ALKAID: Accelerating Three-Party Boolean Circuits by Mixing Correlations and Redundancy

Ye Dong, Xudong Chen, Xiangfu Song✉, Yaxi Yang, Wen-jie Lu, Tianwei Zhang, Jianying Zhou, Jin-Song Dong

***Abstract*—Secure three-party computation (3PC) with semi-honest security under an honest majority offers notable efficiency in computation and communication; for Boolean circuits, each party sends a single bit for every AND gate, and nothing for XOR. However, round complexity remains a significant challenge, especially in high-latency networks. Some works can support multi-input AND and thereby reduce online round complexity, but they require *exponential* communication for generating the correlations in either preprocessing or online phase. How to extend the AND gate to multi-input while maintaining high correlation generation efficiency is still not solved.**

To address this problem, we propose a round-efficient 3PC framework ALKAID for Boolean circuits through improved multi-input AND gate. By mixing correlations and redundancy, we propose a concretely efficient correlation generation approach for small input bits $N < 4$ and shift the correlation generation to the preprocessing phase. Building on this, we create a round-efficient AND protocol for general cases with $N > 4$. Exploiting the improved multi-input AND gates, we design fast depth-optimized parallel prefix adder and share conversion primitives in 3PC, achieved with new techniques and optimizations for better concrete efficiency. We further apply these optimized primitives to enhance the efficiency of secure non-linear functions in machine learning. We implement ALKAID and extensively evaluate its performance. Compared to state of the arts like ABY3 (CCS'2018), Trifecta (PoPETs'2023), and METEOR (WWW'2023), ALKAID enjoys $1.5\times$–$2.5\times$ efficiency improvements for boolean primitives and non-linear functions, with better or comparable communication.

I. INTRODUCTION

Secure multiparty computation (MPC) enables two or more parties to compute a function on their joint data. It ensures that no party can learn anything about other's inputs, except what can be inferred from outputs [1–3]. MPC constructions have been applied in various real-world scenarios, e.g., financial analysis [4, 5], medical data computation [6], and machine learning [7–10]. Specifically, the 3-party with honest majority setting is one of the most promising solutions for practical privacy-preserving applications due to its attractive balance between efficiency and security, and it has been widely adopted in many privacy-preserving applications, including machine/deep learning [9, 11, 12], decision tree [13–15], graph analysis [16], and private inference of large language models [17, 18], and has been included in mainstream open-sourced frameworks [10, 19].

Existing MPC protocols fall into two primary categories: Yao's garbled circuits (GC) [1, 2, 20] and secret sharing-based protocols [3, 21]. In the GC-based approach, the parties use a garbled Boolean circuit to evaluate the function with constant rounds of communication. On the other hand, secret sharing-based protocols secret-share inputs among parties and compute circuits layer-by-layer, requiring multiple rounds of interaction. GC-based protocols are advantageous in high-latency networks because of their constant round communication, but the communication overhead of exchanging garbled Boolean circuits significantly impacts their throughput. In contrast, secret sharing-based protocols have lightweight communication costs and can be highly parallelized [21].

The need for multiple rounds of communication to compute Boolean gates AND is the main performance bottleneck of secret sharing-based protocols, even for the cheapest 3PC solutions. Those existing works are primarily designed for secure 2-input AND gate, and circuit optimizations and parallelization are typically aligned with this designation, leading to at least $\log_2(\ell)$ rounds of online communication for circuits of depth ℓ. This round complexity is particularly problematic for real-time applications in high-latency networks, where excessive communication rounds can severely restrict the throughput. If a *multi-input* AND gate could be securely computed with comparable (online) round complexity to 2-input ones, the circuit depth will be further reduced, resulting in significant online time savings. Consequently, *it is crucial to propose a round-efficient approach to compute multi-input* AND *gate securely, to improve the performance, especially the online time in high-latency networks, for secret sharing-based protocols.*

Existing Solutions. In recent years, there has been increasing research [22–25] aiming to reduce the *online round complexity* of secret sharing-based protocols. They exploit correlations to design secure *multi-input* AND gates, which are applied to construct depth-optimized circuits for better online efficiency. At a high-level, given N inputs $\{x_j\}_{j=1}^N$, a multi-input AND follows:

$$\begin{aligned}\bigwedge_{j=1}^{N} x_j &= \bigwedge_{j=1}^{N} \left((x_j \oplus r_j) \oplus r_j\right) \\ &= \bigoplus_{\mathcal{T}\subseteq\{1,\ldots,N\}} \Big(\bigwedge_{j\notin\mathcal{T}} \underbrace{(x_j \oplus r_j)}_{m_j} \Big) \wedge \Big(\bigwedge_{k\in\mathcal{T}} r_k \Big)' \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \underbrace{\qquad\qquad}_{r_{\mathcal{T}}}\end{aligned} \tag{1}$$

Ye Dong and Jin-Song Dong are with School of Computing, National University of Singapore, Singapore. (email: {dongye,dcsdjs}@nus.edu.sg)

Xudong Chen is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. (email: chenxudong@iie.ac.cn)

Xiangfu Song and Tianwei Zhang are with College of Computing and Data Science, Nanyang Technological University, Singapore (email: {xiangfu.song,tianwei.zhang}@ntu.edu.sg)

Yaxi Yang and Jianying Zhou are with Singapore University of Technology and Design, Singapore (email: {yaxi_yang,jianying_zhou}@sutd.edu.sg)

Wen-jie Lu is with TikTok Inc, Singapore. (email: luwenjie@tiktok.com)

✉Xiangfu Song is the corresponding author.

TABLE I: Existing solutions and ALKAID for ANDing ℓ bits. ● represents secret sharing-fully schemes and ○ denotes dealer-based approaches. Communication is measured by the bits sent by all parties. N is set as the maximum recommended by each work. Prep. is for preprocessing and Comm. denotes communication. κ is the security parameter with $\kappa = 128$.

Framework	# Party	Type	N	ℓ-bit AND			
				Round [Prep.]	Comm. [Prep.]	Comm. [Online]	Round [Online]
[23]	$(2+1)$PC	○	$N \leq 9$	1	$\approx 125\ell$	$\ell/4$	$\log_9(\ell)$
Trifecta [25]	$(2+1)$PC	○	$N \leq 8$	0	0	$\approx 71\ell$	$\log_8(\ell)$
ABY3 [9]	3PC	●	$N = 2$	0	0	3ℓ	$\log_2(\ell)$
METEOR [24]	3PC	●	$N \leq 4$	2	11ℓ	ℓ	$\log_4(\ell)$
ALKAID (Ours)	3PC	●	$N \leq 4$	1	2ℓ	2ℓ	$\log_4(\ell)$

where $\{r_j\}_{j=1}^N$ and $r_{\mathcal{T}} = \bigwedge_{k\in\mathcal{T}} r_k$ for $\forall \mathcal{T} \subseteq \{1, ..., N\}$ are kept secret, and $m_j = x_j \oplus r_j$ is revealed. After generating all correlated randomness $r_{\mathcal{T}}$, the parties can compute $\bigwedge_{j=1}^N x_j$ securely in one round of communication. As m_j is public, the key challenge is computing $\{r_{\mathcal{T}}\}_{\mathcal{T}\subseteq\{1,...,N\}}$ securely and efficiently. Based on the generation methods, existing works can be categorized into two approaches:

i) *Dealer-based* protocols [23, 25, 26] rely on a trusted dealer to generate and distribute correlated randomness. This requires the dealer to send $O(2^N)$ bits in one round and have full access to correlated randomness, so it necessarily assumes that the dealer is a trusted party, which might be unavailable in the real world.

ii) *Secret sharing-fully* schemes [22, 24] allow the parties to independently sample randomness r_j within their respective secret-sharing schemes. Correlated values $r_{\mathcal{T}}$ are computed by executing underlying 2-input AND protocols. This method not only incurs a communication of $O(2^N)$ bits but also introduces an additional round complexity of $\log_2(N)$. Unlike dealer-based approaches, secret sharing-fully methods ensure that no single party gets the full correlations in cleartext, making them more suitable when trusted dealer assumption is unavailable.

On the other hand, [22–24, 26] shift correlation generation to the preprocessing phase to reduce online costs. While these techniques do reduce online round complexity, *they still involve an expensive preprocessing communication complexity of $O(2^N)$ (and secret sharing-fully methods require preprocessing round complexity of $\log_2(N)$)*. To balance the preprocessing and online communication, they all restrict N to a specific bound, *i.e.*, $N \leq 4$. We summarize N limitations, concrete communication and round complexity for ANDing ℓ bits of existing works in Table I. Given the considerable correlation generation costs and the trust assumptions of current multi-input AND protocols, we ask the following question:

Can we reduce the preprocessing communication and round complexity for generating correlations of multi-input AND *while maintaining (comparable) current high online efficiency in the secret sharing-fully setting?*

To answer this question affirmatively, we propose ALKAID, a secure three-party computation framework against semi-honest adversaries. We mix the correlations of random masks and redundancy of replicated secret sharing to balance concrete preprocessing and online communication costs of N-input AND gates for $N \leq 4$. With its guidance, we design efficient 3PC protocols for depth-optimized circuits, including parallel prefix adder (PPA), share conversions, and bit extraction, which finally improve the efficiency of the non-linear function of secure neural network inference. In summary, ALKAID includes the following contributions:

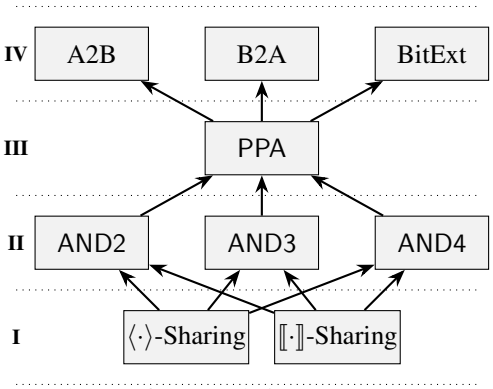

Fig. 1: Dependency of protocols in ALKAID. Layer **I** specifies the sharing semantics, layer **II** presents the $(N \leq 4)$-input AND protocols, layer **III** corresponds to our depth-optimized PPA, and layer **IV** contains the basic 3PC primitives.

- **Cost-Balanced N-Input AND with $N \leq 4$.** We propose practical 3-party protocols for N-input AND with $N \leq 4$. Our approach leverages correlations and redundancy of replicated secret sharing to balance preprocessing and online costs. The preprocessing and online phases merely require *one round of communication with $\leq$ 2 bits per party*, making it suitable for high-latency networks. Compared to prior works [9, 24–26], we achieve an improved trade-off between preprocessing and online costs.
- **Optimized PPA Circuit & Primitives.** We design an efficient protocol for depth-optimized parallel prefix adder (PPA) [22] in 3PC using our N-input AND gates. By exploiting the properties of operators within PPA, we propose specific optimizations to reduce online and preprocessing communication by at least $2\times$ and $1.5\times$, respectively. Since PPA facilitates a bundle of operations in secure computation, our depth-optimized PPA improves the efficiency of widely used 3PC primitives like Arithmetic-Boolean share conversions and bit extraction. We present the dependency of the protocols in Fig. 1.
- **Application & Evaluation.** We implement our protocols in C++ and integrate them into 3PC backend of SecretFlow-SPU [10] to provide our framework ALKAID, and apply it for activation functions of secure neural network inference, handling specific input and output conversions to facilitate its use in Boolean circuits. Experimental results demonstrate that ALKAID accelerates the online phase of 3PC and offers a more balanced trade-off between preprocessing and

TABLE II: Notation table.

Symbols	Descriptions
$\mathcal{P}_i$	party i in 3PC
x	lowercase letter denotes scalar
x_j	the j-th bit of x
$\oplus, \wedge$	bit-wise XOR and AND
$[\cdot]$	3-party linear secret sharing
$\langle\cdot\rangle$	2-out-of-3 replicated secret sharing
$[\![\cdot]\!]$	2-out-of-3 masked replicated secret sharing
AND2/ANDN	AND gate with 2/N inputs
$\mathcal{F}_f$	the ideal functionality of function f
PPA	Parallel Prefix Adder

online phases: ALKAID i) achieves a speedup of 1.5–2.5× compared to [9, 24, 25] for online efficiency of non-linear functions, and ii) even outperforms one of the cheapest solution ABY3 by at least 7% for end-to-end running time in secure evaluation of neural networks and GPT-2. The source code is available: https://github.com/CPS4AI/OpenAlkaid.

Organization. We first introduce the background and preliminary in § II. Then, we present our intuition and techniques for N-input AND gates in § III. Next, the constructions of depth-optimized PPA and 3PC primitives are illustrated in § IV. Afterward, we apply our techniques in the secure non-linear functions of neural network inference in § V. In § VI, we implement our framework and conduct experimental evaluations. Finally, we summarize the related works in § VII and conclude this work in § VIII.

II. Background & Preliminary

A. Notations

We summarize the main notations in Table II. By default, we use the above notations for Boolean sharing, and $[\cdot]^{\mathsf{A}}$, $\langle\cdot\rangle^{\mathsf{A}}$, and $[\![\cdot]\!]^{\mathsf{A}}$ are for the corresponding Arithmetic sharing.

B. Three-Party Computation

We introduce the secret sharing schemes and randomness generation procedures as follows.

1) Linear Secret Sharing (LSS, $[\cdot]$-Sharing): A secret value $x \in \mathbb{Z}_2$ is shared by three random values $r \xleftarrow{\$} \mathbb{Z}_2$, $r' \xleftarrow{\$} \mathbb{Z}_2$, and $r'' = x \oplus r \oplus r'$, where $\mathcal{P}_0$ gets $[x]_0 = r$, $\mathcal{P}_1$ gets $[x]_1 = r'$, and $\mathcal{P}_2$ obtains $[x]_2 = r''$. We refer to it as *Boolean Sharing*. When $x \in \mathbb{Z}_{2^\ell}$ with $\ell > 1$ (*e.g.*, $\ell = 64$), which support *Arithmetic* operations (*e.g.*, $+$, $-$, and $\cdot$) over $\mathbb{Z}_{2^\ell}$, this is *Arithmetic Sharing* and we use notation $[\cdot]^{\mathsf{A}}$.

XOR. For LSS, we use its secure XOR. Let $a, b, c \in \mathbb{Z}_2$ be public constants, $[x]$ and $[y]$ be two secret-shared inputs over $\mathbb{Z}_2$, $[ax \oplus by \oplus c]$ can be computed as $(a[x]_0 \oplus b[y]_0 \oplus c, a[x]_1 \oplus b[y]_1, a[x]_2 \oplus b[y]_2)$, where $\mathcal{P}_i$ can compute its share locally. When $a = 1, b = 1$, and $c = 0$, we get $[x \oplus y]$.

2) Replicated Secret Sharing (RSS, $\langle\cdot\rangle$-Sharing): RSS is constructed on LSS with redundancy. Given secret $x \in \mathbb{Z}_2$, it is also shared by three random values $[x]_0, [x]_1, [x]_2 \in \mathbb{Z}_2$ with $x = [x]_0 \oplus [x]_1 \oplus [x]_2$. $\mathcal{P}_i$ gets two of the three random values as its RSS share, *a.k.a.*, $\langle x\rangle_i = ([x]_i, [x]_{i+1})$.

XOR & AND. Let $a, b, c \in \mathbb{Z}_2$ be public constants, $\langle x\rangle$ and $\langle y\rangle$ be two secret-shared inputs. $\langle ax \oplus by \oplus c\rangle$ can be computed as $(a[x]_0 \oplus b[y]_0 \oplus c, a[x]_1 \oplus b[y]_1, a[x]_2 \oplus b[y]_2)$, where P_i can compute its share locally. When $c_1 = 1, c_2 = 1$, and $c_3 = 0$, we get $\langle x \oplus y\rangle$. On the other hand, computing secure $\langle x \wedge y\rangle$ requires communication among the parties: i) $\mathcal{P}_i$ first computes $[z]_i = [x]_i[y]_i \oplus [x]_{i+1}[y]_i \oplus [x]_i[y]_{i+1}$ locally, ii) Parties then perform *re-sharing* by letting $\mathcal{P}_i$ send $[z']_i = [\alpha]_i \oplus [z]_i$ to $\mathcal{P}_{i-1}$, where $[\alpha]_0 \oplus [\alpha]_1 \oplus [\alpha]_2 = 0$ and are generated by functionality $\mathcal{F}_{\mathsf{ZeroShr}}$ (c.f., § II-C). In the end, $([z']_0, [z']_1), ([z']_1, [z']_2), ([z']_2, [z']_0)$ form $\langle x \wedge y\rangle$.

3) Masked Replicated Secret Sharing (MRSS, $[\![\cdot]\!]$-Sharing): In MRSS [24], $x \in \mathbb{Z}_2$ is shared as $[\![x]\!] = (m_x, \langle r_x\rangle)$ where: i) r is random sampled from $\mathbb{Z}_2$ and RSS-shared among parties, and ii) $m_x = x \oplus r_x$ is revealed to all parties. As no party knows r_x in clear (except the secret owner), revealing m_x will not introduce any leakage of x. MRSS utilizes the same secret-randomness $\langle r\rangle$ to protect inputs and generate correlations.

XOR & AND. Let a, b, and c be public constants, $[\![x]\!]$ and $[\![y]\!]$ be two secret inputs, where $[\![x]\!] = (m_x, \langle r_x\rangle)$ and $[\![y]\!] = (m_y, \langle r_y\rangle)$. It is easy to see that the parties can compute $(am_x \oplus bm_y \oplus c, a\langle r_x\rangle \oplus b\langle r_y\rangle)$ locally. Also, we get $[\![x \oplus y]\!]$ with $a = 1, b = 1$, and $c = 0$. When computing MRSS-based AND2 $[\![x \wedge y]\!]$, the parties work in a *preprocessing/online* paradigm: i) Preprocessing: parties compute $\langle r_{xy}\rangle = \langle r_x\rangle \wedge \langle r_y\rangle$ using RSS-based AND2. ii) Online: parties compute $\langle m_z\rangle = m_x m_y \oplus m_x\langle r_y\rangle \oplus m_y\langle r_x\rangle \oplus \langle r_{xy}\rangle \oplus \langle r_z\rangle$, reveal m_z to all parties, and set the result as $(m_z, \langle r_z\rangle)$, where $\langle r_z\rangle$ is random sampled and $\langle\cdot\rangle$-shared. Dong *et al.* [24] proposed MRSS-based ANDN gates with improved online efficiency with exponential preprocessing communication.

C. Security Model & Ideal Functionalities

Following prior works [9, 24], ALKAID is secure against a semi-honest (*a.k.a.*, honest-but-curious) adversary that corrupts no more than one of the three computing parties. Semi-honest means such an adversary will follow the protocol specifications, but may try to learn other's private information.

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

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

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

Ideal Functionalities. We use several well-established functionalities. The implementations of $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{AND2}}$ and $\mathcal{F}^{[\![\cdot]\!]}_{\mathsf{AND2}}$ are as § II-B2 and § II-B3, respectively. Below, we introduce the randomness generation functionalities.

Randomness Generation. We rely on that parties can generate fresh random elements on demand, without any interaction beyond a short initial phase. Concretely, i) all parties have a common key, ii) $\mathcal{P}_i$ obtains random keys $(\mathsf{key}_i, \mathsf{key}_{i+1})$, such that each pair $(\mathcal{P}_i, \mathcal{P}_{i+1})$ have the common key_{i+1}. We use the

following pseudorandom function (PRF)-based randomness generation procedures from [9, 27]:

- Π_{Rand}: Given a fresh id, $\mathcal{P}_i$ lets $[r]_i = \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id})$ and $[r]_{i+1} = \mathsf{PRF}_{\mathsf{key}_{i+1}}(\mathsf{id})$ so that $([r]_0, [r]_1, [r]_2)$ form valid RSS shares of random r;
- Π_{RandComm}: Each $\mathcal{P}_i$ lets $r = \mathsf{PRF}_{\mathsf{key}}(\mathsf{id})$ for the fresh id, so that all parties have a common random r;
- $\Pi_{\mathsf{RandPair}_{i,j}}$: Each pair of $\mathcal{P}_i$ and $\mathcal{P}_j$ locally computes $r = \mathsf{PRF}_{\mathsf{key}}(\mathsf{id})$, where key is their common key. Therefore, $\mathcal{P}_i$ and $\mathcal{P}_j$ have common random r.
- Π_{ZeroShr}: Each $\mathcal{P}_i$ computes $[\alpha]_i = \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id}) \oplus \mathsf{PRF}_{\mathsf{key}_{i+1}}(\mathsf{id})$. In this way, $[\alpha]_0 \oplus [\alpha]_1 \oplus [\alpha]_2 = 0$.

PRF is defined as $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\ell$, where κ is the computational security parameter, *a.k.a.*, the length of the keys. When requiring a random bit-string, we denote it as $r \xleftarrow{\$} \mathbb{Z}_2^\ell$. Similarly, when needing an ℓ-bits random value from $\mathbb{Z}_{2^\ell}$, we interpret it as $r \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ (and $\oplus$ can be replaced by $+$ or $-$ modulo 2^ℓ).

Note: To achieve information-theoretic security, the parties can locally sample uniform random values and communicate with others to generate correlated randomness. More details can be referred to [27, 28].

D. Preprocessing/Online Paradigm

The preprocessing/online paradigm is widely used in designing practical MPC protocols. Recent works [22–24, 29] make use of input-independent but function-dependent preprocessing phases to further improve online efficiency. This approach necessitates that the parties know the function during preprocessing, which is often met with basic MPC primitives (*e.g.*, share conversions) and some real-world applications like secure neural network inference. We employ an input-independent but function-dependent preprocessing phase.

III. Intuition & Technique for ANDN

We first revisit the RSS and MRSS-based approaches for ANDN, then present the intuitions and techniques, and finally show our concrete ANDN design.

A. Revisiting RSS & MRSS-based Approaches

We revisit the secure ANDN of both approaches and analyze their respective preprocessing and online complexity.

1) RSS-based Approach: RSS-based works [9, 11, 27, 28] use the *redundancy* to compute AND2 securely. As discussed in § II-B2, each AND2 needs one re-sharing to maintain the consistency of sharing semantics for subsequent computation. Re-sharing requires one party to send 1 bit in 1 round. When computing the ANDN gate, these works exploit AND2 and *binary tree*-based optimization to represent ANDN as a binary tree of depth $\log_2(N)$, so parties can compute AND2 gates layer-by-layer: in the i-th layer, the parties compute $N/2^i$ AND2 gates in parallel (*a.k.a.*, 1 round) with communicating $3N/2^i$ bits totally. This method requires all parties to send around $3N$ bits in $\log_2(N)$ rounds.

Protocol Π_{AND4}

Inputs: For $i \in \{0,1,2\}$, $\mathcal{P}_i$ inputs $[\![x]\!]_i = (m_x, \langle r_x \rangle_i)$, $[\![y]\!]_i = (m_y, \langle r_y \rangle_i)$, $[\![u]\!]_i = (m_u, \langle r_u \rangle_i)$, and $[\![v]\!]_i = (m_v, \langle r_v \rangle_i)$.
Outputs: $\mathcal{P}_i$ gets $[\![z]\!]_i = (m_z, \langle r_z \rangle_i)$ with $z = x \wedge y \wedge u \wedge v$.

Preprocessing Phase:

1) Parties compute correlated randomness $\langle r_{xy} \rangle = \langle r_x \rangle \wedge \langle r_y \rangle$ and $\langle r_{uv} \rangle = \langle r_u \rangle \wedge \langle r_v \rangle$ by using $\mathcal{F}_{\mathsf{AND2}}^{\langle \cdot \rangle}$.
2) Parties invoke $\mathcal{F}_{\mathsf{ZeroShr}}$ to let $\mathcal{P}_i$ obtain $[\alpha]_i$ with $[\alpha]_0 \oplus [\alpha]_1 \oplus [\alpha]_2 = 0$.
3) Parties invoke $\mathcal{F}_{\mathsf{Rand}}$, such that $\mathcal{P}_i$ gets random share $\langle r_z \rangle_i = ([r_z]_i, [r_z]_{i+1})$.

Online Phase:

1) Parties compute $\langle xy \rangle = m_x m_y \oplus m_x \langle r_y \rangle \oplus m_y \langle r_x \rangle \oplus \langle r_{xy} \rangle$ and $\langle uv \rangle = m_u m_v \oplus m_u \langle r_v \rangle \oplus m_v \langle r_u \rangle \oplus \langle r_{uv} \rangle$ locally, such that $\mathcal{P}_i$ gets $\langle xy \rangle_i = ([xy]_i, [xy]_{i+1})$ and $\langle uv \rangle_i = ([uv]_i, [uv]_{i+1})$.
2) $\mathcal{P}_i$ locally computes $[z]_i = ([xy]_i \wedge [uv]_i) \oplus ([xy]_{i+1} \wedge [uv]_i) \oplus ([xy]_i \wedge [uv]_{i+1})$.
3) $\mathcal{P}_i$ computes $[m_z]_i = [z]_i \oplus [\alpha]_i \oplus [r_z]_i$ and sends $[m_z]_i$ to $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$, such that each party can reconstruct $m_z = \oplus_{i=0}^2 [m_z]_i$.
4) $\mathcal{P}_i$ sets $[\![z]\!]_i = (m_z, \langle r_z \rangle_i)$ as outputs.

Fig. 2: Protocol for secure ANDing 4 bits.

2) MRSS-based Approach: MRSS-based approaches work in the preprocessing/online paradigm and focus on optimizing online complexity [24]. As discussed in §II-B3, parties compute correlations in the preprocessing phase and consume them in the online phase. This approach can be generally extended to ANDN: Given N secret values $\{[\![x_j]\!] = (m_{x,j}, \langle r_{x,j} \rangle)\}_{j=1}^N$, parties first generate all correlated randomness $\langle r_{\mathcal{T}} \rangle = \wedge_{k \in \mathcal{T}} \langle r_{x,k} \rangle$ for $\mathcal{T} \subseteq \{1, \ldots, N\}$ using RSS-based AND2, then compute $\langle m_z \rangle = \bigoplus_{\mathcal{T} \subseteq \{1,\ldots,N\}} (\wedge_{j \notin \mathcal{T}} m_j) \wedge \langle r_{\mathcal{T}} \rangle \oplus \langle r_z \rangle$, and finally reveal m_z to all parties. The result is $[\![z]\!] = (m_z, \langle r_z \rangle)$ with $z = \wedge_{j=1}^N x_j$. Asymptotically, this approach requires each party to send 1 bit in 1 round during the online phase, but with communicating $(2^N - N - 1)$ bits in $\log_2(N)$ rounds for the preprocessing. It sets $N \leq 4$ to balance preprocessing and online costs and constructs QuadTree for $N > 4$: all parties send N bits in $\log_4(N)$ rounds in online phase, with communicating $11N$ bits in 2 preprocessing rounds.

The analysis illustrates that RSS-based ANDN does not need any preprocessing communication, but it has higher online communication and round complexity than the MRSS-based approach. However, the preprocessing communication complexity of the MRSS-based method is exponential to N so we have to set $N \leq 4$.

B. Our Intuition & Technique

In MPC, we usually guarantee the sharing consistency of inputs and outputs for all gates to support subsequent computation. For example, when computing ANDN, the RSS-based method requires re-sharing to convert intermediate shares back into the RSS format, and the MRSS-based approach needs to reveal the masked values to get the MRSS-shared results. However, when we go beyond the *layer-by-layer* paradigm and look at more layers of gates as a whole, we observe

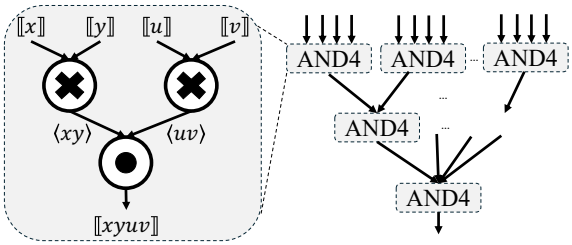

Fig. 3: The illustration of our N-input AND protocol. $\otimes$ denotes the modified MRSS-based AND_2, and $\odot$ denotes RSS-based AND_2. We assume $N = 4^n$ and only need to ensure the sharing consistency of the inputs and the final output.

that *it is sufficient to guarantee the sharing consistency of some necessary gates, but not all, to compute* ANDN *in 3PC*. As illustrated in Fig. 3, we modify the MRSS-based AND2 to generate RSS-shared intermediate results without communication, which support one more layer of RSS-based AND2 followed by one resharing. Therefore, we only need to guarantee the sharing consistency of inputs and final outputs in ANDing 4 bits, saving overhead for processing the sharing semantics of intermediate results.

With this insight, we make use of the correlations of MRSS and redundancy of RSS to compute ANDN for $N \leq 4$. Let $N = 4$, and the secret inputs $[\![x]\!] = (m_x, \langle r_x \rangle), [\![y]\!] = (m_y, \langle r_y \rangle), [\![u]\!] = (m_u, \langle r_u \rangle)$, and $[\![v]\!] = (m_v, \langle r_v \rangle)$ are MRSS-shared inputs. We build our protocol Π_{AND4} using the variants of AND2 of MRSS and RSS as follows:

- **Preprocessing:** In the preprocessing phase, parties first compute correlated randomness $\langle r_{xy} \rangle = \langle r_x \rangle \wedge \langle r_y \rangle$ and $\langle r_{uv} \rangle = \langle r_u \rangle \wedge \langle r_v \rangle$ using RSS-based AND2. Additionally, parties generate zero-sharing $([\alpha]_0, [\alpha]_1, [\alpha]_2)$ and RSS-shared random $\langle r \rangle$, where $\mathcal{P}_i$ has $[\alpha]_i$ and $\langle r \rangle_i$.
- **Online:** We present our mixing correlation and redundancy as follows. i) **Correlations:** instead of directly computing MRSS-shared results, we modify the AND2 of MRSS to output RSS-shared results utilizing the correlated randomness, *i.e.*, $\langle xy \rangle = m_x m_y \oplus m_x \langle r_y \rangle \oplus m_y \langle r_x \rangle \oplus \langle r_{xy} \rangle$, so that the parties can compute $\langle xy \rangle$ and $\langle uv \rangle$ locally (free of communication). ii) **Redundancy:** we observe that $\langle xy \rangle$ and $\langle uv \rangle$ are still of RSS-shared format, so we can exploit the *redundancy* of RSS to compute one AND2. Concretely, parties compute the 3-out-of-3 shares of $z = (xy) \wedge (uv)$ locally, where $\mathcal{P}_i$ holds $[z]_i$ and $z = [z]_0 \oplus [z]_1 \oplus [z]_2$. Thirdly, $\mathcal{P}_i$ computes and sends $[m_z]_i = [z]_i \oplus [\alpha]_i \oplus [r_z]_i$ to $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$, and all parties can reconstruct $m_z = \bigoplus_{i=0}^{2} [m_z]_i$ to set the result as $[\![z]\!] = (m_z, \langle r_z \rangle)$. In this way, we leverage mixing correlations and redundancy to compute AND4 with one round of communication.

Given the correctness of the MRSS and RSS-based AND2, it is easy to verify $z = x \wedge y \wedge u \wedge v$. As $m_z = \bigoplus_{i=0}^{2} [m_z]_i = \bigoplus_{i=0}^{2} ([z]_i \oplus [\alpha]_i \oplus [r_z]_i) = (\bigoplus_{i=0}^{2} [z]_i) \oplus (\bigoplus_{i=0}^{2} [\alpha]_i) \oplus (\bigoplus_{i=0}^{2} [r_z]_i) = z \oplus 0 \oplus r_z = z \oplus r_z$, we have $[\![z]\!] = (m_z, \langle r_z \rangle)$ as an valid MRSS sharing of $(x \wedge y \wedge u \wedge v)$. Our protocol Π_{AND4} is formulated as Figure 2.

1) Security Analysis: In protocol Π_{AND4}, we aim to compute the ideal functionality $\mathcal{F}_{\mathsf{AND4}}$: It takes four MRSS sharings $[\![x]\!]$, $[\![y]\!]$, $[\![u]\!]$, $[\![v]\!]$ with identifiers id_x, id_y, id_u, id_v, respectively, and outputs $[\![z]\!] = [\![x \wedge y \wedge u \wedge v]\!]$ corresponding to a new identifier id_z. We capture the security in Theorem 1 and give the proof in Appendix E.

Protocol Π_{AND2}

Inputs: For $i \in \{0, 1, 2\}$, $\mathcal{P}_i$ inputs $[\![x]\!]_i = (m_x, \langle r_x \rangle_i)$ and $[\![y]\!]_i = (m_y, \langle r_y \rangle_i)$.
Outputs: $\mathcal{P}_i$ gets $[\![z]\!]_i = (m_z, \langle r_z \rangle_i)$ with $z = x \wedge y$.

Preprocessing Phase:
1) Parties invoke $\mathcal{F}_{\mathsf{ZeroShr}}$ to let $\mathcal{P}_i$ obtain $[\alpha]_i$ with $[\alpha]_0 \oplus [\alpha]_1 \oplus [\alpha]_2 = 0$.
2) Parties invoke $\mathcal{F}_{\mathsf{Rand}}$, such that $\mathcal{P}_i$ gets random share $\langle r_z \rangle_i = ([r_z]_i, [r_z]_{i+1})$.

Online Phase:
1) Parties compute $\langle x \rangle = m_x \oplus \langle r_x \rangle$ and $\langle y \rangle = m_y \oplus \langle r_y \rangle$.
2) $\mathcal{P}_i$ locally computes $[z]_i = ([x]_i \wedge [y]_i) \oplus ([x]_{i+1} \wedge [y]_i) \oplus ([x]_i \wedge [y]_{i+1})$.
3) $\mathcal{P}_i$ computes $[m_z]_i = [z]_i \oplus [\alpha]_i \oplus [r_z]_i$ and sends $[m_z]_i$ to $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$, such that each party can reconstruct $m_z = \oplus_{i=0}^{2} [m_z]_i$.
4) $\mathcal{P}_i$ sets $[\![z]\!]_i = (m_z, \langle r_z \rangle_i)$ as outputs.

Fig. 4: Protocol for secure ANDing 2 bits.

Theorem 1 (Security of Π_{AND4})**.** *In the 3-party honest-majority setting, protocol Π_{AND4} securely realizes $\mathcal{F}_{\mathsf{AND4}}$ in the ($\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ZeroShr}}$)-hybrid model in the presence of a static semi-honest adversary $\mathcal{A}$ who corrupts at most one single party among three.*

2) Communication & Round Complexity: We summarize the communication and round complexity of protocol Π_{AND4} in Lemma 1 and give the proof as follows.

Lemma 1. *Protocol Π_{AND4} requires each party to send 2 bits in 1 round during the preprocessing phase, and sends 2 bits in 1 round during the online phase.*

Proof of Lemma 1.. In the preprocessing phase, the parties run RSS-based AND2 twice in parallel, which requires each party to send a total of 2 bits in 1 round. In the online phase, the parties reveal m_z by every party sending 1 bit to each of the other two parties, a total of 2 bits, in 1 round. □

Compared to the RSS-based solution [9, 27], although we incur some preprocessing costs, we reduce online communication (resp. round) by $1.5\times$ (resp. $2\times$) for ANDing 4 bits. Compared to the MRSS-based approach [24], we reduce preprocessing communication (resp. round) by $5.5\times$ (resp. $2\times$) and achieve the same online round complexity and comparable online communication. Our construction achieves a better trade-off of preprocessing and online complexity.

3) Optimizations for $N = 2$ and 3: When ANDing 2 or 3 MRSS-shared inputs, we make the following optimizations:

- **Protocol** Π_{AND2}: To achieve two-input AND functionality $\mathcal{F}_{\mathsf{AND2}}$ with reduced preprocessing costs, we convert the MRSS-shared values into $\langle x \rangle$ and $\langle y \rangle$ locally, and then utilize the redundancy of RSS to compute 3-out-of-3 shared $[x \wedge y]$. Finally, we convert the results into MRSS-shared $[\![x \wedge y]\!]$. Compared to [24], this method does not need any preprocessing communication but requires sending 2 bits

Protocol Π_{AND3}

Inputs: For $i \in \{0,1,2\}$, $\mathcal{P}_i$ inputs $[\![x]\!]_i = (m_x, \langle r_x\rangle_i)$, $[\![y]\!]_i = (m_y, \langle r_y\rangle_i)$, and $[\![u]\!]_i = (m_u, \langle r_u\rangle_i)$.
Outputs: $\mathcal{P}_i$ gets $[\![z]\!]_i = (m_z, \langle r_z\rangle_i)$ with $z = x \wedge y \wedge u$.

Preprocessing Phase:

1) Parties compute correlated randomness $\langle r_{xy}\rangle = \langle r_x\rangle \wedge \langle r_y\rangle$ by using $\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$.
2) Parties invoke $\mathcal{F}_{\mathsf{ZeroShr}}$ to let $\mathcal{P}_i$ obtain $[\alpha]_i$ with $[\alpha]_0 \oplus [\alpha]_1 \oplus [\alpha]_2 = 0$.
3) Parties invoke $\mathcal{F}_{\mathsf{Rand}}$, such that $\mathcal{P}_i$ gets random share $\langle r_z\rangle_i = ([r_z]_i, [r_z]_{i+1})$.

Online Phase:

1) Parties compute $\langle xy\rangle = m_x m_y \oplus m_x\langle r_y\rangle \oplus m_y\langle r_x\rangle \oplus \langle r_{xy}\rangle$ and $\langle u\rangle = m_u - \langle r_u\rangle$ locally, such that $\mathcal{P}_i$ gets $\langle xy\rangle_i = ([xy]_i, [xy]_{i+1})$ and $\langle u\rangle_i = ([u]_i, [u]_{i+1})$.
2) $\mathcal{P}_i$ locally computes $[z]_i = ([xy]_i \wedge [u]_i) \oplus ([xy]_{i+1} \wedge [u]_i) \oplus ([xy]_i \wedge [u]_{i+1})$.
3) $\mathcal{P}_i$ computes $[m_z]_i = [z]_i \oplus [\alpha]_i \oplus [r_z]_i$ and sends $[m_z]_i$ to $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$, such that each party can reconstruct $m_z = \oplus_{i=0}^{2}[m_z]_i$.
4) $\mathcal{P}_i$ sets $[\![z]\!]_i = (m_z, \langle r_z\rangle_i)$ as outputs.

Fig. 5: Protocol for secure ANDing 3 bits.

online. Besides, this results in aligned outputs resharing procedures with Π_{AND4} and Π_{AND3}, and will be helpful when we compute XOR between the results of Π_{AND2} and $\Pi_{\mathsf{AND4}}/\Pi_{\mathsf{AND3}}$. Looking ahead, we will use both Π_{AND2} and MRSS-based AND2 protocol of [24] in designing practical and fast depth-optimized PPA (c.f. § IV-A).

- **Protocol** Π_{AND3}: Given three inputs $[\![x]\!], [\![y]\!], [\![u]\!]$, we adopt a similar approach as Π_{AND4} to achieve functionality $\mathcal{F}_{\mathsf{AND3}}$: we first AND $[\![x]\!]$ and $[\![y]\!]$ using MRSS-based AND2 but keep the result in RSS-shared fashion as $\langle x \wedge y\rangle$ (like step 1) of online phase in protocol Π_{AND4}). At the same time, parties convert $[\![u]\!]$ into RSS sharing $\langle u\rangle$ locally. Finally, all parties compute AND2 gate on $\langle x \wedge y\rangle$ and $\langle u\rangle$ under the realm of RSS and convert the 3-out-of-3 shared $[(x \wedge y) \wedge u]$ into MRSS-shared outputs. In this way, each party only needs to send 1 bit in 1 round during the preprocessing phase and 2 bits in 1 round during the online phase.

Similar to Π_{AND4}, it is easy to see that Π_{AND2} and Π_{AND3} securely realize $\mathcal{F}_{\mathsf{AND2}}$ and $\mathcal{F}_{\mathsf{AND3}}$ in the ($\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ZeroShr}}$)-hybrid model, respectively. The round and communication complexity is analyzed as follows: i) In both protocols, each party sends 2 bits in 1 round for online phase. ii) For the preprocessing phase, Π_{AND2} does not require communication and Π_{AND3} only requires each party to send 1 bit in 1 round.

4) ANDN with $N > 4$: Let n be a parameter and the number of inputs $N = 4^n$. We use protocol Π_{AND4} and QuadTree [30] to build our highly depth-optimized ANDN circuits: Starting from the leaves (*a.k.a.*, inputs), we invoke Π_{AND4} for $N/4^k$ times in parallel (1 round) for the k-th layer AND4 gates, iteratively n sequential layers to reach the root, which gives the final result. When N is not a power of 4, we integrate our protocol Π_{AND2} and Π_{AND3} to process 2- and 3-input AND gates when necessary. We present the detailed protocol with $N > 4$ in Appendix A. It is easy to see the security in the hybrid model. The round and communication complexity is summarized as follows.

Remark 1. *Let M_4, M_3, and M_2 denote the numbers of* AND4, AND3, *and* AND2 *gates in* AND*ing N inputs. Each party sends $2M_4 + M_3$ bits in 1 round for the preprocessing phase, and sends $2(M_4 + M_3 + M_2)$ bits in $\lceil \log_4 N \rceil$ rounds during the online phase. Especially, when $N = 4^n$, we only need $\frac{N-1}{3}$* AND4 *gates. All parties send $2(N-1)$ bits in 1 round for the preprocessing phase, and send $2(N-1)$ bits in $n = \log_4(N)$ rounds for the online phase.*

IV. Improved PPA Circuit & Primitives

We first revisit depth-optimized Parallel Prefix Adder (PPA) circuit (§ IV-A). Then, we apply our improved PPA to design fast 3PC primitives, including Arithmetic-Boolean conversions and most significant bit extraction (§ IV-B and § IV-C).

A. Depth-Optimized PPA & Optimizations

Given two ℓ-bits x and y, the bit-wise representations are $x = x_{\ell-1}\ldots x_1 x_0$ and $y = y_{\ell-1}\ldots y_1 y_0$, where x_j (resp., y_j) denotes the j-th bit of x (resp., y). PPA makes use of the generated (g) and propagated (p) signals to compute the sum of inputs in Boolean circuits [31, 32]. Concretely, given (x_j, y_j), (g_j, p_j) are defined as:

$$g_j = x_j \wedge y_j,\ p_j = x_j \oplus y_j. \tag{2}$$

Patra *et al.* [22] defined operators $\triangledown_N$ and $\blacktriangledown_N$ to compute intermediate signals based on g and p. We revisit $\triangledown_N$ and $\blacktriangledown_N$ for $N = 2, 3, 4$. When $j = 0$, signals can be processed as follows, with indices from 0 to $N-1$:

$$\begin{aligned}
\triangledown_2(g,p) &= g_{j+1} \oplus (p_{j+1} \wedge g_j),\\
\blacktriangledown_2(g,p) &= (g_{j+1} \oplus (p_{j+1} \wedge g_j), p_{j+1} \wedge p_j),\\
\triangledown_3(g,p) &= g_{j+2} \oplus p_{j+2} \wedge (g_{j+1} \oplus (p_{j+1} \wedge g_j))\\
&= g_{j+2} \oplus (p_{j+2} \wedge g_{j+1}) \oplus (p_{j+2} \wedge p_{j+1} \wedge g_j),\\
\blacktriangledown_3(g,p) &= (g_{j+2} \oplus p_{j+2} \wedge (g_{j+1} \oplus (p_{j+1} \wedge g_j)),\\
&\quad\ p_{j+2} \wedge p_{j+1} \wedge p_j)\\
&= (g_{j+2} \oplus (p_{j+2} \wedge g_{j+1}) \oplus (p_{j+2} \wedge p_{j+1} \wedge g_j),\\
&\quad\ p_{j+2} \wedge p_{j+1} \wedge p_j),\\
\triangledown_4(g,p) &= g_{j+3} \oplus p_{j+3}\\
&\quad \wedge (g_{j+2} \oplus (p_{j+2} \wedge (g_{j+1} \oplus p_{j+1} \wedge g_j)))\\
&= g_{j+3} \oplus (p_{j+3} \wedge g_{j+2}) \oplus (p_{j+3} \wedge p_{j+2} \wedge g_{j+1})\\
&\quad \oplus (p_{j+3} \wedge p_{j+2} \wedge p_{j+1} \wedge g_j),\\
\blacktriangledown_4(g,p) &= (g_{j+3} \oplus p_{j+3}\\
&\quad \wedge (g_{j+2} \oplus (p_{j+2} \wedge (g_{j+1} \oplus p_{j+1} \wedge g_j))),\\
&\quad\ p_{j+3} \wedge p_{j+2} \wedge p_{j+1} \wedge p_j)\\
&= (g_{j+3} \oplus (p_{j+3} \wedge g_{j+2}) \oplus (p_{j+3} \wedge p_{j+2} \wedge g_{j+1})\\
&\quad \oplus (p_{j+3} \wedge p_{j+2} \wedge p_{j+1} \wedge g_j),\\
&\quad\ p_{j+3} \wedge p_{j+2} \wedge p_{j+1} \wedge p_j)
\end{aligned} \tag{3}$$

An example of how to use equation (3) in depth-optimized PPA to compute *carry signals* for 16-bit inputs is illustrated in Figure 6. Following this circuit logical, given ℓ-bit x and y, the high-level evaluation of PPA is as follows:

1) Compute $g_j = x_j \wedge y_j$ and $p_j = x_j \oplus y_j$ for $j \in \{0, 1, \ldots, \ell-1\}$.
2) Recursively compute the intermediate signals layer-by-layer from (p_j, g_j) generated in step 1) using the operators of Equation (3). The generated *carry signals* are $\{c_j\}_{j=0}^{\ell-1}$.

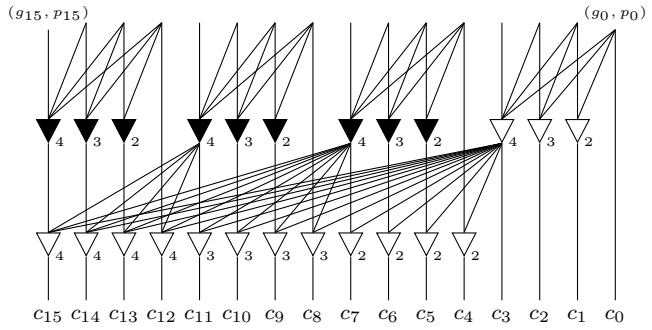

Fig. 6: 16-bit depth-optimized PPA ((g_0, p_0) to (g_{15}, p_{15})).

3) For $j \in \{0, 1, \ldots, \ell-1\}$, compute and output the sum bits $s_0 = p_0$ and $s_j = c_{j-1} \oplus p_j$ when $0 < j \leq \ell - 1$.

With our support for 3PC bitwise XOR, AND2, AND3, and AND4, we can securely implement the operators in Equation (3), ultimately achieving 3PC PPA.

1) Optimizations: Leveraging our proposed protocols and operators in Equation (3), we introduce **XOR-then-Resharing**, **AND-Reusing**, and **Mixed-AND2** to reduce the online and preprocessing communication costs in 3PC:

- **XOR-then-Resharing.** As 3-out-of-3 sharing supports XOR, we make use of the XOR-then-Resharing technique to reduce the invocations of resharing for $\triangledown_3$, $\blacktriangledown_3$, $\triangledown_4$, and $\blacktriangledown_4$. Taking $\triangledown_3$ as an example, instead of resharing $[p_1 g_2]$ and $[p_1 p_2 g_3]$ into MRSS separately and then computing XOR, parties first compute their XOR, then reshare $[p_1 g_2 \oplus p_1 p_2 g_3]$ as $[\![p_1 g_2 \oplus p_1 p_2 g_3]\!]$, and finally compute $[\![g_1]\!] \oplus [\![p_1 g_2 \oplus p_1 p_2 g_3]\!]$. So, we only reshare *one bit* for $\triangledown$-style operators and the first part of $\blacktriangledown$-style operators, making it *independent of the number of inputs*.
- **AND-Reusing.** Recall that protocols Π_{AND3} and Π_{AND4} generate RSS-shared intermediate results. We can re-use these intermediate results to reduce the preprocessing communication costs of $\blacktriangledown_3$, $\triangledown_4$, and $\blacktriangledown_4$. Taking $\blacktriangledown_3$ as an example, we observe that both $p_1 p_2 g_3$ and $p_1 p_2 p_3$ have term $p_1 p_2$. So we can compute RSS-shared $\langle p_1 p_2 \rangle$ once and use it for both AND3. This saves one invocation of RSS-based AND2 in the preprocessing phase. Also, we can apply similar optimizations to $\triangledown_4$ (resp. $\blacktriangledown_4$) to save one (resp. two) preprocessing RSS-based AND2. We reduce the preprocessing communication costs by at least $1.5\times$ than processing each AND3 and AND4 directly and separately. Since the reused intermediate results are of $\langle\cdot\rangle$-shared fashion and not revealed to any party, this technique will not introduce additional security concerns.
- **Mixed-AND2.** In PPA, we need AND2 to compute $\{[\![g_j]\!]\}_{j=0}^{\ell-1}$ and operators $(\triangledown_N, \blacktriangledown_N)$. When computing $\{[\![g_j]\!]\}_{j=0}^{\ell-1}$, we use the MRSS-based approach [24] so that each party only sends ℓ bits in the online phase. This shifts ℓ bits into the preprocessing phase but does not increase communication rounds. For the AND2 gates involved in $(\triangledown_N, \blacktriangledown_N)$, we apply protocol Π_{AND2} described in § III-B3 because we conduct XOR under $[\cdot]$-sharing before resharing (c.f., XOR-then-Resharing). Thus, we save preprocessing communication while keeping the online costs unchanged.

Protocol Π_{PPA}

Inputs: $\mathcal{P}_i$ inputs ℓ-bits Boolean sharings $[\![x]\!]_i$ and $[\![y]\!]_i$.
Outputs: $\mathcal{P}_i$ gets $[\![s]\!]_i$ with $\sum_{j=0}^{\ell-1} 2^j \cdot s_j = x + y$.

Preprocess Phase:

1) Parties jointly run the preprocessing phase of MRSS-based $\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}([\![x_j]\!], [\![y_j]\!])$ [24].
2) Parties run the preprocessing phase of secure operators $\triangledown_2$, $\blacktriangledown_2$, $\triangledown_3$, $\blacktriangledown_3$, $\triangledown_4$, and $\blacktriangledown_4$ that utilized in PPA circuit.

Online Phase:

1) Parties compute $[\![g_j]\!] = \mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}([\![x_j]\!], [\![y_j]\!])$ and $[\![p_j]\!] = [\![x_j]\!] \oplus [\![y_j]\!]$ for $j \in \{0, 1, \ldots, \ell-1\}$.
2) Recursively compute the MRSS-shared intermediate signals using $([\![p]\!], [\![g]\!])$ following the depth-optimized circuit structured as Figure 6. Denote generated *carry signals* as $[\![c]\!]$.
3) Outputting the sum bits $[\![s]\!]$ with $[\![s_0]\!] = [\![p_0]\!]$ and $[\![s_j]\!] = [\![c_{j-1}]\!] \oplus [\![p_j]\!]$ when $0 < j \leq \ell - 1$.

Fig. 7: Protocol for secure parallel prefix adder.

We present protocol Π_{PPA} for 3PC PPA in Figure 7. Without considering the optimizations, we can guarantee the correctness of protocol Π_{PPA} based on [22]. The correctness of XOR-then-Resharing can be derived from the XOR's homomorphism in 3-out-of-3 sharing and MRSS. AND-Reusing only re-uses some intermediate results of Π_{AND3} and Π_{AND4}, and does not change their workflow. Mixed-AND2 adopts two existing AND2 approaches whose correctness has already been proved. So, we can guarantee the correctness of our optimized PPA.

2) Security Analysis: Functionality $\mathcal{F}_{\mathsf{PPA}}$ takes as input two MRSS Boolean sharings $[\![x]\!]$ and $[\![y]\!]$ with identifiers id_x, id_y, respectively, outputs $[\![s]\!] = [\![x + y]\!]$ corresponding to a new identifier id_s. Π_{PPA} achieves $\mathcal{F}_{\mathsf{PPA}}$ securely, its security captured in Theorem 2, and proof is given in Appendix E.

Theorem 2 (Security of Π_{PPA})**.** *In 3-party honest-majority setting, protocol Π_{PPA} securely realizes $\mathcal{F}_{\mathsf{PPA}}$ in the ($\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\mathsf{AND2}}$, $\mathcal{F}_{\mathsf{AND3}}$, $\mathcal{F}_{\mathsf{AND4}}$)-hybrid model in presence of a static semi-honest adversary $\mathcal{A}$ who corrupts at most one party.*

3) Round & Communication: The round and complexity of our Π_{PPA} for $\ell = 4^H$ ($H \geq 1$) is formulated in Lemma 2.

Lemma 2. *For bitwidth $\ell = 4^H$ with integer $H \geq 1$, protocol Π_{PPA} requires 1 round with communication of $3N_3 + 3N_3' + 6N_4 + 9N_4' + 3\ell$ bits totally in the preprocessing phase, while it requires $1 + \log_4(\ell)$ rounds with communication of $6(N_2 + N_3 + N_4) + 12(N_2' + N_3' + N_4') + 3\ell$ bits totally in the online phase. Here N_2, N_2', N_3, N_3', N_4, and N_4' denote the numbers of operators $\triangledown_2$, $\blacktriangledown_2$, $\triangledown_3$, $\blacktriangledown_3$, $\triangledown_4$, and $\blacktriangledown_4$ in the PPA circuit, $N_2 = N_3 = N_4 = \frac{\ell-1}{3}$ and $N_2' = N_3' = N_4' = \frac{\ell \log_4 \ell}{4} - \frac{\ell-1}{3}$.*

Proof of Lemma 2 (Sketch). Benefiting from AND-Reusing, we only need to generate 1 correlated triple for $\blacktriangledown_4$, 2 for $\triangledown_4$, and 3 for $\blacktriangledown_4$. The preprocessing of MRSS-based AND2 requires ℓ bits per party. All correlations can be generated in parallel. Therefore, all parties send $3N_3 + 3N_3' + 6N_4 + 9N_4' + 3\ell$ bits in 1 round during the preprocessing phase. For the online phase, each party sends 2 bits for each $\triangledown$-style operator and 4 bits for each $\blacktriangledown$-style operator. Step 1) of online phase

Protocol Π_{A2B}

Inputs: $\mathcal{P}_i$ inputs Arithmetic sharing $[\![x]\!]^{\mathsf{A}}_i = (m_x, \langle r_x\rangle^{\mathsf{A}}_i)$, where $\langle r_x\rangle^{\mathsf{A}}_i = ([r_x]^{\mathsf{A}}_i, [r_x]^{\mathsf{A}}_{i+1})$.
Outputs: $\mathcal{P}_i$ gets Boolean sharing $[\![x]\!]_i$ for $i \in \{0,1,2\}$.

Preprocessing Phase

1) Parties run preprocessing phase of $\mathcal{F}_{\mathsf{PPA}}([\![u]\!], [\![v]\!])$.
2) $(\mathcal{P}_0, \mathcal{P}_1)$ invoke $\mathcal{F}_{\mathsf{RandPair}_{0,1}}$ to generate $[r_u]_1 \xleftarrow{\$} \mathbb{Z}_2^\ell$, $(\mathcal{P}_0, \mathcal{P}_2)$ invoke $\mathcal{F}_{\mathsf{RandPair}_{0,2}}$ to get $[r_u]_0 \xleftarrow{\$} \mathbb{Z}_2^\ell$, and all parties run $\mathcal{F}_{\mathsf{RandComm}}$ to obtain $[r_u]_2 \xleftarrow{\$} \mathbb{Z}_2^\ell$.
3) Parties $(\mathcal{P}_i, \mathcal{P}_{i+1})$ invoke $\mathcal{F}_{\mathsf{RandPair}_{i,i+1}}$ to generate $[r_v]_{i+1} \xleftarrow{\$} \mathbb{Z}_2^\ell$ for $i \in \{0,1\}$, and all parties run $\mathcal{F}_{\mathsf{RandComm}}$ to get $[r_v]_0 \xleftarrow{\$} \mathbb{Z}_2^\ell$.

Online Phase

1) $\mathcal{P}_0$ computes $u = m_x - [r_x]^{\mathsf{A}}_0$ and broadcast $m_u = u \oplus [r_u]_0 \oplus [r_u]_1 \oplus [r_u]_2$ to all to get $[\![u]\!] = (m_u, \langle r_u\rangle)$.
2) $\mathcal{P}_1$ computes $v = -[r_x]^{\mathsf{A}}_1 - [r_x]^{\mathsf{A}}_2$ and broadcast $m_v = v \oplus [r_v]_0 \oplus [r_v]_1 \oplus [r_v]_2$ to all to get $[\![v]\!] = (m_v, [r_v])$.
3) All parties run the online phase of $\mathcal{F}_{\mathsf{PPA}}([\![u]\!], [\![v]\!])$ to get $[\![x]\!] = [\![u+v]\!]$ and set $[\![x]\!]$ as outputs.

Fig. 8: Protocol for Arithmetic to Boolean conversion.

requires each party to send ℓ bits. In total, all parties send $6(N_2 + N_3 + N_4) + 12(N'_2 + N'_3 + N'_4) + 3\ell$ bits with $1+\log_4(\ell)$ rounds for online phase. For the detailed calculation of (N_2, N_3, N_4) and (N'_2, N'_3, N'_4), see Appendix B. □

B. Arithmetic to Boolean Conversion

Parties can use a *Boolean Adder* circuit to convert an Arithmetic share $[\![x]\!]^{\mathsf{A}}$ in $\mathbb{Z}_{2^\ell}$ to its equivalent ℓ-bits Boolean share in $\mathbb{Z}_2^\ell$. As pointed out by [9], it is more efficient to utilize the depth-optimized variant (*i.e.*, PPA) circuit to obtain a better round complexity. At a high level, given the arithmetic share $[\![x]\!]^{\mathsf{A}} = (m_x, \langle r_x\rangle^{\mathsf{A}})$, party $\mathcal{P}_0$ locally sets $u = m_x - [r_x]^{\mathsf{A}}_0$ followed by sharing u as ℓ-bits $[\![u]\!]$. In parallel, $\mathcal{P}_1$ generates the Boolean sharing $[\![v]\!]$ where $v = -[r_x]^{\mathsf{A}}_1 - [r_x]^{\mathsf{A}}_2$. As $u + v = m_x - [r_x]^{\mathsf{A}}_0 - [r_x]^{\mathsf{A}}_1 - [r_x]^{\mathsf{A}}_2 = x$, the parties can run $\Pi_{\mathsf{PPA}}([\![u]\!], [\![v]\!])$ to get $[\![x]\!] = [\![u]\!] + [\![v]\!]$. The A2B conversion is depicted as Π_{A2B} formally in Figure 8.

Security & Communication. Protocol Π_{A2B} only makes use of $\mathcal{F}_{\mathsf{RandPair}_{i,j}}$, $\mathcal{F}_{\mathsf{RandComm}}$, and $\mathcal{F}_{\mathsf{PPA}}$ in a black-box manner. i) We need to guarantee the security of u when $\mathcal{A}$ corrupts $\mathcal{P}_1$ or $\mathcal{P}_2$: since $[r_u]_1$ is only known to $(\mathcal{P}_0, \mathcal{P}_1)$, m_u is undistinguished from random bit-strings in $\mathbb{Z}_2^\ell$ when $\mathcal{A}$ corrupts $\mathcal{P}_2$; when $\mathcal{A}$ corrupts $\mathcal{P}_1$, we can guarantee the security of u since $[r_u]_0$ is only known to $(\mathcal{P}_0, \mathcal{P}_2)$. ii) Similarly, we can guarantee the security of v as well. The security of Π_{A2B} is easy to see in the $(\mathcal{F}_{\mathsf{RandPair}_{i,j}}, \mathcal{F}_{\mathsf{RandComm}}, \mathcal{F}_{\mathsf{PPA}})$-hybrid model.

Protocol Π_{A2B} results in $2+\lceil\log_4(\ell)\rceil$ online rounds and all parties need to send $6(N_2+N_3+N_4)+12(N'_2+N'_3+N'_4)+7\ell$ bits totally. For the preprocessing phase, all parties send $3N_3+3N'_3+6N_4+9N'_4+3\ell$ bits in 1 round of communication.

1) Most Significant Bit Extraction: PPA can be optimized by removing unnecessary operators to extract the most significant bit from Arithmetic sharing securely (*a.k.a.*, MSB circuit) [22]. Compared to PPA circuit, MSB circuit has the same depth but requires much fewer operators. We build protocol Π_{MSB} similar to Π_{A2B} by replacing the PPA circuit of step 3) in Figure 8 with MSB circuit to extract $[\![\mathsf{MSB}(x)]\!]$. For instance, Figure 9 shows the structure of MSB circuit for 16-bit inputs, which is of depth-2, and only requires two $\triangledown_4$ and $\blacktriangledown_4$, and one $\blacktriangledown_3$, reducing approximately $5\times$ operators compared to 16-bit PPA.

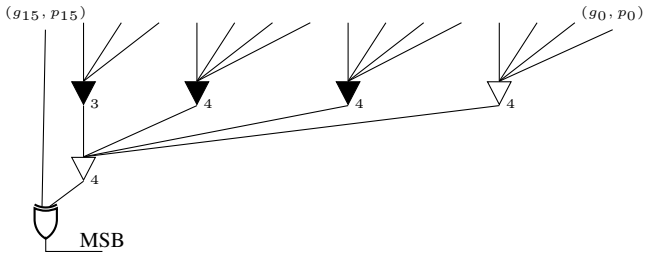

Fig. 9: Depth-optimized MSB extraction circuit for 16 bits.

C. Boolean to Arithmetic Conversion

It may also be required to convert a ℓ-bit value of Boolean sharing to an Arithmetic sharing in $\mathbb{Z}_{2^\ell}$. We can use PPA to achieve this conversion efficiently as well. In the preprocessing phase, we let $(\mathcal{P}_0, \mathcal{P}_1)$ sample random bit-string $u \xleftarrow{\$} \mathbb{Z}_2^\ell$, and $(\mathcal{P}_1, \mathcal{P}_2)$ generate a random bit-string $v \xleftarrow{\$} \mathbb{Z}_2^\ell$. Instead of directly letting $\mathcal{P}_1$ share $(-u-v)$ to get $[\![-u-v]\!]$ as ABY3 [9], we observe $[\![u \oplus v]\!]$ can be expressed as $(m = 0, \langle r\rangle = (0, u, v))$. At the same time, we let $(\mathcal{P}_0, \mathcal{P}_2)$ sample random value $a \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, $(\mathcal{P}_0, \mathcal{P}_1)$ sample $b \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, and $\mathcal{P}_1$ compute and send $c = (u \oplus v) - b$ to $\mathcal{P}_2$, such that we get a valid $\langle a + (u \oplus v)\rangle^{\mathsf{A}} = (a, b, c)$. We save the communication costs of invoking one sharing procedure at the cost of sending c (but still save ℓ bits for preprocessing communication).

During the online phase, given $[\![x]\!]$, the parties first execute $\Pi_{\mathsf{PPA}}([\![x]\!], [\![u \oplus v]\!])$ to get $[\![w]\!] = [\![x + (u \oplus v)]\!]$ and reveal w to both $\mathcal{P}_0$ and $\mathcal{P}_2$. Then, $\mathcal{P}_0$ computes and sends $w + a$ to $\mathcal{P}_1$ ($\mathcal{P}_2$ computes $w + a$ locally). Finally, the result can be set as $[\![x]\!]^{\mathsf{A}} = (w + a, \langle a + (u \oplus v)\rangle^{\mathsf{A}})$. Besides, we observe that for Π_{PPA}, instead of generating MRSS-shared $[\![w]\!]^{\mathsf{A}}$, we modify its last step of PPA circuit to output $[\cdot]$-shared carry signals and then directly reveal w to $\mathcal{P}_0$ and $\mathcal{P}_2$ from 3-out-of-3 sharing. In this way, we can save one round of communication for generating $[\![w]\!]$. Protocol Π_{B2A} is illustrated in Figure 10.

Security & Communication. We analyze the security of protocol Π_{B2A} in two cases: i) When adversary $\mathcal{A}$ corrupts $\mathcal{P}_0$, it will get $w = x + (u \oplus v)$ in clear. Since v is only known to $(\mathcal{P}_1, \mathcal{P}_2)$, w is uniformly random to $\mathcal{A}$. Similarly, when $\mathcal{A}$ corrupts $\mathcal{P}_2$, we can also guarantee the uniformity of w since $\mathcal{P}_2$ does not know u. ii) In case of $\mathcal{A}$ corrupting $\mathcal{P}_1$, it will see $w + a$. As long as $(\mathcal{P}_0, \mathcal{P}_2)$ keep the secrecy of a, $w + a$ is undistinguished from a random value for $\mathcal{A}$. Other steps are either local operators or invoking existing protocols, *i.e.*, $\mathcal{F}_{\mathsf{RandPair}_{i,j}}$ and $\mathcal{F}_{\mathsf{PPA}}$, in the black-box manner. Therefore, protocol Π_{B2A} is secure in $(\mathcal{F}_{\mathsf{RandPair}_{i,j}}, \mathcal{F}_{\mathsf{PPA}})$-hybrid model.

Protocol Π_{B2A} requires $2 + \lceil\log_4 \ell\rceil$ online rounds, and all parties send $4(N_2 + N_3 + N_4) + 12(N'_2 + N'_3 + N'_4) + 4\ell$ bits totally. For the preprocessing phase, all parties send

Protocol Π_{B2A}

Inputs: $\mathcal{P}_i$ inputs Boolean sharing $[\![x]\!]_i = (m_x, \langle r_x \rangle_i)$, where $\langle r_x \rangle_i = ([r_x]_i, [r_x]_{i+1})$.
Outputs: $\mathcal{P}_i$ gets Arithmetic sharing $[\![x]\!]_i^{\mathsf{A}}$ for $i \in \{0, 1, 2\}$.

Preprocessing Phase:

1) $(\mathcal{P}_0, \mathcal{P}_1)$ run $\mathcal{F}_{\mathsf{RandPair}_{0,1}}$ to get random $u \xleftarrow{\$} \mathbb{Z}_2^\ell$, $(\mathcal{P}_1, \mathcal{P}_2)$ execute $\mathcal{F}_{\mathsf{RandPair}_{1,2}}$ generate random $v \xleftarrow{\$} \mathbb{Z}_2^\ell$, such that $[\![u \oplus v]\!] = (m = 0, \langle r \rangle = (0, u, v))$.
2) $(\mathcal{P}_0, \mathcal{P}_2)$ invoke $\mathcal{F}_{\mathsf{RandPair}_{0,2}}$ to get random $a \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, $(\mathcal{P}_0, \mathcal{P}_1)$ call $\mathcal{F}_{\mathsf{RandPair}_{0,1}}$ to obtain random $b \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, and $\mathcal{P}_1$ sends $c = (u \oplus v) - b$ to $\mathcal{P}_2$.
3) Parties run preprocessing phase of $\mathcal{F}_{\mathsf{PPA}}([\![x]\!], [\![u \oplus v]\!])$.

Online Phase:

1) Parties jointly evaluate $\mathcal{F}_{\mathsf{PPA}}([\![x]\!], [\![u \oplus v]\!])$ to get $[w] = [x + (u \oplus v)]$ in 3-out-of-3 sharing, and directly reveal w to $\mathcal{P}_0$ and $\mathcal{P}_2$.
2) $\mathcal{P}_0$ and $\mathcal{P}_2$ locally compute $w + a$, and reveal it to $\mathcal{P}_1$.
3) $\mathcal{P}_i$ sets $[\![x]\!]_i^{\mathsf{A}} = (w + a, \langle a + (u \oplus v) \rangle_i^{\mathsf{A}})$ as outputs.

Fig. 10: Protocol for Boolean to Arithmetic conversion.

$3N_3 + 3N_3' + 6N_4 + 9N_4' + 3\ell$ bits to generate the correlated randomness for Π_{PPA}, and $\mathcal{P}_1$ sends ℓ bits to $\mathcal{P}_2$, which can be executed in parallel (1 round communication).

V. Application to Secure Inference

Secure non-linear functions, *e.g.*, ReLU and Softmax, are widely used in MPC-based secure neural network inference [9, 11]. Existing works have proposed various approximation methods, *e.g.*, piece-wise polynomial approximation [8–10], to efficiently and accurately compute these functions. We denote by $g_{d,m} : \mathbb{R} \to \mathbb{R}$ a piece-wise function defined over $d + 1$ intervals, each represented by a polynomial of degree m as:

$$g_{d,m}(x) = \begin{cases} F_0(x), & \text{if } x \le I_1, \\ F_1(x), & \text{if } x \in (I_1, I_2), \\ \dots \\ F_d(x), & \text{if } x > I_d. \end{cases} \tag{4}$$

Each polynomial $F_i(x) = a_{i,0} + \sum_{j=1}^{m} a_{i,j} x^j$ is defined by at most $m + 1$ coefficients $\{a_{i,j} \in \mathbb{R}\}_j$. However, these approaches still require huge costs for the involved Boolean primitives. Table III profiles the online costs of ReLU and Softmax, which are the most widely used activation functions in neural networks, using ABY3 [10], one of the state-of-the-art MPC frameworks. We find share conversions and MSB-Ext account for over 60% of online rounds and time.

We integrate our proposed protocols, including Π_{A2B}, Π_{B2A}, and Π_{MSB}, into 3PC secure neural network inference backend of the SecretFlow-SPU to provide an optimized framework ALKAID. As shown in Figure 11, the inputs and outputs of secure neural network inference are in the RSS-shared format. We replace the corresponding prior RSS-based implementations for Boolean circuits with our 3PC protocols, and the arithmetic operations, *i.e.*, Matrix Multiplication and Truncation, which are defined on *Arithmetic circuit* for fixed-point values and computed using existing RSS-based protocols [9, 10, 33]. However, our protocols in § IV are designed

TABLE III: Profiling communication (KB), rounds, and time (seconds) of A2B, B2A, MSB-Ext, and total costs of functions ReLU and Softmax using ABY3 (except A2B, B2A and MSB-Ext, total costs include arithmetic addition, multiplication, and *etc.* [10]). The input is of batchsize $|B| = 1000$.

ReLU	# Call	Comm.	Round	Time
A2B	0	0	0	0
MSB-Ext	1	70.300	8	0.363
B2A	0	0	0	0
Total	-	101.563	10	0.507
Percentage (%)	-	69.2%	77.2%	70.9%

Softmax	# Call	Comm.	Round	Time
A2B	1	0.219	8	0.453
MSB-Ext	16	140.625	136	5.781
B2A	2	0.273	13	0.503
Total	-	610.297	236	11.387
Percentage (%)	-	23.1%	62.4%	59.9%

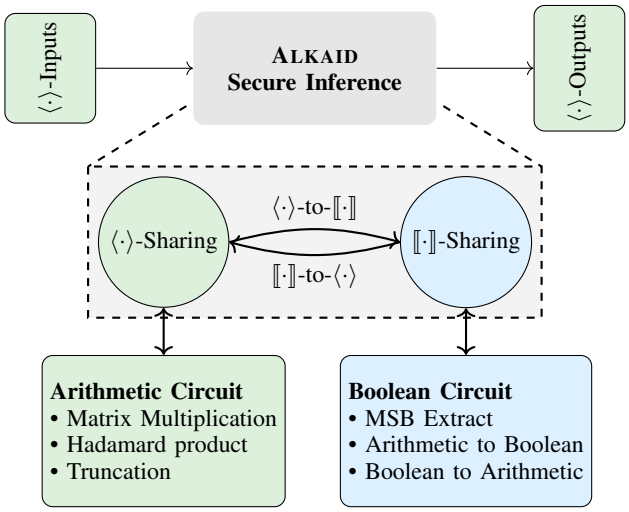

Fig. 11: Overview of the ALKAID framework, bridging RSS and MRSS for efficient secure neural network inference.

with MRSS-shared inputs and outputs, so we need to bridge the RSS and MRSS worlds with the following modifications. Note in ALKAID, we focus on how to apply our proposed primitives to improve the online efficiency of these approaches instead of designing new approximation methods.

- **$\langle\cdot\rangle$-to-$[\![\cdot]\!]$ Processing.** When switching from $\langle\cdot\rangle$-sharing to $[\![\cdot]\!]$-sharing for fast Boolean circuits, we need to convert RSS-shared inputs to MRSS-shared format. For Π_{A2B}, given $\langle x \rangle^{\mathsf{A}}$, $\mathcal{P}_0$ secret-shares $[x]_0$ as $[\![u]\!]$ with $u = [x]_0$, and $\mathcal{P}_1$ secret-shares $[x]_1 + [x]_2$ as $[\![v]\!]$ with $v = [x]_1 + [x]_2$; For protocol Π_{MSB}, given $\langle x \rangle^{\mathsf{A}}$, the parties generate $([\![u]\!], [\![v]\!])$ from $\langle a \rangle^{\mathsf{A}}$ similar to the modifications corresponding to Π_{A2B}. ii) For Π_{B2A}, we add one *resharing* before step 1) of online phase to convert $\langle x \rangle$ to MRSS-shared $[\![x]\!]$.
- **$[\![\cdot]\!]$-to-$\langle\cdot\rangle$ Processing.** We need to convert the MRSS-shared results to the RSS world for other operations. For protocols Π_{A2B} and Π_{MSB}, we modify the lowest $\triangledown$-style operators of PPA and BitExt circuits to output RSS-shared results instead of MRSS ones; For protocol Π_{B2A}, we first let $\mathcal{P}_1$ generate $\langle u \oplus v \rangle^{\mathsf{A}}$, instead of generating (a, b, c), in step 2) of the preprocessing phase. For the online phase, when $w = x + (u \oplus v)$ is revealed to $(\mathcal{P}_0, \mathcal{P}_2)$, $(w, 0, 0)$ is of

a valid RSS of $\langle x + (u \oplus v)\rangle^{\mathsf{A}}$, where $([x + (u \oplus v]_0 = w, [x + (u \oplus v]_1 = 0, [x + (u \oplus v]_2 = 0)$. In this way, the parties can locally compute $\langle x\rangle^{\mathsf{A}} = \langle x+(u\oplus v\rangle^{\mathsf{A}} - \langle u\oplus v\rangle^{\mathsf{A}}$.

The modified protocols $\Pi^{\langle\cdot\rangle}_{\mathsf{A2B}}$, $\Pi^{\langle\cdot\rangle}_{\mathsf{MSB}}$, and $\Pi^{\langle\cdot\rangle}_{\mathsf{B2A}}$, and their security proofs are in Appendix D. We integrate our improved primitives into SecretFlow-SPU to boost non-linear functions and finally lead to faster secure neural network inference.

Remark 2. *In the secure evaluation of non-linear functions, we integrate protocols* $\Pi^{\langle\cdot\rangle}_{\mathsf{A2B}}$, $\Pi^{\langle\cdot\rangle}_{\mathsf{B2A}}$, *and* $\Pi^{\langle\cdot\rangle}_{\mathsf{MSB}}$ *into SecretFlow-SPU and use them to improve the online efficiency of corresponding parts. The other parts are computed using RSS. It is also possible to use our* ANDN *protocols and depth-optimized* Π_{PPA} *to accelerate the online efficiency of other parts, which we leave for future work.*

VI. Implementation & Evaluation

We implement ALKAID and extensive evaluate its performance by answering the following questions.

- **Q1:** What are our communication and running time advantages for depth-optimized PPA? (§ VI-B)
- **Q2:** How about our efficiency for 3PC Boolean primitives? Is it practical to support share conversions? (§ VI-C)
- **Q3:** Can ALKAID support fast secure evaluation of non-linear functions and neural network inference? (§ VI-D)

A. Experimental Setup

Testbed Environments. Experiments are run on a machine with Intel(R) Xeon(R) Silver 4314 CPU @ 2.40 GHz and 500 GB RAM. The Operating System is Ubuntu 22.04.4 LTS with Linux kernel 5.4.0-172-generic. The Wide-Area Network (WAN) environment is emulated using the Linux Traffic Control (tc) command, with configured: i) LAN: bandwidth of 1 Gbps and a round-trip latency of 2 ms. ii) WAN: bandwidth of 160 Mbps and a round-trip latency of 50 ms. Bitwidth is configured as $\ell = 64$ by default.

Baselines. We compare ALKAID to prior works including ABY3 [9], Trifecta [25] and METEOR [24], to show our improvements thoroughly: i) We implement ALKAID on top of the SecretFlow-SPU framework [10] in C++ and Python version-3.10. ii) We re-run ABY3 implemented in SecretFlow-SPU and METEOR in our tested environments. Note that ABY3 and Trifecta do not separate preprocessing and online phases (which incur almost no preprocessing costs), so their total costs are equivalent to online cost. We compare ALKAID with them in both online and total costs. As Trifecta is not publicly available, we set our tested environments to be comparable to theirs for a fair comparison. iii) Unless otherwise stated, ***we report communication costs as the sum of messages sent and received per party***.

B. Benchmarks of Parallel Prefix Adder

We benchmark the online efficiency of PPA. The experiments are performed with bit-width $\ell = 64$ and batchsize $|B| \in \{1, 1000\}$, and the results are illustrated in Table IV. Note that PPA is with MRSS-shared inputs and outputs here.

TABLE IV: Communication (KB) and time (milliseconds) of secure PPA circuit, where bit-width $\ell = 64$ bits and batchsize $|B| = 1$ and 1000. ALKAID-On. is for the online phase, and ALKAID-Tot. indicates the total overhead. *We count the sent and received messages per party as communication costs.*

$\|B\|$	Protocol	Comm.	LAN	WAN
1	ABY3	0.203	5.015	352.078
	Trifecta	4.483	–	213.000
	ALKAID-On.	0.141	2.685	201.828
	ALKAID-Tot.	0.198	3.341	252.214
1000	ABY3	203.125	6.526	375.137
	Trifecta	4378.472	–	339.561
	ALKAID-On.	140.625	10.469	219.770
	ALKAID-Tot.	197.265	11.072	271.105

As shown in Table IV, we compare our communication costs, round complexity, and running time with ABY3 and Trifecta: i) Compared to ABY3, ALKAID reduces communication costs by approximately 1.3×. In terms of running time, ALKAID generally outperforms ABY3, achieving a 1.6-1.7× speedup in WAN, which aligns with its reductions in round complexity. ii) However, the running time gains in LAN are less significant due to ample bandwidth and lower latency, where reductions in communication and rounds do not lead to effective running time savings. Additionally, ALKAIDeven requires more running time when batchsize is 1000. This is because ALKAID requires more local AND and XOR gates in N-input AND protocol and complex circuit logic designs compared to ABY3, introducing some computational and memory access overhead. However, this impact can be mitigated through hardware optimizations. iii) Compared to Trifecta, we achieve comparable round complexity while reducing the communication costs by 28×, since Trifecta needs exponentially more communication to generate correlations during the online phase. Moreover, our approach reduces the running time by around 5%-13%.

Besides, we measure the preprocessing communication and running time in Table IV. Putting preprocessing and online costs together, we analyze the end-to-end efficiency in WAN as follows: Compared to ABY3, even we require ≈ 28% more communication, ALKAID is still 1.2-1.3× faster. Compared to Trifecta, we even reduce total communication by 17× and achieve 1.2× total time improvements in batch processing.

C. Benchmarks of Boolean Primitives

In Table V, we present the online evaluations of 3PC Boolean primitives: A2B, B2A, and MSB-Ext. We mainly compare ALKAID to ABY3: i) Similar to the evaluation analysis of PPA, ALKAID also outperforms ABY3 in online communication and running time for most cases in WAN. Concretely, we reduce the communication of A2B and B2A conversion by 1.2-1.6×, and reduce the running time by approximately 1.2-1.5×. In LAN, we achieve a comparable running time with ABY3 and introduce a little more time for batch evaluation of A2B and B2A conversion. The reason is similar to that of batch evaluation of PPA. ii) Notably, our B2A circuit requires less communication than PPA. For instance, with $\ell = 64$ and $|B| = 1000$, B2A incurs 132.813

TABLE V: Communication (KB) and time (milliseconds) of secure A2B, B2A, and MSB-Ext, where bit-width $\ell = 64$, batchsize $|B| = 1$ and 1000.

$\|B\|$	Primitive	Protocol	Comm.	LAN	WAN
1	A2B	ABY3	0.219	5.659	402.737
		ALKAID-On.	0.164	4.846	302.521
		ALKAID-Tot.	0.221	5.523	352.907
	B2A	ABY3	0.242	6.479	452.797
		ALKAID-On.	0.133	4.823	307.327
		ALKAID-Tot.	0.198	5.574	357.674
	MSB-Ext	ABY3	0.070	5.841	403.133
		METEOR-On.	0.178	10.634	310.495
		METEOR-Tot.	0.662	17.267	418.387
		ALKAID-On.	0.066	4.545	301.979
		ALKAID-Tot.	0.105	5.192	352.190
1000	A2B	ABY3	218.750	7.198	405.648
		ALKAID-On.	164.063	12.704	322.159
		ALKAID-Tot.	220.703	16.197	373.494
	B2A	ABY3	242.188	8.526	455.614
		ALKAID-On.	132.813	12.508	320.074
		ALKAID-Tot.	197.266	15.651	378.805
	MSB-Ext	ABY3	70.313	8.453	407.687
		METEOR-On.	178.176	11.980	329.943
		METEOR-Tot.	662.395	19.392	455.147
		ALKAID-On.	66.406	6.961	305.860
		ALKAID-Tot.	97.656	8.955	356.715

KB in communication, whereas PPA requires 140.625 KB, as a result of the specific optimizations in protocol Π_{B2A} (see Figure 10). However, Π_{B2A} does introduce longer running time due to its deeper circuit and increased rounds. iii) For MSB-Ext, ALKAID reduces round complexity by $1.6\times$, allowing it to remain approximately $1.2\times$ faster than ABY3. Compared to METEOR, our online phase is still more concretely efficient. Even we require twice online communication complexity for N-bit AND (c.f., Table I), ALKAID is 1.1-2.5$\times$ faster and reduces the concrete communication costs by $2.6\times$ for the online phase. *The online communication reductions arise from that* METEOR *is built on* FALCON*, which requires to secret-share bit values in $\mathbb{Z}_p$ with prime $p > \ell$. As a consequence,* METEOR *requires $\lceil \log_2 p \rceil\times$ more concrete online communication.*

Table V also includes the end-to-end communication and running time for each primitive. Our total communication is also comparable or even better than ABY3 and METEOR with our specific optimizations. As METEOR does not open-source the preprocessing implementation, we implement their correlation generation. Moreover, when comparing our total costs to METEOR, we still achieve a communication reduction of approximately $7\times$ and 1.2–2.1$\times$ faster for MSB-Ext. Besides, we reduce total running time by upto $1.3\times$ than ABY3.

D. Performance of Secure Inference

In practical applications, the inputs are usually floating-point values. We truncate the floating-point values as fixed-point representations while maintaining f fractional bits and then encode the fixed-point values as integers within $\mathbb{Z}_{2^\ell}$. Following [10], we set $\ell = 64$ and $f = 18$. Note the inputs and outputs of non-linear functions and secure neural networks inference are of RSS-shared format. We incorporate our modified primitives proposed in § V into the SPU implementation of ABY3 while leveraging existing ABY3-based implementations for other operations. Also, as we only modify the protocols for Boolean circuits and follow existing works for others, *e.g.*, approximation and truncation methods, ALKAID can guarantee the same level of precision and model accuracy as SecretFlow-SPU [10], so we focus on performance evaluation in this work.

TABLE VI: Communication (KB) and running time (milliseconds) of secure activation functions for $|B| = 1$ and 1000. For Softmax, we set $|B| = 10$ and 1000.

$\|B\|$	Functions	Protocol	Comm.	LAN	WAN
1	ReLU	ABY3	0.102	10.003	503.214
		METEOR-On.	0.195	9.472	352.034
		METEOR-Tot.	0.678	17.381	486.033
		ALKAID-On.	0.096	9.078	403.341
		ALKAID-Tot.	0.127	9.865	453.668
	Sigmoid	ABY3	0.552	22.478	1508.344
		ALKAID-On.	0.348	19.398	1064.298
		ALKAID-Tot.	0.418	20.397	1115.638
	GeLU	ABY3	0.806	39.098	2826.476
		ALKAID-On.	0.803	40.098	2518.875
		ALKAID-Tot.	0.897	40.846	2569.745
	Softmax ($\|B\|$=10)	ABY3	7.016	101.349	6272.879
		ALKAID-On.	6.701	87.078	5550.320
		ALKAID-Tot.	7.342	87.857	5601.810
1000	ReLU	ABY3	101.562	14.087	507.467
		METEOR-On.	195.560	11.437	424.098
		METEOR-Tot.	678.776	20.073	554.283
		ALKAID-On.	93.994	14.398	409.341
		ALKAID-Tot.	125.244	16.386	461.362
	Sigmoid	ABY3	465.250	35.431	1520.087
		ALKAID-On.	344.238	32.879	1077.687
		ALKAID-Tot.	414.551	38.897	1133.763
	GeLU	ABY3	834.375	62.031	2836.887
		ALKAID-On.	797.607	52.349	2542.891
		ALKAID-Tot.	894.357	56.350	2599.922
	Softmax	ABY3	610.297	192.319	11387.271
		ALKAID-On.	563.923	160.081	9606.879
		ALKAID-Tot.	626.438	162.862	9657.776

Online Performance. First, we compare ALKAID with prior works in terms of online performance.

- **Non-linear Functions.** We select ReLU, Sigmoid, GeLU, and Softmax for evaluations, and their specifications can be found in Appendix C-A. Table VI illustrates the experiment results for secure activation functions, highlighting the improvements achieved by ALKAID. i) Compared to ABY3, ALKAID demonstrates higher efficiency in online running time across almost all experiments in both LAN and WAN. This is particularly due to our reduction of circuit depth. Roughly, we reduce the circuit depth by approximately 1.2-1.5$\times$. Consequently, our approach reduces online running time by $\approx 1.2\times$ in WAN on average and is more efficient than ABY3 in terms of online communication. ii) For secure ReLU, Compared to METEOR, we reduce the online communication of the function ReLU by $2.0\times$. However, we require slightly more running time than METEOR. This is mainly because our SPU-based implementation generates randomness faithfully, while METEOR prototype simply uses *zero* values for protocol simulation following Falcon. So, we require more computation overhead. Even though our online running time is comparable to METEOR.
- **Secure Inference.** We employ three widely used neural networks: LeNet-5 [34], ResNet-50 [35], and GPT-2 [36], and their specifications are illustrated in Appendix C-B. Following prior works [9, 17], we mainly focus on the

TABLE VII: Communication (MB) and time (seconds) of secure inference of neural networks. The input of LeNet and ResNet-50 is 1 image and output is the label. We measure GPT-2 costs for inputting 8 tokens and generating 1 token.

Model	Protocol	Comm.	LAN	WAN
LeNet-5	ABY3	2.439	0.264	4.833
	METEOR-On.	3.137	0.230	4.445
	METEOR-Tot.	7.312	0.312	5.573
	ALKAID-On.	2.327	0.224	3.982
	ALKAID-Tot.	2.791	0.237	4.079
ResNet-50	ABY3	2347.265	50.207	377.384
	ALKAID-On.	2306.462	49.671	348.023
	ALKAID-Tot.	2649.305	55.314	374.812
GPT-2	PUMA†	459.157	21.065	428.663
	ALKAID-On.	422.192	19.784	395.057
	ALKAID-Tot.	459.926	20.772	398.120

† PUMA is developed on protocol ABY3.

communication and running time of the 3-party inference, *a.k.a.*, we do not include the model deployment cost since this can be done once in the system setup phase. For LeNet-5 (resp., ResNet-50), we input one image from the MNIST dataset [34] (resp., CIFAR-10 [37]) for image classification; for GPT-2, we input 8 tokens and generate 1 new token. Table VII shows the online efficiency of secure neural network inference. Compared to ABY3, ALKAID requires less communication cost and is faster than ABY3 by approximately 1.1-1.2×. Compared to METEOR [24], we achieve 1.35× reduction in communication and approximately 10% time savings for the online phase.

End-to-End Performance Analysis. Tables VI and VII also summarize the total costs of secure activation functions and neural network inference. Our preprocessing involves cheap communication and running time, which account for $< 10\%$ of online costs for most cases. Our total running time outperforms ABY3 and METEOR for all activation functions and NNs. Compared to METEOR, we reduce the total communication by $> 2.6\times$ and total time by 1.2–1.3×. Compared to ABY3, taking secure GPT-2 as an example, we still achieve around 7% reduction in total running time in WAN. Nonetheless, our approach does not require exponential preprocessing communication, due to our improved multi-input AND gate design. ALKAID prioritizes low online latency, making it beneficial in latency-sensitive scenarios over prior works.

VII. RELATED WORK

Secure multiparty computation (MPC) protocols can be constructed from secret sharing [38, 39]. In works [3, 9, 21, 27], the parties securely evaluate one depth of gates of the circuits using the secret sharing-based subprotocols (*e.g.*, XOR and AND) until outputs. It is also possible to design MPC protocols from homomorphic encryption (HE) [40, 41] or GC [2, 20, 42, 43], Secret sharing-based protocols enjoy high-throughput since they require much less (online) overhead and allow multiple instances to be executed in parallel [21, 22].

However, secret sharing-based approaches are still limited by their communication rounds, especially in high-latency networks [21]. Existing works have exploited the function-dependent but input-independent preprocessing/online technique [22, 29] to design multi-input AND (*a.k.a.*, multiplication in arithmetic) protocols. At a high level, this approach mainly designed a masked secret sharing scheme, generated correlations according to the function topology, and finally applied the correlations to improve online efficiency. [29] used the masked secret sharing over framework SPDZ [19, 44, 45] to reduce its online communication rounds and costs. [23, 26] used a dealer to distribute correlated randomness among two computing parties. Following [26], [46–48] designed similar MPC protocols for 3 to 5 parties to resist malicious adversaries in honest-majority. ABY2.0 [22] emulated the requirements for the dealer and built a 2PC framework that supports Arithmetic, Boolean, and Yao's GC, along with their conversion. Brüggemann *et al.* [49] made use of ABY2.0 to improve the efficiency of secure 2PC lookup table evaluation over [50, 51]. Trifecta [25] further constructed a more efficient dealer-based 3PC multi-input AND protocol by putting the preprocessing costs into the online phase. METEOR [24] built another 3PC replicated secret sharing-based multi-input AND protocol on top of [11, 22]. However, the above approaches all require a communication cost exponential to the number of inputs to generate the correlation for multi-input AND. Consequently, most of them are limited to support 4-input AND, except that Trifecta tried 8-inputs AND with much more communication. Function secret sharing (FSS) [52–54] is another MPC technique that could reduce the round complexity significantly and have been used in [55–57], but it requires huge communication costs to distribute the FSS keys.

RSS-based 3PC solutions have been applied to many practical privacy-preserving applications, such as machine learning [9, 11, 12, 17, 33, 58, 59], heavy hitters [60], graph analysis [16, 61], *etc.* Consequently, designing an RSS-based or -compatible fast multi-input AND (even with a lightweight preprocessing/correlation generation phase) and applying it to existing frameworks is meaningful.

VIII. CONCLUSION & FUTURE WORK

We introduce ALKAID a framework for 3-party multi-input AND gates by mixing correlations and redundancy. Following that, we construct several round-efficient 3PC primitives, including depth-optimized PPA and fast share conversions, with specific optimizations to improve efficiency concretely.

Future Work. By integrating these techniques, ALKAID accelerates secure neural network inference. Extending our protocols to arithmetic circuits for multi-input multiplication of integers is straightforward. However, it is non-trivial to extend ALKAID to multi-input multiplication of fixed-point computation since it requires one secure truncation after multiplying two fixed-point values. On the other hand, extending ALKAID to the multi-party setting is both natural and meaningful. At a high level, our multi-input AND protocol is built upon linear secret sharing with redundancy. Given such a linear secret-sharing scheme, constructing masked secret sharing is straightforward. Following the same design philosophy as in our 3PC setting, the correlated randomness required for multi-input AND gates can be efficiently generated using the underlying linear secret sharing. The parties can then leverage these

correlations, together with the redundancy, to support multi-input AND protocols that achieve improved online efficiency while maintaining practical preprocessing costs.

ACKNOWLEDGMENTS

This work is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office, and NSFC, China (Grant 62302118). Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

REFERENCES

[1] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[2] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167.

[3] S. Micali, O. Goldreich, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM New York, 1987, pp. 218–229.

[4] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[5] J. A. Primbs, "Applications of mpc to finance," *Handbook of model predictive control*, pp. 665–685, 2019.

[6] Y. Yang, Y. Tong, J. Weng, Y. Yi, Y. Zheng, L. Y. Zhang, and R. Lu, "Prirange: Privacy-preserving range-constrained intersection query over genomic data," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2379–2391, 2022.

[7] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[8] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 619–631.

[9] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[10] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "Secretflow-spu: A performant and user-friendly framework for privacy-preserving machine learning," in *USENIX Annual Technical Conference*, 2023, pp. 17–33.

[11] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.

[12] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," *arXiv preprint arXiv:2104.10949*, 2021.

[13] J. Bai, X. Song, X. Zhang, Q. Wang, S. Cui, E.-C. Chang, and G. Russello, "Mostree: Malicious secure private decision tree evaluation with sublinear communication," in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 799–813.

[14] D. Bhardwaj, S. Saravanan, N. Chandran, and D. Gupta, "Securely training decision trees efficiently," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4673–4687.

[15] G. Lin, W. Han, W. Ruan, R. Zhou, L. Song, B. Li, and Y. Shao, "Ents: an efficient three-party training framework for decision trees by communication optimization," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4376–4390.

[16] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, "Secure graph analysis at scale," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 610–629.

[17] Y. Dong, W.-j. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, "Puma: Secure inference of llama-7b in five minutes," *arXiv preprint arXiv:2307.12533*, 2023.

[18] H. Wu, W. Fang, Y. Zheng, J. Ma, J. Tan, and L. Wang, "Ditto: Quantization-aware secure inference of transformers upon mpc," in *International Conference on Machine Learning*. PMLR, 2024, pp. 53 346–53 365.

[19] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of ACM SIGSAC conference on computer and communications security*, 2020, pp. 1575–1590.

[20] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *20th USENIX Security Symposium*, 2011.

[21] T. Schneider and M. Zohner, "Gmw vs. yao? efficient secure two-party computation with low depth circuits," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 275–292.

[22] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2.0: Improved mixed-protocol secure two-party computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2165–2182.

[23] S. Ohata and K. Nuida, "Communication-efficient (client-aided) secure two-party protocols and its application," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*. Springer, 2020, pp. 369–385.

[24] Y. Dong, C. Xiaojun, W. Jing, L. Kaiyun, and W. Wang, "Meteor: improved secure 3-party neural network inference with reducing online communication costs," in *ACM Web Conference*, 2023, pp. 2087–2098.

[25] S. Faraji and F. Kerschbaum, "Trifecta: Faster high-throughput three-party computation over wan using multi-fan-in logic gates," *Proceedings on Privacy En-*

hancing Technologies, 2023.

[26] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "Astra: High throughput 3pc over rings with application to secure prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 81–92.

[27] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "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*, 2016, pp. 805–817.

[28] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *Annual international conference on the theory and applications of cryptographic techniques*, 2017, pp. 225–255.

[29] A. Ben-Efraim, M. Nielsen, and E. Omri, "Turbospeedz: Double your online spdz! improving spdz using function dependent preprocessing," in *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer, 2019, pp. 530–549.

[30] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, pp. 1–9, 1974.

[31] A. Beaumont-Smith and C.-C. Lim, "Parallel prefix adder design," in *Proceedings 15th IEEE Symposium on Computer Arithmetic.* IEEE, 2001, pp. 218–225.

[32] D. Harris, "A taxonomy of parallel prefix networks," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*. IEEE, 2003, pp. 2213–2217.

[33] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, pp. 355–375, 2020.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[38] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[39] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 316–334.

[40] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[41] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[42] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*. Springer, 2008, pp. 486–498.

[43] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole: Reducing data transfer in garbled circuits using half gates," in *Advances in Cryptology-EUROCRYPT: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part II 34*. Springer, 2015, pp. 220–250.

[44] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[45] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits," in *18th European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.

[46] A. Patra and A. Suresh, "Blaze: blazing fast privacy-preserving machine learning," *arXiv preprint arXiv:2005.09042*, 2020.

[47] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning." *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 2, pp. 459–480, 2020.

[48] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "Swift: Super-fast and robust privacy-preserving machine learning," in *30th USENIX Security Symposium*, 2021.

[49] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame, "Flute: fast and secure lookup table evaluations," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 515–533.

[50] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *Proceedings of 10th Theory of Cryptography Conference*. Springer, 2013, pp. 600–620.

[51] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," *Cryptology ePrint Archive*, 2018.

[52] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 337–367.

[53] ——, "Function secret sharing: Improvements and extensions," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1292–1303.

[54] ——, "Secure computation with preprocessing via function secret sharing," in *Theory of Cryptography:*

17th International Conference, Proceedings, Part I 17. Springer, 2019, pp. 341–371.
[55] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proceedings on Privacy Enhancing Technologies*, 2022.
[56] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "Sigma: Secure gpt inference with function secret sharing," in *Privacy Enhancing technologies Symposium (PETS)*, June 2024.
[57] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 597–616.
[58] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A gpu platform for secure computation," *Cryptology ePrint Archive*, 2022.
[59] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3pc for binary circuits with application to maliciously-secure dnn inference," in *32nd USENIX Security Symposium*, 2023, pp. 5377–5394.
[60] J. Kilian, A. Madeira, M. J. Strauss, and X. Zheng, "Fast private norm estimation and heavy hitters," in *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC, Proceedings 5*. Springer, 2008, pp. 176–193.
[61] N. Koti, V. B. Kukkala, A. Patra, B. R. Gopal, S. Sangal *et al.*, "Ruffle: Rapid 3-party shuffle protocols," *Proceedings on Privacy Enhancing Technologies*, 2023.

APPENDIX A
PROTOCOL FOR N-INPUTS AND GATE WITH $N > 4$

We present the detail protocol for $N > 4$ inputs AND gate as Figure 12. Note that steps 2)–5) are computed in parallel and preprocessing/online paradigm. We show the protocol in the black-box manner for brevity.

Protocol Π_{ANDN} for $N > 4$

Inputs: N Boolean shared inputs $[\![x_0]\!], [\![x_1]\!], \ldots, [\![x_{N-1}]\!]$.
Outputs: $[\![y]\!]$ with $y = \wedge_{i=0}^{N-1} x_i$
While $N \geq 2$, **do**:
1) $N_4 = \lfloor \frac{N}{4} \rfloor$, $N_3 = \lfloor \frac{N-4N_4}{3} \rfloor$, $N_2 = \lfloor \frac{N-4N_4-3N_3}{2} \rfloor$, $N_1 = N - 4N_4 - 3N_3 - 2N_2$.
2) $[\![z_i]\!] = \Pi_{\mathsf{AND4}}([\![x_{4i}]\!], [\![x_{4i+1}]\!], [\![x_{4i+2}]\!], [\![x_{4i+3}]\!])$ **for** $i \in [N_4]$;
3) $[\![z_{N_4+i}]\!] = \Pi_{\mathsf{AND3}}([\![x_{4N_4+3i}]\!], [\![x_{4N_4+3i+1}]\!], [\![x_{4N_4+3i+2}]\!])$ **for** $i \in [N_3]$.
4) $[\![z_{N_4+N_3+i}]\!] = \Pi_{\mathsf{AND2}}([\![x_{4N_4+3N_3+2i}]\!], [\![x_{4N_4+3N_3+2i+1}]\!])$ **for** $i \in [N_2]$.
5) $[\![z_{N_4+N_3+N_2+i}]\!] = [\![x_{4N_4+3N_3+2N_2+i}]\!]$ **for** $i \in [N_1]$
6) $N = N_4 + N_3 + N_2 + N_1$.
7) $[\![x_i]\!] = [\![z_i]\!]$ **for** $i \in [N]$.

End While
Outputs $[\![y]\!] = [\![x_0]\!]$.

Fig. 12: Protocol for N-input AND gates with $N > 4$.

APPENDIX B
PROOF OF LEMMA 2

Proof. We prove the claimed formulas for the radix-4 depth-optimized PPA with $\ell = 4^H$, where $H \geq 1$. Thus, the formulas below apply to the construction specified in this lemma. Arbitrary word lengths require separate accounting for incomplete boundary blocks or padding.

a) Correctness of the radix-4 prefix construction.: For every bit position $i \in \{0, \ldots, \ell - 1\}$, define the bit-generate and bit-propagate signals as

$$g_i^{(0)} = x_i \wedge y_i, \qquad p_i^{(0)} = x_i \oplus y_i.$$

For an interval $[a, b]$, where $0 \leq a \leq b < \ell$, let $(G_{b:a}, P_{b:a})$ denote its group-generate and group-propagate pair. In particular,

$$P_{b:a} = \bigwedge_{r=a}^{b} p_r^{(0)}, \tag{5}$$

$$G_{b:a} = g_b^{(0)} \oplus \bigoplus_{r=a}^{b-1} \left(g_r^{(0)} \wedge \bigwedge_{v=r+1}^{b} p_v^{(0)} \right). \tag{6}$$

We use the convention that an empty conjunction equals 1 and an empty exclusive-or equals 0. Since

$$g_i^{(0)} \wedge p_i^{(0)} = 0$$

for every i, the terms in (6) are mutually exclusive. Hence, the exclusive-or in (6) is equivalent to the usual disjunction in the carry recurrence.

For two adjacent intervals, their group-generate and group-propagate pairs are combined using the associative prefix operator

$$(G_R, P_R) \circ (G_L, P_L) = (G_R \oplus (P_R \wedge G_L),\ P_R \wedge P_L), \tag{7}$$

where (G_L, P_L) represents the less-significant interval and (G_R, P_R) represents the more-significant interval.

At layer $h \in \{1, \ldots, H\}$, define

$$s_h = 4^h, \qquad t_h = \frac{s_h}{4} = 4^{h-1}.$$

Partition the word into ℓ/s_h consecutive blocks

$$B_{h,b} = \{bs_h, \ldots, (b+1)s_h - 1\}, \qquad 0 \leq b < \frac{\ell}{s_h}.$$

Each block consists of four consecutive subblocks of length t_h.

Consider a position

$$u = bs_h + i, \qquad 0 \leq i < s_h,$$

and define

$$q = \left\lfloor \frac{i}{t_h} \right\rfloor \in \{0, 1, 2, 3\}.$$

Thus, u belongs to the $(q+1)$-st subblock of $B_{h,b}$. For every $r \in \{1, \ldots, q\}$, let

$$e_r = bs_h + rt_h - 1.$$

The index e_r is the last position of the r-th subblock preceding the subblock containing u.

Before dead-signal pruning, the layer-h outputs at position u are

$$p_u^{(h)} = p_u^{(h-1)} \wedge \bigwedge_{r=1}^{q} p_{e_r}^{(h-1)}, \tag{8}$$

$$g_u^{(h)} = g_u^{(h-1)} \oplus \bigoplus_{r=1}^{q} \left(g_{e_r}^{(h-1)} \wedge p_u^{(h-1)} \wedge \bigwedge_{v=r+1}^{q} p_{e_v}^{(h-1)} \right). \tag{9}$$

For $q \geq 1$, Equations (8) and (9) are precisely the expansion of

$$(g_u^{(h-1)}, p_u^{(h-1)}) \circ (g_{e_q}^{(h-1)}, p_{e_q}^{(h-1)}) \circ \cdots \circ (g_{e_1}^{(h-1)}, p_{e_1}^{(h-1)}).$$

For $q = 0$, the pair is forwarded without computation. For $q = 1, 2, 3$, the position invokes a 2-, 3-, or 4-input prefix operator, respectively.

We prove the correctness of the construction by induction on h. At layer 0, the pair $(g_u^{(0)}, p_u^{(0)})$ represents the singleton interval $[u, u]$. Suppose that, at layer $h-1$, the pair at every position represents the local prefix ending at that position within its block of length 4^{h-1}. Then $(g_u^{(h-1)}, p_u^{(h-1)})$ represents the local prefix of the current subblock ending at u, while each $(g_{e_r}^{(h-1)}, p_{e_r}^{(h-1)})$ represents one complete preceding subblock. By the associativity of Equation (7), their composition represents the entire interval

$$[bs_h, u].$$

The invariant therefore holds at layer h. At layer H, there is a single block $[0, \ell - 1]$, and hence

$$g_u^{(H)} = G_{u:0}.$$

Therefore, assuming zero input carry, $g_u^{(H)}$ is exactly the carry signal generated by bit positions 0 through u.

b) Counting the prefix operators.: In each block at layer h, every one of the four subblocks contains exactly

$$t_h = \frac{s_h}{4}$$

positions. Hence, each arity $j \in \{1, 2, 3, 4\}$ occurs $s_h/4$ times per block. Since there are ℓ/s_h blocks, each arity occurs

$$\frac{\ell}{s_h} \cdot \frac{s_h}{4} = \frac{\ell}{4}$$

times in one layer.

The unpruned network therefore contains

$$\frac{\ell H}{4} = \frac{\ell \log_4 \ell}{4} \tag{10}$$

locations of each arity over all H layers.

The optimized circuit omits propagate outputs that cannot contribute to any subsequent generate output. Specifically, at layer h, the propagate outputs in the least-significant block

$$B_{h,0} = \{0, \ldots, 4^h - 1\}$$

are unnecessary. Indeed, according to Equation (7), the propagate signal of the less-significant interval is needed only to construct a larger propagate signal; it is not needed to construct the corresponding larger generate signal. Since the final PPA outputs carry signals but no group-propagate signals, these propagate outputs are dead values and can be removed by a backward liveness argument.

Within $B_{h,0}$, each arity occurs

$$t_h = 4^{h-1}$$

times. Therefore, the number of omitted propagate outputs of each arity over all layers is

$$\sum_{h=1}^{H} 4^{h-1} = \frac{4^H - 1}{4 - 1} = \frac{\ell - 1}{3}. \tag{11}$$

At every nontrivial location where the propagate output is omitted, only the generate output is computed; we denote this operator by $\triangledown_j$. At every other nontrivial location, both the generate and propagate outputs are computed; we denote this operator by $\blacktriangledown_j$. The 1-input case is a direct wire and requires no secure computation.

Combining Equations (10) and (11), for every $j \in \{2, 3, 4\}$, we obtain

$$N_j = \frac{\ell - 1}{3}, \qquad N_j' = \frac{\ell \log_4 \ell}{4} - \frac{\ell - 1}{3}. \tag{12}$$

Consequently,

$$N_2 = N_3 = N_4 = \frac{\ell - 1}{3}$$

and

$$N_2' = N_3' = N_4' = \frac{\ell \log_4 \ell}{4} - \frac{\ell - 1}{3}.$$

c) Preprocessing communication.: All communication quantities below denote the aggregate number of bits sent by the three parties. In the preprocessing phase, the costs of

$$\triangledown_2, \quad \triangledown_3, \quad \triangledown_4$$

are 0, 3, and 6 bits, respectively, while the costs of

$$\blacktriangledown_2, \quad \blacktriangledown_3, \quad \blacktriangledown_4$$

are 0, 3, and 9 bits, respectively. Generating the correlations for the initial ℓ gates

$$g_i^{(0)} = x_i \wedge y_i$$

contributes another 3ℓ bits. Therefore, the total preprocessing communication is

$$3N_3 + 3N_3' + 6N_4 + 9N_4' + 3\ell.$$

All required correlations are independent of the online inputs and can be generated in parallel. Thus, the preprocessing phase requires one communication round.

d) Online communication and rounds.: For every $j \in \{2, 3, 4\}$, each $\triangledown_j$ operator requires 6 aggregate bits in the online phase, while each $\blacktriangledown_j$ operator requires 12 aggregate bits. The initial ℓ AND gates contribute another 3ℓ bits. The bit-propagate signals

$$p_i^{(0)} = x_i \oplus y_i$$

and the final sum bits are computed locally. Hence, the total online communication is

$$6(N_2 + N_3 + N_4) + 12(N_2' + N_3' + N_4') + 3\ell.$$

The initial generate signals are computed in one online round. All prefix operators within one radix-4 layer are evaluated in parallel, and every layer requires one additional round. The online round complexity is therefore

$$1 + H = 1 + \log_4 \ell,$$

where the last equality follows from the assumption $\ell = 4^H$. This proves the lemma. □

APPENDIX C
ACTIVATION FUNCTIONS AND NEURAL NETWORKS

A. Activation Functions

We present the approximation methods for activation functions, including ReLU, Sigmoid, GeLU, and Softmax from existing works as follows:

- $\mathbf{ReLU}(x)$ returns x if $x \geq 0$, and 0 otherwise.
- $\mathbf{Sigmoid}(x) = \frac{1}{1+e^{-x}}$. We use the piece-wise polynomials to approximate Sigmoid. The concrete piece-wise polynomials are:

$$\mathsf{Seg3_Sigmoid}(x) = \begin{cases} 0.5 + 0.125x, & -4 \leq x \leq 4 \\ 1, & x > 4 \\ 0, & x < -4 \end{cases}.$$

- $\mathbf{GeLU}(x) = \frac{x}{2} \cdot \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}} \cdot \left(x + 0.044715 \cdot x^3\right)\right)\right)$. We utilize the piece-wise polynomials proposed by as:

$$\mathsf{Seg4_GeLU}(x) = \begin{cases} 0, & x < T_0 \\ F_0(x) & T_0 \leq x < T_1 \\ F_1(x), & T_1 \leq x \leq T_2 \\ x, & x > T_2 \end{cases},$$

where $(T_0 = -4, T_1 = -1.95, T_2 = 3)$,

$$\begin{aligned} F_0(x) = &-0.011034134030615728x^3 \\ &- 0.11807612951181953x^2 \\ &- 0.42226581151983866x \\ &- 0.5054031199708174) \end{aligned}$$

$$\begin{aligned} F_1(x) = &\, 0.0018067462606141187x^6 \\ &- 0.037688200365904236x^4 \\ &+ 0.3603292692789629x^2 \\ &+ 0.5x \\ &+ 0.008526321541038084 \end{aligned}$$

- $\mathbf{Softmax}(\boldsymbol{x})$ is computed as

$$\mathsf{Softmax}(\boldsymbol{x})_j = \frac{\exp(\boldsymbol{x}_j - \bar{x} - \epsilon)}{\sum_k \exp(\boldsymbol{x}_k - \bar{x} - \epsilon)},$$

where $\bar{x}$ is the maximum element of the input vector $\boldsymbol{x}$. We adopted the following approximation methods to compute function negExp as:

$$\mathsf{negExp}(x) = \begin{cases} 0, & x < T_{\exp} \\ (1 + \frac{x}{2^t})^{2^t}, & x \in [T_{\exp}, 0]. \end{cases}$$

where $T_{\exp} = -14$ and $t = 5$, and the secure Division is computed by GoldSchmidt approximations.

B. Neural Networks

We give the high-level architectures of neural networks as follows.

- **LeNet-5** consists of two convolutional layers, each followed by a ReLU activation function and a pooling layer. The convolutional layers use 5×5 kernels, producing 20 and 50 feature maps respectively, with 2×2 max-pooling layers for down-sampling. Finally, two fully connected layers, with ReLU after the first, map the extracted features to the output classes. LeNet-5 has around 60K parameters.
- **ResNet-50** consists of i) an initial convolutional layer followed by four groups of residual blocks with 3, 4, 6, 3 layers each, using 1×1, 3×3, and 1×1 convolutions. ii) Each residual block includes two convolutional layers and ReLU, with batch normalization applied after each convolution. iii) Finally, a global average pooling layer and a fully connected layer map the features to the output classes. ResNet-50 has around 25.5 million parameters.
- **GPT-2** consists of multiple layers of transformer decoder blocks, each including: i) self-attention layer: multi-head self-attention with 12 to 48 attention heads, depending on the model size, to capture dependencies across the input text. ii) feed-forward layer: a two-layer fully connected network with a GeLU function. iii) and layer normalization: applied after both the self-attention and feed-forward layers to stabilize training. In this work, we use GPT-2 small with 12 blocks and 117 million parameters.

APPENDIX D
INTEGRATION WITH RSS SHARING

In this section, we present our modified protocols for share conversions and MSB extraction that are equipped with RSS-shared inputs and outputs.

Protocol $\Pi^{\langle\cdot\rangle}_{\mathsf{A2B}}$

Inputs: $\mathcal{P}_i$ inputs RSS Arithmetic sharing $\langle x\rangle^{\mathsf{A}}_i = ([x]^{\mathsf{A}}_i, [x]^{\mathsf{A}}_{i+1})$.
Outputs: $\mathcal{P}_i$ gets RSS Boolean sharing $[x]_i$ for $i \in \{0, 1, 2\}$.

Preprocessing Phase:

1) Parties execute the preprocessing phase of Π_{A2B}.

Online Phase:

1) $\mathcal{P}_0$ computes $u = [x]^{\mathsf{A}}_0 + [x]^{\mathsf{A}}_1$ and broadcasts $m_u = u \oplus [r_u]_0 \oplus [r_u]_1 \oplus [r_u]_2$ to all to get $[\![u]\!] = (m_u, \langle r_u\rangle)$.
2) $\mathcal{P}_1$ lets $v = [x]^{\mathsf{A}}_2$ and broadcasts $m_v = v \oplus [r_v]_0 \oplus [r_v]_1 \oplus [r_v]_2$ to all to get $[\![v]\!] = (m_v, [r_v])$.
3) All parties run the online phase of $\mathcal{F}_{\mathsf{PPA}}([\![u]\!], [\![v]\!])$ but outputs RSS-shared $\langle x\rangle = \langle u + v\rangle$.
4) $\mathcal{P}_i$ sets $\langle x\rangle_i$ as outputs.

Fig. 13: Protocol for Arithmetic to Boolean conversion with RSS-shared inputs & outputs.

In Figure 13, we present the modified protocol $\Pi^{\langle\cdot\rangle}_{\mathsf{A2B}}$. It requires the same preprocessing communication costs as Π_{A2B} and each party to send $(N_2+N_3+N_4)+4(N'_2+N'_3+N'_4)+\frac{7}{3}\ell$ bits on average in $2 + \lceil \log_4(\ell) \rceil$ online rounds. Similarly, protocol $\Pi^{\langle\cdot\rangle}_{\mathsf{MSB}}$, which extracts the RSS-shared most significant

Protocol $\Pi_{\mathsf{B2A}}^{\langle\cdot\rangle}$

Inputs: $\mathcal{P}_i$ inputs RSS Boolean sharing $\langle x\rangle_i = (\langle x\rangle_i, \langle x\rangle_{i+1})$.
Outputs: $\mathcal{P}_i$ gets RSS Arithmetic sharing $\langle x\rangle_i^{\mathsf{A}}$ for $i \in \{0,1,2\}$.

Preprocessing Phase:

1) Parties execute the steps 1) and 3) preprocessing phase of Π_{B2A}.
2) After getting u and v, $\mathcal{P}_1$ generate te RSS shared $\langle u+v\rangle^{\mathsf{A}}$.
3) Parties invoke Rand to get random $\langle r_x\rangle$ with $r_x \xleftarrow{\$} \mathbb{Z}_2^\ell$.

Online Phase:

1) Parties compute and reveal $m_x = \langle x\rangle \oplus \langle r_x\rangle$, and denote $[\![x]\!] = (m_x, \langle r_x\rangle)$.
2) Parties jointly evaluate $\mathcal{F}_{\mathsf{PPA}}([\![x]\!], [\![u\oplus v]\!])$ to get $[w] = [x + (u\oplus v)]$ in 3-out-of-3 sharing, and directly reveal w to $\mathcal{P}_0$ and $\mathcal{P}_2$, such that we have $\langle x + (u\oplus v)\rangle^{\mathsf{A}} = (w, 0, 0)$.
3) $\mathcal{P}_i$ locally computes and outputs $\langle x\rangle_i^{\mathsf{A}} = \langle x + (u \oplus v))\rangle_i^{\mathsf{A}} - \langle u\oplus v\rangle_i^{\mathsf{A}}$.

Fig. 14: Protocol for Boolean to Arithmetic conversion with RSS-shared inputs & outputs.

bit from RSS-shared inputs can be constructed as $\Pi_{\mathsf{A2B}}^{\langle\cdot\rangle}$ with replacing the PPA circuit by MSB circuit. In Figure 14, we present protocol $\Pi_{\mathsf{B2A}}^{\langle\cdot\rangle}$. Compared to Π_{B2A}, it additionally requires each party to send ℓ bits in 1 online round, *a.k.a.*, step 1) of the online phase.

It is also easy to see the security of the modified protocols: i) Compared to Π_{A2B}, $\Pi_{\mathsf{A2B}}^{\langle\cdot\rangle}$ only involves introduce some local operations modifications. Therefore, its security can be guaranteed as Π_{A2B}; Similar analysis can be applied to $\Pi_{\mathsf{MSB}}^{\langle\cdot\rangle}$. ii) In terms of $\Pi_{\mathsf{B2A}}^{\langle\cdot\rangle}$, the additional step 1) of online phase reveals m_x, but m_x is masked by secret random $\langle r_x\rangle$. The other modifications are either local operations or invoking existing protocols in a black-box manner, so we can guarantee the security of protocol $\Pi_{\mathsf{B2A}}^{\langle\cdot\rangle}$ as well.

APPENDIX E
SECURITY PROOF

A. Ideal Functionalities

Formally, we present the following well-established functionalities, where each secret sharing is identified by a unique identifier.

- $\mathcal{F}_{\mathsf{Rand}}$ corresponds to securely generating a random RSS sharing. It receives (id) from all parties, randomly samples and outputs an RSS share $(\langle r\rangle_i) = ([r_i], [r_{i+1}])$ to $\mathcal{P}_i$ for $i \in [3]$. When $\mathcal{P}_i$ is corrupted, $[r]_i = \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id})$ and $[r]_{i+1} = \mathsf{PRF}_{\mathsf{key}_{i+1}}(\mathsf{id})$ are generated using PRF keys $(\mathsf{key}_i, \mathsf{key}_{i+1})$ chosen by $\mathcal{A}$.
- $\mathcal{F}_{\mathsf{RandComm}}$ corresponds to securely generating a random value. It receives (id) from all parties, randomly samples and outputs a random r to all parties. In the case $\mathcal{P}_i$ is corrupted, $r = \mathsf{PRF}_{\mathsf{key}}(\mathsf{id})$ is generated using PRF key chosen by adversary.
- $\mathcal{F}_{\mathsf{RandPair_{i,j}}}$ corresponds to securely generating a random value between $\mathcal{P}_i$ and $\mathcal{P}_j$. It receives (id) from both parties, randomly samples and outputs a random r to them. In the case $\mathcal{P}_i$ is corrupted, $[r] = \mathsf{PRF}_{\mathsf{key}}(\mathsf{id})$ is generated using PRF key chosen by the adversary.
- $\mathcal{F}_{\mathsf{ZeroShr}}$ corresponds to securely generating an RSS sharing of 0. It receives (id) from all parties, randomly samples and outputs a share $[\alpha]_i$ to $\mathcal{P}_i$ for $i \in [3]$ under that these share summing up to 0. In the case $\mathcal{P}_i$ is corrupted, $[\alpha]_i = \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id}) \oplus \mathsf{PRF}_{\mathsf{key}_{i+1}}(\mathsf{id})$ using PRF keys $(\mathsf{key}_i, \mathsf{key}_{i+1})$ held by the adversary.
- $\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$ corresponds to securely computing $\wedge$ operation between two RSS-shared bits. It receives id_a and id_b corresponding to RSS sharings $\langle a\rangle$ and $\langle b\rangle$, respectively, samples a new identifier id_c corresponding to RSS sharing $\langle a\wedge b\rangle$. In the case $\mathcal{P}_i$ is corrupted, samples $[a\wedge b]_{i+1} \xleftarrow{\$} \mathbb{Z}_2$ and send it to the adversary.[1]
- $\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}$ corresponds to securely computing $\wedge$ operation between two MRSS-shared bits. It receives id_x and id_y corresponding to MRSS sharings $[\![x]\!]$ and $[\![y]\!]$, respectively, samples a new identifier id_z corresponding to MRSS sharing $[\![x\wedge y]\!]$. In the case $\mathcal{P}_i$ is corrupted, samples $[x\wedge y]_{i-1} \xleftarrow{\$} \mathbb{Z}_2$ and send it to the adversary.[2]

B. Proof of Theorem 1 and 2

Proof of Theorem 1.. Our proof follows the classical real-ideal simulation model proof. For any PPT adversary $\mathcal{A}$ who corrupts one single party $\mathcal{P}_i$ at the beginning of the protocol, we construct a PPT simulator $\mathcal{S}$ that can generate a view for $\mathcal{P}_i$ in the ideal world that is indistinguishable from $\mathcal{P}_i$'s view in the real-world execution.

- (Simulation for the offline phase) For any invocation to RSS-based $\Pi_{\mathsf{AND2}}^{\langle\cdot\rangle}$, Π_{ZeroShr}, and Π_{Rand}, $\mathcal{S}$ simply invoke $\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, and $\mathcal{F}_{\mathsf{ZeroShr}}$, $\mathcal{F}_{\mathsf{Rand}}$, and hand the output of the corrupted party to $\mathcal{A}$, *i.e.*, the shares corresponding to $\langle r_{xy}\rangle, \langle r_{uv}\rangle$, $[\alpha]$, and $\langle r_z\rangle$.
- (Simulation for the online phase) $\mathcal{S}$ simulates the online phase as follows:
  - For computing $\langle xy\rangle$ and $\langle uv\rangle$ from $[\![x]\!]$, $[\![y]\!]$, $[\![u]\!]$, and $[\![v]\!]$, $\mathcal{S}$ can simulate trivially via local computation using all available shares for $\mathcal{P}_i$. Then $\mathcal{S}$ can locally simulate $[z]_i$ similarly.
  - $\mathcal{S}$ randomly samples $[m_z]_{i-1}$ and $[m_z]_{i+1}$ from $\mathbb{Z}_2$ and send them to $\mathcal{A}$ as the simulated messages from $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$. $\mathcal{S}$ then simulates $[\![z]\!]$ trivially, completing the simulation.

Next, we show simulated view above is indistinguishable from real-world execution via a sequence of hybrid games:

- H_0. Game H_0 is identical to the real protocol execution.
- H_1. For any invocation to RSS-based $\Pi_{\mathsf{AND2}}^{\langle\cdot\rangle}$, Π_{ZeroShr}, and Π_{Rand}, $\mathcal{S}$ simply invoke $\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\mathsf{ZeroShr}}$, and $\mathcal{F}_{\mathsf{Rand}}$, and

[1]Note that the adversary can compute $[x\wedge y]_i$ locally using available RSS shares $\langle a\rangle_i$ and $\langle b\rangle_i$ from the corrupted $\mathcal{P}_i$.

[2]The adversary can compute $\langle x\wedge y\rangle_i = ([x\wedge y]_i, [x\wedge y]_{i+1})$ locally using available MRSS shares $[\![a]\!]_i$ and $[\![b]\!]_i$ from the corrupted $\mathcal{P}_i$.

return the desired $\mathcal{P}_i$'s output from these functionalities to $\mathcal{A}$

- H_2. Game H_2 is identical to H_1, except for the following difference: instead of computing the honest parties' shares $[m_z]_{i-1}$ and $[m_z]_{i+1}$ via PRF using the keys held by the honest parties, H_1 randomly samples $[m_z]_{i-1}$ and $[m_z]_{i+1}$ from $\mathbb{Z}_2$. Note that H_2 is exactly the ideal simulated world.

H_0 and H_1 are indistinguishable in ($\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ZeroShr}}$)-hybrid model. Indeed, the outputs of $\mathcal{F}_{\mathsf{AND2}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\mathsf{Rand}}$, and $\mathcal{F}_{\mathsf{ZeroShr}}$ imply indistinguishable simulated views for $\mathcal{A}$. The difference between H_1 and H_2 is on the generation of $[m_z]_{i-1}$ and $[m_z]_{i+1}$. To prove H_1 and H_2 are indistinguishable, we construct the following intermediate games.

- $\mathsf{H}_{1,1}$: Instead of generating $[\alpha]_{i-1} = \mathsf{PRF}_{\mathsf{key}_{i-1}}(\mathsf{id}_\alpha) \oplus \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id}_\alpha)$, $\mathsf{H}_{1,1}$ randomly samples $r_{\mathsf{sim}} \xleftarrow{\$} \mathbb{Z}_2$ and compute $[\alpha]_{i-1} = r_{\mathsf{sim}} \oplus \mathsf{PRF}_{\mathsf{key}_i}(\mathsf{id}_\alpha)$.
- $\mathsf{H}_{1,2}$: The only difference between $\mathsf{H}_{1,2}$ and $\mathsf{H}_{1,1}$ is that $\mathsf{H}_{1,2}$ generate $[m_z]_{i-1} \xleftarrow{\$} \mathbb{Z}_2$.
- $\mathsf{H}_{1,3}$: Instead of generating $[\alpha]_{i+1} = \mathsf{PRF}_{\mathsf{key}_{i+1}}(\mathsf{id}_\alpha) \oplus \mathsf{PRF}_{\mathsf{key}_{i-1}}(\mathsf{id}_\alpha)$, $\mathsf{H}_{1,3}$ randomly samples $[\alpha]_{i+1} \xleftarrow{\$} \mathbb{Z}_2$.
- $\mathsf{H}_{1,4}$: The only difference between $\mathsf{H}_{1,3}$ and $\mathsf{H}_{1,4}$ is that $\mathsf{H}_{1,4}$ generate $[m_z]_{i+1} \xleftarrow{\$} \mathbb{Z}_2$.

Now we prove H_1 is indistinguishable from $\mathsf{H}_{1,4}$. First, suppose there exists a PPT distinguisher $\mathcal{D}$ that can distinguish H_1 and $\mathsf{H}_{1,1}$ with non-negligible probability ϵ, then we can construct an adversary $\mathcal{B}$ to distinguish a PRF from a random function with the same probability: $\mathcal{B}$ queries with an input id_α to an oracle that is either instantiated by a PRF or a random function and obtains o. $\mathcal{B}$ set $r_{\mathsf{sim}} \leftarrow o$ and all other views are generated exactly as H_1. $\mathcal{B}$ then invoke $\mathcal{D}$ over the view and output what $\mathcal{D}$ outputs. Note that when o is generated by a PRF, the view is identically distributed as in H_1. Otherwise, the view is identically distributed as in $\mathsf{H}_{1,1}$. By the security of PRF, H_1 is computationally indistinguishable from $\mathsf{H}_{1,1}$. $\mathsf{H}_{1,1}$ and $\mathsf{H}_{1,2}$ are identically distributed as both $[m_z]_{i-1}$ in $\mathsf{H}_{1,1}$ and $\mathsf{H}_{1,2}$ follow the uniform distribution over $\mathbb{Z}_2$. Following the same argument as above, we prove that $\mathsf{H}_{1,2}$ is computationally indistinguishable from $\mathsf{H}_{1,3}$, and $\mathsf{H}_{1,3}$ is identically distributed as $\mathsf{H}_{1,4}$. Note that $\mathsf{H}_{1,4}$ is exactly the same game as H_2.

Summing all up, the simulated view is computationally indistinguishable from the view from real-world execution. □

Proof of Theorem 2. Given the security of existing and our proposed sub-protocols, *e.g.*, Π_{AND4}, the security of Π_{PPA} is easy to see in the hybrid model. We provide the sketch security proof of Theorem 2 as follows:

- We first prove the security of operators $(\triangledown_N, \blacktriangledown_N)$ in the hybrid model. They are constructed on the top of XOR, AND2, AND3, and AND4 gates and our optimizations. We have developed secure 3PC protocols for $\mathcal{F}_{\mathsf{AND2}}$, $\mathcal{F}_{\mathsf{AND3}}$ and $\mathcal{F}_{\mathsf{AND4}}$, and prove their security. Also, our proposed techniques **XOR-then-Resharing** and AND-**Reusing** only involve local computation, which does not introduce additional communication, and thus does not require simulation. In this way, the security of these operators $(\triangledown_N, \blacktriangledown_N)_{N=2}^4$ can be derived in the ($\mathcal{F}_{\mathsf{AND2}}$, $\mathcal{F}_{\mathsf{AND3}}$, $\mathcal{F}_{\mathsf{AND4}}$)-hybrid model.
- Secondly, **Mixed-**AND2 utilizes $\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}$ to compute signals $[\![g_j]\!]$. And $\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}$ is implemented as protocol $\Pi_{\mathsf{AND2}}^{[\![\cdot]\!]}$ securely.

As PPA circuit only invokes these protocols and operators in black-box manners, we thus can guarantee the security of protocol Π_{PPA} in the ($\mathcal{F}_{\mathsf{AND2}}^{[\![\cdot]\!]}$, $\mathcal{F}_{\mathsf{AND2}}$, $\mathcal{F}_{\mathsf{AND3}}$, $\mathcal{F}_{\mathsf{AND4}}$)-hybrid model. □