Privacy-Preserving Robustness Verification for Neural Networks

Nianyun Song[1,2] **Xiaokun Luan**[3] **Yu Guo**[1] **Rongfang Bie**[1] **Meng Sun**[3] **Xiyue Zhang**[*2]

[1]School of Artificial Intelligence, Beijing Key Laboratory of Artificial Intelligence for Education, Engineering Research Center of Intelligent Technology and Educational Application (Ministry of Education), Beijing Normal University, Beijing, China
[2]School of Computer Science, University of Bristol, Bristol, UK
[3]School of Mathematical Sciences, Peking University, Beijing, China

Abstract

Neural network verification and data privacy are inherently in tension: *verification demands full access to model parameters and input data, yet both are increasingly restricted by privacy regulations and intellectual property constraints*. This tension has left robustness verification impractical in privacy-sensitive domains. In this work, we address this gap with SECURECROWN, the first framework for privacy-preserving neural network robustness verification. Built upon secure two-party computation (2PC), our framework enables a model owner and a data owner to jointly compute certified robustness bounds—revealing only the final result while provably protecting both parties' private data under the semi-honest security model. A key challenge is securely computing the conditional operations in Linear Bound Propagation, where the data-dependent branching is incompatible with standard secure computation protocols. We eliminate branching by formulating conditional logic as continuous arithmetic operations. Additionally, we introduce a Newton–Raphson refinement method to improve numerical stability. Extensive analysis and experiments show that SECURECROWN strictly matches plaintext verification results, while completing in 0.1–200s across varied model sizes and communication settings (LAN/WAN), demonstrating the feasibility of privacy-preserving neural network verification.

1 INTRODUCTION

The field of deep learning (DL) has experienced enormous growth recently, with deep neural networks (DNNs) now deployed in a variety of safety-critical applications, including medical diagnosis [Litjens et al., 2017, Esteva et al., 2017] and healthcare [Esteva et al., 2019, Davenport and Kalakota, 2019]. In these domains, safety and robustness of DNNs are essential. However, DNNs are susceptible to perturbations in their inputs [Szegedy et al., 2014], such as noise and illumination variations [Goodfellow et al., 2015, Hendrycks and Dietterich, 2019]. To ensure DNN robustness with worst-case guarantees, significant progress in verification approaches [Wu et al., 2024, Wang et al., 2021, Singh et al., 2019] has been made, which can certify that a model's prediction remains stable regardless of data noise or environmental changes.

Despite these advances, existing verification techniques assume centralized access to both model parameters and input data in plaintext. This assumption is often unrealistic in privacy-sensitive deployments. In practice, user data may be subject to strict privacy regulations, such as the General Data Protection Regulation (GDPR) [Voigt and Von dem Bussche, 2017]. Meanwhile, proprietary model parameters constitute valuable intellectual property and may not be disclosed to third parties. Thus, the standard verification setting, where both model and data are fully accessible, is incompatible with many privacy-regulated collaborative scenarios.

We consider a two-party verification setting in which a model owner P_0 holds a proprietary DNN, while a data owner P_1 holds sensitive input data. We aim to verify the robustness of the model on the given input such that only the verification result is revealed, while both the model parameters and the input data remain private. Achieving this requires jointly performing robustness verification without exposing either party's confidential information.

Secure multi-party computation (MPC) [Evans et al., 2018] provides a principled cryptographic approach that enables multiple parties to jointly evaluate functions on their private inputs without revealing their data to one another. While MPC has been successfully applied to privacy-preserving inference and training [Knott et al., 2021, Tan et al., 2021, Wagh et al., 2021, Kumar et al., 2020, Gupta et al., 2025,

*Correspondence to XZ (xiyue.zhang@bristol.ac.uk)

Zhang et al., 2021], extending it to robustness verification faces two fundamental challenges: *(i) Data-dependent branching:* Convex relaxation of ReLU activations requires distinct parameters depending on secret-shared activation states. In MPC, resolving these conditionals requires a secure comparison for every neuron, scaling linearly with the total number of neurons and rendering naïve implementations computationally prohibitive. *(ii) Precision sensitivity in deep recursion:* Unlike MPC-based inference, which tolerates minor numerical errors during a single forward pass, verification computes certified bounds via deep backward recursion that amplifies numerical errors. Because safety guarantees depend strictly on the sign of the final margin, these compounded errors can invert near-zero bounds, compromising verification soundness.

To bridge this gap, we propose SECURECROWN, the first privacy-preserving framework for neural network robustness verification built on secure two-party computation (2PC). It enables verification of DNNs while ensuring that sensitive input data remains confidential and proprietary model parameters are not disclosed. Specifically, we develop *(i)* a secure implementation of Linear Bound Propagation (LBP) that computes certified robustness bounds under privacy constraints; *(ii)* a unified branch-free reformulation of the conditional logic in ReLU relaxation and bound propagation, efficiently realized via function secret sharing (FSS); and *(iii)* a Newton–Raphson-based refinement method for reciprocal computation to improve numerical precision and stability. Furthermore, we provide formal security guarantees in the semi-honest adversary model and conduct extensive experiments validating that our protocol achieves verification precision comparable to the plaintext verifier while incurring moderate cryptographic overhead.

2 RELATED WORK

Modern neural network verification methods primarily address robustness analysis in a plaintext setting, where both model parameters and user inputs are assumed to be fully accessible. Representative verifiers include α,β-CROWN [Zhang et al., 2018, Xu et al., 2020, 2021, Wang et al., 2021], CORA [Althoff, 2015], Marabou [Wu et al., 2024, Katz et al., 2019], and NeuralSAT [Duong et al., 2024, 2025]. These approaches rely on techniques such as convex relaxation [Salman et al., 2019], abstract interpretation [Singh et al., 2019], and constraint-solving [Duong et al., 2023] to compute certified robustness bounds or to solve verification queries exactly.

Meanwhile, substantial progress has been made in privacy-preserving machine learning through cryptographic protocols. A prominent line of work employs MPC to enable secure inference. CrypTen [Knott et al., 2021] supports private inference over secret-shared inputs and models in a two-party setting, assuming a semi-honest threat model. CryptGPU [Tan et al., 2021] extends this framework to a three-party system with optimized GPU-based protocols under semi-honest security. Falcon [Wagh et al., 2021] implements strengthened malicious security in a three-party setting and combines SecureNN [Wagh et al., 2019] and ABY^3 [Mohassel and Rindal, 2018] to improve protocol efficiency. ObliviGate [Song et al., 2026] further achieves maliciously secure inference while concealing the model architecture. CrypTFlow [Kumar et al., 2020] proposes an end-to-end toolchain that compiles predefined TensorFlow inference code into MPC protocols, supporting both semi-honest and malicious adversaries. Beyond MPC, zero-knowledge proof systems have been leveraged to provide verifiable inference without revealing model parameters or inputs [Lee et al., 2024, Maheri et al., 2025], while homomorphic encryption (HE) enables neural network inference directly over encrypted data [Lee et al., 2022, Lou and Jiang, 2021].

However, these cryptographic methods focus exclusively on secure or verifiable inference. They do not address robustness verification, which requires computing and propagating symbolic bounds beyond forward inference alone. In contrast, we establish a privacy-preserving framework for neural network robustness verification. To the best of our knowledge, our work is the first to integrate formal robustness verification with cryptographic privacy guarantees.

3 PRELIMINARIES

3.1 LOCAL ROBUSTNESS VERIFICATION

Consider an L-layer feedforward neural network classifier $f : \mathbb{R}^{d_0} \to \mathbb{R}^K$, where d_0 is the input dimension and K is the number of classes. The network is parameterized by weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and bias vectors $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ for layers $l \in \{1, \dots, L\}$, with $d_L = K$. The pre-activation at each layer is defined as $\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$ for $l = 1$, and $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\sigma(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}$ for $l \geq 2$. In this work, we consider the ReLU activation $\sigma(\cdot) = \max(0, \cdot)$.

The network output is $f(\mathbf{x}) = \mathbf{z}^{(L)}$, with predicted class $y = \mathrm{argmax}_k\, f_k(\mathbf{x})$. Given an input $\mathbf{x}_0$ correctly classified as class y, the network is *locally robust* within an ℓ_p-ball of radius ϵ if:

$$f_{y,j} \triangleq \min_{\mathbf{x} \in \mathcal{B}_p(\mathbf{x}_0, \epsilon)} \big(f_y(\mathbf{x}) - f_j(\mathbf{x})\big) > 0, \quad \forall j \neq y, \quad (1)$$

where $\mathcal{B}_p(\mathbf{x}_0, \epsilon) = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_0\|_p \leq \epsilon\}$. Since exactly verifying Eq. (1) is NP-hard for ReLU networks [Katz et al., 2017], verification methods typically compute a certified lower bound $\underline{f}_{y,j}$ via linear relaxation [Salman et al., 2019].

Linear Relaxation. Consider robustness verification under ℓ_p input perturbations $\mathbf{x} \in \mathcal{B}_p(\mathbf{x}_0, \epsilon)$. Let $\underline{z}, \overline{z} \in \mathbb{R}$ be the pre-activation lower and upper bounds for a neuron z, i.e., $\underline{z} \leq z \leq \overline{z}$. To handle the nonlinearity of the activation function, linear relaxation constructs a pair of linear bounding functions—a lower bound $\underline{\alpha}z + \underline{\beta}$ and an upper bound $\overline{\alpha}z + \overline{\beta}$—satisfying $\underline{\alpha}z + \underline{\beta} \leq \sigma(z) \leq \overline{\alpha}z + \overline{\beta}$ for all $z \in [\underline{z}, \overline{z}]$. A commonly used bounding strategy for ReLU in CROWN [Zhang et al., 2018] sets the upper-bound parameters $(\overline{\alpha}, \overline{\beta})$ as:

$$(\overline{\alpha}, \overline{\beta}) = \begin{cases} (1, 0), & \text{if } \underline{z} \geq 0, \\ (0, 0), & \text{if } \overline{z} \leq 0, \\ (\alpha^*, -\alpha^* \underline{z}), & \text{otherwise,} \end{cases} \quad \text{where } \alpha^* = \frac{\overline{z}}{\overline{z} - \underline{z}}, \tag{2}$$

and the lower bound shares the same slope with zero intercept: $(\underline{\alpha}, \underline{\beta}) = (\overline{\alpha}, 0)$.

Linear Bound Propagation. LBP in CROWN-series verifiers computes certified bounds by propagating backward linear relaxations through the network. To bound $\mathbf{z}^{(l)}$ for any layer $l \in \{1, \ldots, L\}$, LBP constructs linear functions such that

$$\mathbf{A}^{(0)}\mathbf{x} + \underline{\mathbf{c}}^{(0)} \leq \mathbf{z}^{(l)} \leq \mathbf{A}^{(0)}\mathbf{x} + \bar{\mathbf{c}}^{(0)}, \ \forall \mathbf{x} \in \mathcal{B}_p(\mathbf{x}_0, \epsilon), \tag{3}$$

where $\mathbf{A}^{(0)} \in \mathbb{R}^{d_l \times d_0}$ and $\underline{\mathbf{c}}^{(0)}, \bar{\mathbf{c}}^{(0)} \in \mathbb{R}^{d_l}$. Since our relaxation uses identical slopes $\underline{\alpha} = \bar{\alpha}$, the coefficient matrix $\mathbf{A}^{(0)}$ is shared for both bounds, while distinct intercepts $\underline{\beta} \neq \bar{\beta}$ yield separate constants $\underline{\mathbf{c}}^{(0)}$ and $\bar{\mathbf{c}}^{(0)}$.

For $l = 1$, since $\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$ is purely an affine transformation of $\mathbf{x}$, we directly obtain $\mathbf{A}^{(0)} = \mathbf{W}^{(1)}$ and $\underline{\mathbf{c}}^{(0)} = \bar{\mathbf{c}}^{(0)} = \mathbf{b}^{(1)}$. For $l \geq 2$, the backward recursion requires relaxation parameters $(\boldsymbol{\alpha}^{(t)}, \underline{\boldsymbol{\beta}}^{(t)}, \bar{\boldsymbol{\beta}}^{(t)})$ for each intermediate layer $t \in \{1, \ldots, l-1\}$, where $\boldsymbol{\alpha}^{(t)} \in \mathbb{R}^{d_t}$ denotes slopes and $\underline{\boldsymbol{\beta}}^{(t)}, \bar{\boldsymbol{\beta}}^{(t)} \in \mathbb{R}^{d_t}$ intercepts; these are determined by the pre-activation bounds via Eq. (2). The recursion maintains $\mathbf{A}^{(t)} \in \mathbb{R}^{d_l \times d_t}$ and $\underline{\mathbf{c}}^{(t)}, \bar{\mathbf{c}}^{(t)} \in \mathbb{R}^{d_l}$ such that $\mathbf{A}^{(t)}\mathbf{z}^{(t)} + \underline{\mathbf{c}}^{(t)} \leq \mathbf{z}^{(l)} \leq \mathbf{A}^{(t)}\mathbf{z}^{(t)} + \bar{\mathbf{c}}^{(t)}$. Starting with $\mathbf{A}^{(l)} = \mathbf{I}_{d_l}$ and $\underline{\mathbf{c}}^{(l)} = \bar{\mathbf{c}}^{(l)} = \mathbf{0}$, the backward updates for $t = l, \ldots, 2$ are given by:

$$\hat{\mathbf{A}}^{(t)} = \mathbf{A}^{(t)}\mathbf{W}^{(t)}, \tag{4}$$

$$\mathbf{A}^{(t-1)} = \hat{\mathbf{A}}^{(t)} \operatorname{diag}(\boldsymbol{\alpha}^{(t-1)}), \tag{5}$$

$$\underline{\mathbf{c}}^{(t-1)} = \underline{\mathbf{c}}^{(t)} + \mathbf{A}^{(t)}\mathbf{b}^{(t)} + \underline{\boldsymbol{\delta}}^{(t-1)}, \tag{6}$$

$$\bar{\mathbf{c}}^{(t-1)} = \bar{\mathbf{c}}^{(t)} + \mathbf{A}^{(t)}\mathbf{b}^{(t)} + \bar{\boldsymbol{\delta}}^{(t-1)}, \tag{7}$$

where the i-th element (for $i \in \{1, \ldots, d_l\}$) of the intercept contribution is $\underline{\delta}_i^{(t-1)} = \sum_{j:\, \hat{A}_{ij}^{(t)} > 0} \hat{A}_{ij}^{(t)} \underline{\beta}_j^{(t-1)} + \sum_{j:\, \hat{A}_{ij}^{(t)} \leq 0} \hat{A}_{ij}^{(t)} \bar{\beta}_j^{(t-1)}$, and $\bar{\boldsymbol{\delta}}^{(t-1)}$ is defined analogously with $\underline{\beta}$ and $\bar{\beta}$ swapped. The recursion terminates at the input layer yielding $\mathbf{A}^{(0)} = \mathbf{A}^{(1)}\mathbf{W}^{(1)}$, $\underline{\mathbf{c}}^{(0)} = \underline{\mathbf{c}}^{(1)} + \mathbf{A}^{(1)}\mathbf{b}^{(1)}$, and $\bar{\mathbf{c}}^{(0)} = \bar{\mathbf{c}}^{(1)} + \mathbf{A}^{(1)}\mathbf{b}^{(1)}$. By applying Hölder's inequality over the ℓ_p-ball $\mathcal{B}_p(\mathbf{x}_0, \epsilon)$ (where $1/p + 1/q = 1$), the symbolic bounds in Eq. (3) are converted into concrete element-wise bounds:

$$\underline{z}_i^{(l)} = \mathbf{A}_{i,:}^{(0)}\mathbf{x}_0 + \underline{c}_i^{(0)} - \epsilon \|\mathbf{A}_{i,:}^{(0)}\|_q, \tag{8}$$

$$\bar{z}_i^{(l)} = \mathbf{A}_{i,:}^{(0)}\mathbf{x}_0 + \bar{c}_i^{(0)} + \epsilon \|\mathbf{A}_{i,:}^{(0)}\|_q. \tag{9}$$

Margin verification. For margin verification between classes y and j, we apply LBP to the output layer ($l = L$) by setting $\mathbf{W}^{(L)} \leftarrow \mathbf{W}_{y,:}^{(L)} - \mathbf{W}_{j,:}^{(L)} \in \mathbb{R}^{1 \times d_{L-1}}$ and $b^{(L)} \leftarrow b_y^{(L)} - b_j^{(L)} \in \mathbb{R}$. With $d_L = 1$, the coefficient matrix $\mathbf{A}^{(t)}$ reduces to a row vector and $\underline{c}^{(t)}$ to a scalar. The certified lower bound is:

$$\underline{f}_{y,j} = \mathbf{A}^{(0)}\mathbf{x}_0 + \underline{c}^{(0)} - \epsilon \|\mathbf{A}^{(0)}\|_q. \tag{10}$$

Local robustness is verified if $\underline{f}_{y,j} > 0$.

3.2 SECURE COMPUTATION PRIMITIVES

In the standard 2PC setting [Yao, 1982, Goldreich et al., 1987], two parties (P_0 and P_1) jointly compute a function over their private inputs, revealing nothing beyond the prescribed output. Computations are performed in the ring $\mathbb{Z}_{2^k}$ of bit-length k (modulo 2^k is omitted for brevity). A real value $v \in \mathbb{R}$ is encoded in two's complement fixed-point format with n_f fractional bits as $\tilde{v} = \lfloor v \cdot 2^{n_f} \rfloor \in \mathbb{Z}_{2^k}$.

Additive Secret Sharing (ASS). A value $x \in \mathbb{Z}_{2^k}$ is additively shared as $\langle x \rangle = \{\langle x \rangle_0, \langle x \rangle_1\}$ such that $x = \langle x \rangle_0 + \langle x \rangle_1$. To generate shares, P_0 samples $\langle x \rangle_0 \in \mathbb{Z}_{2^k}$ uniformly at random and sets $\langle x \rangle_1 = x - \langle x \rangle_0$; each individual share is information-theoretically independent of x. ASS supports linear operations locally. Given shares $\langle x \rangle, \langle y \rangle$ and a public value $c \in \mathbb{Z}_{2^k}$, party P_i ($i \in \{0, 1\}$) computes

$$\langle x + y \rangle_i = \langle x \rangle_i + \langle y \rangle_i, \quad \langle c \cdot x \rangle_i = c \cdot \langle x \rangle_i.$$

For adding a public constant, one party (e.g., P_0) absorbs it locally: $\langle x + c \rangle_0 = \langle x \rangle_0 + c$ and $\langle x + c \rangle_1 = \langle x \rangle_1$.

Secure Multiplication via Beaver Triples. Nonlinear operations require interaction. A standard technique uses a Beaver triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ with $c = a \cdot b$. Given $\langle x \rangle, \langle y \rangle$, parties compute masked differences $e = x - a$ and $d = y - b$ by locally forming $\langle e \rangle = \langle x \rangle - \langle a \rangle$ and $\langle d \rangle = \langle y \rangle - \langle b \rangle$, then opening e and d (i.e., exchanging shares and reconstructing the public values). They then output product shares

$$\langle x \cdot y \rangle_i = \langle c \rangle_i + d \cdot \langle a \rangle_i + e \cdot \langle b \rangle_i + (1 - i) \cdot e \cdot d,$$

where $i \in \{0, 1\}$. This yields $\langle x \cdot y \rangle_0 + \langle x \cdot y \rangle_1 = xy$, followed by a truncation procedure to restore the fixed-point scale, i.e., $\langle \widetilde{xy} \rangle = \langle \lfloor \tilde{x} \cdot \tilde{y} / 2^{n_f} \rfloor \rangle$.

Function Secret Sharing. FSS generalizes secret sharing from values to functions [Boyle et al., 2015]. A two-party FSS scheme for a function family $\mathcal{G}$ consists of:

- $\mathsf{Gen}(1^\lambda, g) \rightarrow (k_0, k_1)$: a randomized key generation algorithm that, given a security parameter λ and a function $g \in \mathcal{G}$, outputs two correlated keys (k_0, k_1).
- $\mathsf{Eval}(i, k_i, x) \rightarrow y_i$: a deterministic evaluation algorithm run by P_i on input x (typically a public masked value in MPC), producing an output share y_i such that $y_0 + y_1 = g(x)$ (over an appropriate output ring).

Intuitively, each key k_i hides the function g, yet the two evaluations add up to the correct value. A key instantiation is the Distributed Comparison Function (DCF), computing $g^{<}_{\theta,\rho}(x) = \rho \cdot \mathbb{I}\{x < \theta\}$ for threshold θ and payload ρ, where $\mathbb{I}$ is the indicator function. DCF is a basic primitive for secure comparison and is commonly used to build higher-level operators such as ReLU and fixed-point truncation.

FSS-Based Secure Evaluation on Masked Values. Many FSS-based 2PC protocols maintain a *masking invariant* on intermediate values. Consequently, for a private value $x \in \mathbb{Z}_{2^k}$, the parties hold a public masked value $\hat{x} = x + r_{\text{in}}$, where r_{in} is a preprocessed random mask unknown to either party. To evaluate $y = g(x)$ while preserving the invariant, the parties generate FSS keys for a shifted function:

$$g_{r_{\text{in}},r_{\text{out}}}(\hat{x}) = g(\hat{x} - r_{\text{in}}) + r_{\text{out}},$$

where r_{out} is a fresh output mask. Each party evaluates its key on $\hat{x}$ to obtain shares of $g_{r_{\text{in}},r_{\text{out}}}(\hat{x})$, which reconstruct to $\hat{y} = g(x) + r_{\text{out}}$. Thus the output remains masked in the same form, enabling secure composition of function evaluations without revealing intermediate plaintexts.

4 PROBLEM FORMULATION

System Setup. We study privacy-preserving robustness verification in a 2PC setting with a *model owner* (P_0) and a *data owner* (P_1). The network architecture (number of layers L and layer dimensions $\{d_l\}_{l=0}^{L}$) is assumed to be public. The model owner P_0 holds a trained neural network f with private parameters $\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$, which represent proprietary intellectual property. The data owner P_1 holds a verification query consisting of input $\mathbf{x}_0 \in \mathbb{R}^{d_0}$, robustness perturbation radius $\epsilon > 0$ (the magnitude of worst-case perturbations to certify against), and the classification target pair (true label y, target label j). During setup, P_0 secret-shares the model parameters, retains $\{\langle \mathbf{W}^{(l)} \rangle_0, \langle \mathbf{b}^{(l)} \rangle_0\}_{l=1}^{L}$, and sends $\{\langle \mathbf{W}^{(l)} \rangle_1, \langle \mathbf{b}^{(l)} \rangle_1\}_{l=1}^{L}$ to P_1. To avoid the $O(K)$ overhead associated with secure array indexing, P_1 encodes the verification target as a difference vector $\mathbf{d} = \mathbf{e}_y - \mathbf{e}_j \in \{-1, 0, 1\}^K$ (where $\mathbf{e}_y$ and $\mathbf{e}_j$ are one-hot vectors). P_1 then secret-shares these encoded inputs, retaining $(\langle \mathbf{x}_0 \rangle_1, \langle \epsilon \rangle_1, \langle \mathbf{d} \rangle_1)$ and sending $(\langle \mathbf{x}_0 \rangle_0, \langle \epsilon \rangle_0, \langle \mathbf{d} \rangle_0)$ to P_0.

Threat Model. We consider the *semi-honest* (honest-but-curious) adversarial model, where both parties follow the protocol specification but may attempt to infer the other party's private input from their view. The protocol operates in a preprocessing model with a trusted dealer $\mathcal{D}$ that distributes correlated randomness (Beaver triples, FSS keys) during an offline phase. The preprocessing depends only on public parameters, so $\mathcal{D}$ learns nothing about private inputs. We assume $\mathcal{D}$ does not collude with either party. In practice, $\mathcal{D}$ can be instantiated via trusted hardware or replaced by a two-party preprocessing protocol.

Problem Statement. Given secret shares of a neural network f and a verification query $(\mathbf{x}_0, \epsilon, \mathbf{d})$ held by the two parties (where y and j are encoded in the difference vector $\mathbf{d}$), design a 2PC protocol that enables P_1 to determine whether the network is certifiably ℓ_∞-robust at $(\mathbf{x}_0, \epsilon)$ for the margin between classes y and j. The parties jointly compute secret shares of the certified lower bound $\underline{f}_{y,j}$ (Eq. (10)). Upon completion, P_0 sends $\langle \underline{f}_{y,j} \rangle_0$ to P_1, who reconstructs $\underline{f}_{y,j} = \langle \underline{f}_{y,j} \rangle_0 + \langle \underline{f}_{y,j} \rangle_1$ and concludes robust if $\underline{f}_{y,j} > 0$, or unknown otherwise.

While the problem formulation above is general, our current protocol instantiates it for fully connected ReLU networks; extensions to other architectures are discussed in Appendix D.

Design Goals. SECURECROWN targets two objectives: **(1) Input and Model Privacy:** The protocol satisfies semi-honest security. P_0 learns nothing about $(\mathbf{x}_0, \epsilon, \mathbf{d})$ beyond public parameters, and P_1 learns only $\underline{f}_{y,j}$. All intermediate quantities remain secret-shared throughout. **(2) Faithful Verification Semantics:** The protocol preserves the decision behavior of the plaintext verifier (Section 3.1). Any numerical error introduced by fixed-point arithmetic must be bounded to prevent unsound certifications.

Security Definition. We formalize semi-honest security in the real/ideal paradigm. Let $\mathsf{inp}_0 = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$ and $\mathsf{inp}_1 = (\mathbf{x}_0, \epsilon, \mathbf{d})$ denote the private inputs of P_0 and P_1, respectively. The ideal functionality is defined as $\mathcal{F}(\mathsf{inp}_0, \mathsf{inp}_1) = (\perp, \underline{f}_{y,j})$, where P_0 receives no output ($\perp$) and P_1 receives the certified lower bound.

Definition 1 (Semi-Honest Security). *Protocol* Π *securely computes* $\mathcal{F}$ *if for each* $i \in \{0, 1\}$*, there exists a probabilistic polynomial-time (PPT) simulator* Sim_i *such that:*

$$\left\{\mathsf{Sim}_i(1^\lambda, \mathsf{inp}_i, \mathcal{F}_i(\mathsf{inp}_0, \mathsf{inp}_1))\right\} \stackrel{c}{\equiv} \left\{\mathsf{View}_i^{\Pi}(1^\lambda, \mathsf{inp}_0, \mathsf{inp}_1)\right\} \tag{11}$$

where View_i^{Π} *denotes the view of* P_i *(input, randomness, received messages),* $\mathcal{F}_i$ *denotes the output to* P_i*, and* $\stackrel{c}{\equiv}$ *denotes computational indistinguishability.*

5 METHODOLOGY

5.1 OVERVIEW

Challenges. Implementing the robustness verification (Section 3.1) in MPC presents two fundamental challenges. *(1) Data-dependent branching:* The relaxation slope α depends on neuron activation state—inactive ($\bar{z} \leq 0$), active ($\underline{z} \geq 0$), or unstable—each requiring distinct computation (Eq. (2)). Similarly, intercept terms $\underline{\boldsymbol{\delta}}$ and $\bar{\boldsymbol{\delta}}$ (Eqs. (6)–(7)) depend on coefficient signs. In MPC, such data-dependent branching requires costly secure comparison, preventing efficient batching across neurons. *(2) Precision sensitivity in deep recursion:* The backward recursion in Eq. (5) updates the coefficient matrix $\mathbf{A}^{(t)}$ across layers, with relaxation slopes $\boldsymbol{\alpha}^{(t)}$ computed via division (Eq. (2)). In fixed-point MPC, division has limited precision, and truncation errors accumulate through the propagation chain, potentially compromising verification accuracy in deep networks.

Our Approach. We address these challenges through two key techniques. *(1) Branch Elimination via ReLU:* We observe that all branching conditions in the verifier are fundamentally sign tests. Since ReLU inherently encodes sign information, we reformulate conditional logic into unified arithmetic expressions where ReLU outputs naturally select the correct terms. This eliminates branching entirely, enabling identical computation paths for all neurons and facilitating efficient vectorized execution in MPC. These ReLU operations are efficiently realized via FSS-based protocols with constant-round communication. *(2) Newton–Raphson Refinement:* To address precision loss in secure division, we apply Newton–Raphson iterations to refine reciprocal estimates, approximately doubling the precision bits per iteration at minimal additional cost.

Protocol Overview. The protocol operates in the preprocessing model: correlated randomness is generated offline based on the public network architecture, reducing the online phase to local computations and share exchanges. Specifically, in the **offline phase**, the trusted dealer $\mathcal{D}$ generates correlated randomness based on the network architecture $\{d_0, \ldots, d_L\}$, including Beaver triples for multiplication, FSS keys with associated input masks for function evaluations, and other correlated randomness required by the sub-protocols. This phase is data-independent, so $\mathcal{D}$ learns nothing about private inputs. The **online phase** begins with **input sharing**, where the parties secret-share their private inputs as described in Section 4. Next, in the **secure computation** phase, the parties jointly execute Π_{Verify} (Algorithm 2): for each hidden layer $l \in \{1, \ldots, L-1\}$, compute pre-activation bounds $\langle \underline{\mathbf{z}}^{(l)} \rangle$, $\langle \overline{\mathbf{z}}^{(l)} \rangle$ via Π_{Backward} (Algorithm 1) and derive relaxation slopes $\langle \boldsymbol{\alpha}^{(l)} \rangle$ via $\Pi_{\boldsymbol{\alpha}}$; then execute a final backward pass using $\langle \mathbf{d} \rangle$ to compute $\langle \underline{f}_{y,j} \rangle$ (Eq. (10)). Finally, in the **output** phase, P_0 sends $\langle \underline{f}_{y,j} \rangle_0$ to P_1, who reconstructs the certified lower bound $\underline{f}_{y,j}$ and concludes local robustness if $\underline{f}_{y,j} > 0$.

5.2 SECURE BUILDING BLOCKS

Secure Multiplication (SecMul, $\mathsf{SecMatMul}$)**.** Secure scalar multiplication $\mathsf{SecMul}(\langle x \rangle, \langle y \rangle) \to \langle xy \rangle$ follows the standard Beaver triple protocol (Section 3.2). For matrix multiplication, given inputs $\mathbf{X} \in \mathbb{Z}_{2^k}^{m \times n}$ and $\mathbf{Y} \in \mathbb{Z}_{2^k}^{n \times p}$, the protocol computes $\mathsf{SecMatMul}(\langle \mathbf{X} \rangle, \langle \mathbf{Y} \rangle) \to \langle \mathbf{XY} \rangle$ by consuming matrix-level Beaver triples $(\langle \mathbf{A} \rangle, \langle \mathbf{B} \rangle, \langle \mathbf{AB} \rangle)$, thereby avoiding the overhead of scalar-wise decomposition.

Secure Arithmetic Right Shift (SecARS)**.** To realign the radix point after fixed-point multiplications—which inherently yield $2n_f$ fractional bits—we must restore the standard n_f-bit precision. We achieve this via $\mathsf{SecARS}(\langle x \rangle, n_f) \to \langle \lfloor x/2^{n_f} \rfloor \rangle$, efficiently instantiated using the DCF-based protocol of Gupta et al. [2025].

Secure ReLU and Absolute Value ($\mathsf{SecReLU}$, SecAbs)**.** Both primitives are built on securely computing the sign indicator $\mu = \mathbb{I}[x < 0]$ via DCF. Parties evaluate preprocessed DCF keys to obtain the secret-shared bit $\langle \mu \rangle$, and subsequently compute $\mathsf{SecReLU}(\langle x \rangle) \to \mathsf{SecMul}(\langle 1-\mu \rangle, \langle x \rangle)$ and $\mathsf{SecAbs}(\langle x \rangle) \to \mathsf{SecMul}(\langle 1-2\mu \rangle, \langle x \rangle)$. Each primitive strictly requires one DCF evaluation followed by one SecMul operation.

Secure Reciprocal ($\mathsf{SecRecip}$)**.** We implement the reciprocal protocol $\mathsf{SecRecip}(\langle x \rangle) \to \langle 1/x \rangle$ in two stages. First, we adapt the FSS-based approach from Gupta et al. [2025] to generate an initial approximation. The input magnitude is decomposed as $|x| \approx (1 + \frac{m}{2^{\ell_m}}) \cdot 2^e$, and a seed $w_0 \approx 1/|x|$ is computed via $w_0 = 2^{-e} \cdot \frac{2^{\ell_m}}{2^{\ell_m} + m}$, where the terms are retrieved via spline and lookup tables. Second, since the lookup-based initialization provides limited precision, we perform n_{iter} Newton–Raphson iterations $w_{i+1} = w_i \cdot (2 - |x| \cdot w_i)$ to refine the result to the target precision. Each iteration roughly doubles the number of accurate bits, ensuring sufficient numerical stability for subsequent verification steps.

5.3 SECURE SLOPE COMPUTATION $\Pi_{\boldsymbol{\alpha}}$

The relaxation slope vector $\boldsymbol{\alpha}^{(l)}$ in Eq. (2) is essential for constructing the linear propagation matrices in LBP. Although the definition is simple, securely evaluating the piecewise logic in Eq. (2) is costly in MPC, as it requires secure comparisons and data-dependent branching per neuron to distinguish active, inactive, and unstable states. To circumvent these overheads and enable efficient execution, we reformulate the piecewise logic into a unified, data-independent arithmetic expression applied element-wise across all neurons in the layer:

$$\boldsymbol{\alpha} = \frac{\mathrm{ReLU}(\bar{\mathbf{z}})}{\mathrm{ReLU}(\bar{\mathbf{z}}) + \mathrm{ReLU}(-\underline{\mathbf{z}}) + \epsilon_s}. \tag{12}$$

This formulation naturally encapsulates the three theoretical cases within a single data-independent execution path: for inactive neurons where $\bar{z} \leq 0$, the vanishing numerator $\mathrm{ReLU}(\bar{z}) = 0$ yields a slope of $\alpha = 0$; for active neurons where $\underline{z} \geq 0$, the term $\mathrm{ReLU}(-\underline{z}) = 0$ results in $\alpha \approx 1$; and for unstable neurons, the expression simplifies to the optimal chord slope $\bar{z}/(\bar{z} - \underline{z})$. To ensure numerical robustness during the MPC execution, we introduce a small public stability constant ϵ_s into the denominator of Eq. (12). This safeguard prevents potential division-by-zero errors that may arise when pre-activation bounds $[\underline{z}, \bar{z}]$ vanish near zero—a common phenomenon in deep neural networks due to the accumulation of conservative bounds during propagation. A theoretical analysis of its impact on the certified bounds is provided in Appendix B.3. The detailed vectorized protocol $\Pi_{\boldsymbol{\alpha}}$, which achieves branching-free computation using SecReLU, SecRecip, and SecMul primitives, is presented in Algorithm 3 in Appendix A.

5.4 SECURE INTERCEPT ACCUMULATION $\Pi_{\boldsymbol{\delta}}$

As established in Eqs. (6)–(7), the intercept contribution vectors $\underline{\boldsymbol{\delta}}^{(t-1)}, \bar{\boldsymbol{\delta}}^{(t-1)} \in \mathbb{R}^{d_l}$ depend on the element-wise signs of the intermediate coefficient matrix $\hat{\mathbf{A}}^{(t)} \in \mathbb{R}^{d_l \times d_{t-1}}$. From Eq. (2), the lower relaxation intercept is $\underline{\beta}_j^{(t-1)} = 0$ for all neurons, while the upper intercept is $\bar{\beta}_j^{(t-1)} = -\alpha_j^{(t-1)} \underline{z}_j^{(t-1)}$ for unstable neurons (and zero otherwise). Since $\underline{\beta}_j^{(t-1)} = 0$, the accumulations simplify significantly: only terms involving $\bar{\beta}$ contribute. By substituting $A_{ij}^{(t-1)} = \hat{A}_{ij}^{(t)} \alpha_j^{(t-1)}$ (Eq. (5)) and noting that $\alpha_j^{(t-1)} \geq 0$ preserves signs, the i-th elements of the lower and upper contributions reduce to $\underline{\delta}_i^{(t-1)} = \sum_{j:\, A_{ij}^{(t-1)} \leq 0} A_{ij}^{(t-1)}(-\underline{z}_j^{(t-1)})$ and $\bar{\delta}_i^{(t-1)} = \sum_{j:\, A_{ij}^{(t-1)} > 0} A_{ij}^{(t-1)}(-\underline{z}_j^{(t-1)})$, respectively.

To avoid expensive secure comparisons in MPC, these conditional summations can be elegantly reformulated into unified matrix-vector multiplications using ReLU operations:

$$\underline{\boldsymbol{\delta}}^{(t-1)} = -\mathrm{ReLU}(-\mathbf{A}^{(t-1)})\mathrm{ReLU}(-\underline{\mathbf{z}}^{(t-1)}), \tag{13}$$

$$\bar{\boldsymbol{\delta}}^{(t-1)} = \mathrm{ReLU}(\mathbf{A}^{(t-1)})\mathrm{ReLU}(-\underline{\mathbf{z}}^{(t-1)}), \tag{14}$$

The protocol $\Pi_{\boldsymbol{\delta}}$ is given in Algorithm 4 (Appendix A).

5.5 SECURE VERIFICATION PROTOCOL

Building on the sub-protocols $\Pi_{\boldsymbol{\alpha}}$ and $\Pi_{\boldsymbol{\delta}}$, we now present the complete secure verification protocol. Algorithm 1 (Π_{Backward}) securely implements the backward recursion (Eqs. (4)–(7)) for a target layer l. Starting with $\langle \mathbf{A} \rangle = \langle \mathbf{I}_{d_l} \rangle$ and $\langle \underline{\mathbf{c}} \rangle = \langle \bar{\mathbf{c}} \rangle = \langle \mathbf{0} \rangle$, the protocol iterates for $t = l, \ldots, 2$, performing coefficient propagation (SecMatMul), slope multiplication (SecMul), intercept accumulation ($\Pi_{\boldsymbol{\delta}}$), and bias addition. Final bounds follow from Hölder's inequality (Eqs. (8)–(9)), with the ℓ_1-norm computed via SecAbs and local summation (for ℓ_∞-perturbations).

Iterative Bound and Slope Computation. As described in Section 3.1, computing bounds at layer l requires slopes from all preceding layers, so the parties compute bounds and slopes iteratively from $l = 1$ to $L - 1$. For $l = 1$, bounds are computed directly: the linear term $\langle \mathbf{W}^{(1)} \mathbf{x}_0 \rangle$ via SecMatMul and the row-wise norm via SecAbs; slopes $\langle \boldsymbol{\alpha}^{(1)} \rangle$ then follow from $\Pi_{\boldsymbol{\alpha}}$. For $l \geq 2$, the parties invoke Π_{Backward} with previously computed slope shares to obtain $\langle \underline{\mathbf{z}}^{(l)} \rangle, \langle \bar{\mathbf{z}}^{(l)} \rangle$, and then derive $\langle \boldsymbol{\alpha}^{(l)} \rangle$ via $\Pi_{\boldsymbol{\alpha}}$.

Algorithm 1 Π_{Backward}: Secure Linear Bound Propagation

Require: Target layer l; $\{\langle \boldsymbol{\alpha}^{(t)} \rangle, \langle \underline{\mathbf{z}}^{(t)} \rangle\}_{t=1}^{l-1}$; $\langle \mathbf{x}_0 \rangle$; $\langle \epsilon \rangle$
Ensure: Concrete bounds $\langle \underline{\mathbf{z}}^{(l)} \rangle, \langle \bar{\mathbf{z}}^{(l)} \rangle$

$\langle \mathbf{A} \rangle \leftarrow \langle \mathbf{I}_{d_l} \rangle$; $\langle \underline{\mathbf{c}} \rangle \leftarrow \langle \mathbf{0} \rangle$; $\langle \bar{\mathbf{c}} \rangle \leftarrow \langle \mathbf{0} \rangle$
for $t = l, \ldots, 2$ **do**
  $\langle \hat{\mathbf{A}} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{W}^{(t)} \rangle)$ ▷ Eq. (4)
  $\langle \mathbf{A} \rangle \leftarrow \mathsf{SecMatMul}(\langle \hat{\mathbf{A}} \rangle, \langle \mathrm{diag}(\boldsymbol{\alpha}^{(t-1)}) \rangle)$ ▷ Eq. (5)
  $\langle \underline{\boldsymbol{\delta}} \rangle, \langle \bar{\boldsymbol{\delta}} \rangle \leftarrow \Pi_{\boldsymbol{\delta}}(\langle \mathbf{A} \rangle, \langle \underline{\mathbf{z}}^{(t-1)} \rangle)$ ▷ Eqs. (13)–(14)
  $\langle \underline{\mathbf{c}} \rangle \leftarrow \langle \underline{\mathbf{c}} \rangle + \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{b}^{(t)} \rangle) + \langle \underline{\boldsymbol{\delta}} \rangle$ ▷ Eq. (6)
  $\langle \bar{\mathbf{c}} \rangle \leftarrow \langle \bar{\mathbf{c}} \rangle + \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{b}^{(t)} \rangle) + \langle \bar{\boldsymbol{\delta}} \rangle$ ▷ Eq. (7)
$\langle \mathbf{A}^{(0)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{W}^{(1)} \rangle)$
$\langle \underline{\mathbf{c}}^{(0)} \rangle \leftarrow \langle \underline{\mathbf{c}} \rangle + \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{b}^{(1)} \rangle)$
$\langle \bar{\mathbf{c}}^{(0)} \rangle \leftarrow \langle \bar{\mathbf{c}} \rangle + \mathsf{SecMatMul}(\langle \mathbf{A} \rangle, \langle \mathbf{b}^{(1)} \rangle)$
$\langle \mathbf{r} \rangle \leftarrow \mathrm{RowSum}(\mathsf{SecAbs}(\langle \mathbf{A}^{(0)} \rangle))$ ▷ Row-wise ℓ_1-norm
$\langle \mathbf{m} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{A}^{(0)} \rangle, \langle \mathbf{x}_0 \rangle)$; $\langle \mathbf{s} \rangle \leftarrow \mathsf{SecMul}(\langle \epsilon \rangle, \langle \mathbf{r} \rangle)$
$\langle \underline{\mathbf{z}}^{(l)} \rangle \leftarrow \langle \mathbf{m} \rangle + \langle \underline{\mathbf{c}}^{(0)} \rangle - \langle \mathbf{s} \rangle$; $\langle \bar{\mathbf{z}}^{(l)} \rangle \leftarrow \langle \mathbf{m} \rangle + \langle \bar{\mathbf{c}}^{(0)} \rangle + \langle \mathbf{s} \rangle$
return $\langle \underline{\mathbf{z}}^{(l)} \rangle, \langle \bar{\mathbf{z}}^{(l)} \rangle$

Algorithm 2 Π_{Verify}: Secure Robustness Verification

Require: Shares $\langle \mathbf{W}^{(l)} \rangle, \langle \mathbf{b}^{(l)} \rangle$ for $l \in [L]$; $\langle \mathbf{x}_0 \rangle$; $\langle \epsilon \rangle$; $\langle \mathbf{d} \rangle$
Ensure: Certified lower bound $\langle \underline{f}_{y,j} \rangle$

/* Iterative Bound and Slope Computation */
$\langle \mathbf{z}^{(1)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{W}^{(1)} \rangle, \langle \mathbf{x}_0 \rangle) + \langle \mathbf{b}^{(1)} \rangle$
$\langle \mathbf{r}^{(1)} \rangle \leftarrow \mathrm{RowSum}(\mathsf{SecAbs}(\langle \mathbf{W}^{(1)} \rangle))$ ▷ Row-wise ℓ_1-norm
$\langle \bar{\mathbf{z}}^{(1)} \rangle \leftarrow \langle \mathbf{z}^{(1)} \rangle + \mathsf{SecMul}(\langle \epsilon \rangle, \langle \mathbf{r}^{(1)} \rangle)$
$\langle \underline{\mathbf{z}}^{(1)} \rangle \leftarrow \langle \mathbf{z}^{(1)} \rangle - \mathsf{SecMul}(\langle \epsilon \rangle, \langle \mathbf{r}^{(1)} \rangle)$
$\langle \boldsymbol{\alpha}^{(1)} \rangle \leftarrow \Pi_\alpha(\langle \bar{\mathbf{z}}^{(1)} \rangle, \langle \underline{\mathbf{z}}^{(1)} \rangle)$
for $l = 2, \ldots, L-1$ **do**
  $\langle \underline{\mathbf{z}}^{(l)} \rangle, \langle \bar{\mathbf{z}}^{(l)} \rangle \leftarrow \Pi_{\mathrm{Backward}}(l, \{\langle \boldsymbol{\alpha}^{(t)} \rangle, \langle \underline{\mathbf{z}}^{(t)} \rangle\}_{t=1}^{l-1})$
  $\langle \boldsymbol{\alpha}^{(l)} \rangle \leftarrow \Pi_\alpha(\langle \bar{\mathbf{z}}^{(l)} \rangle, \langle \underline{\mathbf{z}}^{(l)} \rangle)$
/* Margin Certification */
$\langle \mathbf{W}^{(L)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{d} \rangle^\top, \langle \mathbf{W}^{(L)} \rangle)$
$\langle b^{(L)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{d} \rangle^\top, \langle \mathbf{b}^{(L)} \rangle)$
$\langle \underline{f}_{y,j} \rangle, \_ \leftarrow \Pi_{\mathrm{Backward}}(L, \{\langle \boldsymbol{\alpha}^{(t)} \rangle, \langle \underline{\mathbf{z}}^{(t)} \rangle\}_{t=1}^{L-1})$
return $\langle \underline{f}_{y,j} \rangle$

Margin Certification. After obtaining all slope shares $\langle \boldsymbol{\alpha}^{(1)} \rangle, \ldots, \langle \boldsymbol{\alpha}^{(L-1)} \rangle$, the parties compute the certified margin $\langle \underline{f}_{y,j} \rangle$ via a final backward pass. To avoid $O(K)$ overhead from secure array indexing, P_1 provides a difference vector $\langle \mathbf{d} \rangle$ where $\mathbf{d} = \mathbf{e}_y - \mathbf{e}_j \in \{-1, 0, 1\}^K$. The backward pass initializes $\langle \mathbf{W}^{(L)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{d} \rangle^\top, \langle \mathbf{W}^{(L)} \rangle) \in \mathbb{R}^{1 \times d_{L-1}}$ and $\langle b^{(L)} \rangle \leftarrow \mathsf{SecMul}(\langle \mathbf{d} \rangle^\top, \langle \mathbf{b}^{(L)} \rangle) \in \mathbb{R}$. This reduces the coefficient $\mathbf{A}^{(t)}$ to a row vector in $\mathbb{R}^{1 \times d_t}$, lowering per-layer communication from $O(d^2)$ to $O(d)$. The parties then invoke Π_{Backward} with target layer L to obtain $\langle \mathbf{A}^{(0)} \rangle$ and $\langle \underline{c}^{(0)} \rangle$, from which the certified margin $\langle \underline{f}_{y,j} \rangle$ follows via Eq. (10).

6 THEORETICAL ANALYSIS

Security. The security of SECURECROWN is analyzed under the semi-honest adversary model (Definition 1). The proof constructs a simulator via a hybrid argument over L layers, reducing security to the underlying FSS and ASS primitives. The complete proof of Theorem 6.1 is provided in Appendix B.1. A pathway for upgrading to malicious security, together with an empirical cost estimate, is discussed in Appendix C.

Theorem 6.1 (Security). *Protocol Π_{verify} securely computes $\mathcal{F}_{\text{verify}}$ in the $\mathcal{F}_{\mathcal{D}}$-hybrid model against semi-honest adversaries.*

Complexity. For an L-layer network with maximum width d, SECURECROWN requires $O(L^2)$ rounds, $O(L^2 d^2 k)$ communication, and $O(L^2 d^3)$ computation (Appendix B.2).
Error Analysis. Fixed-point arithmetic introduces error in $\underline{f}_{y,j}$ bounded by a network-depth-dependent multiple of the per-operation rounding unit $\varepsilon_q = 2^{-n_f}$ (Appendix B.3). With maximum weight norm $\rho = \max_i \|\mathbf{W}^{(i)}\|_\infty$, the absolute error in each output bound entry scales as $O(L)\,\varepsilon_q$, $O(L^2)\,\varepsilon_q$, and $O(\rho^L)\,\varepsilon_q$ for the stable ($\rho < 1$), unit-norm ($\rho = 1$), and unstable ($\rho > 1$) regimes, respectively.

7 EXPERIMENTAL EVALUATION

7.1 EXPERIMENTAL SETUP

Implementation. We implement SECURECROWN in C++, with $k = 64$-bit fixed-point representation, $n_f = 26$ fractional bits, stability constant $\epsilon_s = 10^{-4}$, and $n_{\text{iter}} = 1$ Newton–Raphson refinement iteration in SecRecip (the FSS-based seed provides approximately $\lceil n_f/2 \rceil$ accurate bits, and one iteration doubles this to $\geq n_f$ bits). Experiments are run on a machine with an Intel Xeon w7-3445 CPU and 64 GB of RAM, using four CPU threads. We simulate network conditions via Linux `tc`: LAN (10 Gbps bandwidth, 0.05ms round-trip time (RTT)) and WAN (370/600 Mbps, 40/60 ms RTT). Results are averaged over 10 trials. Our replication package is available at https://github.com/songzhibo1/SecureCROWN.

Baseline. As no prior work addresses privacy-preserving robustness verification, we compare against the plaintext CROWN algorithm (Section 3.1), implemented in Python and executed single-threaded. This serves as the correctness oracle for validating numerical fidelity.

Models and Datasets. We evaluate on MNIST [LeCun et al., 1998] and CIFAR-10 [Krizhevsky, 2009] using fully-connected ReLU networks. MNIST models ($2 \times [20]$ to $7 \times [256]$) are from [Zhang et al., 2018] and VNN-COMP [Brix et al., 2023]; CIFAR-10 models ($5 \times [100]$ to $10 \times [200]$) are from ERAN [Singh et al., 2019]. For each configuration, we randomly sample 100 test images, verify those that are correctly classified, and test multiple ϵ values under ℓ_∞ perturbations—including values near the robustness boundary (mixed Robust/Unknown outcomes) and extreme values (all Robust or all Unknown).

Table 1: Fidelity analysis of SECURECROWN. $m \times [n]$: m layers, n neurons each. R: Robust ($\underline{f}_{y,j} > 0$); U: Unknown ($\underline{f}_{y,j} \leq 0$).

Data	Model	Avg. Rob.	ϵ	Result	MRE	Cons.
MNIST	$2 \times [20]$	0.0448	0.015	**All R (99)**	$\mathbf{4.45 \times 10^{-7}}$	**100%**
			0.045	**R (43)**	$\mathbf{9.52 \times 10^{-7}}$	**100%**
				U (56)	8.70×10^{-7}	**100%**
			0.1	All U (99)	3.27×10^{-7}	**100%**
	$3 \times [20]$	0.0304	0.015	**R (93)**	$\mathbf{1.41 \times 10^{-6}}$	**100%**
				U (1)	1.08×10^{-4}	**100%**
			0.030	**R (44)**	$\mathbf{7.18 \times 10^{-6}}$	**100%**
				U (50)	3.68×10^{-6}	**100%**
			0.1	All U (94)	6.85×10^{-7}	**100%**
	$3 \times [256]$	0.0159	0.015	**R (49)**	$\mathbf{6.20 \times 10^{-6}}$	**100%**
				U (51)	1.14×10^{-5}	**100%**
			0.1	All U (100)	9.29×10^{-7}	**100%**
	$5 \times [256]$	0.0149	0.015	**R (50)**	$\mathbf{7.78 \times 10^{-6}}$	**100%**
				U (50)	1.21×10^{-5}	**100%**
			0.1	All U (100)	6.16×10^{-5}	**100%**
	$7 \times [256]$	0.0132	0.015	**R (30)**	$\mathbf{2.66 \times 10^{-5}}$	**100%**
				U (69)	1.32×10^{-2}	**100%**
			0.1	All U (99)	9.18×10^{-2}	**100%**
CIFAR-10	$5 \times [100]$	0.0021	0.002	**R (24)**	$\mathbf{5.23 \times 10^{-5}}$	**100%**
				U (21)	7.78×10^{-5}	**100%**
			0.0078	All U (45)	5.65×10^{-5}	**100%**
	$7 \times [100]$	0.0013	0.001	**R (32)**	$\mathbf{2.82 \times 10^{-4}}$	**100%**
				U (14)	1.50×10^{-4}	**100%**
	$10 \times [200]$	0.0010	0.001	**R (28)**	$\mathbf{1.73 \times 10^{-3}}$	**100%**
				U (24)	1.67×10^{-4}	**100%**

7.2 RESULTS

RQ1: Fidelity. We validate SECURECROWN's correctness by measuring numerical fidelity against the plaintext base-

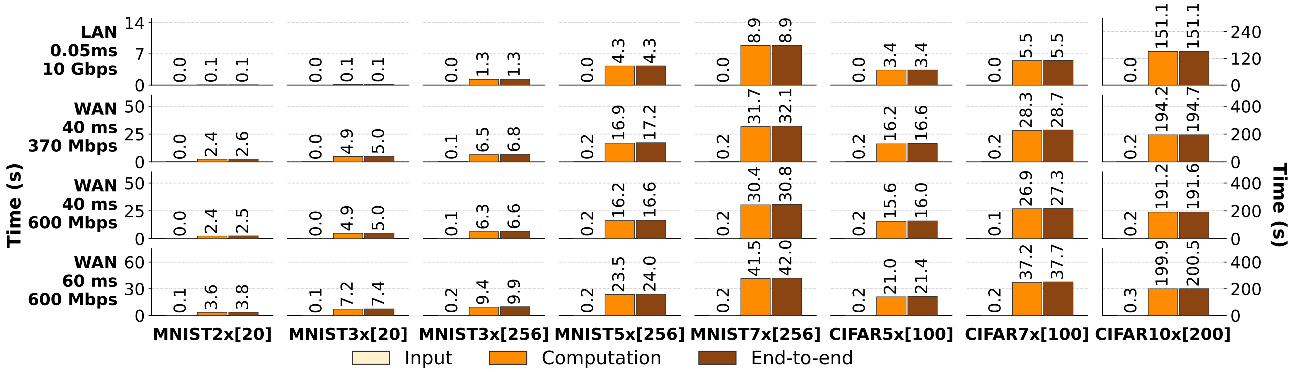

Figure 1: Online latency breakdown across network conditions and model architectures.

Table 2: Online communication volume breakdown (MB).

Dataset	Model	Communication (MB)		
		Input	Comp.	E2E
MNIST	$2 \times [20]$	0.13	0.64	0.77
	$3 \times [20]$	0.14	1.78	1.92
	$3 \times [256]$	2.07	30.16	32.23
	$5 \times [256]$	3.07	109.95	113.02
	$7 \times [256]$	4.07	234.80	238.88
CIFAR-10	$5 \times [100]$	2.63	83.76	86.39
	$7 \times [100]$	2.78	140.92	143.71
	$10 \times [200]$	7.20	596.97	604.17

line using two metrics. The first is *Mean Relative Error* (MRE): MRE $= \frac{1}{N}\sum_{i=1}^{N} |(\underline{f}_{y,j}^{\text{sec},(i)} - \underline{f}_{y,j}^{\text{plain},(i)})/\underline{f}_{y,j}^{\text{plain},(i)}|$, where $\underline{f}_{y,j}^{\text{sec}}$ and $\underline{f}_{y,j}^{\text{plain}}$ denote the SECURECROWN and plaintext certified lower bounds, respectively. The second is *Verification Consistency*, the percentage of samples with identical verification outcomes.

As shown in Table 1, SECURECROWN achieves high numerical fidelity, with MRE scaling predictably with network depth. Our evaluation specifically examines challenging cases near the robustness boundary, as reflected by the mixed Robust/Unknown outcomes in Table 1. For shallow networks (MNIST $2 \times [20]$), MRE remains at $\approx 10^{-7}$. As depth increases, cumulative rounding errors in recursive bound propagation lead to higher MRE—from 6.20×10^{-6} (MNIST $3 \times [256]$) to 1.73×10^{-3} for CIFAR-10 $10 \times [200]$. Despite this, SECURECROWN maintains **100% verification consistency** with the plaintext baseline across all 8 architectures and multiple ϵ configurations, confirming that our approach faithfully preserves the decision behavior of the plaintext verifier.

RQ2: Online Efficiency. We evaluate online efficiency by analyzing runtime latency and communication volume across network conditions and model architectures. We break down the end-to-end (E2E) execution into two phases: (1) *Input Processing*, encompassing secure input sharing and constraint setup; (2) *Computation*, the secure backward bound propagation with FSS-based nonlinear primitives. Figure 1 and Table 2 summarize the results.

Latency. As shown in Figure 1, E2E latency ranges from 0.1s to 200.5s across network conditions and model architectures. Smaller models (MNIST $2 \times [20]$ to $3 \times [256]$) complete within 0.1–9.9s, while CIFAR-10 $10 \times [200]$ requires 151.1–200.5s. The computation phase dominates total latency, with input processing contributing only 0.0–0.3s. Bandwidth variation at fixed RTT (370 vs 600 Mbps, 40ms) yields $<$2% difference, whereas increasing RTT from 40ms to 60ms incurs $\sim$5% overhead. This indicates that round-trip latency accumulates over $O(L^2)$ communication rounds, making RTT the dominant network factor. FSS evaluations are embarrassingly parallel across neurons, so exploiting more cores would further reduce latency for larger models.

Communication. Table 2 details communication volume. E2E communication scales linearly with network complexity: from 0.77 MB (MNIST $2 \times [20]$) to 604.17 MB (CIFAR-10 $10 \times [200]$). The computation phase dominates, accounting for $\approx$98.8% of total volume in the largest model. Input processing remains negligible ($\leq$7.20 MB even for CIFAR-10), confirming that *communication overhead scales with secure operations rather than input dimensions*, ensuring scalability to higher-dimensional data.

RQ3: Offline Preprocessing Overhead. As shown in Table 3, both time and storage scale with network complexity, ranging from 0.17 s / 39 MB (MNIST $2 \times [20]$) to 71.16 s / 31 GB (CIFAR-10 $10 \times [200]$). While storage costs are substantial for deep networks, the preprocessing phase is *one-time and data-independent*—generated offline without access to private inputs. In practical deployment, preprocessing material can be prepared as a batch for multiple planned verification queries; each query consumes a fresh portion of the batch during execution, amortizing the generation cost across those queries.

Table 3: Offline preprocessing overhead (one-time, data-independent).

Dataset	Model	Time (s)	Data (MB)
MNIST	$2 \times [20]$	0.17	39
	$3 \times [20]$	0.41	100
	$3 \times [256]$	4.20	1644
	$5 \times [256]$	13.93	5677
	$7 \times [256]$	29.21	11 898
CIFAR-10	$5 \times [100]$	12.03	4547
	$7 \times [100]$	19.25	7541
	$10 \times [200]$	71.16	31 060

Table 4: Sensitivity to the stability constant ϵ_s on MNIST $5\times[256]$ ($\epsilon = 0.015$).

ϵ_s	Cons.	Avg. MRE	
		R (50)	U (50)
10^{-1}	100%	5.86×10^{-2}	1.40×10^{-1}
10^{-2}	100%	6.47×10^{-3}	1.50×10^{-2}
10^{-3}	100%	6.49×10^{-4}	1.52×10^{-3}
10^{-4}	100%	7.05×10^{-5}	1.62×10^{-4}

RQ4: Sensitivity to the Stability Constant. We vary the stability constant ϵ_s (Eq. (12)) to examine its impact on verification outcomes. Specifically, ϵ_s is varied from 10^{-4} (our default) to 10^{-1} on MNIST $5\times[256]$ ($\epsilon = 0.015$). As shown in Table 4, verification consistency remains 100% across all tested values. At the same time, the reported MREs increase approximately linearly with ϵ_s, indicating that larger stability constants introduce larger numerical deviations from the plaintext bounds. On this benchmark, verification decisions remain unchanged even when ϵ_s is increased to $1000\times$ the default value.

8 CONCLUSION

We propose SecureCROWN, the first privacy-preserving framework for neural network robustness verification in a 2PC setting. Our protocol addresses the key challenges of performing linear relaxation and backward bound computation on protected model parameters and inputs. We provided formal privacy guarantees in the semi-honest adversary model and analyzed the protocol's computational complexity. Experiments on standard benchmarks validate that our method achieves verification fidelity comparable to plaintext verification with moderate cryptographic overhead, representing a promising step toward efficient privacy-preserving verification.

Acknowledgements

This work was supported by the National Science and Technology Major Project (Grant 2022ZD0115901), the University of Bristol Engineering and Physical Sciences Research Council Impact Acceleration Account, and the National Natural Science Foundation of China (Grant Nos. 92582102, 62572013, and 62172019). Nianyun Song is supported by the China Scholarship Council (Grant 202506040106). We thank the anonymous reviewers for their constructive feedback.

References

Matthias Althoff. An introduction to CORA 2015. In *1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 120–151. EasyChair, 2015.

Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology, EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367. Springer, 2015.

Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (VNN-COMP). *Int. J. Softw. Tools Technol. Transf.*, 25 (3):329–339, 2023.

Thomas Davenport and Ravi Kalakota. The potential for artificial intelligence in healthcare. *Future Healthcare Journal*, 6(2):94–98, 2019.

Hai Duong, Linhan Li, ThanhVu Nguyen, and Matthew B. Dwyer. A DPLL(T) framework for verifying deep neural networks. *CoRR*, abs/2307.10266, 2023.

Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B. Dwyer. Harnessing neuron stability to improve DNN verification. *Proc. ACM Softw. Eng.*, 1(FSE):859–881, 2024.

Hai Duong, ThanhVu Nguyen, and Matthew B. Dwyer. NeuralSAT: A high-performance verification tool for deep neural networks. In *Computer Aided Verification, CAV 2025*, volume 15932 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 2025.

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, 2019.

David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2-3):70–246, 2018.

O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proceedings of the Annual ACM Symposium on Theory of Computing, STOC 1987*, pages 218–229. ACM, 1987.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations, ICLR 2015*, 2015.

Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. SHARK: actively secure inference using function secret sharing. In *IEEE Symposium on Security and Privacy, SP 2025*, pages 2472–2490. IEEE, 2025.

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations, ICLR 2019*, 2019.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification, CAV 2017*, volume 10426 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2017.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The Marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification, CAV 2019*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

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

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical Report TR-2009, Department of Computer Science, University of Toronto, 2009.

Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In *IEEE Symposium on Security and Privacy, SP 2020*, pages 336–353. IEEE, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pages 12403–12422. PMLR, 2022.

Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network based on zk-SNARKs. *IEEE Trans. Dependable Secur. Comput.*, 21(4):4254–4270, 2024.

Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42: 60–88, 2017.

Qian Lou and Lei Jiang. HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 7102–7110. PMLR, 2021.

Mohammad Mahdi Maheri, Hamed Haddadi, and Alex Davidson. TeleSparse: Practical privacy-preserving verification of deep neural networks. *Proc. Priv. Enhancing Technol.*, 2025(4):861–880, 2025.

Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 35–52. ACM, 2018.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, 32:9835–9846, 2019.

Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2025*, volume 15696 of *Lecture Notes in Computer Science*, pages 315–335. Springer, 2025.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):1–30, 2019.

Nianyun Song, Fuyi Wang, Rongfang Bie, Yu Guo, Leo Yu Zhang, and Xiaohua Jia. ObliviGate: Towards architecture-oblivious privacy-preserving inference for malicious security. *IEEE Transactions on Services Computing*, 2026.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations, ICLR 2014*, 2014.

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

Paul Voigt and Axel Von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer International Publishing, 2017.

Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.

Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proc. Priv. Enhancing Technol.*, 2021 (1):188–208, 2021.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34: 29909–29921, 2021.

Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In *Computer Aided Verification, CAV 2024*, volume 14682 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2024.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations, ICLR 2021*, 2021.

Andrew C Yao. Protocols for secure computations. In *Proceedings of the Annual Symposium on Foundations of Computer Science, FOCS 1982*, pages 160–164. IEEE, 1982.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31:4939–4948, 2018.

Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Privacy-preserving deep learning based on multiparty secure computation: A survey. *IEEE Internet Things J.*, 8(13):10412–10429, 2021.

Appendix

Nianyun Song[1,2] **Xiaokun Luan**[3] **Yu Guo**[1] **Rongfang Bie**[1] **Meng Sun**[3] **Xiyue Zhang**[*2]

[1]School of Artificial Intelligence, Beijing Key Laboratory of Artificial Intelligence for Education, Engineering Research Center of Intelligent Technology and Educational Application (Ministry of Education), Beijing Normal University, Beijing, China
[2]School of Computer Science, University of Bristol, Bristol, UK
[3]School of Mathematical Sciences, Peking University, Beijing, China

A DETAILED PROTOCOLS

A.1 SECURE RELAXATION SLOPE COMPUTATION

Algorithm 3 details the secure protocol $\Pi_{\boldsymbol{\alpha}}$ for computing the relaxation slope vector $\boldsymbol{\alpha}^{(l)}$, directly implementing the unified arithmetic expression derived in Eq. (12) of the main text. As discussed in Section 5.3, securely evaluating the piecewise conditions of the standard ReLU relaxation introduces prohibitive communication overhead due to data-dependent branching and secure comparisons. By formulating the slope computation as a continuous, data-independent function, our protocol evaluates all neurons uniformly. The protocol first extracts the required positive bounds and the magnitudes of the negative bounds using standard SecReLU operations. After locally incorporating the public stability constant ϵ_s to prevent numerical instability, the division is securely resolved by computing the reciprocal of the denominator via SecRecip, followed by an element-wise SecMul. This branching-free design ensures deterministic execution time and minimizes the use of heavy cryptographic primitives.

Algorithm 3 $\Pi_{\boldsymbol{\alpha}}$: Secure Slope Computation

Require: Secret shares of pre-activation lower bounds $\langle \underline{\mathbf{z}}^{(l)} \rangle \in \mathbb{Z}_{2^k}^{d_l}$, upper bounds $\langle \bar{\mathbf{z}}^{(l)} \rangle \in \mathbb{Z}_{2^k}^{d_l}$, and public stability constant $\epsilon_s \in \mathbb{R}^+$

Ensure: Secret shares of the relaxation slope vector $\langle \boldsymbol{\alpha}^{(l)} \rangle \in \mathbb{Z}_{2^k}^{d_l}$

1: $\langle \mathbf{n} \rangle \leftarrow \mathsf{SecReLU}(\langle \bar{\mathbf{z}}^{(l)} \rangle)$ ▷ Numerator: $\mathrm{ReLU}(\bar{\mathbf{z}}^{(l)})$
2: $\langle \mathbf{d}_{sub} \rangle \leftarrow \mathsf{SecReLU}(-\langle \underline{\mathbf{z}}^{(l)} \rangle)$ ▷ Partial denominator: $\mathrm{ReLU}(-\underline{\mathbf{z}}^{(l)})$
3: $\langle \mathbf{d} \rangle \leftarrow \langle \mathbf{n} \rangle + \langle \mathbf{d}_{sub} \rangle + \epsilon_s$ ▷ Full denominator via local addition (zero communication)
4: $\langle \mathbf{r} \rangle \leftarrow \mathsf{SecRecip}(\langle \mathbf{d} \rangle)$ ▷ Element-wise secure reciprocal: $1/\mathbf{d}$
5: $\langle \boldsymbol{\alpha}^{(l)} \rangle \leftarrow \mathsf{SecMul}(\langle \mathbf{n} \rangle, \langle \mathbf{r} \rangle)$ ▷ Element-wise secure multiplication for division
6: **return** $\langle \boldsymbol{\alpha}^{(l)} \rangle$

A.2 SECURE INTERCEPT ACCUMULATION

Algorithm 4 details the secure protocol $\Pi_{\boldsymbol{\delta}}$ for computing the intercept contribution vectors $\underline{\boldsymbol{\delta}}^{(t-1)}$ and $\bar{\boldsymbol{\delta}}^{(t-1)}$, implementing the reformulated expressions in Eqs. (13)–(14). As discussed in Section 5.4, directly evaluating the conditional summations would necessitate expensive secure comparison protocols. By leveraging the property that the relaxation slopes satisfy $\boldsymbol{\alpha}^{(t-1)} \geq 0$, the updated coefficients $\mathbf{A}^{(t-1)}$ preserve the necessary sign information. The protocol efficiently isolates the positive and negative components of the coefficient matrix $\langle \mathbf{A}^{(t-1)} \rangle$ and the negative components of the pre-activation bounds $\langle \underline{\mathbf{z}}^{(t-1)} \rangle$ using three parallel SecReLU evaluations. The final vectors $\langle \underline{\boldsymbol{\delta}}^{(t-1)} \rangle$ and $\langle \bar{\boldsymbol{\delta}}^{(t-1)} \rangle$ are then obtained via SecMatMul.

*Correspondence to XZ (xiyue.zhang@bristol.ac.uk)

Algorithm 4 Π_{δ}: Secure Intercept Accumulation

Require: Secret shares of coefficients $\langle \mathbf{A}^{(t-1)} \rangle \in \mathbb{Z}_{2^k}^{d_l \times d_{t-1}}$ and pre-activation lower bounds $\langle \underline{\mathbf{z}}^{(t-1)} \rangle \in \mathbb{Z}_{2^k}^{d_{t-1}}$
Ensure: Secret shares of intercept contributions $\langle \underline{\boldsymbol{\delta}}^{(t-1)} \rangle, \langle \bar{\boldsymbol{\delta}}^{(t-1)} \rangle \in \mathbb{Z}_{2^k}^{d_l}$
1: $\langle \mathbf{R}_A^- \rangle \leftarrow \mathsf{SecReLU}(-\langle \mathbf{A}^{(t-1)} \rangle)$ ▷ $\mathrm{ReLU}(-\mathbf{A}^{(t-1)})$, element-wise
2: $\langle \mathbf{R}_A^+ \rangle \leftarrow \mathsf{SecReLU}(\langle \mathbf{A}^{(t-1)} \rangle)$ ▷ $\mathrm{ReLU}(\mathbf{A}^{(t-1)})$, element-wise
3: $\langle \mathbf{r}_z \rangle \leftarrow \mathsf{SecReLU}(-\langle \underline{\mathbf{z}}^{(t-1)} \rangle)$ ▷ $\mathrm{ReLU}(-\underline{\mathbf{z}}^{(t-1)})$
4: $\langle \underline{\boldsymbol{\delta}}^{(t-1)} \rangle \leftarrow -\mathsf{SecMatMul}(\langle \mathbf{R}_A^- \rangle, \langle \mathbf{r}_z \rangle)$ ▷ Eq. (13)
5: $\langle \bar{\boldsymbol{\delta}}^{(t-1)} \rangle \leftarrow \mathsf{SecMatMul}(\langle \mathbf{R}_A^+ \rangle, \langle \mathbf{r}_z \rangle)$ ▷ Eq. (14)
6: **return** $\langle \underline{\boldsymbol{\delta}}^{(t-1)} \rangle, \langle \bar{\boldsymbol{\delta}}^{(t-1)} \rangle$

B DETAILED ANALYSIS

B.1 SECURITY PROOF

Proof. We prove security via a hybrid argument. Let the verification circuit consist of L layers corresponding to the neural network's bound propagation. For each layer l, let F_l denote the FSS scheme used (either for SecReLU, SecMul, or SecRecip), with associated simulator Sim_l.

We construct simulator Sim_0 for corrupted model owner P_0; the case for data owner P_1 is symmetric.

Hybrid Definitions.

- $\mathbf{H}_0$: The simulator knows all inputs and executes the real protocol honestly. This is identical to the real execution.
- $\mathbf{H}_l$ (for $l \in \{1, \dots, L\}$): Execute the real protocol for layers $1, \dots, L-l$. For layers $L-l+1, \dots, L$, use the FSS simulator Sim_l to generate keys and simulate protocol messages.
- $\mathbf{H}_{L+1}$: Same as $\mathbf{H}_L$, but additionally replace P_1's inputs with dummy values. This is the ideal-world execution.

Indistinguishability of Adjacent Hybrids. $\mathbf{H}_l \approx_c \mathbf{H}_{l+1}$ *for* $l \in \{0, \dots, L-1\}$*:* These hybrids differ only in layer $j = L-l$. Suppose a distinguisher $\mathcal{D}$ distinguishes $\mathbf{H}_l$ from $\mathbf{H}_{l+1}$ with non-negligible advantage. We construct an adversary $\mathcal{A}$ that breaks the security of FSS scheme F_j:

1. $\mathcal{A}$ runs the protocol for layers $1, \dots, j-1$ honestly.
2. $\mathcal{A}$ receives an FSS key k_b from the FSS challenger (either real or simulated).
3. $\mathcal{A}$ uses k_b to execute layer j and simulates layers $j+1, \dots, L$.
4. $\mathcal{A}$ outputs $\mathcal{D}$'s guess.

If $\mathcal{D}$ distinguishes, then $\mathcal{A}$ breaks FSS security—a contradiction.

$\mathbf{H}_L \approx_s \mathbf{H}_{L+1}$*:* In both hybrids, P_1's inputs are masked by uniform random shares. Since additive secret sharing is information-theoretically secure, replacing real inputs with dummy inputs produces an identical distribution of shares. Thus these hybrids are *statistically* indistinguishable.

Conclusion. By the hybrid argument, $\mathbf{H}_0 \approx_c \mathbf{H}_{L+1}$, which establishes that the real and ideal executions are computationally indistinguishable. □

B.2 COMPLEXITY ANALYSIS

We analyze the theoretical complexity of SECURECROWN for an L-layer network with maximum width d. Table 5 summarizes the costs of the underlying building blocks.

Table 5: Complexity of secure building blocks. **Rounds** and **Comm.** denote online costs; **Preproc.** denotes offline costs. Parameters: bit-width $k = 64$, fractional bits $n_f = 26$, security parameter $\lambda = 128$, Newton–Raphson iterations $n_{\text{iter}} = 1$.

Primitive	Rounds	Comm. (bits)	Preproc. (bits)
SecMul	1	$2k$ (128)	$3k$ (192)
SecMatMul$_{m \times n \times p}$	1	$2(mn{+}np)k$	$(mn{+}np{+}mp)k$
SecARS	2	$k+2$ (66)	$(k{+}n_f)\lambda + O(k)$
SecReLU	2	$k+1$ (65)	$k\lambda + O(k)$
SecAbs	2	$k+1$ (65)	$k\lambda + O(k)$
SecRecip	6	$O(k)$	$O(k\lambda)$

Sub-protocol Complexity. The complexity of our specialized sub-protocols is derived as follows:

- **Protocol $\Pi_{\boldsymbol{\alpha}}$ (Algorithm 3):** This involves two parallel SecReLU operations (2 rounds), one SecRecip ($4 + 2n_{\text{iter}}$ rounds, including n_{iter} Newton–Raphson iterations), and a final multiplication (1 round). The total is $7 + 2n_{\text{iter}}$ rounds, with communication $O(dk)$.
- **Protocol $\Pi_{\boldsymbol{\delta}}$ (Algorithm 4):** This protocol executes three parallel SecReLU evaluations (2 rounds) followed by two parallel SecMatMul operations (1 round). The total is **3 rounds** with $O(d^2k)$ communication.

End-to-End Complexity. The complete SECURECROWN protocol (Algorithm 2) proceeds layer-by-layer. For a target layer l, Algorithm 1 iterates $l-1$ times. Each iteration involves: SecMatMul (1 round) $\to$ slope multiplication (1 round) $\to \Pi_{\boldsymbol{\delta}}$ (3 rounds), yielding 5 rounds per iteration. The bias accumulation in Lines 6-7 overlaps with $\Pi_{\boldsymbol{\delta}}$: the SecMatMul$(\langle \mathbf{A} \rangle, \langle \mathbf{b} \rangle)$ executes in parallel with the SecReLU operations, and the final addition of $\boldsymbol{\delta}$ is a local operation.

Summing over all layers, the total complexity is:

- **Rounds:** $O(L^2)$. The iterative bound computation contributes $\sum_{l=2}^{L-1} 5(l-1) = \frac{5(L-2)(L-1)}{2} \approx \frac{5}{2}L^2$ rounds. The final margin certification adds $5(L-1) = O(L)$ rounds. Additionally, $\Pi_{\boldsymbol{\alpha}}$ is invoked $L-1$ times, contributing $(L-1)(7+2n_{\text{iter}})$ rounds.
- **Communication:** $O(L^2d^2k)$. This is dominated by the dense matrix multiplications in the iterative backward pass.
- **Computation:** $O(L^2d^3)$, dominated by $O(L^2)$ matrix multiplications of size $d \times d$.
- **Preprocessing:** $O(L^2d^2k)$ bits for matrix Beaver triples and $O(Ldk\lambda)$ bits for FSS keys.

B.3 ERROR ANALYSIS

We analyze the LBP bound computation, which constructs an affine form bound by backward propagation of a coefficient matrix $\mathbf{A}$ and a bias vector $\mathbf{c}$, accumulating constants and relaxation terms layer by layer, and then forms the final upper/lower bound of the target quantity as a function of the input. The final step is to maximize/minimize this affine form over the input perturbation set $\mathcal{X}$, which yields closed-form expressions for the bounds.

We denote by $r(\mathbf{M})$ the vector of row-wise sums of absolute values of a matrix M, i.e., $r(\mathbf{M})_i = \sum_j |\mathbf{M}_{ij}|$. We denote by $\|\mathbf{M}\|_\infty = \max_i r(\mathbf{M})_i$ the induced infinity norm of a matrix $\mathbf{M}$, and by $\|\mathbf{v}\|_1 = \sum_i |v_i|$ the ℓ_1 norm of a vector $\mathbf{v}$. Let $Q_{n_f} : \mathbb{R} \to \mathbb{R}$ denote the fixed-point quantization function that maps a real number to its fixed-point representation with n_f fractional bits, i.e., $Q_{n_f}(y) = 2^{-n_f}\lfloor 2^{n_f} y \rfloor$.

For each layer i, the algorithm uses a diagonal matrix of slopes $\mathrm{diag}(\boldsymbol{\alpha}^{(i)})$ with entries $\alpha_k^{(i)} \in [0, 1]$ to relax the ReLU activation, applied during backward propagation. For a fixed fractional precision parameter n_f, let $\mathrm{ARS}_{n_f}(\cdot)$ denote the arithmetic-right-shift scaling operation that divides its input by 2^{n_f} and rounds towards $-\infty$ (i.e., takes the floor). *We model each call to ARS_{n_f} as introducing an additive element-wise rounding error bounded by a constant ε_q in real units*, where typically $\varepsilon_q = 2^{-n_f}$.

Assumptions Our analysis makes the following assumptions:

- **Assumption 1 (Range safety)**: During execution, every intermediate integer value remains within the representable range of the fixed-point format, i.e., no overflow occurs. Equivalently, the fixed-point computation is a faithful representation of the corresponding real arithmetic plus truncation errors.
- **Assumption 2 (Quantization error model)**: Each call to ARS_{n_f} corresponds to applying Q_{n_f} to the intended real-valued quantity, and thus introduces an additive error bounded by ε_q in real units:

$$Q_{n_f}(y) = y + \eta(y) \quad \text{with} \quad |\eta(y)| \leq \varepsilon_q.$$

Backward propagation operator For the target bound computation, define the ideal backward propagation of coefficient matrices $\{\mathbf{A}^{(i)}\}$ by

$$\mathbf{A}^{(i-1)} = \mathbf{A}^{(i)}\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)}), \quad i = 1, \dots, m,$$

where m is the number of backward steps performed by the algorithm (e.g., $m = L - 1$ for the full backward propagation from the last layer). The starting coefficient $\mathbf{A}^{(m)}$ is initialized as the weight matrix of the current layer.

Let $\{\mathbf{\Lambda}^{(i)}\}$ be the corresponding coefficients computed by the fixed-point implementation, which applies the ARS_{n_f} operation at each multiplication step. The coefficient error at layer i is defined as $\mathbf{E}^{(i)} = \mathbf{\Lambda}^{(i)} - \mathbf{A}^{(i)}$.

Theorem B.1. *Under assumptions 1 and 2, given the slopes $\boldsymbol{\alpha}^{(i)} \in [0, 1]$ element-wise, the coefficient error satisfies the recurrence*

$$\|\mathbf{E}^{(i-1)}\|_\infty \leq \|\mathbf{W}^{(i)}\|_\infty \|\mathbf{E}^{(i)}\|_\infty + \Delta_i, \quad i = 1, \dots, m, \tag{15}$$

where Δ_i is an upper bound on the ℓ_∞ norm of the local rounding injection at step i, induced by the truncations performed at that step. In particular, if step i produces a matrix with q_i columns before truncation, then one may take the conservative bound $\Delta_i \leq q_i \varepsilon_q$. Consequently, if the initial coefficient is computed exactly (i.e., $\mathbf{E}^{(m)} = \mathbf{0}$), then the final coefficient error after m steps is bounded by

$$\|\mathbf{E}^{(0)}\|_\infty \leq \sum_{i=1}^{m} \left(\prod_{j=1}^{i-1} \|\mathbf{W}^{(j)}\|_\infty \right) \Delta_i. \tag{16}$$

If, furthermore, $\|\mathbf{W}^{(i)}\|_\infty \leq \rho$ for all relevant layers, then the error bound simplifies to

$$\|\mathbf{E}^{(0)}\|_\infty \leq \left(\sum_{k=0}^{m-1} \rho^k \right) \max_i \Delta_i = \begin{cases} O(m) \cdot \max_i \Delta_i, & \text{if } \rho = 1, \\ O(\rho^m) \cdot \max_i \Delta_i, & \text{if } \rho > 1, \\ O(1) \cdot \max_i \Delta_i, & \text{if } \rho < 1. \end{cases} \tag{17}$$

Proof. Fix $i \in \{1, \dots, m\}$. The ideal update is $\mathbf{A}^{(i-1)} = \mathbf{A}^{(i)}\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})$, while the actual update is

$$\mathbf{\Lambda}^{(i-1)} = Q_{n_f}(\mathbf{\Lambda}^{(i)}\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})), \tag{18}$$

where we have used the fact that $\alpha_k^{(i-1)}$ is computed to lie in $[0, 1]$ and treated it as the same slope vector used in both the ideal and actual computations. If one wishes to include quantized $\alpha_k^{(i-1)}$ in the error analysis, it appears as an additional additive term.

By Assumption 2, we have

$$\mathbf{\Lambda}^{(i-1)} = \mathbf{\Lambda}^{(i)}\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)}) + \mathbf{\Xi}^{(i)}, \tag{19}$$

where $\mathbf{\Xi}^{(i)}$ is the rounding error matrix injected by the ARS_{n_f} operation, with $|\mathbf{\Xi}_{j,k}^{(i)}| \leq \varepsilon_q$ for all entries. Therefore,

$$\mathbf{E}^{(i-1)} = \mathbf{\Lambda}^{(i-1)} - \mathbf{A}^{(i-1)} = (\mathbf{\Lambda}^{(i)} - \mathbf{A}^{(i)})\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)}) + \mathbf{\Xi}^{(i)} = \mathbf{E}^{(i)}\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)}) + \mathbf{\Xi}^{(i)}. \tag{20}$$

Taking $\|\cdot\|_\infty$ on both sides and using the sub-multiplicative property gives

$$\|\mathbf{E}^{(i-1)}\|_\infty \leq \|\mathbf{E}^{(i)}\|_\infty \|\mathbf{W}^{(i)}\|_\infty \|\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})\|_\infty + \|\mathbf{\Xi}^{(i)}\|_\infty. \tag{21}$$

Since $\alpha_k^{(i-1)} \in [0, 1]$ for all k, we have $\|\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})\|_\infty = \max_k \alpha_k^{(i-1)} \leq 1$, hence

$$\|\mathbf{E}^{(i-1)}\|_\infty \leq \|\mathbf{W}^{(i)}\|_\infty \|\mathbf{E}^{(i)}\|_\infty + \|\mathbf{\Xi}^{(i)}\|_\infty. \tag{22}$$

Define $\Delta_i = \|\boldsymbol{\Xi}^{(i)}\|_\infty$ to be the local error bound at step i, yielding the stated recurrence.

To bound Δ_i, note that $\|\boldsymbol{\Xi}^{(i)}\|_\infty = \max_j \sum_k |\boldsymbol{\Xi}^{(i)}_{j,k}| \leq \max_j \sum_k \varepsilon_q = q_i \varepsilon_q$, where q_i is the number of columns of the matrix.

Finally, unrolling the recurrence with $\mathbf{E}^{(m)} = \mathbf{0}$ gives the stated bound on $\mathbf{E}^{(0)}$:

$$
\begin{aligned}
\|\mathbf{E}^{(0)}\|_\infty &\leq \|\mathbf{W}^{(1)}\|_\infty \|\mathbf{E}^{(1)}\|_\infty + \Delta_1 \\
&\leq \|\mathbf{W}^{(1)}\|_\infty (\|\mathbf{W}^{(2)}\|_\infty \|\mathbf{E}^{(2)}\|_\infty + \Delta_2) + \Delta_1 \\
&= \|\mathbf{W}^{(1)}\|_\infty \|\mathbf{W}^{(2)}\|_\infty \|\mathbf{E}^{(2)}\|_\infty + \|\mathbf{W}^{(1)}\|_\infty \Delta_2 + \Delta_1 \\
&\cdots \\
&\leq \sum_{i=1}^{m} \left(\prod_{j=1}^{i-1} \|\mathbf{W}^{(j)}\|_\infty \right) \Delta_i.
\end{aligned}
$$

The ρ-based bound follows directly from the above by substituting $\|\mathbf{W}^{(j)}\|_\infty \leq \rho$ for all j. $\square$

Based on Theorem B.1, we now derive the error bound for the final upper/lower bound vectors. Write the ideal final bounds over $\mathcal{X}$ as

$$UB = \mathbf{A}^{(0)}\mathbf{x}_0 + \overline{\mathbf{c}}^{(0)} + \epsilon \cdot r(\mathbf{A}^{(0)}), \quad LB = \mathbf{A}^{(0)}\mathbf{x}_0 + \underline{\mathbf{c}}^{(0)} - \epsilon \cdot r(\mathbf{A}^{(0)}),$$

where $r(\mathbf{A}^{(0)})$ is the row-wise ℓ_1-norm vector of $\mathbf{A}^{(0)}$.

During backward propagation, the ideal update is

$$\overline{\mathbf{c}}^{(i-1)} = \overline{\mathbf{c}}^{(i)} + \mathbf{A}^{(i)}\mathbf{b}^{(i)} + \overline{\boldsymbol{\delta}}^{(i-1)}, \tag{23}$$

$$\underline{\mathbf{c}}^{(i-1)} = \underline{\mathbf{c}}^{(i)} + \mathbf{A}^{(i)}\mathbf{b}^{(i)} + \underline{\boldsymbol{\delta}}^{(i-1)}, \tag{24}$$

where $\overline{\boldsymbol{\delta}}^{(i-1)}$ and $\underline{\boldsymbol{\delta}}^{(i-1)}$ are the ReLU relaxation intercept contribution terms. Define $u_k^{(i-1)} = (-\underline{z}_k^{(i-1)})_+ \geq 0$ for all k, where $(x)_+ = \max(0, x)$ denotes the positive part, with the convention $u_k^{(0)} = 0$ since layer 0 is the input layer with no ReLU. Then the intercept contributions are:

$$\overline{\boldsymbol{\delta}}_j^{(i-1)} = \sum_k (\mathbf{A}_{j,k}^{(i-1)})_+ u_k^{(i-1)}, \tag{25}$$

$$\underline{\boldsymbol{\delta}}_j^{(i-1)} = \sum_k (-\mathbf{A}_{j,k}^{(i-1)})_+ u_k^{(i-1)}. \tag{26}$$

The actual update performed by the fixed-point implementation is

$$\overline{\boldsymbol{\gamma}}^{(i-1)} = Q_{n_f}(\overline{\boldsymbol{\gamma}}^{(i)} + \boldsymbol{\Lambda}^{(i)}\mathbf{b}^{(i)} + \hat{\overline{\boldsymbol{\delta}}}^{(i-1)}), \tag{27}$$

$$\underline{\boldsymbol{\gamma}}^{(i-1)} = Q_{n_f}(\underline{\boldsymbol{\gamma}}^{(i)} + \boldsymbol{\Lambda}^{(i)}\mathbf{b}^{(i)} + \hat{\underline{\boldsymbol{\delta}}}^{(i-1)}), \tag{28}$$

where $\hat{\overline{\boldsymbol{\delta}}}^{(i-1)}$ and $\hat{\underline{\boldsymbol{\delta}}}^{(i-1)}$ are computed using the actual coefficient matrix $\boldsymbol{\Lambda}^{(i-1)}$:

$$\hat{\overline{\boldsymbol{\delta}}}_j^{(i-1)} = \sum_k (\boldsymbol{\Lambda}_{j,k}^{(i-1)})_+ u_k^{(i-1)}, \tag{29}$$

$$\hat{\underline{\boldsymbol{\delta}}}_j^{(i-1)} = \sum_k (-\boldsymbol{\Lambda}_{j,k}^{(i-1)})_+ u_k^{(i-1)}. \tag{30}$$

Therefore, the error in the bias term at layer $i-1$, denoted as $\overline{\boldsymbol{\nu}}^{(i-1)} := \overline{\boldsymbol{\gamma}}^{(i-1)} - \overline{\mathbf{c}}^{(i-1)}$, can be decomposed component-wise (for index j) as

$$
\begin{aligned}
\overline{\boldsymbol{\nu}}_j^{(i-1)} &= \overline{\boldsymbol{\gamma}}_j^{(i)} + [\boldsymbol{\Lambda}^{(i)}\mathbf{b}^{(i)}]_j + \hat{\overline{\boldsymbol{\delta}}}_j^{(i-1)} + \Theta_j^{(i)} - \left(\overline{\mathbf{c}}_j^{(i)} + [\mathbf{A}^{(i)}\mathbf{b}^{(i)}]_j + \overline{\boldsymbol{\delta}}_j^{(i-1)} \right) \\
&= \overline{\boldsymbol{\nu}}_j^{(i)} + [\mathbf{E}^{(i)}\mathbf{b}^{(i)}]_j + \sum_k \Big[(\boldsymbol{\Lambda}_{j,k}^{(i-1)})_+ - (\mathbf{A}_{j,k}^{(i-1)})_+ \Big] u_k^{(i-1)} + \Theta_j^{(i)},
\end{aligned} \tag{31}
$$

where $\Theta^{(i)}$ is the rounding error injected by the ARS_{n_f} operation, with $\|\Theta^{(i)}\|_\infty \le \varepsilon_q$. Using the 1-Lipschitz property of $(\cdot)_+$, we can bound the error in the intercept contribution:

$$|\hat{\overline{\boldsymbol{\delta}}}_j^{(i-1)} - \overline{\boldsymbol{\delta}}_j^{(i-1)}| \le \sum_k |(\mathbf{\Lambda}_{j,k}^{(i-1)})_+ - (\mathbf{A}_{j,k}^{(i-1)})_+| u_k^{(i-1)} \le \sum_k |\mathbf{E}_{j,k}^{(i-1)}| u_k^{(i-1)} \le \|\mathbf{E}_{j,:}^{(i-1)}\|_1 \|u^{(i-1)}\|_\infty. \tag{32}$$

Taking the infinity norm over j, we get:

$$\|\hat{\overline{\boldsymbol{\delta}}}^{(i-1)} - \overline{\boldsymbol{\delta}}^{(i-1)}\|_\infty \le \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty. \tag{33}$$

Unrolling the recurrence for $\overline{\boldsymbol{\nu}}^{(0)}$ (with $\overline{\boldsymbol{\nu}}^{(m)} = \mathbf{0}$) gives

$$\|\overline{\boldsymbol{\nu}}^{(0)}\|_\infty = \|\overline{\boldsymbol{\gamma}}^{(0)} - \overline{\mathbf{c}}^{(0)}\|_\infty \le \sum_{i=1}^{m} \Big(\|\mathbf{E}^{(i)}\|_\infty \|\mathbf{b}^{(i)}\|_\infty + \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty \Big) + m\varepsilon_q. \tag{34}$$

A symmetric bound holds for the lower bound bias error $\|\underline{\boldsymbol{\nu}}^{(0)}\|_\infty = \|\underline{\boldsymbol{\gamma}}^{(0)} - \underline{\mathbf{c}}^{(0)}\|_\infty$ (replacing $(\cdot)_+$ by $(-\cdot)_+$ throughout).

Theorem B.2. *Let $\widehat{UB}, \widehat{LB}$ be the fixed-point outputs, and define $\mathbf{E}^{(0)} = \mathbf{\Lambda}^{(0)} - \mathbf{A}^{(0)}$. Under Assumptions 1 and 2, the absolute errors of the final bounds are bounded by:*

$$\|\widehat{UB} - UB\|_\infty \le (\|\mathbf{x}_0\|_\infty + \epsilon)\|\mathbf{E}^{(0)}\|_\infty + \sum_{i=1}^{m} \Big(\|\mathbf{E}^{(i)}\|_\infty \|\mathbf{b}^{(i)}\|_\infty + \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty \Big) + \zeta_u, \tag{35}$$

$$\|\widehat{LB} - LB\|_\infty \le (\|\mathbf{x}_0\|_\infty + \epsilon)\|\mathbf{E}^{(0)}\|_\infty + \sum_{i=1}^{m} \Big(\|\mathbf{E}^{(i)}\|_\infty \|\mathbf{b}^{(i)}\|_\infty + \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty \Big) + \zeta_l, \tag{36}$$

where $u^{(i-1)} = (-\underline{\mathbf{z}}^{(i-1)})_+ \ge 0$ with $u^{(0)} = 0$ (no ReLU at the input layer), and ζ_u, ζ_l are assembly-level quantization residuals bounded by $(m+1)\varepsilon_q$. Therefore, by substituting Theorem B.1 bounds for $\|\mathbf{E}^{(i)}\|_\infty$, one obtains an explicit depth-dependent absolute error bound for both $\widehat{UB}$ and $\widehat{LB}$.

Proof. We prove the upper-bound case; the lower-bound case is identical. The ideal final upper bound is given by

$$UB = \mathbf{A}^{(0)}\mathbf{x}_0 + \overline{\mathbf{c}}^{(0)} + \epsilon \cdot r(\mathbf{A}^{(0)}).$$

The fixed-point implementation computes the final bound as

$$\widehat{UB} = \mathbf{\Lambda}^{(0)}\mathbf{x}_0 + \overline{\boldsymbol{\gamma}}^{(0)} + \epsilon \cdot r(\mathbf{\Lambda}^{(0)}) + \xi_u,$$

where ξ_u represents the rounding error introduced during the final assembly operations (e.g., dot products and additions), with $\|\xi_u\|_\infty \le \varepsilon_q$. Decomposing the difference yields

$$\widehat{UB} - UB = \mathbf{E}^{(0)}\mathbf{x}_0 + \epsilon\big(r(\mathbf{\Lambda}^{(0)}) - r(\mathbf{A}^{(0)})\big) + \overline{\boldsymbol{\nu}}^{(0)} + \xi_u.$$

Taking the infinity norm and applying the triangle inequality gives

$$\|\widehat{UB} - UB\|_\infty \le \|\mathbf{E}^{(0)}\mathbf{x}_0\|_\infty + \epsilon\|r(\mathbf{\Lambda}^{(0)}) - r(\mathbf{A}^{(0)})\|_\infty + \|\overline{\boldsymbol{\nu}}^{(0)}\|_\infty + \|\xi_u\|_\infty.$$

By norm inequalities, we have

$$\|\mathbf{E}^{(0)}\mathbf{x}_0\|_\infty \le \|\mathbf{E}^{(0)}\|_\infty \|\mathbf{x}_0\|_\infty, \quad \|r(\mathbf{\Lambda}^{(0)}) - r(\mathbf{A}^{(0)})\|_\infty \le \|\mathbf{E}^{(0)}\|_\infty.$$

Substituting the bound for $\|\overline{\boldsymbol{\nu}}^{(0)}\|_\infty$ derived earlier, we obtain

$$\begin{aligned}\|\widehat{UB} - UB\|_\infty &\le (\|\mathbf{x}_0\|_\infty + \epsilon)\|\mathbf{E}^{(0)}\|_\infty + \sum_{i=1}^{m} \Big(\|\mathbf{E}^{(i)}\|_\infty \|\mathbf{b}^{(i)}\|_\infty + \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty \Big) + m\varepsilon_q + \varepsilon_q \\ &= (\|\mathbf{x}_0\|_\infty + \epsilon)\|\mathbf{E}^{(0)}\|_\infty + \sum_{i=1}^{m} \Big(\|\mathbf{E}^{(i)}\|_\infty \|\mathbf{b}^{(i)}\|_\infty + \|\mathbf{E}^{(i-1)}\|_\infty \|u^{(i-1)}\|_\infty \Big) + \zeta_u,\end{aligned}$$

where $\zeta_u \le (m+1)\varepsilon_q$. $\square$

Remark 1. The stability constant ϵ_s in Eq. (12) perturbs each relaxation slope from its ideal value, and hence the relaxation bounding functions. For every neuron k, the sup-norm deviation of the perturbed bounding functions over $[\underline{z}_k, \bar{z}_k]$ is at most ϵ_s: for unstable neurons this follows from $\max(|\underline{z}_k|, \bar{z}_k) \le \bar{z}_k - \underline{z}_k$; the active case gives deviation $\epsilon_s \bar{z}_k/(\bar{z}_k + \epsilon_s) \le \epsilon_s$, and the inactive case is exact. Substituting the perturbed relaxations in the backward recursion therefore injects, at each step i, an additional deviation of at most $\|\hat{\mathbf{A}}^{(i)}\|_\infty\, \epsilon_s$ into the accumulated bound values, where $\hat{\mathbf{A}}^{(i)}$ is the intermediate coefficient matrix in Eq. (4) and, as in the proof of Theorem B.1, the relaxation parameters at the remaining layers are treated as fixed. The effect on the final bounds thus follows the same depth-dependent structure as Theorem B.2, with the per-step injection scale ε_q replaced by $\|\hat{\mathbf{A}}^{(i)}\|_\infty\, \epsilon_s$. Since the perturbed relaxation deviates from the ideal one, a verification outcome could in principle change when the certified margin lies within $O(\epsilon_s)$ of zero; empirically, no such change is observed even at $1000\times$ the default ϵ_s (Table 4). A slope–intercept co-adjustment that restores provable soundness of the perturbed relaxation is left for future work.

Corollary B.3 (Asymptotic error bound)**.** *Under the hypotheses of Theorems B.1 and B.2, assume $\mathbf{E}^{(m)} = \mathbf{0}$, $\|\mathbf{W}^{(i)}\|_\infty \le \rho$ and $\Delta_i \le \Delta$ for all i. Let*

$$X \;=\; \|\mathbf{x}_0\|_\infty + \epsilon, \qquad B \;=\; \max_i \|\mathbf{b}^{(i)}\|_\infty, \qquad U \;=\; \max_i \|u^{(i-1)}\|_\infty,$$

and let $S_n = \sum_{k=0}^{n-1} \rho^k$, so that the layer-$t$ coefficient error satisfies $\|\mathbf{E}^{(t)}\|_\infty \le S_{m-t}\,\Delta$. Then, as $m \to \infty$, the error in $\widehat{UB}$ (and identically in $\widehat{LB}$) satisfies

$$\|\widehat{UB} - UB\|_\infty \;\le\; \Big[X \cdot S_m \;+\; B \sum_{i=1}^{m} S_{m-i} \;+\; U \sum_{i=1}^{m} S_{m-i+1}\Big]\Delta \;+\; O(m)\,\varepsilon_q, \tag{37}$$

where the three asymptotic regimes of the leading term (as a function of m) are:

$$\|\widehat{UB} - UB\|_\infty \;=\; \begin{cases} O(m)\cdot(B+U)\Delta \;+\; O(1)\cdot X\Delta \;+\; O(m)\,\varepsilon_q, & \text{if } \rho < 1, \\ O(m^2)\cdot(B+U)\Delta \;+\; O(m)\cdot X\Delta \;+\; O(m)\,\varepsilon_q, & \text{if } \rho = 1, \\ O(\rho^m)\cdot(X+B+U)\Delta \;+\; O(m)\,\varepsilon_q, & \text{if } \rho > 1. \end{cases} \tag{38}$$

In particular, the ζ residuals are $O(m)\,\varepsilon_q$ in all cases.

Interpretation of the unstable regime. The $\rho > 1$ branch in the asymptotic corollary should be read as a conservative worst-case envelope rather than as the typical numerical behavior of SECURECROWN. It is obtained by replacing every layer-dependent amplification factor with the same uniform upper bound ρ and every local rounding injection with the same upper bound Δ. The non-asymptotic analysis above is sharper and remains layer-wise. Indeed, before the conservative inequality $\|\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})\|_\infty \le 1$ is applied, the proof of Theorem B.1 gives

$$\|\mathbf{E}^{(i-1)}\|_\infty \le \kappa_i \|\mathbf{E}^{(i)}\|_\infty + \Delta_i, \qquad \kappa_i := \|\mathbf{W}^{(i)}\mathrm{diag}(\boldsymbol{\alpha}^{(i-1)})\|_\infty. \tag{39}$$

Thus, when $\mathbf{E}^{(m)} = \mathbf{0}$,

$$\|\mathbf{E}^{(0)}\|_\infty \le \sum_{i=1}^{m} \left(\prod_{j=1}^{i-1} \kappa_j\right) \Delta_i. \tag{40}$$

This expression propagates each local error through the actual product of effective layer norms. Consequently, the simplified $O(\rho^m)$ term arises only when these heterogeneous products are upper-bounded by the same worst-case factor at every layer; factors $\kappa_j < 1$ can attenuate errors produced at other layers with $\kappa_j > 1$. The diagonal relaxation matrices further reduce the effective amplification: since $\alpha_k^{(i-1)} \in [0,1]$, the factor κ_i is a row-wise weighted sum of $|\mathbf{W}^{(i)}|$ with weights $\boldsymbol{\alpha}^{(i-1)}$, and is always at most $\|\mathbf{W}^{(i)}\|_\infty$. When many slopes are strictly below one, as for inactive or unstable ReLU neurons, κ_i can be substantially smaller than the raw weight norm of the layer and can even fall below one when $\|\mathbf{W}^{(i)}\|_\infty > 1$.

We do not rule out the vulnerability captured by this worst-case bound: if a network has consistently large effective layer norms, fixed-point rounding errors can be amplified and the numerical precision of the certified bounds can deteriorate. In standard modern training pipelines, however, small initialization, weight decay, and related regularization or normalization techniques typically limit uncontrolled growth of weight norms. Consistent with this interpretation, our experiments do not exhibit the exponential-growth regime: SECURECROWN maintains 100% verification consistency with plaintext CROWN across all evaluated architectures and perturbation settings, including near-boundary instances (Table 1).

C DISCUSSION ON MALICIOUS SECURITY

SECURECROWN currently targets the semi-honest model, which is standard for initial 2PC-based protocol designs. A malicious-secure upgrade could replace unauthenticated Beaver triples with authenticated triples and add MAC-based consistency checks, following SPDZ-style protocols. The trusted-dealer assumption can also be removed by using two-party preprocessing, such as OT-based triple generation, without changing the online verifier logic. These changes preserve the branch-free reformulation of CROWN, but increase both online communication and offline preprocessing costs.

To estimate this overhead, we implemented the same verification algorithm in MP-SPDZ. On MNIST $5\times[256]$, replacing `semi2k` with `spdz2k` increases online time by about $3.2\times$ (183s to 586s) and communication by about $8.7\times$ (1.26TB to 10.88TB). In comparison, our FSS-based semi-honest implementation completes the same task in 4.3s with 113MB communication. This suggests that a specialized malicious-secure variant of SECURECROWN could be substantially more efficient than directly using a generic malicious 2PC backend, although designing and benchmarking such a variant is left for future work.

D EXTENSION TO OTHER NETWORK ARCHITECTURES

The current implementation targets CROWN-style LBP for fully connected ReLU networks. We discuss extensions to other network components according to the type of additional support required.

D.1 COMPONENTS REQUIRING MINIMAL CRYPTOGRAPHIC CHANGES

Batch normalization. Batch normalization (BN) can be folded into the preceding affine layer by P_0 before secret sharing, since all BN parameters are fixed after training. This requires no change to the secure verification protocol.

Residual connections. Residual connections compute $\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$, where $F(\mathbf{x})$ denotes the residual branch. From the secure-computation perspective, this addition is local under additive secret sharing and incurs no communication. Therefore, residual connections require no new cryptographic primitives, although the verifier implementation would need to handle skip connections and merge the propagated bounds from multiple graph branches.

D.2 LINEAR COMPONENTS WITH SCALABILITY BOTTLENECKS

Convolutional layers. Convolutions can be represented as matrix multiplications $\mathbf{y} = W_{\text{toep}}\mathbf{x} + \mathbf{b}$, where $\mathbf{x}$ and $\mathbf{y}$ denote vectorized input and output feature maps, and W_{toep} is the sparse Toeplitz matrix induced by the convolutional kernels. Therefore, convolutions are compatible with our `SecMatMul` interface in principle. However, a naive dense unfolding loses the sparsity of the convolutional kernel. For input and output channel counts c_{in} and c_{out}, input and output spatial sizes s and s', and kernel size κ, the unfolded matrix has dimension $(c_{\text{out}}s'^2)\times(c_{\text{in}}s^2)$, whereas the convolution itself has only $c_{\text{out}}c_{\text{in}}\kappa^2$ independent kernel parameters. Since our Beaver-based `SecMatMul` treats the unfolded matrix as dense, communication scales with the full matrix dimensions rather than with the number of independent kernel parameters. Practical support for CNNs would therefore require secure convolution protocols that exploit kernel sparsity or tiled/sliding-window structure.

Cost projection for ResNet-18. For a CIFAR-10-scale ResNet-18, a naive Toeplitz unfolding of a 64-channel 32×32 convolution can yield matrices up to $65{,}536 \times 65{,}536$. Repeated dense secure matrix multiplications inside LBP would therefore lead to communication on the order of terabytes over the full verification procedure. Exploiting the sparse Toeplitz structure, as done in optimized secure-inference systems such as CryptGPU [Tan et al., 2021] and CrypTFlow [Kumar et al., 2020], is a promising direction for reducing this cost.

D.3 COMPONENTS REQUIRING NEW VERIFICATION RELAXATIONS

Non-ReLU activations. Non-ReLU activations, such as sigmoid or tanh, require activation-specific linear relaxations. Extending SecureCROWN to these activations would therefore require new branch-free reformulations of the corresponding relaxation logic. This does not necessarily require new cryptographic primitives: the underlying FSS primitives already support operations such as spline and exponential evaluations [Gupta et al., 2025], providing a cryptographic foundation for these extensions. The main challenge is deriving secure-computation-friendly bound propagation rules that minimize secure comparisons and communication overhead.

Attention layers. Attention layers compute $\mathrm{softmax}(QK^\top/\sqrt{d_k})V$, where Q, K, and V denote the query, key, and value matrices, respectively, and d_k is the key dimension. This operation involves bilinear terms in $QK^\top$ and the softmax nonlinearity. Unlike ReLU, which is a univariate piecewise-linear function, attention introduces nonlinearities whose tight and scalable verification remains substantially more complex even in plaintext verification [Shi et al., 2025]. Extending SECURECROWN to attention-based architectures would therefore first require mature CROWN-style relaxations for these operations.