we have $[\![x + y]\!] \leftarrow (x_i + y_i, x_j + y_j)$.

P-C addition: For any secret-shared values $[\![x]\!]$ constant $a \in \mathbb{Z}_Q$, we have $[\![a + x]\!] \leftarrow (a + x_i, a + x_j)$.

P-C multiplication: For any secret-shared values $[\![x]\!]$ constant multiplicative $a \in \mathbb{Z}_Q$, we have $[\![ax]\!] \leftarrow (ax_i, ax_j)$.

P-C division: For any secret-shared values $[\![x]\!]$ constant dividend $a \in \mathbb{Z}_Q$, we have $[\![\frac{x}{a}]\!] \leftarrow (\frac{x_i}{a}, \frac{x_j}{a})$.

The arithmetic operations in the AS-ALU are similar to their plaintext counterpart. The primary difference is that the hidden secure results should be maintained on the $\mathbb{Z}_Q$ ring, to ensure ensure the correctness of secure computing results.

4.2 Matrix Multiplication and Accumulation

Using the proposed AS-GEMM and AS-ALU, we can implement other critical 2PC-DNN operators, i.e., matrix multiplication and accumulation (2PC-MMAC). We illustrate how the 2PC-MMAC works in Fig. 3, using (BLOCK_IN, BLOCK_OUT) = (4, 4) as an example. The input and mask E of each party consist of 4 pixels, and they are broadcasted to each column of the 4×4 weight and mask F used for

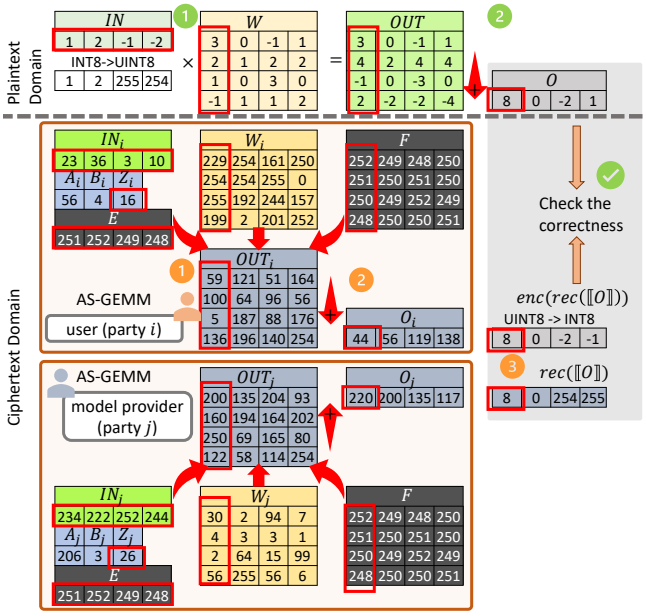

Figure 3: Schematic of 2PC-MMAC. This example includes the computation of MMAC in Plaintext Domain and computation using AS-GEMM (Fig. 2) in Ciphertext Domain.

AS-GEMM computation. Consequently, 16 AS-GEMM operations are computed in parallel.

In **Plaintext Domain**, green ① represents the matrix multiplication of input (IN) and weight (W), where OUT denotes the intermediate results. In green ②, accumulation is performed along the direction of the arrows to produce the final output O.

In **Ciphertext Domain**, orange ① party i and j use AS-GEMM to compute intermediate values OUT_i and OUT_j, respectively. In orange ②, the add instruction in AS-ALU is executed along the direction of the arrows to produce the output O_i and O_j. In orange ③, we compute $\text{rec}(\llbracket O \rrbracket) \leftarrow (O_i + O_j) \bmod Q$ and encode $\text{rec}(\llbracket O \rrbracket)$ with 2's compliment method $enc(\text{rec}(\llbracket O \rrbracket)) = O$, verifying the correctness of 2PC-MMAC. We can observe that in this setup, the 2PC-MMAC will not reveal any original values during the computation.

4.3 Secure-communication Module

Data comparison is a critical and challenging step in 2PC-DNN, in which the two compared values are encrypted and exchanged between the two parties, yet the Yao's classical *millionaires' problem*. Such operation is realized with secure communication in the existing works, which incurs large communication overhead. To facilitate the hardware acceleration, we follow the protocol in [16] to develop a secure-communication Module (Sec-COMM. Module). Our proposed method is detailed in Sec. 4.3.1, which can perform operations like 2PC-ReLU and 2PC-MaxPool. Since this protocol uses arithmetic-to-binary share conversion [15] during the oblivious transfer (OT) [5], we name the developed new 2PC-ReLU methods as **ABReLU**, detailed in Sec.4.4.

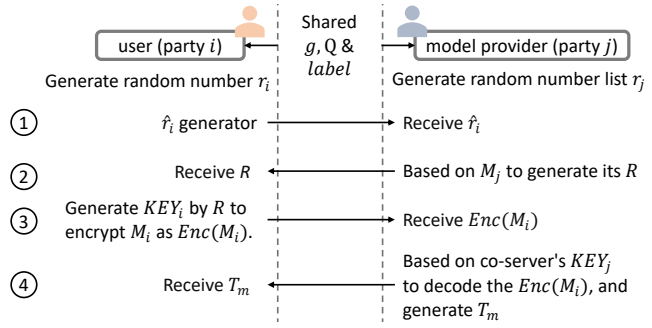

Figure 4: Secure two-party comparison protocol w/ OT-flow.

4.3.1 Secure Two-party Comparison Protocol. We adopt the Diffie-Hellman key exchange protocol (noted as *OT-flow*), to establish a secure information exchange channel between the two parties. The OT-flow uses the multiplicative group of integers modulo Q to mask data for exchange (i.e., communication), and its flow is shown in Fig. 4. In the initialization step, party i and j share the modulus (Q), group number (g), and a non-repeating randomly generated element *label* list of length L, on which the inquiry is an injective non-surjective function: $e2l(\cdot) : x \mapsto label(x)$.

① Party i generates a random number r_i, and uses g to mask r_i as $\hat{r}_i = g^{r_i} \bmod Q$. $\hat{r}_i$ is sent to party j.

② Party j receives $\hat{r}_i$ and generates a 2D matrix R relevant to its own message (M_j). M_j is a $V \times U$ matrix generated from V N-bit signed numbers using the arithmetic-to-binary share conversion machine (A2BM), which is detailed Sec. 4.3.2. A2BM splits an N-bit binary number into U parts, i.e., $M_j(v,u)$ represents the u-th group in the v-th number, where $v \in [0, V-1]$ and $u \in [0, U-1]$. Each element $M_j(v,u)$ is the input for $e2l(\cdot) : x \mapsto label(x)$. The party j generates a list of random numbers r_j of length U. Then, it applies power and *mod* with XOR ($\oplus$) to generate R, following Eq. 2.

$$R(v,u) = (\hat{r}_i^{\,e2l(M_j(v,u))} \bmod Q) \oplus (g^{r_j(u)} \bmod Q) \tag{2}$$

Since the numbers are unsigned and with ℓ-bit ($Q = 2^\ell$), the results of power and *mod* operations are finite, which can be replaced by a hardware-friendly look-up-table in practice.

③ After receiving R, party i also has V N-bit signed numbers to process. These numbers can be extended into a 3D matrix message (M_i), by the A2BM and a secure comparison machine (SCM) detailed in Sec. 4.3.3. M_i is a $V \times U \times L$ matrix with $M_i(v,u,l)$ denoting an element, where $v \in [0, V-1]$, $u \in [0, U-1]$, and $l \in [0, L-1]$. The party i uses R and *label* to generate KEY_i, and encrypt M_i as $Enc(M_i)$ following Eq. 3

$$Enc(M_i(v,u,l)) = M_i(v,u,l) \oplus KEY_i(v,u,l) \tag{3}$$

For hardware-friendly purpose, we generate KEY_i with XOR($\oplus$) operations following Eq. 4.

$$KEY_i(v,u,l) = R(v,u) \oplus \left((\hat{r}_i^{\,e2l(M_i(v,u,l))} \bmod Q)^{r_i} \bmod Q \right) \tag{4}$$

④ Party j generates KEY_j with Eq. 5 to decrypt the message $Enc(M_i)$, i.e., by XORing each element with KEY_j. Note that only the comparison information between M_i and M_j can be correctly

label	M_i / pm_j	$M_i(v,0)$	$M_i(v,1)$	M_i / pm_j	$M_i(v,2)$	$M_i(v,3)$	$M_i(v,4)$
		x_7	x_6		x_5x_4	x_3x_2	x_1x_0
l_0	0	$M_i(v,0,0)$	$M_i(v,1,0)$	00	$M_i(v,2,0)$	$M_i(v,3,0)$	$M_i(v,4,0)$
l_1	1	$M_i(v,0,1)$	$M_i(v,1,1)$	01	$M_i(v,2,1)$	$M_i(v,3,1)$	$M_i(v,4,1)$
l_2				10	$M_i(v,2,2)$	$M_i(v,3,2)$	$M_i(v,4,2)$
l_3				11	$M_i(v,2,3)$	$M_i(v,3,3)$	$M_i(v,4,3)$

Figure 5: A possible value comparison matrix with 8-bit data.

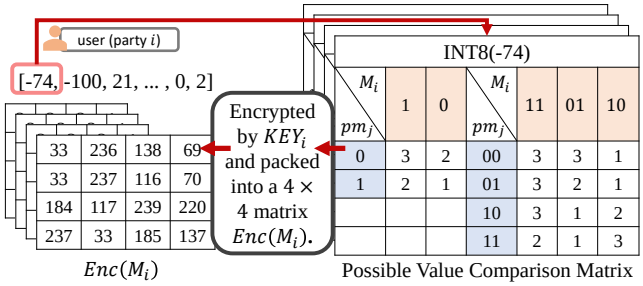

Figure 6: M_i message generation, encryption, packaging.

generated.

$$KEY_j = \hat{r}_i^{r_i \bmod Q} \tag{5}$$

After decryption, party j generates a T_m matrix stored in OUP-MSK Buffer, as the comparison result to inform party i.

4.3.2 Arithmetic-to-Binary Share Conversion Machine. We develop an arithmetic-to-binary share conversion machine (*A2BM*) to convert the arithmetic shares into binary shares, by grouping the bits. Considering an arithmetic share with ℓ-bit, we use "||" to split them into U groups. For example, an arithmetic share of signed 8-bit integer (INT8) format can be separated into five groups: $x \leftarrow x_7||x_6||x_5x_4||x_3x_2||x_1x_0$, where $x \in \mathbb{Z}_{2^8}$. The two most significant bits $x7$ and x_6 are associated with $(1,2)-OT$, while the rest of the bits are associated with $(1,4)-OT$. In general, a group with ℓ_s sub-bits follows the $(1,2^{\ell_s})-OT$ scheme.

4.3.3 Secure Comparison Machine. We develop a secure comparison machine (SCM) to generate a possible value comparison matrix (M_i) after using A2BM. We provide a formulated matrix for the INT8 data type in Fig. 5. Each INT8 value is separated to five groups: $M_i(v,u,l), u \in \{0,1,2,3,4\}$, and them each has a number of possible comparison value pm_j, e.g. $M_i(v,2)$ has 4 pm_j in 00, 01, 10, 11. The comparison result is shown in Eq. 6.

$$M_i(v,u,l) = \begin{cases} 1, & M_i(v,u) < pm_j \\ 2, & M_i(v,u) = pm_j \\ 3, & M_i(v,u) > pm_j \end{cases} \tag{6}$$

We use an INT8 data (-74) as an example to explicitly show the mechanism of possible value comparison matrix in Fig. 6. In the beginning, A2BM separates INT8(-74) into five groups. The most significant bit (MSB) is the sign bit. After applying A2BM, all encryption and calculation operations will depend on **Step ③** in the OT-flow. After recovering the message, the OT-flow employs an extra step to pack the encrypted message into a matrix. For example, in Fig. 6, we pack each INT8 value into a UINT8 4 × 4 matrix.

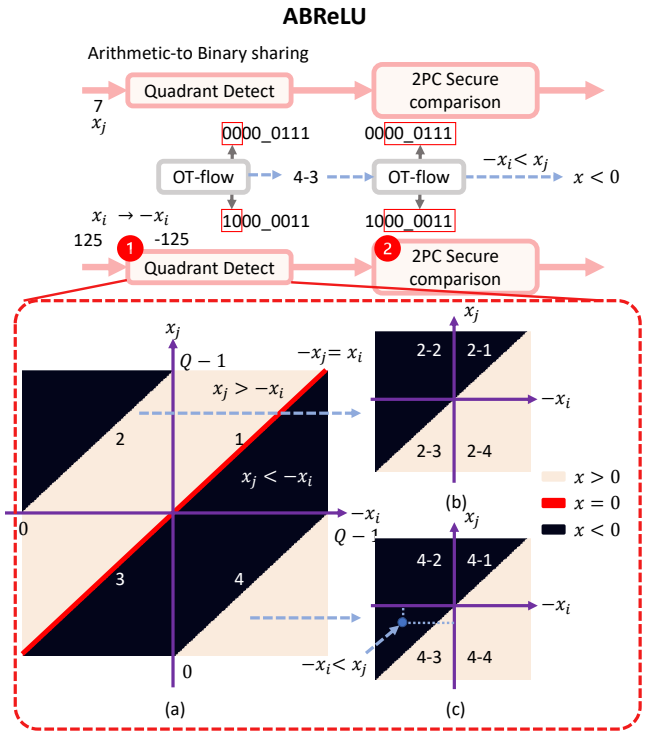

Figure 7: Evaluation of $x \leftarrow (x_i + x_j) \bmod Q$. (a) x_i and $-x_j \in [0, Q-1]$, corresponding to the values of x. (b) Subdivision of the 2^{nd} quadrant. (c) Subdivision of the 4^{th} quadrant.

Consequently, the formulated possible value comparison matrix will convert one 8-bit value into a 4 × 4 8-bit matrix. Assuming we have an ℓ-bit value, and where $U = \lfloor \ell/2 \rfloor + 1$, the packaged matrix size is $\lceil \ell/2 \rceil \times 4$. As the example shown in Fig. 6, we have $\ell = 8$ and $U = 5$. We can find that the two leftmost groups each have only two possible values, while the three groups on the right have four possible values. Therefore, we can combine the data of the two leftmost groups and use a 4 × 4 matrix to pack the data efficiently.

4.4 Arithmetic-to-Binary Sharing based ReLU

While it is straightforward to conduct the ReLU(x) operation in the plaintext domain, e.g., ReLU(x) = x for $x > 0$, it is complicated and challenging to do so in the ciphertext domain. In previous works that employ Yao's Garbled Circuit [21], the embedded secret-sharing from one party also includes the sign of the data. However, in the additive secret-sharing based method, the effect of the sign bit and the "mod" operation could not be ignored. We have to take the signed secret information for ReLU computation, and cannot directly apply the traditional secure comparison method [15]. For example, if $(-x_i, x_j) = (-100, 5)$, represented in INT8 binary, the traditional secure comparison will lead to a wrong conclusion that $-100 > 5$, as their binary representations are $(1001\_1100)_b$ and $(0000\_0101)_b$, respectively. In fact for 2PC-ReLU we need to compare "$(x_i + x_j)$ mod Q" vs. "0", not "-x_i" vs. "x_j".

To solve this problem, we first explore all possible results of "$(x_i + x_j)$ mod Q", based on the range of x_i and x_j. Considering the

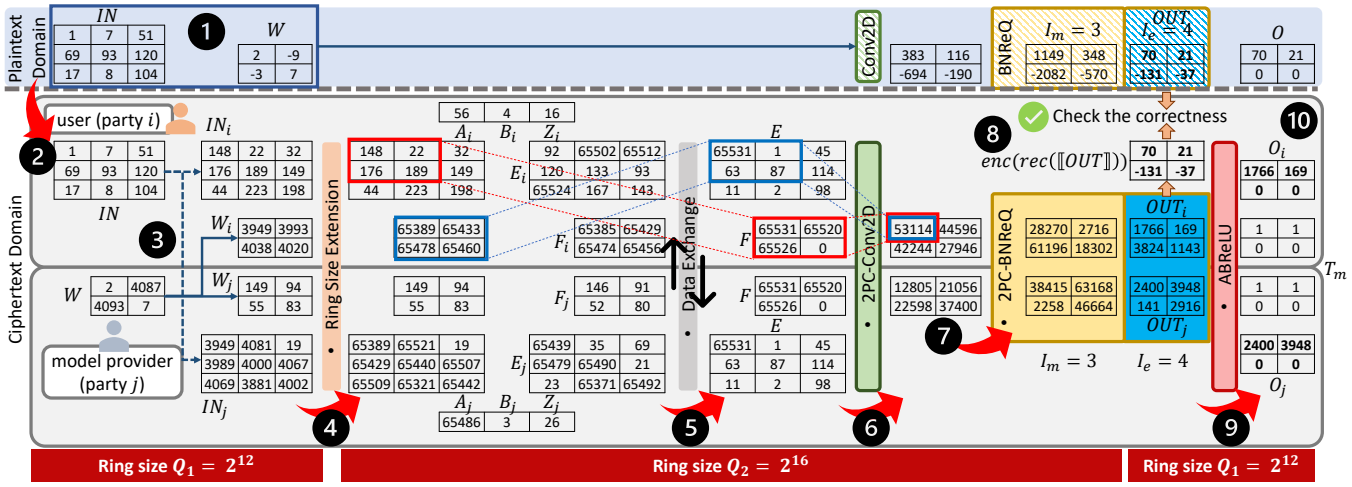

Figure 8: DNN Building block demonstration, exploring the ring size change and computation correctness.

quantized DNNs in the plaintext domain, the input of ReLU is a ℓ-bit signed number encoded by 2's complement method. Taking $-x_i$ and x_j as coordinates, we calculate "$x \leftarrow (x_i + x_j) \bmod Q$" and evaluate x. As shown in Fig. 7 (a), the light pixels represent $x > 0$, the dark pixels represent $x < 0$, and the red pixels represent $x = 0$. Therefore, we can use the most significant 2 bits of $-x_i$ and x_j to determine the quadrant where the hidden x falls. For example, if x falls in the 1^{st} and 3^{rd} quadrants, we can directly determine the sign of x by comparing $-x_i$ and x_j, to achieve higher inference efficiency.

However, the evaluation of x will be more complicated, if it falls in the 2^{nd} and 4^{th} quadrants, as shown in Fig. 7 (b) and (c). More specifically, if x falls in the 2-2, 2-4, 4-2, and 4-4 sub-quadrants, we can still directly determine its sign as "-" and "+", respectively. While x falls in the 2-1, 2-3, 4-3, and 4-3 sub-quadrantsm, we could not directly know its sign. To fully address this issue, we propose a novel arithmetic-to-binary sharing-based ReLU design (*ABReLU*), which conducts the secret-sharing-based ReLU in two steps: Red ① Quandrant detection, which determines the quadrant using the most significant 2 bits; Red ② 2PC secure comparison, which uses OT-flow (Sec. 4.3.1) to compare the value of $-x_i$ and x_j.

We take $(x_i, x_j) = (125, 7)$ as an example to demonstrate how ABReLU works, where $-x_i = -125$. Party i and j convert $-x_i$ and x_j into 8-bit: $(-x_i, x_j) = (1000\_0011, 0000\_0111)_b$. The separation operator splits $-x_i \leftarrow 1||0||00||00||11$, and $x_j \leftarrow 0||0||00||01||11$ into 5 groups. ABReLU will find that x falls in the 4-3 quadrant, by comparing the first two groups, then it compares each of the remaining groups from left to right using the OT-flow and gets $-x_i < x_j$. Based on this result, it concludes that $x < 0$. By reconstructing $\text{rec}[\![x]\!] = -124$, we can check the correctness of the ABReLU output.

We take $(x_i, x_j) = (-2, -2)$ as an other example, from which we get $(-x_i, x_j) \leftarrow (0||0||00||00||10, 1||1||11||11||10)_b$. By comparing the most significant bits, we find that the hidden x falls in the 2-2 quadrant. Since all x possible values in the 2-2 quadrant are less than 0, we can directly conclude that $x < 0$. Reconstructing $\text{rec}[\![x]\!] = -4$, we can verify the correctness of the ABReLU result.

4.5 Security of AQ2PNN

Our overall implementations do not alter the underlying 2PC algorithm or the secret-sharing mechanisms, and the proposed ABReLU are developed based on ABY [15] and OT [12]. Therefore, the algorithm modifications are merely post-processing operations over the secret shares, and the security guarantee of AQ2PNN follows the security of the original 2PC protocol and OT. That is, as long as the adversary is incapable of acquiring the randomness required to generate secret-sharing, the probability of recovering sensitive data and breaching the protocol remains negligibly small [52].

5 AQ2PNN: ADAPTIVE QUANTIZATION

Most SOTA solutions employ fix-bit ring for the DNN models, e.g., DELPHI [37] and Falcon [53] utilize 32-bit, while CryptGPU [51] utilizes a fixed 64-bit ring. However, these large and fixed bit ring sizes lead to significant computation and communication overhead for the 2PC-DNN inference. Differently, AQ2PNN dynamically adjusts the bit-width of the model at different stages, to reduce the overhead associated with the quantized 2PC-DNN inference.

5.1 Quantized Model Inference

We first take the DNN model building block in Fig. 9 (a) as an example, to illustrate the model quantization in plaintext domain. This block consists of 2D convolution (Conv2D), batch normalization (BN), and re-quantization (ReQ) operators. The initial input of Conv2D is 8-bit, which is then extended to 32-bit. By using ReQ operators to perform truncation, we finally obtain an 8-bit output. Such a consistency of output and input bit-widths is necessary to ensure the proper implementation of a quantized DNN model. We adopt this design philosophy in our proposed adaptive quantization. To further reduce the number of involved parameters and improve the inference efficiency, we follow the scheme in Hawq-v3 [59] to

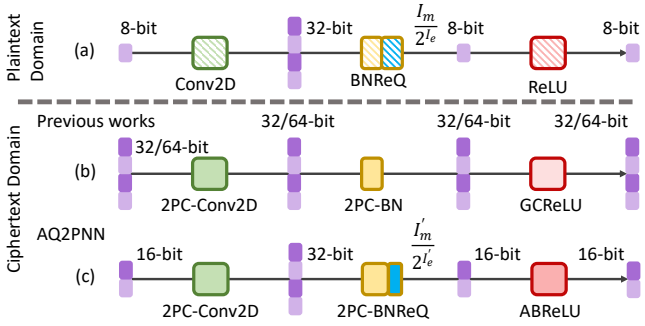

Figure 9: Comparison among different DNN frameworks. (a) Default plaintext DNN inference with an 8-bit quantized model. (b) Commonly used 2PC-DNN inference flow. (c) AQ2PNN inference flow with 16-bit quantized model.

combine the BN and ReQ operations into *BNReQ*, during the training stage. Specifically, we employ a scaling factor I_m and truncation bit I_e to guide the training and combination. These parameters will also be used in the ciphertext domain quantization model.

For model quantization in the ciphertext domain, one key design consideration is the ring size Q. Assuming a plaintext model has input and weight values of ℓ-bit, when this model is converted to a 2PC model, we need to choose a larger ring size to carry the original ℓ-bit secret value, i.e., to avoid bit overflow along the inference. Since otherwise, the bit-width of intermediate results along the inference will expand and possibly overflow the ring, which leads to incorrect inference results. Most existing works, e.g., the CPU or CPU+GPU-based, choose to use the maximum ring size of 32 or 64-bit to fit their underlying ISA systems, as shown in Fig. 9 (b). These methods, although could ensure the correctness of the 2PC-DNN inference, but also introduce large communication and computation overhead associated with the model size.

To address these concerns, we propose an adaptive model quantization scheme. Our proposed quantization scheme characterizes the distribution of run-time activation, through a post-training quantization process, and determines the required bit-width to minimize the quantization error while reducing 2PC operator computation overflow probability. We conduct statistical analysis on the bit-width to avoid overflow. Our experimental results have shown that for a plaintext model has input and weight values of ℓ-bit, a suitable ring size is $2^{\ell+4}$. For example, in a 12-bit plaintext model, a 16-bit carrier (i.e., a ring size of $2^{12} \times 2^4 = 2^{16}$) is used to generate additive secret shares. We illustrate the proposed adaptive quantization scheme in Fig. 9 (c) using 16-bit quantization.

To demonstrate the inference process using our proposed adaptive quantization, we extend Fig. 9(a) (c), and provide a quantization building block demonstration in Fig. 8 to demonstrate the workflow: ① Our proposed adaptive quantization method is trained by the model provider party j in the plaintext domain, yielding an 8-bit quantized model. ② Both the user (party i) and the model provider utilize a larger ring carrier ($Q_1 = 2^{12}$) to expand their data. ③ Two paries generate additive secret shares and deploy them. ④ The Ring size extension function is enabled to extend $Q_1 = 2^{12}$ to $Q_2 = 2^{16}$. ⑤ Both parties exchange data and compute masks E and F. ⑥ The 2PC-Conv2D operator, which primarily relies on AS-GEMM, is employed as described in Sec. 4.1.2. ⑦ The 2PC-BNReQ is facilitated by scaling and truncation, generating internal results OUT_i and OUT_j for each party. ⑧ We recover and encode $[\![OUT]\!]$, comparing it with the plaintext OUT to verify correctness. ⑨ The ABReLU operator is activated, generating the mask T_m. ⑩ The outputs of the two parties, O_i and O_j, are extracted as the result of this block.

In summary, our proposed adaptive quantization-based inference mainly goes through the following operation:

• **Ring Size Extension.** In the Plaintext Domain, the data bit-width of convolution continues to expand due to a large number of consecutive MAC, as shown in Fig. 9(a), where it expands to 32-bit. Therefore, 2PC-Conv2D in the Chipertext Domain requires a larger ring to carry the data. Fig. 8 visualizes the change of ring size from $Q_1 = 2^{12}$ to $Q_2 = 2^{16}$, where ring size extension is based on the sign extension. For example, the 12-bit number in Q_1 is 1111_0110_1101, and in Q_2, it becomes 1111_1111_0110_1101.

• **Data Exchange.** As discussed in Sec. 4.1.2, due to the variability of input, the masked value E also changes in each inference. Therefore, data exchange is needed here to allow both parties to obtain the updated E matrix.

• **2PC-Conv2D.** This stage involves extensive use of AS-GEMM. To accelerate this process, we employ blocking techniques to divide the data into blocks for computation. The results are stored in the AS-OUP Buffer.

• **2PC-BNReQ.** The 2PC-BNReQ operator uses the P-C multiplication and P-C division logic from AS-ALU. Then, we truncate the ring size from Q_2 back to Q_1.

• **ABReLU.** The ABReLU (Sec.4.4) operation generates the mask T_m (Fig. 4) for $[\![OUT]\!] \leftarrow (OUT_i, OUT_j)$ and produces $[\![O]\!] \leftarrow (O_i, O_j)$ as input for the next layer.

5.2 Quantized Model Accuracy Evaluation

We evaluate the performance of our proposed adaptive quantization scheme on MNIST [33], CIFAR10 [30], and ImageNet [31] datasets with different model architectures, and report the inference accuracy in Tab. 2. Following the previous works [24, 51, 59], we adopt several representative models like ResNet18/50 [20] and VGG16 [48] in our evaluation. We use PyTorch [43] to perform the model training and quantization. For ImageNet, we perform the quantization on the pre-trained model from PyTorchCV [50] and keep the architecture unchanged to show the performance of the models after the quantization. For CIFAR10, we first train the full precision model following the same architecture in CryptGPU[51], but using only one linear layer for the final output, to achieve the SOTA model accuracy. Then we apply our proposed quantization on the trained model.

Tab. 2 compares the performance of our proposed adaptive quantization scheme with other SOTA works. The baseline indicates the full precision (32-bit floating-point) model accuracy, and the following rows correspond to two quantized inferences in Fig. 9(b) (previous works [29, 37, 51, 53]) and Fig. 9(c) (AQ2PNN). In previous works, their input and output are quantized to a unified 32-bit or 64-bit fixed point format, and utilize an extra scaling function to constrain the activation value range. Differently, our proposed scheme in Fig. 9(c) reduces the OT-flow cost with little accuracy

Table 2: Inference accuracy (%) on different models for MNIST, CIFAR10 and ImageNet with proposed quantization

Dataset	Model	Baseline (float32)	Previous works [51, 53]	**AQ2PNN (16-bit)**
MNIST	LeNet5	99.26	96.85	**99.34**
	AlexNet	99.09	97.42	**99.11**
CIFAR10	VGG16	92.28	91.98	**91.69**
	ResNet18	93.02	92.79	**93.06**
ImageNet	VGG16	73.02	72.73	**72.08**
	ResNet18	73.06	72.87	**72.59**
	ResNet50	77.72	77.47	**76.24**

Table 3: AQ2PNN vs. VTA (plaintext DNN): a comparison of resource consumption.

	LUT	FF	DSP	BRAM
AQ2PNN	120k × 2	207k × 2	1536 × 2	310 × 2
VTA	24.2k	26.8k	268	136.5

drop. Specifically, the AQ2PNN results are obtained on all listed models with 16-bit output sent to the ABReLU operator, which is **1/2** of the data width in DELPHI [37] and Falcon [53] inference framework, and is **1/4** of the data width in CryptGPU [51] framework. We discuss the trade-off between communication cost and model utility using different extracted output bit-widths in Sec. 6.5.

6 AQ2PNN: EXPERIMENTAL EVALUATION

We use two ZCU104 FPGA development boards to serve as the computing resource for two parties. ZCU104 integrates all common SoC-FPGA components, such as an SRAM FPGA, an ARM processor, DRAM controller/buses, and peripheral I/Os, which enable us to emulate a practical 2PC setup. We implement AQ2PNN on these two FPGAs and connect them with Ethernet LAN at a bandwidth of 1000Mbps. The hardware resource consumption is reported in Tab. 3.

6.1 Overall Evaluation

We compare AQ2PNN (16-bit quantized 2PC-DNN solution) with the following SOTA solutions:

- **Falcon** [53], a standard 3PC-DNN framework, which demonstrates outstanding performance in DNN models of small and medium sizes.
- **Cryptflow** [32], we use its ABY2-based [44] 2PC-DNN for fair performance comparison.
- **CryptGPU** [51], we utilize its 2PC-DNN setting and apply a 2-out-of-2 secret-sharing scheme for comparison.

In particular, we consider four important metrics: (1) Throughput (Tput.), with unit frames-per-second (fps). (2) The communication overhead (Comm.), with the corresponding unit specified in the measurement, e.g., Mebibyte (MiB) (3) Power consumption (Power), with unit Watt (W). (4) Energy efficiency (Efficiency), using fps-per-watt (fps/W) as its unit. Note that all **Tput.** and **Power** measurements represent the average (Avg.) power consumption, while executing the target solutions for 1,000 inference iterations.

Table 4: Comparison between AQ2PNN and SOTA works.

Model (Datasets)	Metrics	Tput. (fps)	Comm. (MiB)	Power (W)	Efficientcy (fps/W)
Small Size Model					
LeNet5 (MNIST)	Falcon	26.316	2.29	133 × 3	0.065354
	AQ2PNN (16-bit)	**16.68**	**0.95**	**7.2 × 2**	**1.158333**
AlexNet (MNIST/ CIFAR10)	Falcon	9.091	4.02	139 × 3	0.021801
	AQ2PNN (16-bit)	**6.081**	**1.2**	**7.4 × 2**	**0.410878**
Medium Size Model					
VGG16 (CIFAR10)	Falcon	0.694	40.45	185 × 3	0.001250
	CryptGPU	0.467	56.20	289 × 2	0.000807
	AQ2PNN (16-bit)	**0.352**	**28.87**	**7.7 × 2**	**0.022857**
Large Size Model					
ResNet50 (ImageNet)	Cryptflow	0.039	6900	178 × 2	0.000110
	CryptGPU	0.107	3080	306 × 2	0.000175
	AQ2PNN (16-bit)	**0.071**	**1120**	**7.7 × 2**	**0.004610**
VGG16 (ImageNet)	CryptGPU	0.106	2750	315 × 2	0.000168
	AQ2PNN (16-bit)	**0.038**	**1410**	**7.7 × 2**	**0.002468**

All solutions adhere to the platform configurations specified in the original papers. For example, the CryptGPU platform's configuration includes a single NVIDIA Tesla V100 GPU with 16 GB of GPU memory, 8 Intel Xeon E5-2686 v4 (2.3 GHz) CPUs, and 61 GB of RAM [51].

We show the comparison results in Tab. 4, that AQ2PNN outperforms other SOTA works in terms of energy efficiency for all small, medium, and large-size datasets. In the **small-size model** comparison, only Falcon [53] reported the corresponding results, where the efficiency of AQ2PNN is ~ 18× higher than Falcon. For **medium-size models**, AQ2PNN achieves 18.3× and 28.3× higher efficiency than Falcon [53] and CryptGPU [51], respectively. In the **large-size model** comparison, Falcon is no longer applicable, while Cryptflow [32] and CryptGPU [51] reported results. For ResNet50 (ImageNet), the efficiency of AQ2PNN is 41.9× and 26.3× higher than Cryptflow [32] and CryptGPU [51], respectively.

Besides, we also find that the efficiency advantage of AQ2PNN over CryptGPU [51] in VGG16 (ImageNet) is reduced to 14.7×. Upon analysis, we find that VGG16 contains more max pooling layers than ResNet50, leading to performance degradation. Further details can be found in Sec 6.5. It is worth noting that our proposed AQ2PNN is built on an FPGA setup, which only offers a comparable throughput, albeit slightly lower, with much-saved power and cost. As expected, in ResNet50 (ImageNet) evaluation, the achievable throughput of AQ2PNN is ~ 66% of that in CryptGPU [51], while the batch size is 1. This is mainly due to the clock frequency (200 MHz) of our adopted ZCU104 FPGA (cost ~ $1,500) is much lower than that (1.23~1.38 GHz) of the NVIDIA Tesla V100 GPU (cost ~ $10,000). Overall, AQ2PNN has significantly improved energy efficiency (about 14.7× ~ 41.9×) and cost compared to all previous works.

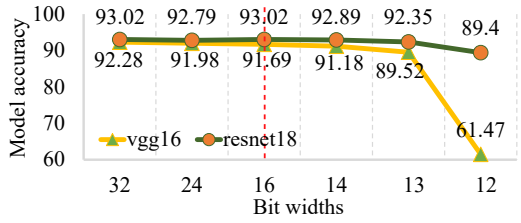

Figure 10: CIFAR10 Accuracy (%) with different bit-widths

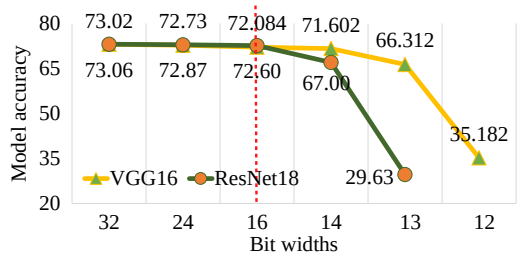

Figure 11: ImageNet Accuracy (%) with different bit-widths

6.2 Communication Overhead Analysis

Our experimental results in Tab. 4 demonstrate that AQ2PNN significantly reduces the communication data volume across different model sizes. For **small-size models**, we observe a reduction of 2.41× and 3.35× for LeNet5 and AlexNet. For **medium-size models**, AQ2PNN reduces communication data volume by 1.4× and 1.94×, compared to Falcon [53] and CryptGPU [51], respectively. In the **large-size models**, AQ2PNN reduces the communication data volume by 2.75× and 1.95× for ResNet50 and VGG16 architectures compared to CryptGPU [51], and by 6.16× for ResNet50 inference compared to Cryptflow [32].

Trade-off analysis: We analyze ResNet18 and VGG16 to explore the trade-off between quantized model accuracy and bit-width selection for ABReLU. The accuracy of models with different hyperparameters is obtained through a retraining process. The flexibility of selecting a bit-width provides a great trade-off between model utility and secure inference communication and computation. Fig. 10 and Fig. 11 show the model accuracy for ResNet18 and VGG16 models on CIFAR10 and ImageNet, with different output bit-widths selected for 2PC-Conv2D or 2PC-FC output and ABReLU input (quantization scheme refers to Fig. 9 (c)), from which we can observe that our proposed hardware-friendly quantization introduces trivial accuracy drop of only ~ 1%, when reducing the number of bit-width from 32 to 16. If we push the accuracy degradation to 6%, we only need to use 14-bit for the ABReLU function input. The "sweet spot" is identified as 16-bit or 14-bit in Fig. 10 and Fig. 11.

6.3 Operator-wise Performance Profiling

To further imporve the performance of AQ2PNN, we also apply operator-wise profiling. Taking ResNet50 (ImageNet) as an example, its architecture consists of 16 building blocks, to demonstrate the operator-wise performance of AQ2PNN, we select the operators

Table 5: Operator-wise performance profiling

bit width	Latency(ms)			Comm. (MiB)
	2PC-Conv2D-6	ABReLU-6	2PC-BNReQ-6	
32	42.76	140.01	13.87	36.92
16	40.12	65.83	10.65	18.46

Table 6: Validation set accuracy for ResNet18, ResNet50, and VGG16 over the ImageNet dataset using Max pooling vs. Average pooling (retrained).

Model	ResNet18	ResNet50	VGG16
Average Pooling	65.234	70.42	68.24
Max Pooling	72.872	77.47	72.73

within the 6^{th} building block as a case study, which includes operators 2PC-Conv2D-6, ABReLU-6, and 2PC-BNReQ-6. We measure the latency (ms) and communication data volume (MiB) improvement by AQ2PNN in Tab. 5, for different operators. The benefits of our proposed adaptive quantization schemes (Fig. 9 (c)) on ABReLU are significant. For example, when the bit-width is reduced by 2×, the latency of ABReLU also decreases by ~ 2×. This is because the bit-width in our proposed scheme, is directly related to the communication data volume of the ABReLU. The total communication overhead also includes the exchange of $[\![E]\!]$ in 2PC-Conv2D-6 that is also related to the bit-width. Therefore, the overall communication overhead is reduced by 2× as well. In 2PC-Conv2D-6, the primary overhead comes from computation, so the improvement in latency is not that significant. Similarly, the improvements in 2PC-BNReQ-6 are also small, as this operator does not require communication and can be completed using the AS-ALU. Thus, we can conclude that quantization can help ABReLU to reduce execution time by ~ 2×.

6.4 Scalability Evaluation

We explore the scalability of AQ2PNN regarding the model size in Tab. 4. First, the model size is related to the depth of the model. We compare AlexNet and VGG16 for inferring CIFAR10, both with the same input image size of 32 × 32. VGG16 has 2.6× more layers than AlexNet, resulting in a 17.27× reduction in throughput and a 24× increase in communication data volume. We compare the impact of scaling the input image size on AQ2PNN model inference, using the same architecture. When the input image size is increased by 49×, the communication overhead also increases by ~ 49×, and the throughput only decreases by 9.26×. However, expanding the model regarding input image size does not result in a significant throughput loss. This is because the number of handshaking remains the same. Although the communication data volume increases, the continuous transmission and computation combined with the increased communication and computation can mitigate such loss.

6.5 Optimization and Trade-off

We explore further optimization of AQ2PNN throughput, by modifying the model structure. The most straightforward method is to replace the Max pooling layer with the Average pooling layer. Since Max pooling introduces more communication and computational

Table 7: The model accuracy (%) and throughput (Tput.) for ImageNet on ResNet18 models with different output bits lengths applying Max Pooling and Avg Pooling.

Bits	Max pooling			Average pooling		
	Top-1 (%)	Tput. (fps)	Comm. (MiB)	Top-1 (%)	Tput. (fps)	Comm. (MiB)
32	73.06	0.157	894	65.23	86.48	618
24	72.87	0.198	520	64.79	86.16	361
16	72.60	0.243	246	64.93	86.30	172
14	67.00	0.276	194	54.04	78.64	136
12	29.63	0.311	147	19.86	40.33	104

Table 8: The model accuracy (%) and throughput (Tput.) for ImageNet on VGG16 models with different output bits lengths applying Max Pooling and Avg. Pooling.

Bits	Max pooling			Average pooling		
	Top-1 (%)	Tput. (fps)	Comm. (MiB)	Top-1 (%)	Tput. (fps)	Comm. (MiB)
32	73.02	0.030	5216	68.24	0.040	3145
24	72.73	0.033	3015	68.27	0.041	1823
16	72.08	0.038	1412	68.17	0.045	858
14	71.6	0.043	1104	66.64	0.050	673
12	35.18	0.049	835	11.37	0.061	809

cost in the 2PC-DNN scenario [53], while Average pooling can be completed using only AS-ALU without communication, users can choose different operations for model utility and inference throughput. Tab. 6 shows the retraining results for the 16-bit quantization on ImageNet by replacing all Max pooling layers with Average pooling.

For the ResNet18/50 model with one Max pooling layer for the input feature maps, the retraining results show that the single layer change leads to a $\sim$ 7% accuracy drop. In the VGG16 model that uses more Max pooling layers processing intermediate results, replacing them all with Average pooling also leads to a 2.61%, and 4.49% accuracy drop, respectively.

Tab. 7 and Tab. 8 show ImageNet results for ResNet18 and VGG16 on accuracy, communication data volume, and throughput for different representations. On ResNet18, the general model accuracy (Top-1 and Top-5) slightly decreases when we reduce the number of bit representations, while the throughput increases. Note that using the proposed AQ2PNN (16-bit), the Top-1 accuracy drop is within $\sim$ 0.94% compared to the 32-bit baseline. By reducing the output to 16-bit, the throughput is $\sim$ 35.2% and $\sim$ 20.1% higher than the 32-bit implementation on ResNet18 and VGG16 using Max pooling, respectively. The sweat spot locates at 16-bit, i.e., there is a small accuracy degradation but significant speedup. While the number of bit representations is reduced, the communication overhead decreases accordingly. On ResNet18 (ImageNet) and VGG16 (ImageNet) with Max pooling, the communication data volume reduction is $\sim$ 3.6× and 3.7×, respectively, when we reduce the number of bits to 16.

Max pooling support is critical for large and complex tasks in 2PC-DNN. We observe from Tab. 6, 7 and 8, that there is a significant accuracy loss (up to 7.67% for ResNet18 on ImageNet using 16 bits), if we change the Max pooling to Average pooling. Our observation aligns well with the common knowledge and practice, that DNNs (e.g., VGG series and ResNet series) prefer Max pooling over Average pooling. We also observe that the throughput overhead for all the listed bit representations are large. Note that it is a common practice to trade significant throughput for even 0.5% accuracy in developing a deep learning system [1, 34, 55, 56]. With our proposed adaptive quantization scheme, a well-chosen bit-width (e.g., 16 bits) for the intermediate outputs can achieve a similar inference time with Max pooling, compared to using only Average pooling, while still maintaining high accuracy.

7 CONCLUSION

This paper presents AQ2PNN, an end-to-end two-party framework that enables privacy-preserving deep learning on FPGAs. AQ2PNN targets the root performance bottleneck of SOTA solutions and constructively applies an adaptive quantization scheme to generate lightweight 2PC-DNN models in the ciphertext domain. AQ2PNN also employs a novel ABReLU scheme and associates it with adaptive quantization profiling, to significantly reduce the communication overhead. All the proposed methods are deployed on FPGA and evaluated with prevalent datasets and model architectures. We leverage the fine-grained reconfigurability of FPGA devices for acceleration purposes. The experimental results demonstrate that AQ2PNN outperforms all SOTA GPU-based solutions in terms of energy efficiency up to 29×, with similar or higher throughput and accuracy.

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation under Grants 2153690, 2319962, 2239672, 2326597, 2247891, 2247892, and 2247893.

REFERENCES

[1] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. 2019. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 715–731.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.

[3] Jeff Barnes. 2015. Azure machine learning. *Microsoft Azure Essentials. 1st ed, Microsoft* (2015).

[4] Donald Beaver. 1992. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO'91: Proceedings 11*. Springer, 420–432.

[5] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '95)*. Springer-Verlag, Berlin, Heidelberg, 97–109. http://dl.acm.org/citation.cfm?id=646760.706018

[6] Amos Beimel. 2011. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*. Springer, 11–46.

[7] Ekaba Bisong. 2019. *Building machine learning and deep learning models on Google cloud platform*. Springer.

[8] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society, 313–313.

[9] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[10] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799* (2018).

[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 409–437.

[12] Tung Chou and Claudio Orlandi. 2015. The simplest protocol for oblivious transfer. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 40–58.

[13] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. 2001. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*. Springer, 280–300.

[14] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 142–156.

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

[16] Juan Garay, Berry Schoenmakers, and José Villegas. 2007. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*. Springer, 330–342.

[17] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20 (2007), 51–83.

[18] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Stanford university.

[19] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[21] Kai Huang, Mehmet Gungor, Xin Fang, Stratis Ioannidis, and Miriam Leeser. 2019. Garbled circuits in the cloud using fpga enabled nodes. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.

[22] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure {Two-Party} Computation Using Garbled Circuits. In *20th USENIX Security Symposium (USENIX Security 11)*.

[23] Mitsuru Ito, Akira Saito, and Takao Nishizeki. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 72, 9 (1989), 56–64.

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[25] Ameet V Joshi. 2020. Amazon's machine learning toolkit: Sagemaker. In *Machine Learning and Artificial Intelligence*. Springer, 233–243.

[26] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*. 1651–1669.

[27] Seny Kamara, Payman Mohassel, and Mariana Raykova. 2011. Outsourcing Multi-Party Computation. *IACR Cryptology ePrint Archive* 2011 (2011), 272.

[28] Joe Kilian. 1988. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 20–31.

[29] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[32] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[34] Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2021. CSCNN: Algorithm-hardware Co-design for CNN Accelerators using Centrosymmetric Filters. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 612–625.

[35] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 619–631.

[36] Qian Lou and Lei Jiang. 2021. HEMET: a homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International conference on machine learning*. PMLR, 7102–7110.

[37] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*. 2505–2522.

[38] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 161–174.

[39] Jianqiao Mo, Jayanth Gopinath, and Brandon Reagen. 2023. HAAC: A Hardware-Software Co-Design to Accelerate Garbled Circuits. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[40] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.

[41] P. Mohassel and Y. Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. 19–38. https://doi.org/10.1109/SP.2017.12

[42] Thierry Moreau, Tianqi Chen, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. VTA: an open hardware-software stack for deep learning. *arXiv preprint arXiv:1807.04188* (2018).

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[44] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. {ABY2.0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation. In *30th USENIX Security Symposium (USENIX Security 21)*. 2165–2182.

[45] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.

[46] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. 2015. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 896–902.

[47] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[48] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[49] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

[50] Oleg Sémery. 2021. *PyTorchCV Library*. https://pypi.org/project/pytorchcv/.

[51] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1021–1038.

[52] Vinod Vaikuntanathan and Prashant Nalini Vasudevan. 2015. Secret sharing and statistical zero knowledge. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 656–680.

[53] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229* (2020).

[54] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2022. IncShrink: architecting efficient outsourced databases using incremental mpc and differential privacy. In *Proceedings of the 2022 International Conference on Management of Data*. 818–832.

[55] Erwei Wang, James J Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter YK Cheung, and George A Constantinides. 2019. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 1–39.

[56] Xingbin Wang, Boyan Zhao, Rui Hou, Amro Awad, Zhihong Tian, and Dan Meng. 2021. NASGuard: a novel accelerator architecture for robust neural architecture search (NAS) networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 776–789.

[57] Pierre-Francois Wolfe, Rushi Patel, Robert Munafo, Mayank Varia, and Martin Herbordt. 2020. Secret sharing MPC on FPGAs in the datacenter. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 236–242.

[58] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[59] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. 2021. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*. PMLR, 11875–11886.

[60] Xun Yi, Russell Paulet, and Elisa Bertino. 2014. Homomorphic encryption. In *Homomorphic encryption and applications*. Springer, 27–46.