Conan: Secure and Reliable Machine Learning Inference Against Malicious Service Providers

Hanxiao Chen, *Member, IEEE*, Hongwei Li, *Fellow, IEEE*, Meng Hao, *Member, IEEE*, Pengzhi Xing, *Graduate Student Member, IEEE*, Jia Hu, *Graduate Student Member, IEEE*, Wenbo Jiang, *Member, IEEE*, Tianwei Zhang, *Member, IEEE*, and Guowen Xu, *Senior Member, IEEE*

***Abstract*—In the Machine Learning as a Service paradigm, a service provider (e.g., a server) hosting a model offers inference APIs to clients, who can send their queries and receive the inference results. While most recent secure inference works focus on addressing privacy issues, they overlook the importance of checking the service quality and reliability. A malicious server may deviate from the protocol specification to deliberately provide incorrect services such as using low-quality models. Thus, it is necessary to design new solutions to empower clients to verify the server's model accuracy and inference integrity while protecting both parties' privacy. We present Conan, a new secure and reliable inference framework against malicious servers to achieve accuracy verification, inference integrity, and privacy simultaneously. In Conan, the server first commits to the model and proves in zero-knowledge that the committed model achieves the claimed accuracy. Then both parties perform secure inference on the committed model against the malicious server. To instantiate the above framework, we design generic maliciously secure two-party computation (2PC) protocols with a fixed corrupted party, which may be of independent interest. Our protocols achieve high efficiency by utilizing the advantage that the semi-honest party can check the behavior of the corrupted party. Furthermore, they support both arithmetic and Boolean circuit evaluation, a crucial attribute for secure inference on complicated machine learning models. We implement the fixed-corruption 2PC protocols for our secure and reliable inference. The experimental results show 1 ∼ 2 orders of magnitude improvements over conventional maliciously secure protocols in terms of communication and computation costs.**

***Index Terms*—Secure inference, secure two-party computation, zero-knowledge proof, privacy, integrity.**

Received 2 June 2025; revised 19 November 2025; accepted 12 December 2025. Date of publication 24 December 2025; date of current version 26 January 2026. This work was supported in part by Sichuan Science and Technology Program under Grant 2024ZHCG0188; in part by the National Natural Science Foundation of China under Grant 62020106013 and Grant 62502079; in part by China Postdoctoral Science Foundation under Grant BX20240053; and in part by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity Research and Development Program and CyberSG Research and Development Cyber Research Program Office. 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 the CyberSG Research and Development Program Office, Singapore. The associate editor coordinating the review of this article and approving it for publication was Dr. Yang Zhang. *(Corresponding author: Meng Hao.)*

Hanxiao Chen, Hongwei Li, Pengzhi Xing, Jia Hu, Wenbo Jiang, and Guowen Xu are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: hanxiao.chen@uestc.edu.cn; hongweili@uestc.edu.cn; p.xing@std.uestc.edu.cn; jiahu@std.uestc.edu.cn; wenbo_jiang@uestc.edu.cn; guowen.xu@foxmail.com).

Meng Hao is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: menghao303@gmail.com).

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

This article has supplementary downloadable material available at https://doi.org/10.1109/TIFS.2025.3648121, provided by the authors.

Digital Object Identifier 10.1109/TIFS.2025.3648121

I. Introduction

MACHINE Learning as a Service (MLaaS) has attracted wide attention and facilitated various real-world applications [1], [2], [3]. In this paradigm, service providers (e.g., a remote server) usually deploy and provide inference services to clients under a pay-as-you-go pricing model. Generally, the server hosts a model M with parameters $\mathbf{W}$ and provides inference APIs to the public. Clients can send queries $\mathbf{x}$ to the server and receive the corresponding results $\mathsf{M}(\mathbf{W}, \mathbf{x})$. Due to the intellectual property and privacy concerns, it has become an urgent requirement to protect the privacy of both clients' queries and the server's model parameters. To this end, several works proposed *secure inference* in the semi-honest setting [4], [5], [6] based on secure two-party computation (2PC) techniques. These schemes ensure that the server learns nothing about $\mathbf{x}$, while the client learns nothing about $\mathbf{W}$ beyond $\mathsf{M}(\mathbf{W}, \mathbf{x})$ and what can be deduced from $\mathsf{M}(\mathbf{W}, \mathbf{x})$.

However, existing secure inference solutions lack mechanisms to verify whether the service provider is faithfully adhering to the protocol specification. On the one hand, as shown in recent works [7], [8], a malicious adversary may corrupt the server, leading it to utilize low-quality models for rendering services, instead of the claimed high-quality ones. This strategy effectively saves the server's computation resources, but the resulting inaccurate or incorrect inference results may cause severe consequences, such as misleading medical diagnoses [2]. On the other hand, there are instances where the server deviates from the protocols, aiming to steal the client's private inputs [9], [10]. Worse still, this deviation can be performed in an undetectable manner, namely, the server steals the client's inputs while still delivering seemingly correct inference services. A detailed analysis is provided in Section VIII-B. These vulnerabilities highlight the critical need for stricter security guarantees within inference services.

Given the aforementioned discussions, we aim to address an important problem, namely *secure and reliable inference resilient to malicious servers*. Our ultimate objective is to ensure correct inference services on the claimed accurate model while preserving the client's and server's privacy. We detail the specific requirements in this setting as follows.

(1) *Accuracy*: Before requesting inference services, the client should be able to verify whether the accuracy of the server's model meets the claimed threshold. (2) *Integrity*: The client should be able to detect any deviations by the server from the specified protocols on the claimed model with an overwhelming probability. (3) *Privacy*: The proposed scheme should prevent the malicious server from stealing the client's inputs and outputs while protecting the privacy of the server's model.

Existing solutions fail to fully meet the above requirements or possess certain limitations. For example, general-purpose maliciously secure 2PC protocols, like SPDZ [11] and its variants [12], [13], [14], [15], [16], [17], can achieve privacy, but cannot verify the model's accuracy and ensure that inferences are executed on the claimed model (i.e., integrity goal). In addition, many recent works explore the use of zero-knowledge (ZK) proofs for ML inference tasks [7], [8], where the server (as the prover) proves to the client (as the verifier) that the inference is performed on the claimed model without leaking the server's model. However, such methods cannot protect the privacy of the verifier's secrets (i.e., the client's query samples). More related work is discussed in Section VIII. Overall, a seemingly straightforward solution is to combine ZK proofs with maliciously secure 2PC techniques to address all requirements. Unfortunately, these paradigms are inherently challenging to reconcile, as 2PC relies on secret sharing between parties, whereas ZK assumes the secret resides solely with the prover.

To fill the gap, we propose a new secure and reliable framework called Conan, composed of two components. The first is *commit and accuracy verify*. The server first commits to the model parameters and proves in zero-knowledge that the committed model achieves the claimed accuracy on a client-chosen testing dataset. The second is *secure inference on the committed model against malicious servers*, built on our generic maliciously secure 2PC protocols with a fixed corrupted party. We carefully address the compatibility of the two components. Our protocols achieve high efficiency by utilizing the advantage that the semi-honest party can check the behavior of the corrupted party, while supporting both arithmetic and Boolean circuit evaluation. It is important for secure inference since ML models generally consist of alternating linear and non-linear layers, which require being evaluated in arithmetic and Boolean circuits, respectively.

To carefully merge ZK and 2PC techniques, our protocols are constructed upon *authenticated shares* (refer to Section II-A), which are infrequently explored concurrently. To evaluate authenticated multiplication in arithmetic circuits, prior works utilize SPDZ-style protocols [11], [12], [14], but their costs are prohibitively high. Benefitting from our fixed corruption setting, our insight is to split the authenticated multiplication on secret-shared values into two sub-multiplications on non-shared values such that we can invoke an efficient ZK proof to verify the computation's correctness. Note that the idea of splitting multiplication is fundamentally different from Delphi [18] and Muse [19], which aim to move heavy cryptographic operations offline. To evaluate arbitrary non-linear functions securely, our protocols for Boolean circuits adopt the garbled circuits (GC) paradigm (refer to Section II-C) because GC is inherently secure against a malicious evaluator [20], allowing us to assign the corrupted party as the evaluator. To achieve better performance, we present two constructions for an improved GC-based solution. Further technical details are provided in Section III. Our contributions can be summarized as follows.

- We provide the first secure and reliable inference framework against malicious servers, Conan, which simultaneously guarantees the model accuracy, inference integrity, and privacy of both parties.
- The core of our framework consists of generic maliciously secure 2PC protocols with a fixed corrupted party, supporting both arithmetic and Boolean circuit evaluation.
- Extensive experiments demonstrate that our protocols outperform general-purpose maliciously secure protocols by up to 163× in communication and 67× in computation.

II. Preliminaries

Notation: Let κ and λ be the computational and statistical security parameters, respectively. $a := b$ denotes a is assigned by b. For $n \in \mathbb{N}$, $[n] := \{1, \ldots, n\}$. For any $a \leq b$, $[a, b] := \{a, \ldots, b\}$. $x \leftarrow S$ denotes the operation of sampling x randomly from a finite set S. We use bold lower-case letters like $\mathbf{x}$ for column vectors and bold upper-case letters like $\mathbf{X}$ for matrices. For any two distributions X and Y, $X \approx_c Y$ denotes computational indistinguishability. $\mathsf{negl}(\cdot)$ denotes a negligible function. For a value x and $i \in [|x|]$, $x[i]$ denotes the i-th bit of x and $x[1]$ is the least significant bit.

Fixed-point Encoding and Truncation: ML inference performs computations on floating-point numbers, whereas our protocols work for integers in a field $\mathbb{F}_p$ (e.g., $\mathbb{Z}_p$). To represent a floating-point number $x \in \mathbb{Q}$ in $\mathbb{Z}_p$, similar to prior works [4], [19], [20], we encode it as a fixed-point integer $a := \lfloor x \cdot 2^{st} \rfloor \bmod p$ with scale st. As shown in the recent work [4], there is no loss in accuracy in fixed-point representations. Fixed-point arithmetic is performed on the encoded input values and the same scale st is maintained for all the intermediate results. To this end, we need to truncate intermediate values and our protocols can do this for free within garbled circuits.

A. Arithmetic Secret Sharing and It-MACs

We utilize 2-out-of-2 arithmetic secret sharing [21] over a finite field $\mathbb{F}_p$, combined with IT-MACs [22], [23] to ensure malicious security. Given a secret $x \in \mathbb{F}_p$, the arithmetic secret shares in Conan are denoted as $x_{\mathcal{C}} \in \mathbb{F}_p$ and $x_{\mathcal{S}} \in \mathbb{F}_p$, held by $\mathcal{C}$ and $\mathcal{S}$, respectively, such that $x = x_{\mathcal{C}} + x_{\mathcal{S}}$ in $\mathbb{F}_p$. The security satisfies that given $x_{\mathcal{C}}$ or $x_{\mathcal{S}}$, x is perfectly hidden. In addition, similar to existing maliciously secure protocols [14], [20], [24], we use IT-MACs [22] to authenticate values over $\mathbb{F}_p$. Let $\Delta \in \mathbb{F}_p$ be a randomly sampled global key, a value $x \in \mathbb{F}_p$ owned by $\mathcal{S}$ can be authenticated by giving $\mathcal{C}$ a uniform key $K_x \in \mathbb{F}_p$ and giving $\mathcal{S}$ a MAC tag $M_x \in \mathbb{F}_p$, such that $M_x := K_x + \Delta \cdot x \in \mathbb{F}_p$. Although full-maliciously secure protocols [11], [23] require Δ to be secret-shared between parties, in Conan we have the semi-honest party $\mathcal{C}$ pick and hold Δ, similar to

[19] and [20]. There are two properties of IT-MACs [11], [22]: information-theoretic binding and additively homomorphic.

- Information-theoretic security. Authentication based on IT-MACs provides information-theoretic security, which means that the probability that a malicious adversary forges the value x to x' is $1/|\mathbb{F}_p|$, where we require $p \geq 2^\lambda$.
- Additive homomorphism. Authentication based on IT-MACs is additively homomorphic. Taking authenticated values as an example, given two public constants a_1, a_2 and two authenticated values $[\![x_1]\!], [\![x_2]\!]$, the authenticated value $[\![a_1 \cdot x_1 + a_2 \cdot x_2]\!]$ can be computed locally by the two parties.

In Conan, $\langle x \rangle$ denotes the *authenticated values* based on IT-MACs as above. $[x]$ denotes the *authenticated secret shares* of x, structured as a tuple $(x_\mathcal{C}, x_\mathcal{S}, K_{x_\mathcal{S}}, M_{x_\mathcal{S}})$, where $x = x_\mathcal{C} + x_\mathcal{S} \in \mathbb{F}_p$, $M_{x_\mathcal{S}} = K_{x_\mathcal{S}} + \Delta \cdot x_\mathcal{S} \in \mathbb{F}_p$, $(x_\mathcal{C}, K_{x_\mathcal{S}})$ and $(x_\mathcal{S}, M_{x_\mathcal{S}})$ are held by $\mathcal{C}$ and $\mathcal{S}$, respectively.

B. Additively Homomorphic Encryption

An additively homomorphic encryption (AHE) scheme consists of four algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$.

- $\mathsf{KeyGen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$: given security parameter κ, it returns a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m) \rightarrow c$: given pk and a message $m \in \mathbb{F}_p$, it returns a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, c) \rightarrow m$: given sk and a ciphertext c, it returns a message m.
- $\mathsf{Eval}(\mathsf{pk}, c_1, c_2, g) \rightarrow c_3$: given pk, two ciphertexts c_1 and c_2 with $c_1 := \mathsf{Enc}(\mathsf{pk}, m_1)$ and $c_2 := \mathsf{Enc}(\mathsf{pk}, m_2)$, and a linear function g, it returns a ciphertext c_3 with $g(m_1, m_2) = \mathsf{Dec}(\mathsf{sk}, c_3)$. This corresponds to the additive homomorphism of AHE.

An AHE scheme satisfies the following properties: correctness, semantic security, and function privacy.

- Correctness. An AHE is correct, if for each message $m \in \mathbb{F}_p$, it holds that $\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, x)) \neq x \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)] \leq negl(\kappa)$.
- Semantic security. An AHE is semantically secure, if for any two messages $m_1, m_2 \in \mathbb{F}_p$ and all PPT adversaries $\mathcal{A}$, it holds that $\{\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1)\} \approx_c \{\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_2)\}$, where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$.
- Function privacy. Informally, an AHE satisfies function privacy, which guarantees that c_3 does not leak any information about the function g even given sk, where $c_3 \leftarrow \mathsf{Eval}(\mathsf{pk}, c_1, c_2, g)$. Refer to works [25], [26] for the formal definition.

C. Garbled Circuits

Garbled circuits (GC) [27] is a general scheme for securely evaluating an arbitrary Boolean circuit. It contains a pair of algorithms $(\mathsf{Garble}, \mathsf{GCEval})$ as follows.

- $\mathsf{Garble}(1^\kappa, f) \rightarrow (\mathsf{GC}, \{\{\mathsf{lab}^{\mathsf{in}}_{i,j}\}_{i\in[n]}, \{\mathsf{lab}^{\mathsf{out}}_{i,j}\}_{i\in[m]}\}_{j\in\{0,1\}})$: given the security parameter κ and a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, it returns a garbled circuit GC, a set of input labels $\{\mathsf{lab}^{\mathsf{in}}_{i,j}\}_{i\in[n], j\in\{0,1\}}$, and a set of output labels $\{\mathsf{lab}^{\mathsf{out}}_{i,j}\}_{i\in[m], j\in\{0,1\}}$.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Upon receiving $m_0 \in \{0,1\}^n$ and $m_1 \in \{0,1\}^n$ from the sender and $b \in \{0,1\}$ from the receiver, send m_b to the receiver.

Fig. 1. Functionality for OT.

- $\mathsf{GCEval}(\mathsf{GC}, \{\hat{\mathsf{lab}}^{\mathsf{in}}_i\}_{i\in[n]}) \rightarrow \{\hat{\mathsf{lab}}^{\mathsf{out}}_i\}_{i\in[m]}$: given GC and labels $\{\hat{\mathsf{lab}}^{\mathsf{in}}_i\}_{i\in[n]}$ corresponding to an input $x \in \{0,1\}^n$, it returns labels $\{\hat{\mathsf{lab}}^{\mathsf{out}}_i\}_{i\in[m]}$ corresponding to the output $f(x) \in \{0,1\}^m$.

A GC scheme satisfies the properties: correctness, security, and authenticity.

- Correctness. For any function f and $x \in \{0,1\}^n$, $\mathsf{GCEval}(\mathsf{GC}, \{\mathsf{lab}^{\mathsf{in}}_{i,x[i]}\}_{i\in[n]}) = \{\mathsf{lab}^{\mathsf{out}}_{i,f(x)[i]}\}_{i\in[m]}$.
- Security. For any function f and $x \in \{0,1\}^n$, the view of $\mathsf{Sim}(1^\kappa, f)$ generated by a simulator Sim is computationally indistinguishable to $(\mathsf{GC}, \{\mathsf{lab}^{\mathsf{in}}_{i,x[i]}\}_{i\in[n]})$.
- Authenticity. For any function f and $x \in \{0,1\}^n$, given $(\mathsf{GC}, \{\mathsf{lab}^{\mathsf{in}}_{i,x[i]}\}_{i\in[n]})$, it is infeasible to guess $\{\mathsf{lab}^{\mathsf{out}}_{i,1\oplus f(x)[i]}\}_{i\in[m]}$.

Our protocol uses the instantiation of garbling schemes with *point-and-permute* (PaP) optimization [28]. For each AND gate in GC, the garbler generates a table with four ciphertexts, namely a label of each input wire is used as a key to encrypt the corresponding label of the output wire. Therefore, the evaluator needs to test all four ciphertexts to find the correct one. The idea of PaP is to interpret a part of the label as a *pointer* to the table, where the encryption will be placed. Specifically, for each wire w with label $\{\mathsf{lab}_{w,j}\}_{j\in\{0,1\}}$, PaP prepends $\mathsf{lab}_{w,j}$ as $k_{w,j} \| p_{w,j} \in \{0,1\}^{\kappa-1} \times \{0,1\}$ with the constraint $p_{w,0} \oplus p_{w,1} = 1$. With this technique, the evaluator only decrypts the uniquely pointed ciphertext.

D. Oblivious Transfer

Conan utilizes 1-out-of-2 oblivious transfer (OT), where a *sender* inputs two n-bit messages $m_0, m_1 \in \{0,1\}^n$ and a *receiver* inputs a choice bit $b \in \{0,1\}$. At the end of the protocol, the receiver obtains m_b and the sender receives nothing. The corresponding functionality $\mathcal{F}_{\mathsf{OT}}$ is shown in Figure 1. Conan requires OT that is secure against a semi-honest sender and a malicious receiver. To achieve this, we use the OT instantiation proposed in [29].

E. Zero-Knowledge Proof for Inner Products

QuickSilver [24] provides a state-of-the-art ZK protocol for proving the inner product $z = \mathbf{x} \cdot \mathbf{y}$. Formally, a *prover* and a *verifier* hold the authentication of $\mathbf{x}$ and $\mathbf{y}$, and a public z. The prover aims to prove to the verifier that $g_z(\mathbf{x}, \mathbf{y}) := \mathbf{x} \cdot \mathbf{y} - z$ is equal to 0. The main idea is that the following equation holds

$$\overbrace{g_z(\mathbf{K_x}, \mathbf{K_y})}^{\text{denoted as } b} - z \cdot \Delta^2 = \overbrace{(\mathbf{M_x} \cdot \mathbf{M_y} - z)}^{\text{denoted as } a_0} - \overbrace{(\mathbf{M_y} \cdot \mathbf{x} + \mathbf{M_x} \cdot \mathbf{y})}^{\text{denoted as } a_1} \cdot \Delta. \quad (1)$$

Functionality $\mathcal{F}_{\mathsf{VOLE}}$

Parameters: A finite field $\mathbb{F}_p$.
Initialize: Upon receiving (init) from $\mathcal{C}$ and $\mathcal{S}$, sample $\Delta \leftarrow \mathbb{F}_p$ and send Δ to $\mathcal{C}$.
Extend: Upon receiving (extend, n) from $\mathcal{C}$ and $\mathcal{S}$, do the following:

1) Sample $\mathbf{k} \leftarrow \mathbb{F}_p^n$.
2) If $\mathcal{S}$ is honest, sample $\mathbf{s} \leftarrow \mathbb{F}_p^n$ and compute $\mathbf{m} := \mathbf{k} + \mathbf{s} \cdot \Delta \in \mathbb{F}_p^n$. Otherwise, receive $\mathbf{s}, \mathbf{m} \in \mathbb{F}_p^n$ from the adversary and recompute $\mathbf{k} := \mathbf{m} - \mathbf{s} \cdot \Delta \in \mathbb{F}_p^n$.
3) Output $(\mathbf{s}, \mathbf{m})$ to $\mathcal{S}$ and $\mathbf{k}$ to $\mathcal{C}$.

Fig. 2. Functionality for VOLE.

Observe that (b, Δ) is known to the verifier and (a_0, a_1) is known to the prover. Thus, this equation is a linear relationship, verifiable via the following method.

BatchCheck: Note that n such relationships can be checked simultaneously by employing a random linear combination. Suppose the prover owns $\{a_{0,i}, a_{1,i}\}_{i\in[n]}$ and the verifier owns $\{b_i\}_{i\in[n]}$ and Δ such that $b_i = a_{0,i} - a_{1,i} \cdot \Delta$. The two parties first generate a random vector oblivious linear evaluation (VOLE) [30], [31] relationship $b^* = a_0^* - a_1^* \cdot \Delta$, where the prover holds (a_0^*, a_1^*) and the verifier holds (b^*, Δ). The functionality $\mathcal{F}_{\mathsf{VOLE}}$ is presented in Figure 2. Then, the verifier samples a random χ and sends it to the prover, who computes and sends $A_0^* := \sum_{i=1}^n a_{0,i} \cdot \chi^i + a_0^*$ and $A_1^* := \sum_{i=1}^n a_{1,i} \cdot \chi^i + a_1^*$ to the verifier. Finally, the verifier computes $B^* := \sum_{i=1}^n b_i \cdot \chi^i + b^*$ and then checks whether $B^* = A_0^* - A_1^* \cdot \Delta$. If the underlying values (i.e., $\mathbf{x}, \mathbf{y}, z$) are not computed correctly, then the above relationship can hold only with probability $(2 + n)/|\mathbb{F}_p|$.

III. Technique Overview

A. Threat Model

Conan works in the malicious-server setting. Namely, the adversary can corrupt the client but is restricted to be semi-honest, i.e., following the protocol specification strictly. Alternatively, it can corrupt the server and behave maliciously, i.e., deviating from the protocol arbitrarily [11], [14]. Specifically, the malicious server may trick the client by using a low-quality model and stealing the client's queries by carefully manipulating the inference process [7], [8]. Besides, the semi-honest client also attempts to infer the server's model parameters based on the received messages. Therefore, our goal is to ensure correct inference services on the claimed accurate model while preserving the privacy of both parties.

We would like to clarify that, similar to existing secure inference works based on secure two-party computation [6], [10], [18], [32], [33], [34] and zero-knowledge proof [7], [8], [35], our protocols can protect against explicit privacy leaks of inference samples and model parameters during the inference process. Therefore, malicious privacy attacks, including model extraction and model inversion attacks, fall outside the scope of all 2PC-based secure inference protocols, regardless of whether the client is malicious [7], [10], [35]. As we have discussed in the Introduction, such protocols ensure that the server learns nothing about queries $\mathbf{x}$, while the client learns nothing about model parameters $\mathbf{W}$ beyond $\mathsf{M}(\mathbf{W}, \mathbf{x})$ and what can be deduced from $\mathsf{M}(\mathbf{W}, \mathbf{x})$, where M denotes the target model with parameters $\mathbf{W}$. There is currently a fundamental gap between machine learning attacks and cryptography-based secure protocols, as they address different aspects and belong to separate research areas. To enhance existing secure inference protocols with adversarial robustness, a promising approach is to adapt corresponding defenses into the cryptographic environment. This might be a valuable direction for future research.

B. Our Secure and Reliable Inference Framework

Figure 3 shows a high-level overview of Conan, which comprises two phases. Phase ❶: *Commit and accuracy verify*. The server $\mathcal{S}$ first commits to the claimed model and then proves in zero-knowledge to the client $\mathcal{C}$ that it achieves the claimed accuracy on a client-chosen testing dataset. This can be achieved using any ZK proof protocol that follows the "commit-and-prove" paradigm, as long as the committed model remains compatible with the subsequent maliciously secure inference procedure in Conan. Therefore, we instantiate this phase using VOLE-based ZK proofs, which utilize information-theoretic message authentication codes (IT-MACs) [22], [23] to commit to the model parameters w with the following form: $\mathcal{C}$ holds (Δ, K_x) and $\mathcal{S}$ holds (x, M_x) such that $M_x = K_x + \Delta \cdot x$. Phase ❷: *Secure inference on the committed model against malicious servers*. Once accuracy is verified, $\mathcal{C}$ requests inference services on its private inputs using the committed model. To ensure correct execution by $\mathcal{S}$, we propose a generic maliciously secure 2PC protocol with fixed corruption as follows, operating over committed values with rigorous integrity and privacy guarantees.

Maliciously secure 2PC protocol with fixed corruption: This setting has gained ever-growing interest in recent works [10], [19], [20]. Our protocols are crucially built on the observation that the semi-honest client can check the possibly malicious behavior of the corrupted server. Note that each value x in our protocols is in the form of authenticated secret shares (refer to Section II-A), i.e., $\mathcal{C}$ holds $(x_{\mathcal{C}}, K_{x_{\mathcal{S}}})$ and $\mathcal{S}$ holds $(x_{\mathcal{S}}, M_{x_{\mathcal{S}}})$ such that $x = x_{\mathcal{C}} + x_{\mathcal{S}}$ and $M_{x_{\mathcal{S}}} = K_{x_{\mathcal{S}}} + \Delta \cdot x_{\mathcal{S}}$. Our protocols can evaluate both arithmetic and Boolean circuits, and all their inputs/outputs follow the above authenticated sharing form. This is important since ML models generally consist of alternating linear layers (such as convolution and fully connected layers) and non-linear layers (e.g., ReLU, Maxpooling). The former is evaluated with arithmetic circuits, while the latter is complicated and requires invoking Boolean circuits [15], [19], [20].

(1) New protocols for arithmetic circuits. We focus on multiplication, as addition over authenticated shares can be easily evaluated via the additive homomorphism. A straightforward method is to utilize maliciously secure multiplication schemes such as SPDZ-family [11], [12], [14]. However, the cost is unsatisfactory due to their reliance on expensive maliciously secure OT or fully FHE with ZK proofs to generate

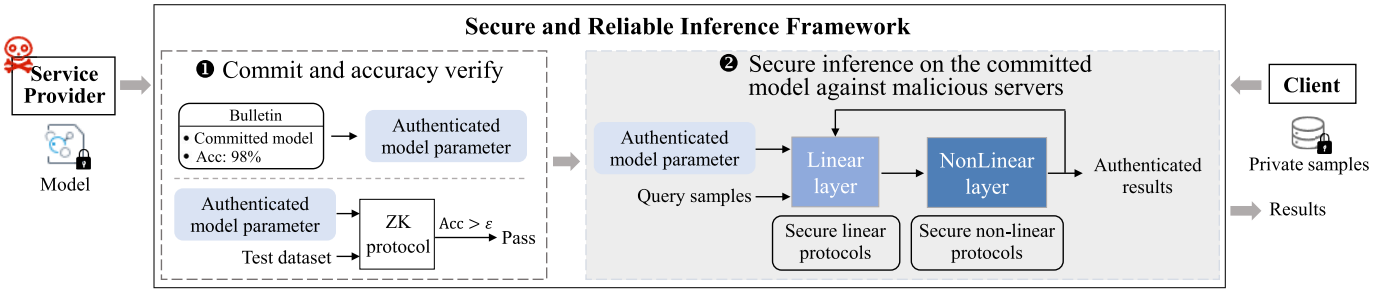


Fig. 3. A high-level Overview of Conan.

authenticated Beaver triples. To remedy this, we make the following attempts.

Attempt 1. We try to utilize ZK proofs such as QuickSilver [24] to evaluate multiplication, where the malicious server acts as the prover and the semi-honest client as the verifier. However, ZK proofs are ill-suited for our setting, as they assume the input resides solely with the prover. In contrast, secure inference requires evaluating a function on secret-shared inputs from both the client and the server. This limitation cannot be addressed without compromising privacy.

Attempt 2. We explore extending advanced maliciously secure 2PC frameworks such as Overdrive [14] to the simpler 2PC setting where only one party is malicious. In this setting, the MAC key Δ can be owned by the semi-honest party, rather than being secret-shared among the parties as in Overdrive. This allows us to bypass distributed decryption, triple sacrifice, and certain ZK proofs during the Beaver triple generation process, by leveraging the client's knowledge of Δ [19]. However, despite these optimizations, we still need to generate authenticated triples based on expensive primitives including maliciously secure OT and FHE with ZK proofs.

Final solution. Our final idea lies in allowing malicious $\mathcal{S}$ to execute the possibly incorrect evaluation, followed by semi-honest $\mathcal{C}$ verifying the correctness of $\mathcal{S}$'s behavior. A pivotal component of our design is to solely authenticate $x_{\mathcal{S}}$, the secret share held by $\mathcal{S}$, using the global key Δ, instead of authenticating the original secret x as in SPDZ-family [11], [12]. We now outline the idea for computing $z = x \cdot y$, where x, y, and z are in the form of authenticated secret shares. Specifically, expanding the product of shares, z can be represented as $z = x_{\mathcal{C}} \cdot y_{\mathcal{C}} + x_{\mathcal{S}} \cdot y_{\mathcal{S}} + x_{\mathcal{S}} \cdot y_{\mathcal{C}} + x_{\mathcal{C}} \cdot y_{\mathcal{S}}$. The first two items $\mu := x_{\mathcal{C}} \cdot y_{\mathcal{C}}$ and $\nu := x_{\mathcal{S}} \cdot y_{\mathcal{S}}$ can be locally evaluated by $\mathcal{C}$ and $\mathcal{S}$, respectively. Since a malicious $\mathcal{S}$ may compute ν incorrect, $\mathcal{C}$ must verify ν is indeed the product of $x_{\mathcal{S}}$ and $y_{\mathcal{S}}$. The main advantage of our authentication form is that this verification can be performed efficiently using ZK proofs in QuickSilver [24]. The remaining cross-terms, $\omega := x_{\mathcal{S}} \cdot y_{\mathcal{C}}$ and $\delta := x_{\mathcal{C}} \cdot y_{\mathcal{S}}$, can be evaluated efficiently by utilizing the additive homomorphism of IT-MACs combined with a standard MAC check. Because IT-MACs (including those used in QuickSilver) support amortized checking, our maliciously secure multiplication protocol achieves efficiency comparable to that of semi-honest protocols.

We note that some recent works [19], [20] also designed secure multiplication protocols against a single malicious party. However, these protocols are designed for multiplying an authenticated value by a plain (unauthenticated) value. Consequently, they cannot be directly extended to our setting, where both multiplicands are in authenticated form.

(2) Optimized protocols for Boolean circuits. Given an input x, a Boolean circuit computing a function f_B is formalized as $y := f_B(x)$. We require a maliciously secure Boolean circuit evaluation protocol with fixed corruption while ensuring computation correctness and input consistency. Inspired by SIMC [20], we adopt the GC paradigm because GC is inherently secure against a malicious evaluator, allowing us to assign the corrupted party as the evaluator.

We present two constructions for an improved GC-based solution. The first is a label-based multiplication protocol in GC. SIMC [20] evaluates this operation by using GC's output labels as one-time pad encryption keys and then transferring two encrypted messages between parties. We optimize the communication cost by exploiting the correlation between these two messages. The idea is to convert a specific message into a designed randomness related to the corresponding GC label and, based on it, construct another message according to the correlation. The second is a most significant bit (MSB)-based comparison protocol in $\mathbb{F}_p$. Comparison is the fundamental block for non-linear functions in ML [4], [6]. Our insight is that it can be evaluated for free by extracting the input's MSB in GC, without affecting correctness. We provide a rigorous theoretical analysis of its feasibility.

IV. Maliciously Secure 2PC Protocol With Fixed Corruption

In this section, we present our 2PC protocols secure against a designated malicious party, realizing the functionality defined in Figure 4. The core of our construction relies on tailored protocols for arithmetic and Boolean circuits, which may be of independent interest.

A. Protocols for Arithmetic Circuits

As described in Section III, here we focus on the evaluation of authenticated multiplication gates, i.e., $[z] = [x] \cdot [y]$. Given that we have already discussed the intuition of our multiplication protocol in Section III, we now directly describe the protocol details. As presented in Figure 5, the protocol can be divided into two steps: secure multiplication evaluation and correctness verification.

Functionality $\mathcal{F}_{\mathsf{2PC}}$

This functionality is parameterized by a field $\mathbb{F}_p$ with $\ell = \lceil \log p \rceil$.

Input: On input $(\mathsf{Input}, \mathcal{S}, vid, x)$ from $\mathcal{S}$ and $(\mathsf{Input}, \mathcal{S}, vid)$ from $\mathcal{C}$, where vid is a fresh identifier and $x \in \mathbb{F}_p$, store (vid, x). It is symmetrical for $\mathcal{C}$'s input.

Arithmetic Circuit: On command $(\mathsf{Arithmetic}, f_A, vid_1, \ldots, vid_n, vid)$ from both parties, where $f_A : \mathbb{F}_p^n \to \mathbb{F}_p$ is an arithmetic circuit, $vid_1, \ldots, vid_n$ are defined identifiers and vid is a fresh identifier, retrieve $(vid_1, x_1), \ldots, (vid_n, x_n)$ and store (vid, y) with $y = f_A(x_1, \ldots, x_n)$. Check that $[x]$ is valid and abort if not.

Boolean Circuit: On command $(\mathsf{Boolean}, f_B, vid_1, \ldots, vid_n, vid)$ from both parties, where $f_B : \{0,1\}^{\ell \cdot n} \to \{0,1\}^{\ell}$ is a Boolean circuit, $vid_1, \ldots, vid_n$ are defined identifiers and vid is a fresh identifier, retrieve $(vid_1, x_1), \ldots, (vid_n, x_n)$ and store (vid, y) with $y = f_B(x_1, \ldots, x_n)$ and $y \in \mathbb{F}_p$. Check that $[x]$ is valid and abort if not.

Output: On input $(\mathsf{Output}, \mathcal{P}, vid)$ from both parties, where $\mathcal{P} \in \{\mathcal{C}, \mathcal{S}\}$ and vid is a defined identifier, retrieve (vid, x) and output x to $\mathcal{P}$.

Fig. 4. Ideal functionality for our maliciously secure 2PC protocol with fixed corruption.

1) Secure Multiplication Evaluation: Step 1 in Figure 5 gives the details of our secure multiplication evaluation. Given $x = x_{\mathcal{C}} + x_{\mathcal{S}}$ and $y = y_{\mathcal{C}} + y_{\mathcal{S}}$, we have $z = x_{\mathcal{C}} \cdot y_{\mathcal{C}} + x_{\mathcal{S}} \cdot y_{\mathcal{S}} + x_{\mathcal{S}} \cdot y_{\mathcal{C}} + x_{\mathcal{C}} \cdot y_{\mathcal{S}}$. Therefore, we can compute these four items separately and then simply sum them up to obtain z. Specifically, the first two items $\mu := x_{\mathcal{C}} \cdot y_{\mathcal{C}}$ and $\nu := x_{\mathcal{S}} \cdot y_{\mathcal{S}}$ can be locally evaluated by $\mathcal{C}$ and $\mathcal{S}$, respectively. For the last two items $\omega := x_{\mathcal{S}} \cdot y_{\mathcal{C}}$ and $\delta := x_{\mathcal{S}} \cdot y_{\mathcal{C}}$, both parties can obtain $[\omega]$ and $[\delta]$ by utilizing standard AHE schemes and the additive homomorphism of IT-MACs (steps 1c and 1d). It is worth noting that unlike prior maliciously secure protocols such as SPDZ-family [11], [12], [14] that authenticate the true intermediate values during the protocol execution, here we only authenticate the values owned by malicious $\mathcal{S}$ using a single global key Δ held by $\mathcal{C}$. Benefiting from this, we can leverage the additive homomorphism of IT-MACs and the ZK proofs provided in Section II to verify the computation correctness efficiently as discussed below.

2) Correctness Verification: Step 2 in Figure 5 details our correctness verification procedure. Given that $\mathcal{S}$ is malicious, it is necessary to check both the input consistency (for $x_{\mathcal{S}}$ and $y_{\mathcal{S}}$) and computation correctness (i.e., $\nu = x_{\mathcal{S}} \cdot y_{\mathcal{S}}$ and $z = x \cdot y$). Since $x_{\mathcal{S}}$ and $y_{\mathcal{S}}$ are authenticated using IT-MACs with information-theoretic security, we ensure input consistency by verifying their MAC relations, i.e., $M_{x_{\mathcal{S}}} = K_{x_{\mathcal{S}}} + \Delta \cdot x_{\mathcal{S}}$ and $M_{y_{\mathcal{S}}} = K_{y_{\mathcal{S}}} + \Delta \cdot y_{\mathcal{S}}$. Notably, these consistency checks, along with the verification of $\nu = x_{\mathcal{S}} \cdot y_{\mathcal{S}}$, can be proved together using the BatchCheck mechanism described in Section II. Further, verifying $z = x \cdot y$ reduces to simply checking the validity of the IT-MAC for z. This is enabled by the additive homomorphism of the authentication scheme. Recall that z is computed as $z = \mu + \nu + \omega + \delta$. With the exception of ν (which is verified explicitly as described above), all other terms are derived via additively homomorphic operations. Consequently, if any error is introduced during this process, the IT-MAC for z will inevitably be invalid.

B. Protocols for Boolean Circuits

A Boolean circuit f_B takes input $[x]$ and outputs $[y]$ with $y := f_B(x)$. As noted in Section III, in the maliciously secure 2PC setting with fixed corruption, a common choice for instantiating this circuit is using GC [19], [20], since GC is inherently secure against a malicious evaluator. Specifically, we can use a standard semi-honest secure GC to implement f_B and transmit the labels corresponding to $\mathcal{C}$'s share receiver-malicious OT. SIMC [20] introduces an advanced GC-based protocol with two designs. More details can be found in Section 3 in [20].

- To ensure input consistency, SIMC re-generates an authentication for x and later verifies the equality of two independently generated authentications on x. It is worth to noticing that different from our protocol, SIMC maintains the following authentication form: $\mathcal{C}$ holds $(x_{\mathcal{C}}, K_x)$ and $\mathcal{S}$ holds $(x_{\mathcal{S}}, M_x)$ such that $x = x_{\mathcal{C}} + x_{\mathcal{S}}$ and $M_x = K_x + \Delta \cdot x$.
- To avoid expensive field multiplications (i.e., $\Delta \cdot y$ and $\Delta \cdot x$) within GC, SIMC garbles a circuit, that given $x_{\mathcal{C}}$ and $x_{\mathcal{S}}$, only computes y and x (and not their IT-MACs). Then it performs field multiplication by using the output labels of the GC as one-time pad masks. Briefly, taking $\Delta \cdot y$ as an example, we have $\Delta \cdot y = \sum_{i \in [\ell]} \Delta \cdot y[i] \cdot 2^i$ in $\mathbb{F}_p$. To compute shares of $\Delta \cdot y[i]$, $\mathcal{C}$ picks a random $r_i \in \mathbb{F}_p$ and mask r_i with $\mathsf{lab}^y_{i,0}$ and $\Delta + r_i$ with $\mathsf{lab}^y_{i,1}$. $\mathcal{C}$ sends two messages $(r_i + \mathsf{lab}^y_{i,0}, (\Delta + r_i) + \mathsf{lab}^y_{i,1})$ to $\mathcal{S}$, who can only unmask one of the two using the owned label as its share. Meanwhile, $\mathcal{C}$ sets its share as $-r_i \in \mathbb{F}_p$.

For better performance, we propose two optimized constructions for GC-based Boolean circuit evaluation, and the detailed protocol is presented in Figure 6. The first construction is a communication-efficient field multiplication procedure, described in Figure 7. Specifically, we parse the label $\mathsf{lab}^x_{i,j}$ as the concatenation $k^x_{i,j} \| p^x_{i,j}$. Leveraging the PaP technique (refer to Section II), our key insight is that $\mathcal{C}$ can set the message placed in the first position (i.e., $p^x_{i,j} = 0$) to $H(\mathsf{lab}^x_{i,j}) + H(\mathsf{lab}^x_{i,j})$ for each $i \in [\ell]$, by carefully designing the value r_i, and the other one can be derived from this r_i. For example, if $p^x_{i,0} = 0$, the message $r_i + H(\mathsf{lab}^x_{i,0})$ is placed in the first position. $\mathcal{C}$ sets r_i to $H(\mathsf{lab}^x_{i,0})$ and thus another message $u_i = \alpha + H(\mathsf{lab}^x_{i,0}) + H(\mathsf{lab}^x_{i,1})$. After obtaining u_i, $\mathcal{S}$ can learn the share of $\alpha \cdot x[i]$, i.e., $\hat{r}_i$ in step 4, based on the owned label. Meanwhile, $\mathcal{C}$ sets another share as $-r_i$. Compared with the method of SIMC [20], this optimization reduces communication overhead by half.

We further pay special attention to the comparison operation, an important building block for non-linear functions in ML, such as ReLU and Maxpooling. SIMC [20] evaluates it straightforwardly using GC, which requires ℓ multiplication

Protocol Π_{2PC} (Part 1)

Parameters: A finite field $\mathbb{F}_p$; an AHE scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$; $\mathcal{F}_{\mathsf{VOLE}}$; $\mathcal{F}_{\mathsf{OT}}^{\kappa}$; a GC scheme $(\mathsf{Garble}, \mathsf{GCEval})$; $\ell := \lceil \log p \rceil$. Semi-honest client $\mathcal{C}$ and malicious server $\mathcal{S}$.

Initialization:

1) $\mathcal{C}$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{AHE.KeyGen}(1^{\kappa})$ and sends pk to $\mathcal{S}$.
2) $\mathcal{C}$ and $\mathcal{S}$ send (Init) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns a uniformly random $\Delta \in \mathbb{F}_p$ to $\mathcal{C}$.

Input:

1) For each input x from $\mathcal{C}$, $\mathcal{C}$ samples $x_{\mathcal{S}}, M_{x_{\mathcal{S}}} \leftarrow \mathbb{F}_p$, and computes $x_{\mathcal{C}} := x - x_{\mathcal{S}} \in \mathbb{F}_p$ and $K_{x_{\mathcal{S}}} := M_{x_{\mathcal{S}}} - \Delta \cdot x_{\mathcal{S}} \in \mathbb{F}_p$. $\mathcal{C}$ sends $(x_{\mathcal{S}}, M_{x_{\mathcal{S}}})$ to $\mathcal{S}$ and both parties hold $[x]$.
2) For each input y from $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{S}$ send (Extend) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $(r, M_r) \in \mathbb{F}_p^2$ to $\mathcal{S}$ and $K_r \in \mathbb{F}_p$ to $\mathcal{C}$. $\mathcal{S}$ sends $y - r \in \mathbb{F}_p$ to $\mathcal{C}$ and then both parties compute $[y]$, where $\mathcal{S}$ holds $y_{\mathcal{S}} := r$ and $M_{y_{\mathcal{S}}} := M_r$, and $\mathcal{C}$ holds $y_{\mathcal{C}} := y - r$ and $K_{y_{\mathcal{S}}} := K_r$.

Arithmetic Circuit: Without loss of generality, the following protocol assumes that the arithmetic circuit f_A consists of only one multiplication gate[a].

1) Two parties hold $[x], [y]$ and evaluate the multiplication $z = x \cdot y$ to obtain $[z]$ as follows:
   a) $\mathcal{C}$ computes $\mu := x_{\mathcal{C}} \cdot y_{\mathcal{C}}$ locally.
   b) $\mathcal{C}$ and $\mathcal{S}$ send (Extend) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $(r, M_r) \in \mathbb{F}_p^2$ to $\mathcal{S}$ and $K_r \in \mathbb{F}_p$ to $\mathcal{C}$. $\mathcal{S}$ computes $\nu := x_{\mathcal{S}} \cdot y_{\mathcal{S}}$ and sends $t := \nu - r \in \mathbb{F}_p$ to $\mathcal{C}$. $\mathcal{S}$ and $\mathcal{C}$ set $M_\nu := M_r$ and $K_\nu := K_r - \Delta \cdot t \in \mathbb{F}_p$, respectively.
   c) $\mathcal{C}$ sends $\mathsf{AHE.Enc}(\mathsf{pk}, y_{\mathcal{C}})$ to $\mathcal{S}$, who samples $\omega_{\mathcal{S}}, M_{\omega_{\mathcal{S}}} \leftarrow \mathbb{F}_p$, computes $e_1 := \mathsf{AHE.Enc}(\mathsf{pk}, x_{\mathcal{S}} \cdot y_{\mathcal{C}} - \omega_{\mathcal{S}}), e_2 := \mathsf{AHE.Enc}(\mathsf{pk}, M_{\omega_{\mathcal{S}}} - M_{x_{\mathcal{S}}} \cdot y_{\mathcal{C}})$ and sends (e_1, e_2) to $\mathcal{C}$. Then $\mathcal{C}$ sets $\omega_{\mathcal{C}} := \mathsf{AHE.Dec}(\mathsf{sk}, e_1)$ and $K_{\omega_{\mathcal{S}}} := \mathsf{AHE.Dec}(\mathsf{sk}, e_2) + K_{x_{\mathcal{S}}} \cdot y_{\mathcal{C}} + \Delta \cdot \omega_{\mathcal{C}} \in \mathbb{F}_p$. In the honest case, both parties hold $[\omega]$, where $\omega = x_{\mathcal{S}} \cdot y_{\mathcal{C}}$.
   d) $\mathcal{C}$ sends $\mathsf{AHE.Enc}(\mathsf{pk}, x_{\mathcal{C}})$ to $\mathcal{S}$, who samples $\delta_{\mathcal{S}}, M_{\delta_{\mathcal{S}}} \leftarrow \mathbb{F}_p$, computes $e_3 := \mathsf{AHE.Enc}(\mathsf{pk}, x_{\mathcal{C}} \cdot y_{\mathcal{S}} - \delta_{\mathcal{S}}), e_4 := \mathsf{AHE.Enc}(\mathsf{pk}, M_{\delta_{\mathcal{S}}} - M_{y_{\mathcal{S}}} \cdot x_{\mathcal{C}})$ and sends (e_3, e_4) to $\mathcal{C}$. Then $\mathcal{C}$ sets $\delta_{\mathcal{C}} := \mathsf{AHE.Dec}(\mathsf{sk}, e_3)$ and $K_{\delta_{\mathcal{S}}} := \mathsf{AHE.Dec}(\mathsf{sk}, e_4) + K_{y_{\mathcal{S}}} \cdot x_{\mathcal{C}} + \Delta \cdot \delta_{\mathcal{C}} \in \mathbb{F}_p$. In the honest case, both parties hold $[\delta]$, where $\delta = x_{\mathcal{C}} \cdot y_{\mathcal{S}}$.
   e) $\mathcal{C}$ sets $z_{\mathcal{C}} := \mu + \omega_{\mathcal{C}} + \delta_{\mathcal{C}} \in \mathbb{F}_p$ and $K_{z_{\mathcal{S}}} := K_\nu + K_{\omega_{\mathcal{S}}} + K_{\delta_{\mathcal{S}}} \in \mathbb{F}_p$.
   f) $\mathcal{S}$ sets $z_{\mathcal{S}} := \nu + \omega_{\mathcal{S}} + \delta_{\mathcal{S}} \in \mathbb{F}_p$ and $M_{z_{\mathcal{S}}} := M_\nu + M_{\omega_{\mathcal{S}}} + M_{\delta_{\mathcal{S}}} \in \mathbb{F}_p$.
2) Two parties check the correctness of the above protocol as follows[b]:
   a) Following QuickSilver [24], $\mathcal{S}$ computes $M := M_{x_{\mathcal{S}}} \cdot M_{y_{\mathcal{S}}} \in \mathbb{F}_p$ and $d := y_{\mathcal{S}} \cdot M_{x_{\mathcal{S}}} + x_{\mathcal{S}} \cdot M_{y_{\mathcal{S}}} - M_\nu \in \mathbb{F}_p$, and $\mathcal{C}$ computes $K := K_{x_{\mathcal{S}}} \cdot K_{y_{\mathcal{S}}} + \Delta \cdot K_v \in \mathbb{F}_p$.
   b) $\mathcal{C}$ samples $\chi_0, \chi_1 \leftarrow \mathbb{F}_p$ and sends them to $\mathcal{S}$.
   c) $\mathcal{C}$ and $\mathcal{S}$ send (Extend) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $(r, M_r) \in \mathbb{F}_p^2$ to $\mathcal{S}$ and $K_r \in \mathbb{F}_p$ to $\mathcal{C}$. $\mathcal{S}$ computes $M^* := M \cdot \chi_0 + M_{z_{\mathcal{S}}} \cdot \chi_1 + M_r$ and $d^* := d \cdot \chi_0 + z_{\mathcal{S}} \cdot \chi_1 + r$, and send (M^*, d^*) to $\mathcal{C}$.
   d) $\mathcal{C}$ computes $K^* := K \cdot \chi_0 + K_{z_{\mathcal{S}}} \cdot \chi_1 + K_r$ and checks $M^* = K^* + d^* \cdot \Delta$. If the check fails, $\mathcal{C}$ outputs false and aborts.

[a]Note that protocols for generic circuits f_A can be directly achieved by combining the above multiplication protocol and the additive homomorphism of authenticated secret sharing.

[b]This check can be performed only once when verifying multiple multiplication gates.

Fig. 5. Maliciously secure 2PC protocol with fixed corruption in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model.

gates. We show an important idea that it can be evaluated for free in GC without affecting correctness. Specifically, in a finite field $\mathbb{F}_p$, this operation can be formalized as checking $s > \lfloor \frac{p}{2} \rfloor$, where s is a representation of a signed input (refer to Section II for details). This is achieved by replacing $s > \lfloor \frac{p}{2} \rfloor$ with the MSB of s. i.e., the ℓ-th bit of s. Here, $\mathsf{MSB}(s) = 0$ if $s < 2^{\ell-1}$ or 1 otherwise. One may doubt that it is true if s is in a ring like $\mathbb{Z}_{2^\ell}$ [36], [37] but is problematic in $\mathbb{Z}_p$ where p is prime with $|p| = \ell$. Now we give a formal analysis in Lemma 1 to clarify this insight.

Lemma1: Given a finite field $\mathbb{F}_p$ with $\ell := \lceil \log p \rceil$ and the input $s \in [0, 2^{\ell_s}] \cup [p - 1 - 2^{\ell_s}, p)$ with $2^{\ell_s} < \lfloor \frac{p}{2} \rfloor$, it holds that $1\left\{s > \lfloor \frac{p}{2} \rfloor\right\} = \mathsf{MSB}(s)$ when $2^{\ell_s} \le p - 1 - 2^{\lceil \log p \rceil - 1}$.

Proof: Consider the following two cases.

- Case 1: $s \in [0, 2^{\ell_s}]$. It holds $1\left\{s > \lfloor \frac{p}{2} \rfloor\right\} = 0$ due to $s \le 2^{\ell_s} < \lfloor \frac{p}{2} \rfloor$. Moreover, it holds that $\mathsf{MSB}(s) = 0$ due to $s < \lfloor \frac{p}{2} \rfloor < 2^{\ell-1}$. Therefore, if $s \in [0, 2^{\ell_s}]$, then $1\left\{s > \lfloor \frac{p}{2} \rfloor\right\} = \mathsf{MSB}(s)$ unconditionally.
- Case 2: $s \in [p - 1 - 2^{\ell_s}, p)$. It holds $1\left\{s > \lfloor \frac{p}{2} \rfloor\right\} = 1$ due to $s \ge p - 1 - 2^{\ell_s} > p - 1 - \lfloor \frac{p}{2} \rfloor \ge \lfloor \frac{p}{2} \rfloor$. Moreover, it holds that $\mathsf{MSB}(s) = 1$ when $s \ge p - 1 - 2^{\ell_s} \ge 2^{\ell-1}$. Therefore, if $s \in [p - 1 - 2^{\ell_s}, p)$, then $1\left\{s > \lfloor \frac{p}{2} \rfloor\right\} = \mathsf{MSB}(s)$ under the condition of $p - 1 - 2^{\ell_s} \ge 2^{\ell-1}$, i.e., $2^{\ell_s} \le p - 1 - 2^{\lceil \log p \rceil - 1}$. □

We also give a graphical explanation in Figure 8. Without loss of generality, we assume the real input $s \in [0, 2^{\ell_s}] \cup [p - 1 - 2^{\ell_s}, p)$ lies in the blue area according to the

Protocol Π_{2PC} (Part 2)

Boolean Circuit: Without loss of generality, the following protocol assumes that the Boolean circuit f_B consists of only single-input and single-output[a].

1) Two parties hold $[x]$ and evaluate a Boolean circuit f_B to obtain $[y]$ with $y = f_B(x)$:
   a) $\mathcal{C}$ computes $(\mathsf{GC}, \{\mathsf{lab}_{i,j}^{x_\mathcal{C}}, \mathsf{lab}_{i,j}^{x_\mathcal{S}}, \mathsf{lab}_{i,j}^{x}, \mathsf{lab}_{i,j}^{y}\}_{i\in[\ell],j\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\kappa, C)$, where C is a Boolean circuit that takes $x_\mathcal{C}, x_\mathcal{S}$ as input and outputs $x = x_\mathcal{C} + x_\mathcal{S}$ in $\mathbb{F}_p$ and $y = f_B(x)$.
   b) For $i \in [\ell]$, both parties invoke $\mathcal{F}_{\mathsf{OT}}^\kappa$, where $\mathcal{C}$ is the sender with input $(\mathsf{lab}_{i,0}^{x_\mathcal{S}}, \mathsf{lab}_{i,1}^{x_\mathcal{S}})$ and $\mathcal{S}$ is the receiver with input $x_\mathcal{S}[i]$, and $\mathcal{S}$ obtains $\{\hat{\mathsf{lab}}_i^{x_\mathcal{S}}\}_{i\in[\ell]}$, where $\hat{\mathsf{lab}}_i^{x_\mathcal{S}} = \mathsf{lab}_{i,x_\mathcal{S}[i]}^{x_\mathcal{S}}$.
   c) $\mathcal{C}$ sets $\hat{\mathsf{lab}}_i^{x_\mathcal{C}} := \mathsf{lab}_{i,x_\mathcal{C}[i]}^{x_\mathcal{C}}$ for $i \in [\ell]$, and then sends $(\mathsf{GC}, \{\hat{\mathsf{lab}}_i^{x_\mathcal{C}}\}_{i\in[\ell]})$ to $\mathcal{S}$.
   d) $\mathcal{S}$ computes $\{\hat{\mathsf{lab}}_i^{x}, \hat{\mathsf{lab}}_i^{y}\}_{i\in[\ell]} \leftarrow \mathsf{GC.Eval}(\mathsf{GC}, \{\hat{\mathsf{lab}}_i^{x_\mathcal{S}}, \hat{\mathsf{lab}}_i^{x_\mathcal{C}}\}_{i\in[\ell]})$, where $\hat{\mathsf{lab}}_i^{x} = \mathsf{lab}_{i,x[i]}^{x}$ and $\hat{\mathsf{lab}}_i^{y} = \mathsf{lab}_{i,y[i]}^{y}$ for $i \in [\ell]$.
   e) The parties invoke GCMul in Figure 7 with inputs $(\{\mathsf{lab}_{i,j}^{y}\}_{i\in[\ell],j\in\{0,1\}}, 1)$ from $\mathcal{C}$ and $\{\hat{\mathsf{lab}}_i^{y}\}_{i\in[\ell]}$ from $\mathcal{S}$, which returns $y_\mathcal{C}$ to $\mathcal{C}$ and $y_\mathcal{S}$ to $\mathcal{S}$.
   f) The parties invoke GCMul in Figure 7 with inputs $(\{\mathsf{lab}_{i,j}^{y}\}_{i\in[\ell],j\in\{0,1\}}, \Delta)$ from $\mathcal{C}$ and $\{\hat{\mathsf{lab}}_i^{y}\}_{i\in[\ell]}$ from $\mathcal{S}$, which returns $a_\mathcal{C}$ to $\mathcal{C}$ and $a_\mathcal{S}$ to $\mathcal{S}$. Then $\mathcal{C}$ sets $K_{y_\mathcal{S}} := \Delta \cdot y_\mathcal{C} - a_\mathcal{C} \in \mathbb{F}_p$ and $\mathcal{S}$ sets $M_{y_\mathcal{S}} := a_\mathcal{S}$. In the honest case, both parties hold $[y]$.
   g) The parties invoke GCMul in Figure 7 with inputs $(\{\mathsf{lab}_{i,j}^{x}\}_{i\in[\ell],j\in\{0,1\}}, \Delta)$ from $\mathcal{C}$ and $\{\hat{\mathsf{lab}}_i^{x}\}_{i\in[\ell]}$ from $\mathcal{S}$, which returns $b_\mathcal{C}$ to $\mathcal{C}$ and $b_\mathcal{S}$ to $\mathcal{S}$. Then $\mathcal{C}$ sets $K'_{x_\mathcal{S}} := \Delta \cdot x_\mathcal{C} - b_\mathcal{C} \in \mathbb{F}_p$ and $\mathcal{S}$ sets $M'_{x_\mathcal{S}} := b_\mathcal{S}$.
2) Two parties check the correctness of $x_\mathcal{S}$ and $y_\mathcal{S}$ as follows:
   a) $\mathcal{S}$ computes $M := M_{x_\mathcal{S}} - M'_{x_\mathcal{S}} \in \mathbb{F}_p$ and $\mathcal{C}$ computes $K := K_{x_\mathcal{S}} - K'_{x_\mathcal{S}} \in \mathbb{F}_p$.
   b) $\mathcal{C}$ samples $\chi_0, \chi_1 \leftarrow \mathbb{F}_p$ and sends them to $\mathcal{S}$.
   c) $\mathcal{C}$ and $\mathcal{S}$ send (Extend) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $(r, M_r) \in \mathbb{F}_p^2$ to $\mathcal{S}$ and $K_r \in \mathbb{F}_p$ to $\mathcal{C}$. $\mathcal{S}$ computes $M^* := M \cdot \chi_0 + M_{y_\mathcal{S}} \cdot \chi_1 + M_r$ and $d^* := y_\mathcal{S} \cdot \chi_1 + r$, and sends (M^*, d^*) to $\mathcal{C}$.
   d) $\mathcal{C}$ computes $K^* := K \cdot \chi_0 + K_{y_\mathcal{S}} \cdot \chi_1 + K_r$ and checks that $M^* = K^* + d^* \cdot \Delta$. If the check fails, $\mathcal{C}$ outputs false and aborts.

Output:

1) If $\mathcal{C}$ is to learn x, $\mathcal{S}$ sends $(x_\mathcal{S}, M_{x_\mathcal{S}})$ to $\mathcal{C}$, who checks $M_{x_\mathcal{S}} = K_{x_\mathcal{S}} + \Delta \cdot x_\mathcal{S} \in \mathbb{F}_p$. If the check fails, $\mathcal{C}$ outputs false and aborts. Otherwise, $\mathcal{C}$ outputs $x := x_\mathcal{S} + x_\mathcal{C} \in \mathbb{F}_p$.
2) If $\mathcal{S}$ is to learn x, $\mathcal{C}$ sends $x_\mathcal{C}$ to $\mathcal{S}$, who outputs $x := x_\mathcal{S} + x_\mathcal{C} \in \mathbb{F}_p$.

[a]This protocol extends easily for Boolean circuits that take multiple inputs and produce multiple outputs.

Fig. 6. Maliciously secure 2PC protocol with fixed corruption in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model.

encoding of a finite field in Section II. The problematic area, i.e., $1\{s > \lfloor \frac{p}{2} \rfloor\} \neq \mathsf{MSB}(s)$, is indicated in grey. However, these two parts will not overlap with the condition $2^{\ell_s} \leq p - 1 - 2^{\lceil \log p \rceil - 1}$. We emphasize that this condition is always satisfied, especially for maliciously secure machine learning schemes [7], [19], [20], [38]. The reason is that to ensure a negligible failure probability in checking IT-MACs, it requires at least $\lfloor \log p \rfloor \geq \lambda$, where λ is the statistical security parameter. Nevertheless, the bitlength ℓ_s of intermediate values in neural network inference is usually much smaller than λ [4], [5], [18]. Thus, we directly follow general parameter settings for ℓ_s, p without making any adaptive modifications.

Theorem 1: Given a GC scheme $(GC.Garble, GC.Eval)$ and an AHE scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ satisfying the properties defined in Section II, the protocol Π_{2PC} securely realizes the functionality $\mathcal{F}_{\mathsf{2PC}}$ against a malicious adversary corrupting $\mathcal{S}$ or a semi-honest adversary corrupting $\mathcal{C}$ in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model.

Proof: The proof is given in Appendix A. □

C. Scalability of Our Protocols

Our protocols can be deployed in various applications to ensure privacy and integrity, not merely for secure inference in machine learning. In the following, we provide two scenarios to demonstrate this generalizability.

The first application is malicious traffic filtering. This application involves joint malicious traffic filtering between an internet service provider, acting as the semi-honest party, and a security agency, acting as the malicious party. The semi-honest service provider may try to infer the agency's private threat intelligence from intermediate results. Meanwhile, the malicious agency may compromise the filtering integrity and steal the service provider's user logs. Our protocol can be used to deliver strict correctness verification against the agency's malicious behaviors, simultaneously ensuring data confidentiality and filtering integrity.

The second application is secure genomic data analysis for drug discovery. A biotech company, acting as the semi-honest party, collaborates with a pharmaceutical company, acting as

Procedure GCMul

Parameter: A finite field $\mathbb{F}_p$; a random oracle $\mathsf{H} : \{0,1\}^\kappa \to \mathbb{F}_p$; $\ell := \lceil \log p \rceil$.
Input: $\mathcal{C}$ inputs labels $\{\mathsf{lab}^x_{i,j}\}_{i\in[\ell],j\in\{0,1\}} \in \{0,1\}^\kappa$, $\alpha \in \mathbb{F}_p$. $\mathcal{S}$ inputs $\{\hat{\mathsf{lab}}^x_i\}_{i\in[\ell]} \in \{0,1\}^\kappa$ where $\hat{\mathsf{lab}}^x_i = \mathsf{lab}^x_{i,x[i]}$.
Output: $\mathcal{C}$ and $\mathcal{S}$ learns $y_\mathcal{C}$ and $y_\mathcal{S}$, respectively, where $y = \alpha \cdot x$ in $\mathbb{F}_p$.
Procedure:

1) For $i \in [\ell]$ and $j \in \{0,1\}$, $\mathcal{C}$ parses $\mathsf{lab}^x_{i,j}$ as $k^x_{i,j}||p^x_{i,j} \in \{0,1\}^{\kappa-1} \times \{0,1\}$.
2) For $i \in [\ell]$, if $p^x_{i,0} = 0$, $\mathcal{C}$ computes $r_i := \mathsf{H}(\mathsf{lab}^x_{i,0})$, and then sets $u_i := r_i + \alpha + \mathsf{H}(\mathsf{lab}^x_{i,1}) \in \mathbb{F}_p$. If $p^x_{i,1} = 0$, $\mathcal{C}$ sets $r_i := \mathsf{H}(\mathsf{lab}^x_{i,1}) - \alpha \in \mathbb{F}_p$ and $u_i := r_i + \mathsf{H}(\mathsf{lab}^x_{i,0}) \in \mathbb{F}_p$.
3) $\mathcal{C}$ sends $\{u_i\}_{i\in[\ell]}$ to $\mathcal{S}$, and sets $y_\mathcal{C} := -\sum_{i\in[\ell]} r_i \cdot 2^{i-1} \in \mathbb{F}_p$.
4) For $i \in [\ell]$, $\mathcal{S}$ parses $\hat{\mathsf{lab}}^x_i$ as $\hat{k}^x_i||\hat{p}^x_i \in \{0,1\}^{\kappa-1} \times \{0,1\}$. If $\hat{p}^x_i = 0$, $\mathcal{S}$ sets $\hat{r}_i := \mathsf{H}(\hat{\mathsf{lab}}^x_i)$. Otherwise, $\mathcal{S}$ sets $\hat{r}_i := u_i - \mathsf{H}(\hat{\mathsf{lab}}^x_i) \in \mathbb{F}_p$. Then $\mathcal{S}$ learns $y_\mathcal{S} = \sum_{i\in[\ell]} \hat{r}_i \cdot 2^{i-1} \in \mathbb{F}_p$.

Fig. 7. Procedure for label-based multiplication in GCs.

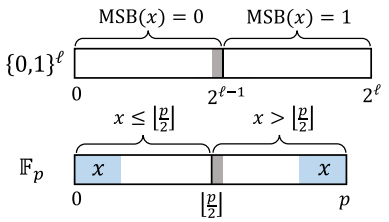


Fig. 8. Relationship between $1\{x > \lfloor \frac{p}{2} \rfloor\}$ and $\mathsf{MSB}(x)$. The shaded region denotes the inconsistency between the MSB and the real value's sign representation, i.e., $1\{x > \lfloor \frac{p}{2} \rfloor\} \neq \mathsf{MSB}(x)$.

the malicious party, for drug target identification. The malicious pharmaceutical company may disrupt the analysis for self-benefit (compromising correctness) and steal the biotech company's sensitive genomic data. Conversely, the semi-honest biotech company may attempt to infer the proprietary drug structures of another company from intermediate results. Our protocol can be used to prevent malicious behaviors of the pharmaceutical company and protect the sensitive information and intellectual property of both companies.

V. The Conan Framework

This section presents Conan, a secure and reliable inference protocol against malicious servers. The full protocol Π_{SecInf} is given in Figure 11, which consists of two phases, i.e., *commit and accuracy verify* and *secure inference on the committed model against malicious servers*. In the following, we provide a technical description for each phase.

A. Commit and Accuracy Verify

In this phase, $\mathcal{S}$ pre-processes the claimed model to make it verifiable in a secure manner, and then $\mathcal{C}$ verifies its accuracy.

Parameters: A set of public test samples $(X, Y) = \{(x_i, y_i)\}_{i\in[n]}$. An accuracy metric Acc and an accuracy threshold ϵ.
Receive $(\mathsf{AccProve}, \{\mathbf{W}_i, [\![\mathbf{W}_i]\!]_0\}_{i\in[t]})$ from the server and $(\mathsf{AccProve}, \{[\![\mathbf{W}_i]\!]_1\}_{i\in[t]}, \Delta)$ from the client. Define $\mathbf{W} := (\mathbf{W}_1, \ldots, \mathbf{W}_t)$. Send true to the client if $\mathsf{Acc}(\mathbf{W}, (X, Y)) \geq \epsilon$ and false otherwise.

Fig. 9. Functionality $\mathcal{F}_{\mathsf{AccProve}}$ for accuracy proof.

Parameters: A neural network model M with t linear layers and $t-1$ non-linear layers.
Receive the model parameter $\{\mathbf{W}_i\}_{i\in[t]}$ from the server and the input $\mathbf{x}$ from the client. Send $\mathsf{M}(\{\mathbf{W}_i\}_{i\in[t]}, \mathbf{x})$ to the receiver.

Fig. 10. Functionality $\mathcal{F}_{\mathsf{SecInf}}$ for secure and reliable inference protocol against malicious servers.

Specifically, two parties first invoke the Input procedure of $\mathcal{F}_{\mathsf{2PC}}$ to convert $\mathcal{S}$'s model parameters $\{\mathbf{W}_i\}_{i\in[t]}$ into an authenticated format, which can be seamlessly integrated into ZK-based accuracy proof [24], [38]. Then $\mathcal{C}$ verifies whether the model meets a predetermined accuracy threshold, based on a labeled testing dataset. We instantiate this step using QuickSilver [24], the state-of-the-art VOLE-based ZK protocol, and adopt conversion protocols between authenticated arithmetic and Boolean values from Mystique [7]. Note that this procedure only needs to be executed once, and therefore, its overhead can be amortized.

For better efficiency, we introduce optimizations tailored to secure inference. Specifically, unlike Mystique, which requires conversion between fixed-point numbers (used in linear layers) and floating-point numbers (used in non-linear layers) within ZK protocols, we consistently use fixed-point evaluation throughout, avoiding this costly conversion. This design follows most existing secure inference protocols [6], [11], [19], [20], [37], which show that fixed-point arithmetic has minimal and somewhat even positive [4], [6] impact on model accuracy. The functionality of this phase is given in Figure 9.

B. Secure Inference on the Committed Model Against Malicious Servers

Once the model accuracy is verified, $\mathcal{C}$ can request inference services on its private inputs $\mathbf{x}$. Such inference services are allowed to be executed multiple times on different private samples. This process is instantiated utilizing our maliciously secure 2PC protocol in Section IV, where alternate linear and non-linear layers are evaluated sequentially based on our arithmetic and Boolean circuit protocols, respectively. Note that the correctness verification step in the Arithmetic procedure of $\mathcal{F}_{\mathsf{2PC}}$ ensures that the model parameters used in each linear layer are consistent with those in the above accuracy verification phase.

Protocol Π_{SecInf}

Parameters: A model M with t linear layers (denoted by $f_{A,i}$ for $i \in [t]$) and $t-1$ non-linear layers (denoted by $f_{B,i}$ for $i \in [t-1]$); $\mathcal{F}_{\mathsf{VOLE\text{-}ZK}}$; $\mathcal{F}_{\mathsf{2PC}}$.
Input: $\mathcal{S}$ holds the model parameter $\{\mathbf{W}_i\}_{i\in[t]}$. $\mathcal{C}$ holds the input $\mathbf{x}$.
Output: $\mathcal{C}$ learns the inference result $\mathbf{y}$, where $\mathbf{y} := \mathsf{M}(\{\mathbf{W}_i\}_{i\in[t]}, \mathbf{x})$.
Protocol:

1) **Commit and Accuracy-Prove:**
   a) For $i \in [t]$, $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Input}, \mathcal{S})$ and $(\mathsf{Input}, \mathcal{S}, \mathbf{W}_i)$, respectively, to $\mathcal{F}_{\mathsf{2PC}}$, which returns $[\mathbf{W}_i]$ to both parties.
   b) $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{AccuacyProve}, \{[\mathbf{W}_i]\}_{i\in[t]})$ to $\mathcal{F}_{\mathsf{VOLE-ZK}}$. $\mathcal{C}$ aborts if $\mathcal{F}_{\mathsf{VOLE-ZK}}$ returns false, and continues otherwise.
2) **Secure Inference:**
   a) **First Linear Layer:**
      - $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Input}, \mathcal{C}, \mathbf{x})$ and $(\mathsf{Input}, \mathcal{C})$, respectively, to $\mathcal{F}_{\mathsf{2PC}}$, which returns $[\mathbf{x}]$ to both parties.
      - $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Arithmetic}, f_{A,0}, [\mathbf{W}_0], [\mathbf{x}])$ to $\mathcal{F}_{\mathsf{2PC}}$, which return $[\mathbf{v}_0]$ to both parties.
   b) For each $i \in [t-1]$, two parties alternately execute linear and non-linear layers:
      - **Non-linear Layer**: $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Boolean}, f_{B,i}, [\mathbf{v}_i])$ to $\mathcal{F}_{\mathsf{2PC}}$, which returns $[\mathbf{x}_{i+1}]$ to both parties.
      - **Linear Layer**: $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Arithmetic}, f_{A,i+1}, [\mathbf{W}_{i+1}], [\mathbf{x}_{i+1}])$ to $\mathcal{F}_{\mathsf{2PC}}$, which returns $[\mathbf{v}_{i+1}]$ to both parties.
3) **Output:** $\mathcal{C}$ and $\mathcal{S}$ send $(\mathsf{Output}, \mathcal{C}, [\mathbf{v}_t])$ to $\mathcal{F}_{\mathsf{2PC}}$, which returns $\mathbf{v}_t$ to $\mathcal{C}$. $\mathcal{C}$ outputs $\mathbf{y} := \mathbf{v}_t$.

Fig. 11. Secure and reliable inference protocol against malicious servers.

Figure 11 details our secure inference protocol Π_{SecInf} on the committed model against malicious servers, and the corresponding functionality is given in Figure 10. Compared to the general protocol Π_{2PC}, we make the following optimizations customized for the model inference setting. Without loss of generality, we assume the dimension of a model parameter matrix $\mathbf{W}_i$ for $i \in [t]$ is $m \times n$ and the dimension of each linear layer's input $\mathbf{x}_i$ is n.

- In linear layers, the model parameter $\{\mathbf{W}_i\}_{i\in[t]}$ is owned by $\mathcal{S}$, which can not be secret-shared between the two parties. Specifically, for each $i \in [t]$, to authenticate $\mathbf{W}_i$ in the Input procedure, $\mathcal{S}$ sends $\mathbf{W}_i - \mathbf{R}_i \in \mathbb{F}_p^{m\times n}$ to $\mathcal{C}$, and then both parties compute the authentication $\langle \mathbf{W}_i \rangle$ as $\langle \mathbf{R}_i \rangle + (\mathbf{W}_i - \mathbf{R}_i) \in \mathbb{F}_p^{m\times n}$, where $\langle \mathbf{R}_i \rangle$ is a random VOLE matrix. Then given $\mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{W}_i \cdot (\mathbf{x}_{i\mathcal{S}} + \mathbf{x}_{i\mathcal{C}}) = \mathbf{W}_i \cdot \mathbf{x}_{i\mathcal{S}} + \mathbf{W}_i \cdot \mathbf{x}_{i\mathcal{C}}$, the multiplication evaluation in the Arithmetic procedure of protocol Π_{2PC} can also be simplified to only computing the authenticated value of $\mathbf{W}_i \cdot \mathbf{x}_{i\mathcal{S}}$ using the method in step 1b and the authenticated secret share of $\mathbf{W}_i \cdot \mathbf{x}_{i\mathcal{C}}$ using the method in step 1c.
- In linear layers, as described above, both parties evaluate $\boldsymbol{\nu}_i := \mathbf{W}_i \cdot \mathbf{x}_{i\mathcal{S}}$ for each $i \in [t]$ that contains $m \cdot n$ element-wise multiplications. Obviously, $m \cdot n$ correctness verifications are required, which are expensive since the parameter dimension m is usually large. We can transform checking the above $m \cdot n$ multiplications into checking only n multiplications without sacrificing soundness [7], by proving $\mathbf{a}^\mathsf{T} \cdot \mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{a}^\mathsf{T} \cdot \boldsymbol{\nu}_i$, where $\mathbf{a} \in \mathbb{F}_p^m$ is uniformly sampled by $\mathcal{C}$ and sent to $\mathcal{S}$. Based on the additive homomorphism, both parties can easily compute the authentication of $\mathbf{b} := \mathbf{a}^\mathsf{T} \cdot \mathbf{W}_i$ and $\mathbf{c} := \mathbf{a}^\mathsf{T} \cdot \mathbf{v}_i$. Thus, only $\mathbf{b} \cdot \mathbf{x}_i = \mathbf{c}$ with the dimension n needs to be proved.
- In non-linear layers, we can optimize the communication cost when truncation is required (refer to Section II for details). Assuming the truncated bitlength is s, both parties only need to communicate the last $\ell - s$ items in each instance of GCMul in the Boolean procedure, and both parties set $\alpha \cdot x = \sum_{i\in[st+1,\ell]} \alpha \cdot x[i] \cdot 2^{i-st-1}$ in $\mathbb{F}_p$. With this optimization, a GCMul instance costs $\ell(\ell - st)$ bits of communication, rather than ℓ^2 bits in the general solution.
- We can check the consistency and correctness of all layers in a batch using a random linear combination for better communication performance, instead of checking layer by layer. Note that all our checks in Π_{SecInf} can be generalized to checking that $M_x = K_x + \Delta \cdot x$ holds where x is owned by $\mathcal{S}$ or x should be 0. To check k such relations, where $\mathcal{S}$ and $\mathcal{C}$ own $\{x_i, M_{x_i}\}_{i\in[k]}$ and $\{\Delta, K_{x_i}\}_{i\in[k]}$ respectively, $\mathcal{C}$ first samples uniform elements $\{\chi_i\}_{i\in[k]}$ and sends them to $\mathcal{S}$, and then checks that $\sum_{i\in[k]} M_{x_i} \cdot \chi_i = \sum_{i\in[k]} K_{x_i} \cdot \chi_i + \Delta \cdot \left(\sum_{i\in[k]} x_i \cdot \chi_i\right)$ holds.

1) Analysis on Scalability for Large-Scale Models: Below, we clarify the computational and communication overhead these client-side operations introduce as the model scales. Our protocol overhead on the client-side is linearly related to the model depth and the size of each layer, which inherently allows for easy scalability to large-scale models. Specifically, the model can be divided into linear and non-linear layers. The linear layers are evaluated using our arithmetic circuit protocol. For each multiplication gate, the client requires 6 local multiplications and 6 homomorphic encryption/decryption operations, with a communication overhead of 1 field element and 2 ciphertexts. When evaluating multiple multiplication gates, the number of local multiplications and communicated field elements increases linearly. However, the homomorphic operations can be efficiently batched. If the batch size is k, the total number of encryption/decryption operations required for n multiplication gates is $\lceil (n/k) \rceil \times 6$, and the communication overhead is $\lceil (n/k) \rceil \times 2$ ciphertexts. The non-linear layers are evaluated using our Boolean circuit protocol. The cost associated with this evaluation is perfectly linear with the number of non-linear operations, as each operation necessitates a complete execution through the Boolean circuit.

Theorem 2: The protocol Π_{SecInf} securely realizes the functionality $\mathcal{F}_{\mathsf{SecInf}}$ against a malicious adversary $\mathcal{A}$ corrupting the server or a semi-honest adversary $\mathcal{A}$ corrupting the client in the $(\mathcal{F}_{VOLE\text{-}ZK}, \mathcal{F}_{\mathsf{2PC}})$-hybrid model.

Proof: The security of the corrupted client in the semi-honest adversary setting is straightforward. We therefore focus on the case of a malicious adversary $\mathcal{A}$ corrupting the server. We construct a simulator Sim that has access to the ideal functionalities $\mathcal{F}_{VOLE\text{-}ZK}$, $\mathcal{F}_{\mathsf{2PC}}$ and interacts with the environment $\mathcal{Z}$. Note that $\mathcal{Z}$ chooses the inputs of both parties, and at the end of the execution also learns the outputs of both parties. We prove that the simulated interaction by Sim is indistinguishable from the real interaction. The simulator Sim works as follows.

Simulating the commit and accuracy-prove phase: Sim emulates the input command of the functionality $\mathcal{F}_{\mathsf{2PC}}$, hence Sim holds the global key Δ and can compute all correct MACs of $\mathcal{A}$. When receiving $\mathbf{W}_i$ from $\mathcal{A}$, Sim samples $[\mathbf{W}_i]_{\mathcal{S}} \leftarrow \mathbb{F}_p^2$ and sends them to $\mathcal{A}$. Sim emulates the accuracy-prove command of the functionality $\mathcal{F}_{VOLE\text{-}ZK}$ and receives $[\mathbf{W}_i]^*_{\mathcal{S}}$ from $\mathcal{A}$. If $\mathcal{F}_{VOLE\text{-}ZK}$ returns false, Sim sends $\perp$ to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$ and aborts.

Simulating the secure inference phase: Sim emulates the input command of the functionality $\mathcal{F}_{\mathsf{2PC}}$. Sim samples $[\mathbf{x}]_{\mathcal{S}} \leftarrow \mathbb{F}_p^2$ and sends them to $\mathcal{A}$. Sim emulates the arithmetic command of the functionality $\mathcal{F}_{\mathsf{2PC}}$ and receives $[\mathbf{W}_0]^*_{\mathcal{S}}$, $[\mathbf{x}]^*_{\mathcal{S}}$ from $\mathcal{A}$. If $\mathcal{F}_{\mathsf{2PC}}$ aborts, Sim sends $\perp$ to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$ and aborts. Sim samples $[\mathbf{v}_0]_{\mathcal{S}} \leftarrow \mathbb{F}_p^2$ and sends them to $\mathcal{A}$. For $i \in [t-1]$, Sim emulates the Boolean circuit command of the functionality $\mathcal{F}_{\mathsf{2PC}}$ and receives $[\mathbf{v}_i]^*_{\mathcal{S}}$ from $\mathcal{A}$. If $\mathcal{F}_{\mathsf{2PC}}$ aborts, Sim sends $\perp$ to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$ and aborts. Otherwise, Sim samples $[\mathbf{x}_{i+1}]_{\mathcal{S}} \leftarrow \mathbb{F}_p^2$ and sends them to $\mathcal{A}$. Sim emulates the arithmetic circuit command of the functionality $\mathcal{F}_{\mathsf{2PC}}$ and receives $[\mathbf{W}_{i+1}]^*_{\mathcal{S}}$, $[\mathbf{x}_{i+1}]^*_{\mathcal{S}}$ from $\mathcal{A}$. If $\mathcal{F}_{\mathsf{2PC}}$ aborts, Sim sends $\perp$ to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$ and aborts. Otherwise, Sim samples $[\mathbf{v}_{i+1}]_{\mathcal{S}} \leftarrow \mathbb{F}_p^2$ and sends them to $\mathcal{A}$.

Simulating the output phase: Sim emulates the output command of the functionality $\mathcal{F}_{\mathsf{2PC}}$ and receives $[\mathbf{v}_t]^*_{\mathcal{S}}$ from $\mathcal{A}$. If $\mathcal{F}_{\mathsf{2PC}}$ aborts, Sim sends $\perp$ to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$ and aborts. Otherwise, Sim extracts the input $\mathbf{W}_i$ of $\mathcal{A}$ and sends them to the functionality of $\mathcal{F}_{\mathsf{SecInf}}$.

Now we argue that $\mathcal{Z}$ cannot distinguish between the real and ideal executions. Clearly, the view of adversary $\mathcal{A}$ simulated by Sim has the identical distribution as its view in the real-world execution. The reason is that the received messages of $\mathcal{A}$ are $[\mathbf{W}_i]_{\mathcal{S}}$, $[\mathbf{x}]_{\mathcal{S}}$, $[\mathbf{v}_i]_{\mathcal{S}}$, $[\mathbf{x}_i]_{\mathcal{S}}$ and they are uniformly random in both real and simulated executions. Moreover, whenever the execution in the real-world execution aborts, the execution in the ideal-world execution aborts as well. This completes the proof. □

VI. Experimental Evaluation

A. Experiment Setup

1) Implementation: We implemented Conan in C++. Conan is built on top of the EMP toolkit [39] and Seal homomorphic encryption library [40]. The EMP toolkit provides ZK protocols for accuracy verification and GC protocols for non-linear layers in secure inference. The Seal library implements the AHE scheme for linear layers in secure inference. In line with existing secure inference works [6], [20], we simulate both LAN and WAN settings. Under LAN, the bandwidth is 584MBps and the echo latency is 21ms. Under WAN, the bandwidth is 44MBps and the echo delay is 40ms. All our experiments are performed on AWS c5.9xlarge instances with Intel Xeon 8000 series CPUs at 3.6GHz, and we run all of Conan's components using a single thread. Note that our protocols can directly benefit from parallelization through multi-threading. Conan is implemented over a 44-bit prime field and provides 128 bits of computational security and 40 bits of statistical security.

TABLE I
End-to-End Communication (MB) and Runtime (Seconds) Overhead of Our Conan Framework

	MNIST			CIFAR10		
Overhead	Comm.	LAN	WAN	Comm.	LAN	WAN
phase (1)	101.79	58.54	61.98	128.92	70.59	74.01
phase (2)	122.90	7.51	11.44	1510.10	43.54	75.71

2) Datasets and Models: Consistent with existing maliciously secure protocols [19], [20], we evaluate Conan on two datasets and model architectures: (1) A 2-layer convolutional neural network (CNN) for MNIST, based on MiniONN [41], with Average Pooling in place of Max Pooling. (2) A 7-layer CNN for CIFAR10, also from MiniONN [41].

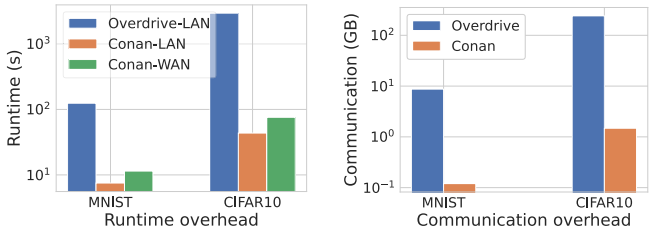
3) Baselines: Since there are no provably secure protocols for server-malicious secure inference, we follow prior works [10], [19] to use a general-purpose maliciously secure framework, Overdrive [14], as our baseline. We report the experimental results of Overdrive from Figure 9 in [19], which illustrates its performance with 8 threads under LAN.

B. End-to-End Evaluation

As detailed in Section V, Conan contains two phases: (1) commit and accuracy verify, and (2) secure inference on the committed model against malicious servers. We evaluate both under LAN and WAN settings, with results shown in Table I. To highlight efficiency, we report the overhead of accuracy verification and secure inference on a single input. Note that the overhead of phase (1) could always be amortized over multiple secure inferences since it runs only once. Results show that Conan is efficient in different datasets and network settings. For example, phase (1) completes in less than 75 seconds on both datasets over LAN and WAN. Phase (2) only takes 11.44 seconds for MNIST and 75.71 seconds for CIFAR10 over WAN.

C. Performance of Secure Inference Against Malicious Servers

We now focus on the performance of phase (2) in Conan, i.e., secure inference on the committed model against malicious servers, which is the main bottleneck as identified in prior work [10], [14]. We first give a comparison with the state-of-the-art maliciously secure framework, Overdrive [14], followed by showing the microbenchmarks of our secure inference protocol. We further report the performance of our protocol when extending to the offline-online setting.



(a) Runtime (b) Communication

Fig. 12. Runtime (seconds) and communication (GB) comparison with Overdrive [14].

TABLE II
COMMUNICATION (MB) AND RUNTIME (SECONDS) OVERHEAD BREAKDOWN OF OUR SECURE INFERENCE

	MNIST			CIFAR10		
Overhead	Comm.	LAN	WAN	Comm.	LAN	WAN
Linear layers	38.96	6.44	8.35	107.83	30.57	35.44
Non-linear layers	83.94	1.07	3.09	1402.27	12.97	40.27
Total	122.90	7.51	11.44	1510.10	43.54	75.71

TABLE III
COMMUNICATION (MB) AND RUNTIME (SECONDS) COST OF OUR SECURE INFERENCE IN THE OFFLINE-ONLINE SETTING

	MNIST			CIFAR10		
Overhead	Comm.	LAN	WAN	Comm.	LAN	WAN
Offline phase	190.85	6.40	8.90	2790.03	46.04	90.47
Online phase	10.24	0.80	3.45	235.52	7.86	25.21
Total	201.09	7.20	12.35	3025.55	53.90	115.68

1) Comparison With Overdrive [14]: Figure 12 compares the runtime and communication performance of Conan and Overdrive. We utilize the same network setting as Overdrive, but with a single thread rather than 8 threads in Overdrive. Conan shows significant improvement. It is 16.54× and 67.36× faster, and reduces communication by 72.48× and 163.73× on MNIST and CIFAR10, respectively. Notably, Conan under WAN still outperforms Overdrive under LAN, highlighting the efficiency of our secure inference protocol against malicious servers.

2) Performance Breakdown: Table II gives the communication and runtime performance breakdown of our protocol. We observe that non-linear layers dominate the communication overhead, e.g., up to 68% and 92%, respectively. The main reason is that our linear layer protocol mainly uses communication-efficient additively homomorphic encryption and significantly reduces communication overhead. Moreover, a large number of non-linear functions like ReLU are evaluated in these two models, e.g., 173K ReLUs in CIFAR10. A similar phenomenon is also observed in prior secure inference works [19], [20]. For runtime, the non-linear layers often require less runtime than the linear layers, mainly due to our customized circuit designs and evaluation protocols. For example, under CIFAR10, the non-linear layers take 12.97 seconds while the linear layers take 30.57 seconds over LAN.

3) Performance in the Offline-Online Setting: Similar to recent secure inference works [18], [19], [20], [41], our protocols can be extended into an offline-online setting. In the offline phase when client queries are unknown, the parties precompute all query-independent operations and generate correlated randomness, shifting most cryptographic cost offline. We briefly introduce how to extend our protocols in this setting. (1) For linear layers, $\mathbf{W} \cdot \mathbf{r}$ (and hence all AHE-related operations) is moved offline, leaving only lightweight local computation and minimal communication online. (2) For non-linear layers, GC and labels are prepared offline, with the server performing only lightweight circuit evaluation online (refer to Section 4.4 in [20]). As reported in Table III, the offline phase dominates the overall overhead. In particular, 94% and 92% of the communication overhead come from the offline phase for MNIST and CIFAR10, respectively.

VII. DISCUSSION

A. Possible Optimizations for Better Efficiency

Several strategies might be adopted to enhance the practicality of Conan for MLaaS deployment. We discuss these mitigation strategies in the following.

1) Offline-Online Paradigm: Following established secure inference practices [18], [33], heavy cryptographic operations such as randomness and multiplication triplet generation can be offloaded to an input-independent offline phase. This ensures the resulting online phase involves only lightweight operations like reconstruction and local computation, thereby significantly reducing the latency perceived by the client. Our experimental results in Table III in Section VI have demonstrated its effectiveness.

2) Quantization: A dominant factor affecting runtime in secure inference is the bitlength of inputs and intermediate values. To mitigate this, a possible solution is to integrate quantization techniques [8], [42], which allow inference to be performed on lightweight and low-bitlength models while preserving comparable accuracy. Integrating these techniques into our framework offers a promising direction for substantial performance gains.

3) Parallelization and Hardware Acceleration: Further engineering optimizations can leverage parallelization and specialized hardware acceleration such as GPUs. In neural network inference, a vast number of operations are independent (i.e., matrix multiplication) and can be executed concurrently, allowing for significant reductions in total runtime.

B. Generalization on Transformer-Based Models

In Section VI, Conan is experimentally applied to traditional machine learning models to evaluate its usability. Traditional models remain fundamentally important for real-world applications. Moreover, the fundamental components of conventional models (e.g., matrix multiplication and comparisons) are general-purpose and can be readily extended to generative models. Furthermore, Conan is a general secure inference framework. This generic nature ensures its direct applicability to guarantee security and privacy for a wide range of model architectures, including generative models.

In the following, we discuss the importance, generalization, and barriers to its generalization.

1) Importance of Secure Inference on Traditional Models: While the growing popularity of generative models, traditional machine learning like classification tasks remain fundamentally important for numerous real-world applications. Critical sectors such as healthcare [43], [44] and finance [34] heavily rely on these classification architectures for tasks such as diagnosis and fraud detection. Furthermore, achieving practical efficiency while maintaining strict security remains a substantial challenge, even for these established classification networks [32]. Conan can ensure a secure, efficient, and reliable inference service for this fundamental problem, to achieve accuracy verification, inference integrity, and privacy simultaneously.

2) Application of Conan to Transformer-Based Models: As detailed in Section V, Conan consists of two phases, i.e., commit and accuracy verify, and secure inference. In the first phase, both parties first authenticate model parameters using the QuickSilver mechanism [24], and then verify model accuracy using the Mystique technique [7]. The former operates on single domain elements, while the latter can transform any function into a specific Boolean circuit for evaluation. Crucially, they are independent of the model architecture and thus directly applicable to Transformer-based models. In the second phase, secure inference is achieved utilizing our 2PC protocols, which can evaluate both arithmetic and Boolean circuits. In transformer-based models, linear operations can be evaluated using our protocols for arithmetic circuits. Non-linear operations like GELU and Sigmoid can be transformed into general Boolean circuits for evaluation, as arbitrary functions can be represented using Boolean circuits. In conclusion, Conan can be applied to transformer-based models to provide secure inference services.

3) The Barriers to Its Generalization: In the transformer-based model architecture, the primary obstacle lies in its complex non-linear components. These functions are difficult to implement efficiently using generic Boolean circuits because their representation requires a significant number of AND gates, introducing considerable overhead. For example, complex primitives like exponentiation, division, and reciprocal square root necessitate $3 \sim 11K$ multiplication gates in Boolean circuit representations [7], [35]. Existing secure inference works for transformers [32], [45] typically employ spline approximations and the lookup table technique to handle costly non-linear functions. Nevertheless, balancing computation, communication, and accuracy remains a critical challenge in secure inference for transformers [33].

VIII. Related Works

A. Review of Related Works

A number of recent works have attempted to design specialized protocols for performing secure inference. These protocols generally fall into the following categories according to the type of threat model: semi-honest security and malicious security. We discuss each of these categories as follows.

1) Semi-Honest Security: Most of existing secure inference protocols [4], [5], [18], [37], [41], [46], [47], [48], [49], [50] focus on semi-honest security. In this setting, the client and the server follow protocol specification honestly but attempt to infer more knowledge about each other from the received messages. These protocols provide privacy guarantees by utilizing various advanced secure computation techniques, such as secret sharing [36], oblivious transfer [51], garbled circuits [27], and homomorphic encryption [52]. They achieve high efficiency either by designing customized cryptographic protocols based on these techniques [4], [6], [41], [46] or by modifying the model architecture such as approximating ReLU activations with low-degree polynomials [18], [37], [48], [53] or quantizing network weights [47], [54], [55], [56].

2) Malicious Security: Considering a more realistic inference service scenario, malicious parties may arbitrarily deviate from the pre-established protocols. Several solutions have been proposed in this setting. Specifically, the general-purpose 2PC protocols for computing arithmetic circuits [11], [12], [13], [14], binary circuits [57], and mix circuits [15], [58] provide privacy and correctness guarantees in an environment where both parties are malicious. However, as mentioned in Section I, these protocols incur intolerable overhead and cannot provide model accuracy and integrity guarantees.

Recently, MUSE [19] was proposed, which works in the setting of malicious clients and semi-honest servers, and later SIMC [20] further improved its efficiency. They consider that clients may maliciously modify the intermediate results during the execution of inference protocols, to steal the server's model parameters. Thus, consistency verification strategies were proposed to ensure whether the input of each layer has been tampered with. Such protocols cannot be employed in Conan due to different adversarial settings.

Besides, ZK proofs have been explored for verifiable ML inference [7], [8], [59], [60], [61], [62]. These approaches allow the server to provide the client with proof validating that inferences are indeed performed in the claimed accurate model (i.e., accuracy and integrity goals). Nevertheless, they can not meet the client's privacy requirement due to the need for the client's queries to be uploaded in plaintext.

B. Discussion on the Security of Applying Existing Semi-Honest Protocols Into Our Setting

It is worth noting that existing semi-honest inference protocols, such as ABY [36], Delphi [18], CrypTFlow2 [4], and Cheetah [6], cannot be adapted to our setting, because they fail to protect privacy and integrity. As introduced in Section I, the server operating under these protocols can deviate from the protocols, aiming to steal the client's private inputs. Even worse, this deviation can be conducted in an undetectable manner, namely, the server steals the client's inputs while delivering seemingly correct inference services. Moreover, some solutions [10] attempt to combine an accuracy check of inference results with these semi-honest protocols to resist malicious servers. However, the adversary can always bypass the checks and return correct results, while stealing the client's private inputs.

The main reason is that the adopted semi-honest inference protocols are incapable of defending malicious adversaries, such as the garbler in GC [63] and the receiver in OT protocols [29]. We note that GC and OT are indispensable techniques for evaluating non-linear layers in secure inference. For the non-linear layer evaluation protocols used in the advanced works, ABY/Delphi employs GC, while CrypTFlow2/Cheetah utilizes OT. Below, we show two counter-examples from GC and OT techniques such that the server can stealthily steal the client's inputs or intermediate values, and meanwhile deliver the correct inference to the client.

1) Undetectable Privacy Leakages From GC: The evaluation of a non-linear function f takes as input $\langle x\rangle_0$ from the server and $(\langle x\rangle_1, r)$ from the client, where r is uniformly sampled, and outputs $\langle y\rangle_0 := f(x) - r$ to the server and $\langle y\rangle_1 := r$ to the client. Before discussing privacy leakages from malicious deviations, we first introduce benign GC-based non-linear protocols in ABY and Delphi, where the server acts as the garbler, and the client acts as the evaluator.

1) The server constructs a garbled circuit $\tilde{C}$ for computing $f(x) - r$. It sends $\tilde{C}$ and the labels for $\langle x\rangle_0$ to the client and simultaneously, the server and the client exchange labels for $\langle x\rangle_1$ and r via OT.
2) The client evaluates $\tilde{C}$ using the received labels and learns the labels of $f(x) - r$. It then sends this output to the server.
3) The server decrypts these labels using its decryption table to learn $f(x) - r$.

There is a malicious strategy for the server to learn $f(x)$ in plaintext. In particular, the server constructs $\tilde{C}$ for computing $f(x)$ rather than $f(x) - r$. This can be achieved by the server setting both two labels of r as the label of 0 in Step 1), regardless of its true value. The server then will obtain $f(x)$ in Step 3). It's worth emphasizing that using the intermediate value $f(x)$, the server can compute the correct output z of the last linear layer in plaintext. In the corresponding evaluation of the last linear layer, the server modifies the parameter W to 0, resulting in the secret shares $\langle z'\rangle_0$, $\langle z'\rangle_1$ of the output to satisfy $\langle z'\rangle_0 + \langle z'\rangle_1 = 0$. Finally, the server sets $\langle z'\rangle_0 := \langle z'\rangle_0 + z$. Therefore, the final inference result is $\langle z'\rangle_0 + \langle z'\rangle_1 = z$, which is still correct.

2) Undetectable Privacy Leakages From OT: Now we analyze the security of OT used in the semi-honest protocols of these solutions. As shown in prior works [9], widely used semi-honest OTs such as the protocols [51], [63] are insecure against the malicious receiver. We take the semi-honest OT protocol [51] as an example. Assuming the sender's message is (x_0, x_1) and the receiver's choice bit is b, the protocol [51] performs as follows:

1) Given a group $(\mathbb{G}, q, g)$ for which the DDH is hard, the receiver samples a random $\alpha \in \mathbb{Z}_q$ and a random $h \in \mathbb{G}$ and then sets $(h_0, h_1) := (g^\alpha, h)$ if $b = 0$ and (h, g^α) otherwise. The receiver sends (h_0, h_1) to the sender.
2) The sender samples a random $r \in \mathbb{Z}_q$, sets $u := g^r$ and computes (v_0, v_1) such that $v_0 := x_0 \oplus \mathsf{KDF}(h_0^r)$ and $v_1 := x_1 \oplus \mathsf{KDF}(h_1^r)$, where KDF is a key derivation function. The sender sends (u, v_0, v_1) to the receiver.
3) The receiver computes and outputs $x_b := v_b \oplus \mathsf{KDF}(u^\alpha)$.

The malicious receiver can learn both the sender's messages without affecting the protocol's correctness. In particular, the receiver sets h as g^β in Step 1) and hence knows its exponent, rather than sampling it uniformly random. After that, in Step 3), the receiver can utilize α, β to learn both x_0, x_1. Once the malicious server acts as the receiver, it can learn the OT protocol's inputs in secure inference. For example, the MUX protocol has been used in both CrypTFlow2 and Cheetah, in which the server will serve as the receiver and learn the protocol inputs of the MUX protocol. Moreover, this deviation does not affect the correctness of inference results.

IX. Conclusion

This work proposes Conan, a novel framework for secure and reliable inference in MLaaS. Conan ensures correct inference on the claimed accurate model while preserving the privacy of both client and server. At the core of Conan are new secure inference protocols tailored to the malicious-server threat model. We evaluate Conan on representative benchmarks, demonstrating its efficiency. In the future, we aim to support more complex models and inference tasks such as transformer-based large language models (LLMs). Unfortunately, even under semi-honest adversary settings, secure inference on LLMs still remains a major performance bottleneck. For example, the state-of-the-art solution [32] requires 13.87 minutes and 5.64 GB of communication to generate a single token on LLaMA-7B with 8 input tokens.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[2] A. Esteva et al., "A guide to deep learning in healthcare," *Nature Med.*, vol. 25, no. 1, pp. 24–29, 2018.
[3] M. Popel et al., "Transforming machine translation: A deep learning system reaches news translation quality comparable to human professionals," *Nature Commun.*, vol. 11, no. 1, p. 4381, Sep. 2020.
[4] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.
[5] D. Rathee et al., "SiRnn: A math library for secure RNN inference," in *Proc. IEEE SP*, Apr. 2021, pp. 1003–1020.
[6] Z. Huang, W.-J. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Secur.*, 2022, pp. 809–826.
[7] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *Proc. USENIX Secur.*, May 2021, pp. 730–518.
[8] T. Liu, X. Xie, and Y. Zhang, "ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 2968–2985.
[9] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer extensions with security for malicious adversaries," in *Proc. EUROCRYPT*, 2015, pp. 673–701.
[10] C. Dong et al., "Fusion: Efficient and secure inference resilient to malicious servers," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023.
[11] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. CRYPTO*, 2012, pp. 643–662.
[12] 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 *Proc. ESORICS*, 2013, pp. 1–18.
[13] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 830–842.
[14] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *Proc. EUROCRYPT*, 2018, pp. 158–189.

[15] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Schöll, "Improved primitives for MPC over mixed arithmetic-binary circuits," in *Proc. CRYPTO*, 2020, pp. 823–852.
[16] J. H. Cheon, D. Kim, and K. Lee, "MHz2k: MPC from he over $\mathbb{Z}_{2^\kappa}$ with new packing, simpler reshare, and better ZKP," in *Proc. CRYPTO*, 2021, pp. 426–456.
[17] M. Rivinius, P. Reisert, S. Hasler, and R. Küsters, "Convolutions in overdrive: Maliciously secure convolutions for MPC," in *Proc. PETs*, 2023, pp. 321–353.
[18] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proc. USENIX Secur.*, Nov. 2020, pp. 27–30.
[19] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *Proc. USENIX Secur.*, 2021, pp. 2201–2218.
[20] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "SIMC: ML inference secure against malicious clients at semi-honest cost," in *Proc. USENIX Secur.*, 2022, pp. 1361–1378.
[21] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[22] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Proc. EUROCRYPT*, 2011, pp. 169–188.
[23] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proc. CRYPTO*, 2012, pp. 681–700.
[24] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 2986–3001.
[25] Y. Ishai and A. Paskin, "Evaluating branching programs on encrypted data," in *Proc. TCC*, 2007, pp. 575–594.
[26] F. Bourse, R. D. Pino, M. Minelli, and H. Wee, "FHE circuit privacy almost for free," in *Proc. CRYPTO*, 2016, pp. 62–89.
[27] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. FOCS*, 1986, pp. 162–167.
[28] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proc. 22nd Annu. ACM Symp. Theory Comput.*, 1990, pp. 503–513.
[29] M. Keller, E. Orsini, and P. Schöll, "Actively secure OT extension with optimal overhead," in *Proc. CRYPTO*, 2015, pp. 724–741.
[30] E. Boyle et al., "Efficient two-round OT extension and silent non-interactive secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 291–308.
[31] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1074–1091.
[32] W.-J. Lu et al., "BumbleBee: Secure two-party inference framework for large transformers," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2025, pp. 1–18.
[33] K. Gupta et al., "SIGMA: Secure GPT inference with function secret sharing," *Proc. Privacy Enhancing Technol.*, vol. 2024, no. 4, pp. 61–79, Oct. 2024.
[34] Z. Xiang, T. Wang, and D. Wang, "Preserving node-level privacy in graph neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 4714–4732.
[35] M. Hao et al., "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *Proc. USENIX Secur.*, 2024, pp. 3819–3836.
[36] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015.
[37] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.
[38] C. Weng, K. Yang, Z. Yang, X. Xie, and X. Wang, "AntMan: Interactive zero-knowledge proofs with sublinear communication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 2901–2914.
[39] *EMP Toolkit*. Accessed: Dec. 24, 2025. [Online]. Available: https://github.com/emp-toolkit
[40] *Seal Homomorphic Encryption Library*. Accessed: Dec. 24, 2025. [Online]. Available: https://github.com/Microsoft/SEAL
[41] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 619–631.
[42] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[43] J. Ran and D. Li, "A faster privacy-preserving medical image diagnosis scheme with machine learning," *J. Imag. Informat. Med.*, vol. 38, no. 5, pp. 2716–2728, Jan. 2025.
[44] F. Wang et al., "TrustMIS: Trust-enhanced inference framework for medical image segmentation," in *Proc. ECAI*, 2024, pp. 105–112.
[45] A. Y. L. Kei and S. S. M. Chow, "SHAFT: Secure, handy, accurate and fast transformer inference," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2025.
[46] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. USENIX Secur.*, 2018, pp. 1651–1669.
[47] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. USENIX Secur.*, 2019, pp. 1501–1518.
[48] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "NGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. Comput. Frontiers*, 2019, pp. 3–13.
[49] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, "SecFloat: Accurate floating-point meets secure 2-party computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 576–595.
[50] N. Jovanovic, M. Fischer, S. Steffen, and M. Vechev, "Private and reliable neural network inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 1663–1677.
[51] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM CCS*, 2013, pp. 535–548.
[52] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC*, May 2009, pp. 169–178.
[53] R. Dathathri et al., "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2019, pp. 142–156.
[54] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1231–1247.
[55] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," in *Proc. PETs*, 2020, pp. 355–375.
[56] L. W. Folkerts, C. Gouert, and N. G. Tsoutsos, "REDsec: Running encrypted discretized neural networks in seconds," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Jun. 2023.
[57] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang, "Optimizing authenticated garbling for faster secure two-party computation," in *Proc. CRYPTO*, 2018, pp. 365–391.
[58] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure MPC over rings with applications to private machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1102–1120.
[59] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero knowledge proofs for decision tree predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 2039–2053.
[60] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "VeriML: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2524–2540, Oct. 2021.
[61] S. Lee et al., "VCNN: Verifiable convolutional neural network based on zk-SNARKs," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 4254–4270, 2020.
[62] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences," *Cryptol. ePrint Arch.*, 2021.
[63] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. CRYPTO*, 2009, pp. 145–161.