Range-Arithmetic: Verifiable Deep Learning Inference on an Untrusted Party

Ali Rahimi[†], **Babak Khalaj**[†], **Mohammad Ali Maddah-Ali**[*]

[†] Department of Electrical Engineering, Sharif University of Technology
[*] Department of Electrical Engineering, University of Minnesota

Abstract

Verifiable computing (VC) has gained prominence in decentralized machine learning systems, where resource-intensive tasks like deep neural network (DNN) inference are offloaded to external participants due to blockchain limitations. This creates a need to verify the correctness of outsourced computations without re-execution. We propose Range-Arithmetic, a novel framework for efficient and verifiable DNN inference that transforms non-arithmetic operations, such as rounding after fixed-point matrix multiplication and ReLU, into arithmetic steps verifiable using sum-check protocols and concatenated range proofs. Our approach avoids the complexity of Boolean encoding, high-degree polynomials, and large lookup tables while remaining compatible with finite-field-based proof systems. Experimental results show that our method not only matches the performance of existing approaches, but also reduces the computational cost of verifying the results, the computational effort required from the untrusted party performing the DNN inference, and the communication overhead between the two sides.

1 Introduction

By leveraging blockchain technology for transparent coordination, incentives, and auditability, decentralized machine learning (ML) systems enable participants to collaboratively train or infer using models without relying on centralized infrastructure. This shift not only enhances scalability and fault tolerance, but also lays the groundwork for trustworthy and autonomous AI applications across open and dynamic environments.

A core challenge in decentralized ML systems is ensuring the integrity of computations that are outsourced to external, and potentially untrusted, executors. When decentralized ML systems delegate heavy ML tasks, such as inference or training, to off-chain workers, they must be able to trust the correctness of the results without re-executing the entire computation, which is often prohibitively expensive. This need has led to the application of verifiable computing (VC) techniques in this domain.

VC techniques enable a *verifier* to confirm the correctness of a computation performed by an untrusted *prover* without re-executing the computation. The three main metrics for evaluating a verifiable computing algorithm are: the size of the communication (or *proof*), the computational cost for the verifier, and the prover's effort in generating the proof. Interactive proofs (IPs) are one of the promising approaches to verifiable computing [23, 14, 4]. As illustrated in Figure 1, in this approach, the prover sends a claimed result which the verifier challenges over several rounds. The verifier can detect, with high probability, if the claimed result is incorrect, whether due to a discrepancy between the model computed by the prover and the model expected by the verifier, errors during computation, or intentional misrepresentation by the prover.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

VC techniques have gained significant attention from both industry and academia, particularly for secure, trustless, and verifiable outsourced machine learning computation. Notable initiatives include EZKL [9], Polygon zkEVM [20], Accountable Magic [1], zkAGI [28], Noya [17], ZKML Systems [29], and Provably AI [21]. Recent academic work has also contributed with surveys and protocols focusing on *verifiable ML* [19, 26, 16, 15, 25].

One major critique of VC schemes is that they operate over finite fields, often integers modulo a large prime p. Furthermore, these schemes are generally restricted to arithmetic computations, that is, computations expressible as compositions of additions and multiplications over the chosen field. However, many widely used operations in ML computations, such as fixed-point arithmetic and non-linear activation functions like ReLU, do not naturally conform to this arithmetic model, making them difficult to represent and verify efficiently within conventional VC frameworks.

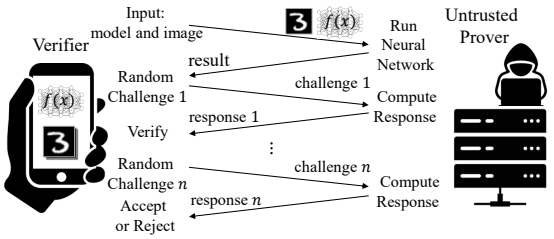

Figure 1: A schematic overview of general interactive verifiable computing, where the goal is to compute a function $f(\cdot)$ at input x (e.g., performing inference using model $f(\cdot)$ on input x). The prover executes the computation and generates a proof of correctness by responding to challenges posed by the verifier.

To address the above challenge, various solutions have been explored in the literature. Mystique [24] transforms computations over fixed-point and floating-point numbers into bitwise operations on large binary circuits, which significantly increases the prover's computational overhead. In [6], the authors propose embedding truncation verification into high-degree polynomial equations over the ring $\mathbb{Z}_{p^e}$, where p is a large prime and e denotes the number of digits in the number. Although theoretically sound, this method incurs substantial computational overhead. In [10], a commit-and-prove scheme is introduced to control the magnitude of rounding errors. However, this approach leads to significant circuit expansion, resulting in considerable overhead for both the prover and the verifier. Similarly, [11] explores the MPC-in-the-head paradigm, where a secure multi-party computation is simulated and the interactions between virtual parties are committed to the verifier. While this technique enables verifiable rounding operations, it substantially increases both computational and communication complexities. Finally, the method proposed by Dao and Thaler [7] improves polynomial evaluation techniques widely used in verifiable computation. Nevertheless, it still requires large finite fields to prevent overflow, introducing significant arithmetic overhead. This approach is conceptually similar to the integer-scaling technique employed in SafetyNets [12].

Our Contribution: In this paper, we propose a novel framework that alleviates core obstacles in realizing fully verifiable machine learning. We introduce `Range-Arithmetic`, a method that fuses sum-check protocols for arithmetic verification with range proofs for rounding and activation functions. By representing rounding and ReLU through concatenated range constraints, our framework maintains compatibility with finite-field arithmetic and avoids the need for extensive preprocessing, Boolean circuit encodings, or large lookup tables. We also show how these operations can be combined in a unified framework and used together to verify ML inference tasks without sacrificing efficiency or generality.

We compare the complexities of our approach with those of existing methods in Table 1. For numerical comparisons, however, we focus on the state-of-the-art method proposed by Dao and Thaler [7], which is the latest in a sequence of progressively improving works and has gained popularity in practice. Since [7] does not support the ReLU activation function, we focus this comparison on the inference task for a linear neural network. In our scheme, a rounding operation is applied after each multiplication, whereas the method in [7] requires enlarging the finite field to support such operations. As illustrated in Figure 2, the proposed `Range-Arithmetic` achieves superior performance compared to the state-of-the-art method [7], particularly as the number of layers increases. Due to the lack of well-maintained codebases, numerical comparisons with other methods were found to be practically infeasible. Nevertheless, we believe Table 1 provides a sufficiently conclusive basis for comparing the proposed method with the remaining approaches.

The remainder of the paper is organized as follows. Section 2 reviews the necessary prerequisites and presents an overview of fixed-point arithmetic, including its conversion to finite field representations.

Table 1: Comparison of verifiable computation methods for verifying the multiplication of two $n \times n$ matrices with rounding, evaluated in terms of verifier and prover computational complexity, communication complexity, and support for ReLU verification.

Method	Prover Compl.	Verifier Compl.	Comm. Compl.	Verifying ReLU
Mystique [24]	$O\left(n^3 \log n\right)$	$O\left(n^2\right)$	$O\left(n^2\right)$	Yes
Ring-based [6]	$O\left(pen^3\right)$	$O\left(n^2\right)$	$O\left(\log n\right)$	No
Commit-and-Prove [10]	$O\left(n^3 \log n^3\right)$	$O\left(n^{1.5}\right)$	$O\left(n^2\right)$	No
MPC-in-head [11]	$O\left(n^3\right)$	$O\left(n^3\right)$	$O\left(n^3\right)$	No
Integer-scaling [7]	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(\log n\right)$	No
Proposed Method	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(\log n\right)$	Yes

In [6], p is the prime base and e is the number of digits used to represent numbers in the ring $\mathbb{Z}_{p^e}$.

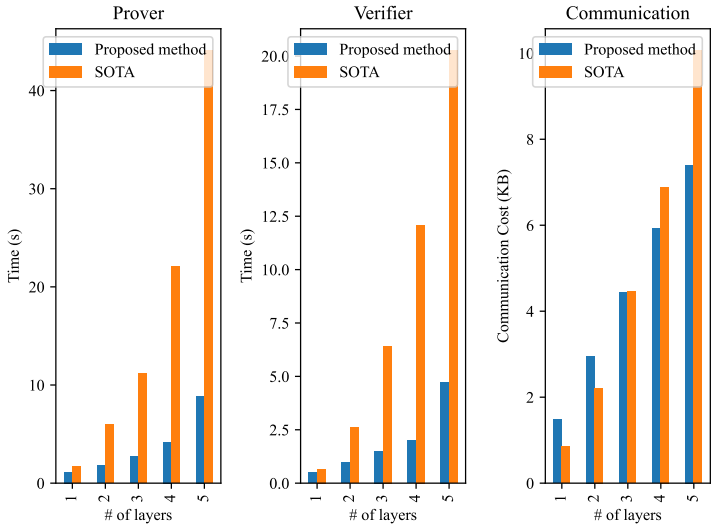

Figure 2: Comparison of prover and verifier runtimes, as well as communication costs, for verifiable inference on a linear neural network, relative to the state-of-the-art method in [7].

We then describe the verification algorithms, covering vector multiplication, polynomial commitment, the Sum-Check protocol, and range-proof techniques. Our proposed algorithm is detailed in Section 3. Section 4 presents the results of our simulations and evaluates the algorithm's performance from multiple perspectives. Finally, Section 5 discusses potential directions for future research.

2 Preliminaries

Notation. Vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$. Matrices are denoted by boldface uppercase letters, e.g., $\mathbf{A}$. Polynomials are denoted by lowercase letters, for example, a. We denote the inner product between two vectors by $\langle \mathbf{a}, \mathbf{b} \rangle$. For a finite field $\mathbb{F}$, a group $\mathbb{G}$, a given vector $\mathbf{a} \in \mathbb{F}^n$, and a vector $\mathbf{g} \in \mathbb{G}^n$ of generators, $\mathbf{g}^{\mathbf{a}} = \prod_{i=1}^{n} g_i^{a_i}$. For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$, the element-wise product is represented by $\mathbf{a} \circ \mathbf{b}$ which is equal to $(a_1 b_1, \ldots, a_n b_n)$. For $a, b \in \mathbb{R}$, $[a, b]$ denotes the set of all numbers $x \in \mathbb{R}$, where $a \leq x \leq b$. For integers $1 \leq u \leq v \leq n$, and a vector $\mathbf{a} = (a_1, \ldots, a_n)$, then $\mathbf{a}[u-1, v]$ denotes the sub-vector $(a_u, \ldots, a_v)$ of the vector $\mathbf{a}$. Similarly, $a[:u] := (a_1, \ldots, a_u)$, and $\mathbf{a}[u:] := (a_{u+1}, \ldots, a_n)$.

Definition 2.1. *For any prime number $p > 2$, the **symmetric finite field**, denoted $\mathbb{F}_p$ (or simply $\mathbb{F}$), can be represented by the symmetric complete residue system $\left\{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\right\}$, which serves as an alternative to the standard system $\{0, 1, \ldots, p-1\}$. Field operations such as addition*

and multiplication are performed modulo p, with the results mapped back into this symmetric representative set.

Definition 2.2. *Let $e : \{0,1\}^v \to \mathbb{F}$ be a function defined over the Boolean hypercube for some integer v. The **multilinear extension** (MLE) of e, denoted $\tilde{e} : \mathbb{F}^v \to \mathbb{F}$, is a multivariate polynomial satisfying the following properties:*

- *For all $b_1, \ldots, b_v \in \{0,1\}$, we have $\tilde{e}(b_1, \ldots, b_v) = e(b_1, \ldots, b_v)$.*
- *The polynomial $\tilde{e}(x_1, \ldots, x_v)$ is* multilinear*, i.e., it is linear in each input variable individually.*

It has been shown that $\tilde{e}$ can be uniquely written as [23]:

$$\tilde{e}\left(x_1, \ldots, x_v\right) := \sum_{\mathbf{y} \in \{0,1\}^v} e(\mathbf{y}) . \prod_{i=1}^{v} \left(x_i y_i + (1 - x_i)\left(1 - y_i\right)\right), \tag{1}$$

for any $(x_1, \ldots, x_v) \in \mathbb{F}^v$.

Lemma 2.1. ***(Schwartz–Zippel Lemma [22, 27])*** *Let $f, g : \mathbb{F}^v \to \mathbb{F}$ be distinct multilinear polynomials. If $\mathbf{r}$ is sampled uniformly at random from $\mathbb{F}^v$, then the probability that $f(\mathbf{r}) = g(\mathbf{r})$ is at most $\frac{v}{|\mathbb{F}|}$; that is,*

$$\Pr[f(\mathbf{r}) = g(\mathbf{r})] \leq \frac{v}{|\mathbb{F}|}.$$

2.1 Review of fixed-point arithmetic

In this paper, we focus on fixed-point arithmetic, a widely used technique for managing overflow. In this approach, rounding is applied after each multiplication to discard the least significant bits, thereby preventing unbounded growth in memory usage. Specifically, each fixed-point number consists of three components: a sign bit, t bits for the integer part, and s bits for the fractional part.

Let p be a prime with at least $s + t + 3$ bits. We represent a real number a' as a field element $a = 2^s a' \in \mathbb{F}_p$. When multiplying two fixed-point numbers with s fractional bits, the result may contain up to $2s$ fractional bits. To retain the target precision, a rounding operation truncates the s least significant bits and rounds to the nearest fixed-point number.

This rounding can be equivalently performed in the field representation using the operator $\mathfrak{R}$, defined as:

$$\mathfrak{R}(x) = \frac{x + 2^{s-1} - \left(x + 2^{s-1} \bmod 2^s\right)}{2^s}.$$

An example of this rounding process is illustrated in Figure 5.

2.2 Inner-product argument

Consider a system involving a prover and a verifier. Let $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$ be two publicly known vectors, consisting of $2n$ generators from a group $\mathbb{G}$. The prover sends two elements to the verifier: $P \in \mathbb{G}$ and $c \in \mathbb{F}$, with the goal of convincing the verifier that they possess two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ such that

$$P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} \quad \text{and} \quad c = \langle \mathbf{a}, \mathbf{b} \rangle.$$

To verify this claim, the verifier runs Algorithm 2 (Appendix 5), inspired by the Bulletproofs protocol [5]. This algorithm is communication-efficient: the prover transmits only $2\log_2(n)$ group elements, rather than the full vectors $\mathbf{a}$ and $\mathbf{b}$, which would require sending $2n$ elements. For simplicity, we assume that n is a power of two.

2.3 Polynomial commitment

Consider a system comprising a prover and a verifier. The prover holds a multilinear polynomial q, and the verifier wishes to evaluate q at an arbitrary point $\mathbf{z} \in \mathbb{F}^v$. A naïve approach would have the prover transmit all coefficients of q, allowing the verifier to compute the evaluation locally. However, this incurs high communication cost and substantial computational effort for the verifier.

A more efficient alternative is to use a polynomial commitment scheme, as described in Algorithm 3. In this setting, the evaluation $q(\mathbf{z})$ is computed via an inner product. Let $\mathbf{z} = [z_1, \ldots, z_v]$. Then $q(\mathbf{z})$ can be written as:

$$q(\mathbf{z}) = \sum_{\mathbf{i} \in \{0,1\}^v} a_{\mathbf{i}} \cdot \chi_{\mathbf{i}}(\mathbf{z}),$$

where a_i are the coefficients of q, and $\chi_i(\mathbf{z})$ are the Lagrange basis polynomials evaluated at $\mathbf{z}$. Each basis polynomial is defined as:

$$\chi_{\mathbf{i}}(\mathbf{z}) := \prod_{j=1}^{v} \left(z_j i_j + (1 - z_j)(1 - i_j)\right),$$

for $\mathbf{i} = (i_1, \ldots, i_v) \in \{0,1\}^v$.

The prover commits to the coefficients $\mathbf{a} = [a_0, \ldots, a_{2^v-1}]$ by sending a group element $P_1 = \mathbf{g}^{\mathbf{a}}$, where $\mathbf{g}$ is a vector of public generators. The verifier then checks whether $q(\mathbf{z}) = \langle \mathbf{a}, \mathbf{b} \rangle$, where $\mathbf{b} = [\chi_0(\mathbf{z}), \ldots, \chi_{2^v-1}(\mathbf{z})]$.

2.4 The sum-check protocol

Suppose the prover and verifier share knowledge of a v-variate polynomial $f(x_1, \ldots, x_v)$, where each variable has finite degree. The **sum-check protocol**, used within an interactive proof (IP) system, enables the prover to send a value $w \in \mathbb{F}$ to the verifier and convince her that

$$w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots, x_v).$$

This protocol proceeds in v rounds, with each round consisting of one message from the prover and one from the verifier. The prover's total computational cost is only a small multiple of the cost of directly computing w, making the protocol highly efficient compared to alternative approaches [23]. A detailed description is provided in Algorithm 4 in Appendix 5.

In the final step of Algorithm 4, the verifier must evaluate the multivariate polynomial f at a randomly chosen point. However, in our setting, downloading all individual coefficients of $f(\mathbf{x})$ would incur prohibitive communication overhead. Furthermore, as we later show, the polynomial $f(\mathbf{x})$ admits a decomposition:

$$f(\mathbf{x}) = f^{(1)}(\mathbf{y}_1, \mathbf{x}) \cdot f^{(2)}(\mathbf{x}, \mathbf{y}_2),$$

where $\mathbf{y}_1$ and $\mathbf{y}_2$ are fixed inputs, and both $f^{(1)}$ and $f^{(2)}$ are multilinear functions.

To mitigate communication cost, we employ binding commitments to the coefficients of $f^{(1)}$ and $f^{(2)}$, denoted P_1 and P_2, respectively, using the commitment scheme described in the previous subsection. These commitments allow the verifier and prover to independently execute two instances of Algorithm 3 to evaluate $f^{(1)}$ and $f^{(2)}$ at the desired points. The verifier then computes $f(\mathbf{x})$ as the product of the two evaluations.

This leads to Algorithm 5, which augments the classical sum-check protocol with polynomial commitments to reduce communication overhead. However, this efficiency gain comes at the cost of increased computational effort for the verifier in the final step. Overall, the total communication complexity of the protocol is dominated by Step 6 of Algorithm 5, and scales with the sum of the degrees of the involved polynomials, i.e., $O\left(\sum_{i=1}^{v} \deg(f_i)\right)$ [2].

2.5 Aggregated range proof

Let $\mathbf{g} \in \mathbb{G}^{mn}$ be a globally known vector consisting of mn generators of the group $\mathbb{G}$. The prover sends an element $P \in \mathbb{G}$ to the verifier and claims knowledge of a vector $\mathbf{v} \in \mathbb{F}^m$ such that:

$$P = \mathbf{g}_{[:m]}^{\mathbf{v}}, \quad \text{where } v_j \in [0, 2^n - 1] \text{ for all } j \in [1, m].$$

Algorithm 6 in Appendix 5, based on the Bulletproofs protocol [5], enables the prover to efficiently convince the verifier of the validity of this claim.

3 The Proposed `Range-Arithmetic` Scheme

In this section, we introduce the core components of our proposed scheme. We begin by addressing the verification of arithmetic operations, with a focus on matrix multiplication. This is motivated by the fact that many fundamental operations in neural networks, including fully connected layers, convolutions, and batch normalization, can be expressed as matrix multiplications [3, 13]. We then present our approach for handling non-arithmetic operations, specifically rounding, and describe how arithmetic and non-arithmetic components are integrated through a composition step. Next, we extend the framework to support verification of the ReLU activation function. Finally, we outline the complete procedure for verifying inference in neural networks.

3.1 Verification of Matrix Multiplication Concatenated with Rounding

Consider a scenario where the prover holds two matrices, $\mathbf{A} \in \mathbb{F}^{n\times m}$ and $\mathbf{B} \in \mathbb{F}^{m\times k}$. The prover computes their product $\mathbf{C} = \mathbf{AB}$, and then applies a rounding operation to obtain $\mathbf{C}' = \mathfrak{R}(\mathbf{C})$. The goal is for the prover to convince the verifier of the correctness of these computations without requiring the verifier to recompute them.

The goal is to develop a verification algorithm that meets the following criteria:

- **Correctness:** If the prover executes the computation faithfully, an honest verifier should accept the result.
- **Soundness:** If the prover attempts to deviate from the correct computation, the verifier should reject the result with high probability.
- **Efficiency:** The verification protocol should be efficient in terms of both communication and computation. Specifically, the communication complexity between the prover and verifier should be $O(\log nmk)$, the prover's computational complexity should be $O(nmk)$, and the verifier's computational complexity should be $O(nm + mk)$.

3.1.1 Verification of Matrix Multiplication

Inspired by established methods in the literature [7, 23], we reformulate the verification of matrix multiplication $\mathbf{C} = \mathbf{AB}$ as a sum-check verification problem. Consider three matrices $\mathbf{A} \in \mathbb{F}^{n\times m}$, $\mathbf{B} \in \mathbb{F}^{m\times k}$, and $\mathbf{C} \in \mathbb{F}^{n\times k}$. For simplicity, we assume that m, n, and k are powers of two. Matrix rows and columns are indexed from 0 to one less than the total number of rows and columns, respectively.

To reference matrix elements, we define the following functions: $f_{\mathbf{A}} : \{0,1\}^{\log n\times\log m} \to \mathbb{F}$, $f_{\mathbf{B}} : \{0,1\}^{\log m\times\log k} \to \mathbb{F}$, and $f_{\mathbf{C}} : \{0,1\}^{\log n\times\log k} \to \mathbb{F}$ These functions take the binary representations of row and column indices as input and return the corresponding matrix element. For instance, if $\mathbf{A}$ is a 4×4 matrix with element $a_{0,2} = 57$, then $f_{\mathbf{A}}([0,0],[1,0]) = 57$, where $[0,0]$ specifies the first row and $[1,0]$ specifies the third column.

Under this formulation, verifying the correctness of the matrix multiplication reduces to checking that, for all $\mathbf{i} \in 0,1^{\log n}$ and $\mathbf{j} \in 0,1^{\log k}$, the following equality holds:

$$f_{\mathbf{C}}(\mathbf{i},\mathbf{j}) = \sum_{\boldsymbol{\ell}\in 0,1^{\log m}} f_{\mathbf{A}}(\mathbf{i},\boldsymbol{\ell}) \cdot f_{\mathbf{B}}(\boldsymbol{\ell},\mathbf{j}). \tag{2}$$

As we saw in Definition 2.2, we can construct MLEs (Multilinear Extensions) $\tilde{a} : \mathbb{F}^{\log n\times\log m} \to \mathbb{F}$, $\tilde{b} : \mathbb{F}^{\log m\times\log k} \to \mathbb{F}$, and $\tilde{c} : \mathbb{F}^{\log m\times\log k} \to \mathbb{F}$ based on functions $f_{\mathbf{A}}$, $f_{\mathbf{B}}$, and $f_{\mathbf{C}}$, respectively. These MLEs are multilinear polynomials that, for binary inputs, yield outputs resembling those of $f_{\mathbf{A}}$, $f_{\mathbf{B}}$, and $f_{\mathbf{C}}$, respectively. Therefore, the above equality is equivalent to establishing the following equality for $\mathbf{i} \in \{0,1\}^{\log n}$ and $\mathbf{j} \in \{0,1\}^{\log k}$

$$\tilde{c}(\mathbf{i},\mathbf{j}) = \sum_{\boldsymbol{\ell}\in\{0,1\}^{\log m}} \tilde{a}(\mathbf{i},\boldsymbol{\ell})\tilde{b}(\boldsymbol{\ell},\mathbf{j}). \tag{3}$$

We aim to verify (3) for all values of $\mathbf{i}$ and $\mathbf{j}$. We observe that since $\tilde{a}$ is linear with respect to the variable $\mathbf{i}$ and $\tilde{b}$ is linear with respect to the variable $\mathbf{j}$, the polynomial on the right-hand side of the

equation is linear with respect to variables $\mathbf{i}$ and $\mathbf{j}$. As we discussed, MLE is unique. Therefore, if $\tilde{c}$ is equal to the right-hand side for all possible binary values of $\mathbf{i} \in \{0,1\}^{\log n}$ and $\mathbf{j} \in \{0,1\}^{\log k}$, then (3) is of the form of equality of two polynomials. If matrix multiplication $\mathbf{C} = \mathbf{AB}$ has been performed correctly, then (3) must hold for all values in the domain $\mathbf{i} \in \mathbb{F}^{\log n}$ and $\mathbf{j} \in \mathbb{F}^{\log k}$. Thus, we can use the lemma 2.1, which allows us to verify the equality of two polynomials with high probability by checking their equality at a random point. Assume $\mathbf{r}_1 \in \mathbb{F}^{\log n}$ and $\mathbf{r}_2 \in \mathbb{F}^{\log k}$ are chosen uniformly at random. Then if

$$\tilde{c}(\mathbf{r}_1, \mathbf{r}_2) = \sum_{\boldsymbol{\ell} \in \{0,1\}^{\log m}} \tilde{a}(\mathbf{r}_1, \boldsymbol{\ell}).\tilde{b}(\boldsymbol{\ell}, \mathbf{r}_2) \tag{4}$$

holds, the matrix multiplication $\mathbf{C} = \mathbf{AB}$ has been computed correctly with the probability of at least $\frac{|\mathbb{F}|-v}{|\mathbb{F}|}$.

In (4), the left-hand side is a single number, while the right-hand side is actually a sum over the polynomial $\gamma(\boldsymbol{\ell}) := \tilde{a}(\mathbf{r}_1, \boldsymbol{\ell}).\tilde{b}(\boldsymbol{\ell}, \mathbf{r}2)$, for all values $\boldsymbol{\ell} \in \{0,1\}^{\log m}$. Therefore, we can reduce the matrix multiplication verification problem to a sum-check problem over $\sum_{\boldsymbol{\ell}} \gamma(\boldsymbol{\ell})$ and apply Algorithm 5 to verify it. It is observed that $\gamma(\boldsymbol{\ell})$ represents the multiplication of two multilinear functions $\tilde{a}(\mathbf{r}_1, \boldsymbol{\ell})$ and $\tilde{b}(\boldsymbol{\ell}, \mathbf{r}_2)$. The prover can compute the commitment to the coefficients of these functions as outlined in Section 2.3.

3.1.2 Verification of Rounding

Here, we elucidate how the range proof algorithm ensures the validity of the operation $\mathfrak{R}$. It is worth recalling that $\mathfrak{R}(x) = \frac{x+2^{s-1}-(x+2^{s-1} \mod 2^s)}{2^s}$, and when $\mathfrak{R}$ is applied to a matrix or vector, the operation is executed on each element individually. We note that $-2^{s-1} \leq (x + 2^{s-1} \mod 2^s) - 2^{s-1} < 2^{s-1}$, which constitutes the truncated portion. In the rounding process, this part is discarded, which makes the numerator of the fraction $\frac{x+2^{s-1}-(x+2^{s-1} \mod 2^s)}{2^s}$ a multiple of 2^s. Subsequently, division by 2^s moves its binary representation as s bits to the right. Suppose that matrix $\mathbf{A} \in \mathbb{F}^{\log n \times \log m}$ has been rounded to $\mathbf{A}' = \mathfrak{R}(\mathbf{A})$. Let $\mathbf{E} := \mathbf{A} - 2^s \times \mathbf{A}'$ denote the discarded part during the rounding process, and let $\mathbf{D} := \mathbf{A} - \mathbf{E}$ denote the numerator of the fraction in $\mathfrak{R}$. The correctness of the rounding process is equivalent to ensuring that (1) all entries e_i of $\mathbf{E}$ lie within the interval $-2^{s-1} \leq e_i < 2^{s-1}$, and (2) all elements d_i of $\mathbf{D}$ are integer multiples of 2^s.

The first condition limits the discarded number during rounding to a small, standard interval. The second condition ensures $a_i - e_i$ is divisible by 2^s without wrap-around. To prevent wrap-around, we require $a'_i = \frac{d_i}{2^s}$ to fall within $-2^{t+1} \leq a'_i < 2^{t+1}$, guaranteeing at least s trailing zeros in d_i. This condition suffices to avoid wrap-around, leading to two necessary range proofs, which are checked using an aggregated range proof algorithm. The correctness of this transformation is formally stated in the following theorem. A detailed proof is provided in Appendix 5.

Theorem 3.1. *Let a be a fixed-point number with 1 sign bit, t integer bits, and s fractional bits. Suppose the prover sends integers a'_1 and e_1 such that*

$$a \equiv e_1 + 2^s \cdot a'_1 \pmod{p},$$

where p is a prime with at least $t + s + 3$ bits. Assume the values satisfy the following bounds:

$$-2^{t+1} \leq a'_1 < 2^{t+1}, \quad \textit{and} \quad -2^{s-1} \leq e_1 < 2^{s-1}.$$

Then, the verifier can conclude that $a'_1 = \mathfrak{R}(a)$, where $\mathfrak{R}(a)$ denotes the fixed-point rounding of a to the nearest integer after truncating the fractional part.

3.1.3 Concatenation of Matrix Multiplication and Rounding

We now explain how to combine the arithmetic and rounding components using the techniques described above. Suppose that the prover possesses two matrices, $\mathbf{A} \in \mathbb{F}^{n \times m}$ and $\mathbf{B} \in \mathbb{F}^{m \times k}$. After computing their product, $\mathbf{C} = \mathbf{AB}$, the prover performs rounding operations on the output, resulting in $\mathbf{C}' = \mathfrak{R}(\mathbf{C})$. By executing Algorithm 1 on the input $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{mn}$, $u \in \mathbb{G}$, $\mathbf{A} \in \mathbb{F}^{n \times m}$, and $\mathbf{B} \in \mathbb{F}^{m \times k}$, the prover convinces the verifier of the accuracy of these computations without transmitting the large matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{C}'$. In the input tuple, $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{\tau}$ and $u \in \mathbb{G}$ are some

Algorithm 1 Verifying a calculation with both arithmetic and non-arithmetic layers

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{\tau}$, $u \in \mathbb{G}$, $\mathbf{A} \in \mathbb{F}^{n \times m}$, $\mathbf{B} \in \mathbb{F}^{m \times k}$.
Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{j}$, $u \in \mathbb{G}$.
Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{j}$, $u \in \mathbb{G}$, $\mathbf{A} \in \mathbb{F}^{n \times m}$, $\mathbf{B} \in \mathbb{F}^{m \times k}$.
Ensure: Verifier receives P_A, P_B, P_C, and $P_{C'}$ as commitments to specific matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{C}'$, respectively. Then Verifier accepts that the relations $\mathbf{C} = \mathbf{AB}$ and $\mathbf{C}' = \mathfrak{R}(\mathbf{C})$ hold between them.

1: Prover computes the commitments of $\mathbf{A}$ and $\mathbf{B}$ as P_A and P_B respectively. then sends P_A and P_B to Verifier.
2: Prover calculates $\mathbf{C} = \mathbf{AB}$.
3: Prover computes the commitment of $\mathbf{C}$ as P_C then sends P_C to Verifier.
4: Verifier selects two random vectors $\mathbf{r}_1 \in \mathbb{F}^{\log n}$ and $\mathbf{r}_2 \in \mathbb{F}^{\log k}$ and sends $\mathbf{r}_1$ and $\mathbf{r}_2$ to Prover.
5: Prover constructs the polynomials $\tilde{a} : \mathbb{F}^{\log nm} \to \mathbb{F}$, $\tilde{b} : \mathbb{F}^{\log mk} \to \mathbb{F}$, $\tilde{c} : \mathbb{F}^{\log nk} \to \mathbb{F}$ and $\gamma(\mathbf{z}) : \mathbb{F}^{\log m} \to \mathbb{F}$ according to the description provided in the Section 3.1.1.
Note: The commitments to the coefficients of the polynomial $\tilde{a}$, $\tilde{b}$, and $\tilde{c}$ are represented by P_A, P_B, and P_C respectively, all of which are already held by the verifier. Furthermore, it is given that $\gamma(\mathbf{z}) = \tilde{a}(\mathbf{r}_1, \mathbf{z})\tilde{b}(\mathbf{z}, \mathbf{r}_2)$.
6: Prover and Verifier run Algorithm 5 on the input $\big(\mathbf{g}_{[:m]}, \mathbf{h}_{[:m]}, u, P_A, P_B, \gamma\big)$. Verifier obtains w.
7: Prover and Verifier run Algorithm 3 on the input $\big(\mathbf{g}_{[:nk]}, \mathbf{h}_{[:nk]}, u, P_C, \tilde{c}, (\mathbf{r_1}, \mathbf{r_2})\big)$. Verifier obtains $\tilde{c}(\mathbf{r_1}, \mathbf{r_2})$.
8: Verifier checks $w = \tilde{c}(\mathbf{r_1}, \mathbf{r_2})$.
9: Prover computes $\mathbf{C}' = \mathfrak{R}(\mathbf{C})$.
10: Prover computes the commitment of $\mathbf{C}'$ as $P_{C'}$ then sends $P_{C'}$ to Verifier.
11: Prover and Verifier run Algorithm 6 on the input
$\Big(\mathbf{g}_{[:nks]}, \mathbf{h}_{[:nks]}, u, P_C / P_{C'}^{2^s} \times \mathbf{g}_{[:nk]}^{2^{s-1}} \in \mathbb{G}, \mathbf{C} + 2^{s-1} \in \mathbb{F}^{nk}\Big)$.
▷ Note that $P_C / P_{C'}^{2^s}$ is the commitment to $E = \mathbf{C} - 2^s \times \mathbf{C}'$. This step verifies for all elements of $\mathbf{E}$ we have $-2^{s-1} \leq e_i < 2^{s-1}$.
12: Prover and Verifier run Algorithm 6 on the input
$\Big(\mathbf{g}_{[:nk(t+1)]}, \mathbf{h}_{[:nk(t+1)]}, u, P_{C'} \times \mathbf{g}_{[:nk]}^{2^{t+1}} \in \mathbb{G}, \mathbf{C} + 2^{t+1} \in \mathbb{F}^{nk}\Big)$.
▷ This step verifies for all elements of $\mathbf{C}'$ we have $-2^{t+1} \leq c'_i < 2^{t+1}$.
If all checks pass, Verifier accepts that P_A, P_B, P_C, and $P_{C'}$ are commitments to certain matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{C}'$, respectively, where the relations $\mathbf{C} = \mathbf{AB}$ and $\mathbf{C}' = \mathfrak{R}(\mathbf{C})$ hold between them.

globally known generators. Here, τ denotes the maximum value among $(nks, nk(t+1), mk, mn)$. This algorithm leverages all the methodologies introduced and examined in prior subsections.

Our proposed algorithm is adaptable, allowing the prover to reuse it for matrices $\mathbf{A}$ and $\mathbf{B}$ from prior computations or for matrix $\mathbf{C}'$ in subsequent computations without transmitting large intermediary matrices to the verifier. Additionally, the algorithm requires no preprocessing and avoids the need to encode computations using large sparse matrices, unlike techniques such as R1CS or arithmetic circuits [18].

3.2 Verification of ReLU activation function

We verify the ReLU function, defined as $\text{ReLU}(x) = \max\{0, x\} = \frac{x+|x|}{2}$, by applying it element-wise to a matrix. Suppose the verifier has a commitment to a matrix $\mathbf{A} \in \mathbb{F}^{n \times k}$ and receives a commitment to $\mathbf{B} \in \mathbb{F}^{n \times k}$. To verify that $\mathbf{B} = \text{ReLU}(\mathbf{A})$, the prover first computes $\mathbf{Y} = |\mathbf{A}|$ and commits to it. The prover then proves that all elements of $\mathbf{Y}$ are non-negative using a Range proof. Next, the prover proves that for each element a_i in $\mathbf{A}$ and y_i in $\mathbf{Y}$, $a_i^2 = y_i^2$ using the following equation:

$$\vec{0} = \sum_{\mathbf{s} \in \{0,1\}^{\log nk}} \tilde{I}\left(\mathbf{x}, \mathbf{s}\right) . \left(\tilde{a}^2\left(\mathbf{s}\right) - \tilde{y}^2\left(\mathbf{s}\right)\right) \quad \forall \mathbf{x} \in \{0,1\}^{\log nk} \tag{5}$$

In (5), matrices are encoded using MLE, where $\tilde{I}$ is the MLE of the identity matrix, and $\tilde{a}$ and $\tilde{y}$ encode the elements of $\mathbf{A}$ and $\mathbf{Y}$, respectively. This equation ensures that each element in $\mathbf{Y}$ equals $|\mathbf{A}|$. By the uniqueness of MLEs, the right-hand side polynomial is zero. Instead of checking the equation at every point, the prover and verifier use the sum-check protocol at a random point $\mathbf{r}_1$ chosen by the verifier.

$$0 = \sum_{\mathbf{x} \in \{0,1\}^{\log nk}} \tilde{I}\left(\mathbf{s}, \mathbf{x}\right) . \left(\tilde{a}^2\left(\mathbf{x}\right) - \tilde{y}^2\left(\mathbf{x}\right)\right) \tag{6}$$

According to (6), the sum-check is performed on the polynomial $f(\mathbf{x}) = \tilde{I}\left(\mathbf{s}, \mathbf{x}\right) . \left(\tilde{a}^2\left(\mathbf{x}\right) - \tilde{y}^2\left(\mathbf{x}\right)\right)$. The details are provided in Algorithm 7. We used Algorithm 8 to make ReLU verifiable, which incorporates Algorithm 7 and other components mentioned in this subsection.

3.3 Verification process of the neural network

To verify the inference of a neural network, we consider the model as a sequence of layers, each corresponding to an operation, e.g., matrix multiplication or ReLU activation. Each layer is verified independently. The prover first sends a commitment of the layer's output to the verifier and then uses this commitment to prove that the corresponding computation was performed correctly, following the proposed algorithm. This modular approach enables the composition of verification algorithms across different layers, allowing them to be reused and combined in arbitrary order and quantity. Please note that output of each layer is the input to the next layer and verifier has access to the inputs. It is important to note that the output of each layer serves as the input to the subsequent layer. An illustrative case study demonstrating this process is presented in Section 4.

4 Experimental results

This section evaluates the performance of our proposed algorithm for fixed-point matrix multiplication, in which standard matrix multiplication is followed by a rounding step. The evaluation focuses on three key metrics: the prover's runtime, the verifier's runtime, and the communication cost between them. All algorithms are implemented in Python and tested on an Asus X515 laptop. The implementation is available on GitHub[1].

Figure 3 presents the performance of Algorithm 1 in verifying the relation $\mathbf{C}' = \mathfrak{R}(\mathbf{AB})$, where $\mathbf{A}$ and $\mathbf{B}$ are 64×64 matrices. As the numeric range increases, the cost of the rounding operation—dominated by the complexity of Algorithm 6—surpasses that of the matrix multiplication.

Figure 4 illustrates the effect of matrix size on the prover and verifier runtimes. As expected, both increase with matrix size, while the communication overhead grows logarithmically.

Finally, we compare our method to the state-of-the-art [7], which verifies sequential matrix multiplications of depth n. Unlike method of [7], which lacks support for rounding and incurs significant storage overhead, our approach integrates rounding efficiently and reduces computational cost for both prover and verifier as the depth increases. This advantage is depicted in Figure 2.

4.1 A Case Study

In the previous section, we evaluated the performance of our proposed method for verifying matrix multiplication. Building on this foundation, we now demonstrate its application to verifying the inference process of a neural network trained on the MNIST dataset [8]. The network consists of four fully connected layers, each performing matrix multiplication using fixed-point arithmetic, followed by a ReLU activation function. The input vector has a dimension of 784, and the model contains approximately 10,000 parameters.

All computations are carried out in fixed-point arithmetic, with weights and inputs represented using 6 integer bits and 8 fractional bits. The trained model achieves an accuracy of approximately 95%. For verification, matrix multiplications are checked using Algorithm 1, while ReLU activations are verified using a combination of range proofs and the sum-check protocol, as detailed in Algorithm 8

[1] https://github.com/trainingzk/Range-Arithmetic

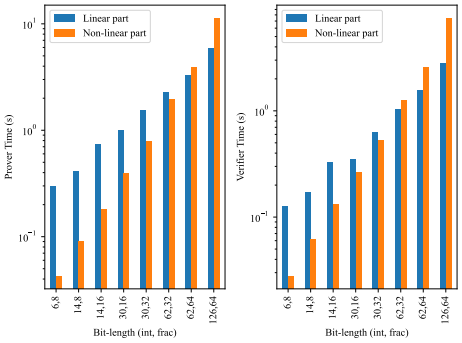

Figure 3: Runtime of the arithmetic and non-arithmetic parts for the verifier and the prover in the matrix multiplication.

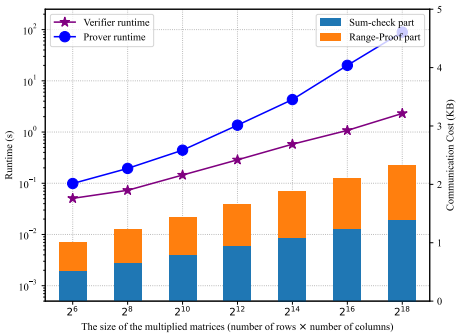

Figure 4: Impact of matrix size on the prover and the verifier runtime, as well as communication load.

and Subsection 3.2. The prover's runtime is approximately 230 milliseconds, and the verifier's runtime is approximately 154 milliseconds.

5 Future directions

In this paper, we focused on the verifying inference on neural network. Future work includes enriching the model to support a wider range of network architectures and layers, incorporating privacy-preserving features, benchmarking against alternative verification schemes, exploring additional evaluation metrics, and extending the approach to other collaborative or trust-sensitive environments such as federated learning.

References

[1] Accountable Magic. Accountable magic: Ai alignment and governance, 2025. URL `https://www.accountablemagic.com/`.

[2] Suyash Bagad, Yuval Domb, and Justin Thaler. The sum-check protocol over fields of small characteristic. *Cryptology ePrint Archive*, 2024.

[3] Yoshua Bengio, Ian Goodfellow, Aaron Courville, et al. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[4] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 742–773. Springer, 2021.

[5] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[6] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Interactive proofs for rounding arithmetic. *IEEE Access*, 10:122706–122725, 2022.

[7] Quang Dao and Justin Thaler. More optimizations to sum-check proving. *Cryptology ePrint Archive*, 2024.

[8] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[9] EZKL. EZKL: Zero-knowledge machine learning, 2025. URL `https://ezkl.xyz/`.

[10] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct zero knowledge for floating point computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1203–1216, 2022.

[11] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1880–1894, 2023.

[12] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] Chanyang Ju, Hyeonbum Lee, Heewon Chung, Jae Hong Seo, and Sungwook Kim. Efficient sum-check protocol for convolution. *IEEE Access*, 9:164047–164059, 2021.

[14] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. Snargs and ppad hardness from the decisional diffie-hellman assumption. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II*, pages 470–498. Springer, 2023.

[15] Alan Li, Qingkai Liang, and Mo Dong. Sparsity-aware protocol for zk-friendly ml models: Shedding lights on practical zkml. *Cryptology ePrint Archive*, 2024.

[16] Hidde Lycklama, Alexander Viand, Nikolay Avramov, Nicolas Küchler, and Anwar Hithnawi. Artemis: Efficient commit-and-prove snarks for zkml. *arXiv preprint arXiv:2409.12055*, 2024.

[17] Noya. Noya: Ai-powered infrastructure for zero-knowledge proofs, 2025. URL `https://noya.ai/`.

[18] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

[19] Zhizhi Peng, Taotao Wang, Chonghe Zhao, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. A survey of zero-knowledge proof based verifiable machine learning. *arXiv preprint arXiv:2502.18535*, 2025.

[20] Polygon. Polygon zkEVM: Ethereum scaling with zero-knowledge proofs, 2025. URL https://polygon.technology/polygon-zkevm.

[21] Provably AI. Provably: Trustless ai verification, 2025. URL https://provably.ai/.

[22] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[23] Justin Thaler et al. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.

[24] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.

[25] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. pvcnn: Privacy-preserving and verifiable convolutional neural network testing. *IEEE Transactions on Information Forensics and Security*, 18:2218–2233, 2023.

[26] Qianyi Zhan, Yuanyuan Liu, Zhenping Xie, and Yuan Liu. Validating the integrity for deep learning models based on zero-knowledge proof and blockchain. In *Blockchain and Web3 Technology Innovation and Application Exchange Conference*, pages 387–399. Springer, 2024.

[27] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

[28] zkAGI. zkAGI: Zero-knowledge artificial general intelligence, 2025. URL https://www.zkagi.ai/.

[29] ZKML Systems. ZKML: Zero-knowledge machine learning systems, 2025. URL https://www.zkml.systems/.

Appendix A. Details of Algorithms

Algorithm 2 Verification of Inner-product

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $u, P \in \mathbb{G}$, $c \in \mathbb{F}$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$.
Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $u, P \in \mathbb{G}$, $c \in \mathbb{F}$.
Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $u, P \in \mathbb{G}$, $c \in \mathbb{F}$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$.
Ensure: Verifier accepts $P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}$ and $c = \langle \mathbf{a}, \mathbf{b} \rangle$.

Verifier sends a random number $x \in \mathbb{F}$ to Prover.
Both Prover and Verifier compute $u_{\text{new}} = u^x$ and $P_{\text{new}} = Pu_{\text{new}}^c \in \mathbb{G}$. ▷ u is a generator of the group $\mathbb{G}$.
Run the function INNERPRODUCT on the input $(\mathbf{g}, \mathbf{h}, u_{\text{new}}, P_{\text{new}}, \mathbf{a}, \mathbf{b})$.
function INNERPRODUCT($\mathbf{g}, \mathbf{h}, u, P, \mathbf{a}, \mathbf{b}$) ▷ Verifier accepts $P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a}, \mathbf{b} \rangle}$.
    if $n = 1$ **then**
        Prover sends $a, b \in \mathbb{F}$ to Verifier.
        Verifier checks whether $P = g^a h^b u^{ab}$. If the equation is true, Verifier accepts; otherwise Verifier rejects.
    else
        Prover computes $n' = \frac{n}{2}$, $c_L = \left\langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \right\rangle \in \mathbb{F}$, $c_R = \left\langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \right\rangle \in \mathbb{F}$, $L = \mathbf{g}_{[n':]}^{\mathbf{a}_{[:n']}} \mathbf{h}_{[:n']}^{\mathbf{b}_{[n':]}} u^{c_L} \in \mathbb{G}$, $R = \mathbf{g}_{[:n']}^{\mathbf{a}_{[n':]}} \mathbf{h}_{[n':]}^{\mathbf{b}_{[:n']}} u^{c_R} \in \mathbb{G}$.
        Prover sends $L, R \in \mathbb{G}$ to Verifier.
        Verifier sends a random number $x \in \mathbb{F}$ to Prover.
        Both Prover and Verifier compute $\mathbf{g}' = \mathbf{g}_{[:n']}^{x^{-1}} \circ \mathbf{g}_{[n':]}^{x} \in \mathbb{G}^{n'}$, $\mathbf{h}' = \mathbf{h}_{[:n']}^{x} \circ \mathbf{h}_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$, $P' = L^{x^2} P R^{x^{-2}} \in \mathbb{G}$.
        Prover computes $\mathbf{a}' = x\mathbf{a}_{[:n']} + x^{-1}\mathbf{a}_{[n':]} \in \mathbb{F}^{n'}$ and $\mathbf{b}' = x^{-1}\mathbf{b}_{[:n']} + x\mathbf{b}_{[n':]} \in \mathbb{F}^{n'}$.
        Recursively run the function INNERPRODUCT on the input $(\mathbf{g}', \mathbf{h}', u, P', \mathbf{a}', \mathbf{b}')$.
    end if
end function

Algorithm 3 Polynomial commitment scheme

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1 \in \mathbb{G}$, the v-variate multilinear polynomial $q : \mathbb{F}_p^v \to \mathbb{F}_p$, $\mathbf{z} \in \mathbb{F}^v$.
Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1 \in \mathbb{G}$, $\mathbf{z} \in \mathbb{F}^v$.
Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1 \in \mathbb{G}$, the v-variate multilinear polynomial $q : \mathbb{F}_p^v \to \mathbb{F}_p$, $\mathbf{z} \in \mathbb{F}^v$.
Ensure: Verifier accepts $P_1 = \mathbf{g}^{\mathbf{a}}$ and obtains $q(\mathbf{z})$.

Prover and Verifier compute $P = P_1 \times \mathbf{h}^{\mathbf{b}}$, where $\mathbf{b} = (\chi_1(\mathbf{z}), \ldots, \chi_{2^v}(\mathbf{z}))$. ▷ χ_i are defined in the subsection 2.3.
Prover computes $q(\mathbf{z})$ and sends it to Verifier.
Prover and Verifier run Algorithm 2 on the input $(\mathbf{g}, \mathbf{h}, u, P, q(\mathbf{z}), \mathbf{a}, \mathbf{b})$. ▷ $u \in \mathbb{G}$ and $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$ are some generators of $\mathbb{G}$.

Algorithm 4 Verification protocol for matrix multiplication

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$.

Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$.

Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$.

Ensure: Verifier receives w and accepts $w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$.

1: Prover sends $w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$ to Verifier.
2: Prover sends the polynomial $f_1(X) = \sum_{(x_2,\ldots,x_v) \in \{0,1\}^{v-1}} f(X, x_2, \ldots, x_v)$
3: Verifier checks $w = f_1(0) = f_1(1)$.
4: Verifier sends a random number $r_1 \in \mathbb{F}_p$ to Prover.
5: **for** $i = 2, \ldots, v-1$ **do**
6: Prover sends the polynomial $f_i(X) = \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_v)$ to Verifier.
7: Verifier checks $g_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$.
8: Verifier sends a random number $r_i \in \mathbb{F}_p$ to Prover.
9: **end for**
10: Prover sends $f_v(X) = f(r_1, \ldots, r_{v-1}, X)$ to Verifier.
11: Verifier checks $f_{v-1}(r_{v-1}) = f_v(0) + f_v(1)$.
12: Verifier selects a number $r_v \in \mathbb{F}_p$ uniformly at random from $\mathbb{F}_p$.
13: Verifier computes $f(r_1, \ldots, r_v)$ and checks $f_v(r_v) = f(r_1, \ldots, r_v)$.
If all checks pass, Verifier accepts that w is correctly calculated.

Algorithm 5 Verification Protocol for Matrix Multiplication Using Dual Polynomial Commitments.

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$.
We have $f(\mathbf{x}) = f^{(1)}(\mathbf{y}_1, \mathbf{x}).f^{(2)}(\mathbf{x}, \mathbf{y}_2)$, where $f^{(1)}$ and $f^{(2)}$ are two multilinear functions, and $\mathbf{y}_1, \mathbf{y}_2$ are fixed vectors.

Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$.

Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$.

Ensure: Verifier receives w and accepts $w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$.

1: Prover sends $w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$ to Verifier.
2: Prover sends the polynomial $f_1(X) = \sum_{(x_2,\ldots,x_v) \in \{0,1\}^{v-1}} f(X, x_2, \ldots, x_v)$
3: Verifier checks $w = f_1(0) = f_1(1)$.
4: Verifier sends a random number $r_1 \in \mathbb{F}_p$ to Prover.
5: **for** $i = 2, \ldots, v-1$ **do**
6: Prover sends the polynomial $f_i(X) = \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_v)$ to Verifier.
7: Verifier checks $g_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$.
8: Verifier sends a random number $r_i \in \mathbb{F}_p$ to Prover.
9: **end for**
10: Prover sends $f_v(X) = f(r_1, \ldots, r_{v-1}, X)$ to Verifier.
11: Verifier checks $f_{v-1}(r_{v-1}) = f_v(0) + f_v(1)$.
12: Verifier sends a random number $r_v \in \mathbb{F}_p$ to Prover.
13: Prover and Verifier run Algorithm 3 on the input $\left(\mathbf{g}, \mathbf{h}, u, P_1, f^{(1)}, (\mathbf{y}_1, r_1, \ldots, r_v)\right)$ for Verifier to obtain $f^{(1)}(\mathbf{y}_1, r_1, \ldots, r_v)$.
14: Prover and Verifier run Algorithm 3 on the input $\left(\mathbf{g}, \mathbf{h}, u, P_2, f^{(2)}, (r_1, \ldots, r_v, \mathbf{y}_2)\right)$ for Verifier to obtain $f^{(2)}(r_1, \ldots, r_v, \mathbf{y}_2)$.
15: Verifier computes $f(r_1, \ldots, r_v) = f^{(1)}(r_1, \ldots, r_v).f^{(2)}(r_1, \ldots, r_v)$ and checks $f_v(r_v) = f(r_1, \ldots, r_v)$.
If all checks pass, Verifier accepts that w is correctly calculated.

Figure 5: Lines 1 and 2 depict the fixed-point numbers a' and b'. Lines 3 and 4 illustrate the rounding operation applied to their multiplication.

Algorithm 6 Aggregated Range Proof Protocol

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{mn}$, $u, P \in \mathbb{G}$, $\mathbf{v} \in \mathbb{F}^m$
Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{mn}$, $u, P \in \mathbb{G}$.
Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{mn}$, $u, P \in \mathbb{G}$, $\mathbf{v} \in \mathbb{F}^m$.
Ensure: Verifier accepts $P = \mathbf{g}_{[:m]}^{\mathbf{v}}$ and $v_j \in [0, 2^n - 1]$ for all $j \in [1, m]$.

1: Prover forms $\mathbf{a_L} \in \mathbb{F}^{mn}$ by concatenating the bits of all v_j such that $\left\langle \mathbf{2}^n, \mathbf{a}_{L[(j-1)n:jn-1]} \right\rangle = v_j$ for all $j \in [1, m]$.
2: Prover computes $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^{mn} \in \mathbb{F}^{mn}$ and $A = \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R} \in \mathbb{G}$.
3: Prover sends A to Verifier.
4: Verifier sends two random numbers $y, z \in \mathbb{F}$ to Prover.
5: Prover forms $\boldsymbol{\ell} = (\mathbf{a}_L - z \times \mathbf{1}^{nm}) \in \mathbb{F}^{nm}$, $\mathbf{r} = \mathbf{y}^{nm} \circ (\mathbf{a}_R + z \times \mathbf{1}^{nm}) + \sum_{j=1}^{m} z^{1+j} \left(\mathbf{0}^{(j-1)n} || \mathbf{2}^n || \mathbf{0}^{(m-j)n} \right) \in \mathbb{F}^{nm}$.
6: Prover computes $q(z) = \langle \mathbf{z}^m, \mathbf{v} \rangle$ and sends it to Verifier.
7: Verifier and Prover compute $P' = \mathbf{h}_{[:m]}^{\mathbf{z^m}}$.
8: Verifier and Prover run Algorithm 2 on the inputs $\left(\mathbf{g}_{[:m]}, \mathbf{h}_{[:m]}, u, P \times P', q(z), \mathbf{v}, \mathbf{z}^m \right)$. ▷ To ensure the correctness of $q(z)$.
9: Verifier and Prover compute $\delta(y, z) = (z - z^2) \times \langle \mathbf{1}^{mn}, \mathbf{y}^{mn} \rangle - \sum_{j=1}^{m} z^{j+2} \times \langle \mathbf{1}^n, \mathbf{2}^n \rangle$ and $\mathbf{h}' = \left(h_1, h_2^{\left(y^{-1}\right)}, h_3^{\left(y^{-2}\right)}, \ldots, h_{mn}^{\left(y^{-mn+1}\right)} \right)$.
10: Verifier and Prover compute $t = \delta(y, z) + z^2 \times q(z)$ ▷ t is the claimed $\langle \boldsymbol{\ell}, \mathbf{r} \rangle$.
11: Verifier and Prover compute $P'' = A \times \mathbf{g}^{-z} \times \mathbf{h'}^{z \times \mathbf{y}^{mn}} \times \prod_{j=1}^{m} \mathbf{h'}_{[(j-1)n:jn-1]}^{z^{j+1} \times \mathbf{2}^n}$. ▷ P'' is the claimed commitment for $\boldsymbol{\ell}$ and $\mathbf{r}$.
12: Verifier and Prover run Algorithm 2 on the inputs $(\mathbf{g}, \mathbf{h}', u, P'', t, \mathbf{r}, \boldsymbol{\ell})$. ▷ To ensure $t = \langle \boldsymbol{\ell}, \mathbf{r} \rangle$ and P'' is correct.

Algorithm 7 Sum-check protocol for equality check

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$

We have $f(\mathbf{x}) = \tilde{I}(\mathbf{s}, \mathbf{x}) \cdot \left(\tilde{a}^2(\mathbf{x}) - \tilde{y}^2(\mathbf{x})\right)$, where $\tilde{a}$ and $\tilde{y}$ are two MLEs, and $\mathbf{s}$ is a fixed vector.

Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$.

Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{2^v}$, $u, P_1, P_2 \in \mathbb{G}$, and the v-variate multilinear polynomial $f : \mathbb{F}_p^v \to \mathbb{F}_p$.

Ensure: Verifier accepts $0 = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$.

1: Prover sends $w = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$ to Verifier.
2: Prover sends the polynomial $f_1(X) = \sum_{(x_2, \ldots, x_v) \in \{0,1\}^{v-1}} f(X, x_2, \ldots, x_v)$
3: Verifier checks $w = f_1(0) = f_1(1)$.
4: Verifier sends a random number $r_1 \in \mathbb{F}_p$ to Prover.
5: **for** $i = 2, \ldots, v-1$ **do**
6: Prover sends the polynomial $f_i(X) = \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_v)$ to Verifier.
7: Verifier checks $g_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$.
8: Verifier sends a random number $r_i \in \mathbb{F}_p$ to Prover.
9: **end for**
10: Prover sends $f_v(X) = f(r_1, \ldots, r_{v-1}, X)$ to Verifier.
11: Verifier checks $f_{v-1}(r_{v-1}) = f_v(0) + f_v(1)$.
12: Verifier sends a random number $r_v \in \mathbb{F}_p$ to Prover.
13: Prover and Verifier run Algorithm 3 on the input $(\mathbf{g}, \mathbf{h}, u, P_1, \tilde{a}, (r_1, \ldots, r_v))$ for Verifier to obtain $\tilde{a}(r_1, \ldots, r_v)$.
14: Prover and Verifier run Algorithm 3 on the input $(\mathbf{g}, \mathbf{h}, u, P_2, \tilde{y}, (r_1, \ldots, r_v))$ for Verifier to obtain $\tilde{y}(r_1, \ldots, r_v)$.
15: Verifier computes $\tilde{I}(s_1, \ldots, s_v, r_1, \ldots, r_v)$.
16: Verifier commputes $f(r_1, \ldots, r_v) = \tilde{I}(s_1, \ldots, s_v, r_1, \ldots, r_v)\left(\tilde{a}(r_1, \ldots, r_v) - \tilde{y}(r_1, \ldots, r_v)\right)$ and checks $f_v(r_v) = f(r_1, \ldots, r_v)$.
If all checks pass, Verifier accepts $0 = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_v \in \{0,1\}} f(x_1, \ldots x_v)$.

Algorithm 8 Verifying ReLU activation layer

Require: The algorithm inputs are: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{nk}$, $u \in \mathbb{G}$, $\mathbf{A} \in \mathbb{F}^{n \times k}$, $P_A \in \mathbb{G}$.

Require: Verifier has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{nk}$, $u \in \mathbb{G}$, $P_A \in \mathbb{G}$.

Require: Prover has $\mathbf{g}, \mathbf{h} \in \mathbb{G}^{nk}$, $u \in \mathbb{G}$, $\mathbf{A} \in \mathbb{F}^{n \times k}$, $P_A \in \mathbb{G}$.

Ensure: Verifier receives P_B as the commitment to a matrix $\mathbf{B}$. Then Verifier accepts that the relation $\mathbf{B} = \frac{\mathbf{A}+|\mathbf{A}|}{2}$ holds.

1: Prover computes $\mathbf{Y} = |\mathbf{A}|$. ▷ The symbol $|.|$ denotes the absolute value.
2: Prover computes the commitment of $\mathbf{Y}$ as P_Y. Then sends P_Y to Verifier.
3: Prover and Verifier run Algorithm 6 on the input $\left(\mathbf{g}, \mathbf{h}, u, P_Y \in \mathbb{G}, \mathbf{Y} \in \mathbb{F}^{nk}\right)$. ▷ To verify all elements of $\mathbf{Y}$ are positive.
4: Prover computes $\mathbf{B} = \frac{\mathbf{A}+|\mathbf{A}|}{2}$ and its commitment, P_B, then sends P_B to Verifier.
5: Verifier selects a random vectors $\mathbf{s} \in \mathbb{F}^{\log nk}$ and sends $\mathbf{s}$ to Prover.
6: Prover constructs the polynomials $\tilde{a} : \mathbb{F}^{\log nk} \to \mathbb{F}$, $\tilde{y} : \mathbb{F}^{\log nk} \to \mathbb{F}$, $\tilde{I} : \mathbb{F}^{2 \log nk} \to \mathbb{F}$ and $f(\mathbf{x}) : \mathbb{F}^{\log nk} \to \mathbb{F}$ according to the description provided in Subsection 3.2.

Note: The commitments to the coefficients of the polynomial $\tilde{a}$, $\tilde{b}$, and $\tilde{y}$ are represented by P_A, P_B, and P_Y respectively, all of which are already held by the verifier. Furthermore, it is given that $f(\mathbf{x}) = \tilde{I}(\mathbf{s}, \mathbf{x}) \cdot \left(\tilde{a}^2(\mathbf{x}) - \tilde{y}^2(\mathbf{x})\right)$.

7: Prover and Verifier run Algorithm 7 on the input $(\mathbf{g}, \mathbf{h}, u, P_A, P_Y, f)$. Verifier accepts $\mathbf{Y} = |\mathbf{A}|$.
8: Verifier checks $P_B^2 = P_A \times P_Y$. ▷ To check $\mathbf{B} = \frac{\mathbf{A}+|\mathbf{A}|}{2}$ holds.
If all checks pass, Verifier accepts P_B as the commitment to a matrix $\mathbf{B}$, where $\mathbf{B} = \text{ReLU}(\mathbf{A})$.

Appendix B. Theorems and Proofs

Theorem .1. *Let a be a fixed-point number with 1 sign bit, t integer bits, and s fractional bits. Suppose the prover sends integers a'_1 and e_1 such that the following conditions hold:*

- ***Condition 1:*** $a \equiv e_1 + 2^s \cdot a'_1 \pmod{p}$*, where p is a prime with at least $t + s + 3$ bits.*
- ***Condition 2:*** $-2^{t+1} \leq a'_1 < 2^{t+1}$.
- ***Condition 3:*** $-2^{s-1} \leq e_1 < 2^{s-1}$.

Then the verifier can conclude that $a'_1 = \mathfrak{R}(a)$.

Proof. According to the definition,

$$\mathfrak{R}(a) = \frac{a + 2^{s-1} - (a + 2^{s-1} \mod 2^s)}{2^s},$$

if the prover has performed the calculations correctly, they would obtain the values $a'_2 = \mathfrak{R}(a)$ and $e_2 = (a + 2^{s-1} \mod 2^s) - 2^{s-1}$. It is straightforward to verify that these two values satisfy the aforementioned conditions. We will now proceed to demonstrate that $a'_2 = a'_1$ and $e_1 = e_2$.

From Condition 1, we observe that

$$a \equiv e_1 + 2^s \cdot a'_1 \equiv e_2 + 2^s \cdot a'_2 \pmod{p},$$

which implies that

$$p \mid (e_2 - e_1) + 2^s \times (a'_2 - a'_1).$$

Moreover, from Condition 2, it follows that

$$-2^{t+1} \leq a'_2 \leq 2^{t+1} - 1,$$

and

$$-2^{t+1} + 1 \leq -a'_1 \leq 2^{t+1},$$

which leads to

$$-2^{t+2} + 1 \leq a'_2 - a'_1 \leq 2^{t+2} - 1.$$

In addition, from Condition 3, we know that

$$-2^{s-1} \leq e_2 \leq 2^{s-1} - 1,$$

and

$$-2^{s-1} + 1 \leq -e_1 \leq 2^{s-1},$$

which implies that

$$-2^s + 1 < e_2 - e_1 < 2^s - 1.$$

Thus, we have the following inequality:

$$-2^{t+s+2} + 2^s \leq 2^s(a'_2 - a'_1) \leq 2^{t+s+2} - 2^s.$$

From these inequalities, we deduce that

$$-2^{t+s+2} + 1 \leq (e_2 - e_1) + 2^s \times (a'_2 - a'_1) \leq 2^{t+s+2} - 1.$$

Since p has at least $s + t + 3$ bits, we can conclude that

$$2^{s+t+2} \leq p.$$

Thus, since p divides $(e_2 - e_1) + 2^s \times (a'_2 - a'_1)$, and this expression is smaller in magnitude than p, the only possible solution is

$$(e_2 - e_1) + 2^s \times (a'_2 - a'_1) = 0.$$

Given the bounds on e_1 and e_2 as well as the multiple of 2^s, it follows that

$$e_1 = e_2,$$

and consequently,

$$a'_1 = a'_2.$$

□