An Efficient and Extensible Zero-knowledge Proof Framework for Neural Networks

Tao Lu
lutao2020@zju.edu.cn
Zhejiang University

Haoyu Wang
whaoyu@zju.edu.cn
Zhejiang University

Wenjie Qu
wenjiequ@u.nus.edu
National University of Singapore

Zonghui Wang
zhwang@zju.edu.cn
Zhejiang University

Jinye He
jin-yehe@outlook.com
National University of Singapore

Tianyang Tao
tianyangtao@u.nus.edu
National University of Singapore

Wenzhi Chen
chenwz@zju.edu.cn
Zhejiang University

Jiaheng Zhang
jhzhang@nus.edu.sg
National University of Singapore

ABSTRACT

In recent years, cloud vendors have started to supply paid services for data analysis by providing interfaces of their well-trained neural network models. However, customers lack tools to verify whether outcomes supplied by cloud vendors are correct inferences from particular models, in the face of lazy or malicious vendors. The cryptographic primitive called zero-knowledge proof (ZKP) addresses this problem. It enables the outcomes to be verifiable without leaking information about the models. Unfortunately, existing ZKP schemes for neural networks have high computational overheads, especially for the non-linear layers in neural networks.

In this paper, we propose an efficient and extensible ZKP framework for neural networks. Our work improves the performance of the proofs for non-linear layers. Compared to previous works relying on the technology of bit decomposition, we convert complex non-linear relations into range and exponent relations, which significantly reduces the number of constraints required to prove non-linear layers. Moreover, we adopt a modular design to make our framework compatible with more neural networks. Specifically, we propose two enhanced range and lookup proofs as basic blocks. They are efficient in proving the satisfaction of range and exponent relations. Then, we constrain the correct calculation of primitive non-linear operations using a small number of range and exponent relations. Finally, we build our ZKP framework from the primitive operations to the entire neural networks, offering the flexibility for expansion to various neural networks.

We implement our ZKPs for convolutional and transformer neural networks. The evaluation results show that our work achieves over 168.6× (up to 477.2×) speedup for separated non-linear layers and 41.4× speedup for the entire ResNet-101 convolutional neural network, when compared with the state-of-the-art work, Mystique. In addition, our work can prove GPT-2, a transformer neural network with 117 million parameters, in 287.1 seconds, achieving 35.7× speedup over ZKML, which is a state-of-the-art work supporting transformer neural networks.

1 INTRODUCTION

In the past decade, neural networks in deep learning have gained unprecedented focus in various fields, such as medical examination [28, 41], industrial fault detection [31, 33], natural language processing [48, 56, 61], and so on. Therefore, many customers are content with subscribing to a paid service known as Machine-Learning-as-a-Service (MLaaS), where cloud vendors such as Amazon [2], Google [26], and OpenAI [42] offer cloud-based platforms for data analysis through their deep learning models.

However, in the traditional MLaaS framework, customers lack tools to verify whether outcomes supplied by cloud vendors are indeed predictions of particular models because well-trained models are regarded as intellectual property and secret to customers. This lack of transparency leads to the outcomes provided by cloud vendors being untrusted, subsequently limiting the application of MLaaS in safety-critical domains such as medical examination, where any erroneous result may lead to serious consequences.

The cryptographic primitive called Zero-Knowledge Proof (ZKP) addresses this problem. It enables the owner of the secret model to convince customers that the results are correct predictions, while preserving the model's privacy. Considerable efforts [10, 16, 17, 35, 38, 40, 55, 60] have been directed toward developing ZKPs for neural networks, but their high computational costs make them hard to deploy practically. For example, a state-of-the-art work called zkCNN [40] requires more than 50 seconds to complete a proof for the inference of the VGG-11 model [53] with a single CIFAR-10 image [12] as input. It is thousands of times longer than performing the inference without zero-knowledge proofs.

We notice that the main focus of these previous works [17, 38, 40, 60, 68] is on the proof for linear layers in neural networks, such as convolutional and fully-connected layers. However, their inefficient schemes for non-linear layers hinder further improvement of the overall performance. For example, Mystique [60], a well-known ZKP scheme for convolutional neural networks (CNNs) [45], requires more than 90 percent of the total runtime for non-linear layer proofs. Moreover, in the era of large language models, transformer neural networks such as GPT [48, 49] are playing an increasingly important role. Nevertheless, most previous ZKP schemes [16, 17, 40, 55] for CNNs cannot be extended to handle complex non-linear layers in transformers, such as Softmax and GELU layers. While Mystique [60] and ZKML [10] are capable of generating proofs for complex non-linear layers, their huge computational

costs hinder their practical deployment, with the process taking several hours to produce non-linear layer proofs for GPT-2 [49].

In summary, the challenges related to non-linear layers are twofold. First, generating proofs for these non-linear layers is a significant performance bottleneck, reducing the overall efficiency of proving entire neural networks. Second, ZKP solutions [16, 40, 55] tailored for a specific non-linear layer struggle to be used in other non-linear layers. This limits the extension to more neural networks. Addressing the challenges associated with non-linear layers is crucial to developing ZKPs for neural networks.

In this paper, we propose an efficient and extensible ZKP framework for neural networks. In this framework, we employ Vector Oblivious Linear Evaluation (VOLE) as our proving backend. The VOLE backend is adept at proving large-scale computation with a small memory requirement, and thus it is well-suited for constructing zero-knowledge proofs for large-scale neural networks.

Our work addresses the challenges related to non-linear layer proofs. We employ the range and lookup proofs to prove the correct calculation of these non-linear layers, with our technique that utilizes a small number of range and exponent relations to constrain the correctness of non-linear operations. Compared to the bit decomposition method used in previous works [16, 38, 40, 55, 60], which requires decomposing each input and output of non-linear operations into Boolean values and proving that they satisfy particular Boolean circuits with hundreds of relations, our approach greatly reduces the required relations and improves the proving efficiency. Moreover, we adopt a modular design to make our framework compatible with more neural networks. Specifically, we propose two enhanced VOLE-based range and lookup proofs as basic blocks. They are efficient in proving the satisfaction of range and exponent relations. Then, we constrain the correct calculation of primitive non-linear operations using a small number of range and exponent relations. Finally, we build our ZKP framework from the primitive operations to the entire neural networks, offering the flexibility for expansion to various neural networks.

Our proposed VOLE-based range and lookup proofs are of independent interest. For the range proof, we follow previous works [13, 14] that leverage Legendre's theorem, which implies integers that can be decomposed into three integer squares are positive. Unfortunately, the step of identifying these three squares is a performance bottleneck in our proving framework. Hence, we optimize the searching method in [47], improving its expected time complexity from $O(b^2)$ to $O(b)$, where b denotes the bit length of integers. For the lookup proof, we adopt the subset lemma proposed in [19] to prove all lookup elements are in an exponent table, where the most time-consuming step is to sort the lookup elements. We perform the sorting process by grouping together elements with identical values, which is the optimal sorting method for our setting, where the elements exhibit only a few distinct values.

Our contributions. The following is a summary of contributions:

- We propose a ZKP framework for neural networks. Our framework achieves the following properties at the same time.
  - ***Efficient.*** Our work improves performance by optimizing non-linear layer proofs. We convert complex non-linear relations into range and exponent relations, which significantly reduces the number of constraints required to prove non-linear layers and thus improves the proving efficiency.
  - ***Practical.*** Our work is carefully designed to be compatible with existing well-studied neural networks that have high-accuracy performance. For example, we can straightforwardly generate proofs for various neural networks [30] in PyTorch [46], a well-known machine learning library.
  - ***Extensible.*** Our work adopts a modular design, offering the flexibility for expansion to various neural networks. We construct our ZKP framework from the fundamental primitive operations to the entire neural networks, such as convolutional and transformer neural networks.
- We propose two VOLE-based range and lookup proofs as the basic blocks of our ZKP framework for neural networks. They are of independent interest and fully compatible with other VOLE-based proofs [60, 63]. By improving the searching method to identify three squares in the range proof and employing an optimal sorting method tailored to our setting in the lookup proof, both proofs demonstrate outstanding efficiency and are competitive with other existing range and lookup proofs.
- We implement our zero-knowledge proofs for convolutional and transformer neural networks. The evaluation results show that our implementation achieves over 168.6× (up to 477.2×) speedup for separated non-linear layers and 41.4× speedup for the entire ResNet-101 convolutional neural network, when compared with the state-of-the-art VOLE-based work, Mystique [60]. In addition, our work can prove GPT-2, a transformer neural network with 117 million parameters, in 287.1 seconds achieving 35.7× speedup over ZKML [10], which is a state-of-the-art work that supports transformer neural networks.

1.1 Related Works

Zero-knowledge proof for neural networks. Considerable efforts have been directed toward developing zero-knowledge proofs for neural networks. Most previous works [16, 17, 35, 38, 40, 60, 68] primarily focus on the proof for linear layers using different proving backends such as zkSNARK [20, 27], GKR protocols [25, 62], and VOLE [3, 63]. To generate non-linear layer proofs, these works rely on bit decomposition, whose huge computational costs hinder their practical deployment. Recently, [17] proposed a ZENO language to provide a ZKP framework for neural networks, but it only supports CNNs and cannot be extended to other neural networks like transformers. The concurrent works [10, 29, 54] employ lookup proofs to construct non-linear layer proofs. Specifically, ZKML [10] employs Halo2 [67] as the ZKP backend to support transformer neural networks, but it requires several hours to generate a proof for GPT-2 [49]. The work [29] utilizes the lookup proof that is based on ZK-RAM [64] to build non-linear layer proofs. However, their scheme cannot be directly deployed into neural networks due to the lack of compatibility with existing networks in practice. zkLLM [54] is a GPU-accelerated scheme customized for transformer neural networks, while our work targets providing an efficient and extensible framework for various neural networks on CPUs. All of the above works are about the inference procedure of neural networks. Other recent works [21, 52, 55] attempt to generate proofs for the training procedure of machine learning models.

Range and lookup proofs. Range and lookup proofs serve as the basic blocks in numerous private-preserving applications [7, 22]. Many recent works have been dedicated to advancing these two types of proofs. For range proofs, existing works [8, 13, 14, 22, 57, 58] typically are divided into two categories [11] involving digital decomposition and square decomposition. Our range proof adopts square decomposition, whereas previous works [13, 14, 22] utilize the searching method in [47] to identify decomposed squares. This searching method becomes a performance bottleneck in our proving framework. Thus, we further optimize this method and improve its efficiency. For lookup proofs, previous works [15, 18, 51, 65, 66] mainly focus on optimizing proofs involving large lookup tables. Conversely, the lookup table used in our setting is relatively small. Hence, we adopt the subset lemma introduced in [19] to build the VOLE-based lookup proof, which is well-suited for our setting.

2 DESIGN OVERVIEW

At a high level, we design zero-knowledge proof (ZKP) for neural networks in machine-learning-as-a-service (MLaaS) setting, where we enable the service provider to supply, alongside the prediction outcome, a proof that convinces customers the outcome is really generated by the intended model. In addition, this proof leaks no information about the model's parameters.

To make our work compatible with a broader range of neural networks, we adopt a modular design. Figure 1 outlines the overview of our framework. First, we propose two enhanced range and lookup proofs as basic blocks. They are efficient in proving the satisfaction of range and exponent relations. Then, we constrain the correct calculation of non-linear operations with a small number of range and exponent relations. Finally, we assemble the design modules to build our ZKP scheme for the entire neural network. Below, we give insights into our design modules.

Basic block 1: Range proof. Range proof is the basic block to prove a secret value x is within a certain range, such as $[0, B]$, where B is a range bound. In our setting, the requirement for millions of range proofs makes their efficiency significantly affect the performance of the entire framework. However, existing range proofs [8, 13, 14, 57, 58] cannot meet our efficiency requirement.

Our range proof is constructed based on the three-square decomposition lemma, which states that there exist three integers $\{y_i\}_{i\in[1,3]}$ such that $4x(B-x)+1 = \sum_{i=1}^{3} y_i^2$ if and only if x being in the range $[0, B]$. To efficiently find these three integers, we optimize the searching method in [47], improving its expected time complexity from $O(b^2)$ to $O(b)$, where b denotes the bit length of x. Our method initially identifies the largest even integer y_1 such that $4x(B-x)+1 = y_1^2 + q$ and q is a prime. Next, Fermat's two-square theorem states that the prime q can be expressed as the sum of two squares when $q \equiv 1 \pmod 4$. We determine two integers y_2 and y_3 from a precomputed table containing two-square decompositions of primes. This table does not need to be large because the prime q is small in most of our cases. In the case when q is large, we can still use the method in [47] to find the decomposed integers.

One remaining issue is that the above approach only works on integers, not on finite field elements. We bridge this gap by leveraging the shortness test [13] to ensure that x and $\{y_i\}_{i\in[1,3]}$ are small enough so that the phenomenon of wrapping around in the finite field cannot occur. In this case, the finite field elements can effectively be treated as integers. Section 4 presents more details about our range proof.

Basic block 2: Lookup proof. Lookup proof is the basic block to prove a secret vector $\boldsymbol{x}$ belongs to a public table $\{\boldsymbol{t}_i\}_{i\in[1,d]}$, where d denotes the table size. In our setting, we utilize lookup proof to prove the satisfaction of exponent relations. As shown in Figure 2, the pair of two numbers appearing in the exponent table infers their exponent relation, where the exponent table holds integers and their corresponding powers. In our setting, we only need a small exponent table with a few hundred elements. Therefore, we should design our lookup proof that is suitable for this setting.

We construct our lookup proof based on the subset lemma [19], which states that the equivalence of two polynomials g and h leads to an ordered set $\mathbf{X} := \{\boldsymbol{x}_i\}_{i\in[1,n]}$ being the subset of a table $\mathbf{T} := \{\boldsymbol{t}_i\}_{i\in[1,d]}$, indicating each $\boldsymbol{x}_i$ belongs to the table $\mathbf{T}$. The polynomial g is related to the public table $\mathbf{T}$ and the set $\mathbf{X}$ known by the prover, while the polynomial h is associated with the set $\mathbf{S}$, representing the union set $\{\mathbf{X}, \mathbf{T}\}$ sorted by $\mathbf{T}$, which is not readily available to the prover.

Therefore, the prover must obtain $\mathbf{S}$ before proving the equivalence of two polynomials. Although using the quick sort algorithm to obtain $\mathbf{S}$ has a time complexity of $O(n \log n)$, we perform the sorting process by grouping together elements with identical values, which is the optimal sorting method for our setting where the elements exhibit only a few distinct values. This adjustment allows us to obtain $\mathbf{S}$ in $O(n)$ time complexity. With $\mathbf{S}$ prepared, we prove the equivalence of two polynomials using the polynomial proof studied in [63]. Section 5 gives details about our lookup proof.

Constraints for non-linear operations. Our work utilizes the range and exponent relations to constrain the correct calculation of non-linear operations used in neural networks. We complete the constraining process with two steps. First, we employ the range and exponent relations to constrain the correct calculation of primitive non-linear operations, such as round and bit-shift operations, with details in Section 6.1. Second, We employ these primitive operations to build the constraints for non-linear layers in neural networks, with details in Section 6.2. For an intuitive demonstration, we show a simple example to explain the logic behind our method.

In this example, we use $B = 2^b$ and $0 \le x - By < B$ to constrain the result y calculated from b-bit right shift on integer x. Logically, the b-bit right shift means discarding the last b bits of x. Therefore, the difference between x and By corresponds to the value of the last b bits, which is less than $B = 2^b$. Therefore, we have $0 \le x - By < B$. Furthermore, we could prove the integer y satisfying the above two relations is unique. If there is another $\bar{y}$ satisfying $0 \le x - B\bar{y} < B$, we set $\bar{y} = y + t$, where t is a non-zero integer. Then, we have $0 \le x - B(y+t) < B$, which conflicts with $0 \le x - By < B$.

In summary, our method only requires a single range relation and a single exponent relation to constrain the correct calculation of the right shift operation. We achieve a reduction by around two orders of magnitude compared to the previous works using bit decomposition. Such works prove each bit of the input x and output y is correctly decomposed and the previous bits in x are the same as the bits in y, requiring hundreds of relations in total.

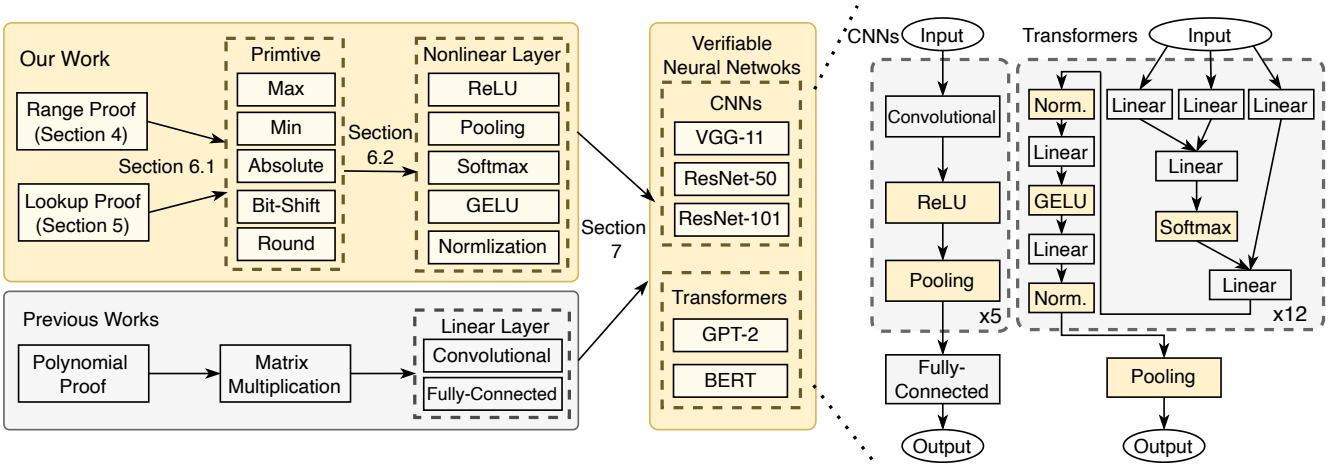

Figure 1: A design overview of our framework.

Assemble together. We assemble the design modules into our ZKP framework for neural networks. The insight of this step is straightforward. Briefly, we convert the non-linear relations in neural networks into range and exponent relations, whose satisfaction is proven through our range and lookup proofs, respectively. For the linear relations in neural networks, we convert them to matrix multiplication, and its correct calculation is proven through polynomial proofs discussed in previous studies [60, 63]. This approach ensures that both linear and non-linear relations are addressed within our ZKP framework.

However, an issue remains. Typically, the calculation of neural networks involves floating-point operations, but these operations are not ZKP-friendly. For example, merely proving the addition of two floating-point numbers requires more than a hundred constraints [23]. Large-scale neural networks cannot tolerate such overhead. As a solution, existing ZKP schemes [16, 38, 40, 60] for neural networks, including ours, opt for approximating these floating-point numbers to integers or fixed-point numbers.

To minimize the loss of accuracy caused by approximation, we design our ZKP framework to make it compatible with existing approximated neural networks that have high accuracy. For example, our framework can work with the approximated ResNet-101 networks in PyTorch [46], a prominent library for machine learning. It achieves an accuracy of 93.79% on the CIFAR-10 dataset, dropped by only 0.04% compared to the original ResNet-101 network.

3 PRELIMINARIES

3.1 Notation

We use λ and ρ to denote the computational and statistical security parameters, respectively. We use $\mathbb{F}_p$ to denote a finite field with prime order p. We use $x \leftarrow S$ to denote that sampling x uniformly at random from a finite set S. For $a, b \in \mathbb{N}$, we denote the set $\{a, ..., b\}$ by $[a, b]$, and the set $\{a, ..., b-1\}$ by $[a, b)$. We employ the finite field $\mathbb{F}_p$ to encode the signed integers between $[-\frac{p-1}{2}, \frac{p-1}{2}]$ in a natural way. We use bold lower-case letters like $\boldsymbol{x}$ to denote vectors, and x_i is the i-th element in $\boldsymbol{x}$.

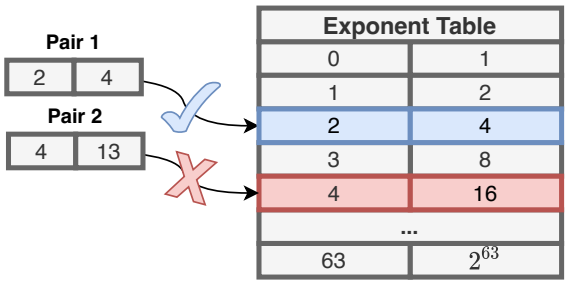

Figure 2: The exponent table used to prove the exponent relations between two numbers.

3.2 VOLE and IT-MACs

Vector Oblivious Linear Evaluation (VOLE) [5] is a secure two-party protocol for generating correlations that consist of a global key $\Delta \in \mathbb{F}_p$ and three vectors $(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{k})$ with the relations $\mathsf{m}_i = \mathsf{k}_i - \Delta \cdot x_i$ for $i \in [1, n]$, where $\boldsymbol{x}$, $\boldsymbol{m}$, and $\boldsymbol{k}$ belong to $(\mathbb{F}_p)^n$. In our setting, the VOLE correlation is used as Information Theoretic Message Authentication Codes (IT-MACs) to authenticate values over $\mathbb{F}_p$. Specifically, let $\Delta \in \mathbb{F}_p$ be a global key, known only by the verifier $\mathcal{V}$. A value $x \in \mathbb{F}_p$ known by the prover $\mathcal{P}$ is authenticated following the functionality in Figure 3 by giving $\mathcal{V}$ a local key $\mathsf{k} \in \mathbb{F}_p$ and giving $\mathcal{P}$ a tag $\mathsf{m} \in \mathbb{F}_p$. We denote the authentication code by $[x] = (x, \mathsf{m}, \mathsf{k})$, where $\mathcal{P}$ holds (x, m) and $\mathcal{V}$ holds k.

The random values can be efficiently authenticated by the recent LPN-based VOLE protocols [6, 59], exhibiting communication complexity that is sublinear relative to the number of random values. To authenticate the non-random value x, $\mathcal{P}$ and $\mathcal{V}$ begin by getting an authenticated random value $[r]$. Next, $\mathcal{P}$ sends the difference $d = x - r \in \mathbb{F}_q$ to $\mathcal{V}$ and then both parties compute $[x] = [r] + d$, leveraging the additive homomorphic property inherent in VOLE correlations. Specifically, given $[x_1], ..., [x_n]$ and public coefficients $c_0, c_1, ..., c_n$, the parties can locally compute $[y] = \sum_{i=1}^{n} c_i[x_i] + c_0$ without any communication.

Functionality $\mathsf{Auth}(x)$

This functionality should be performed after the global key Δ being generated. On input $x \in \mathbb{F}_p$, this functionality interacts with the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, and outputs the authentication code $[x] = (x, \mathsf{m}, \mathsf{k})$ that satisfys $\mathsf{m} = \mathsf{k} - \Delta \cdot x$.

(1) If $\mathcal{V}$ is honest, sample $\mathsf{k} \leftarrow \mathbb{F}_p$. Otherwise, receive $\mathsf{k} \in \mathbb{F}_p$ from the adversary.
(2) If $\mathcal{P}$ is honest, compute $\mathsf{m} := \mathsf{k} - \Delta \cdot x \in \mathbb{F}_p$. Otherwise, receive $\mathsf{m} \in \mathbb{F}_p$ from the adversary and recompute $\mathsf{k} := \mathsf{m} + \Delta \cdot x \in \mathbb{F}_p$.
(3) Output $[x]$ to parties by sending (x, m) to $\mathcal{P}$ and k to $\mathcal{V}$.

Figure 3: The functionality for generating the authentication code using the VOLE correlation.

3.3 Zero-knowledge Proof

Zero-knowledge proof (ZKP) is a cryptographic primitive that allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a statement is true without leaking any information about the prover's secret witness. Same as other VOLE-based zero-knowledge proofs [63], we present the definition of zero-knowledge proof in the Universal Composability (UC) framework [9] with the ideal functionality $\mathcal{F}_{\mathsf{authZK}}$ shown in Figure 4. UC framework is the general-purpose security framework for the analysis of cryptographic protocols. Briefly, we say that a protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if we cannot distinguish the world where the prover and the verifier execute the real protocol Π and the world where two parties perform the ideal functionality $\mathcal{F}$. A more formal description about UC framework are shown in Appendix A.

Recent works [4, 63] have explored efficient VOLE-based proving protocols that achieve the polynomial command of the ideal functionality $\mathcal{F}_{\mathsf{authZK}}$. These protocols follow the commit-and-prove paradigm, where they consider authentication codes as a form of commitment and then prove the committed values satisfy particular relations by running a consistency-check procedure. In this paper, we support proving protocols that achieve other commands of the ideal functionality listed in Figure 4 using the same commit-and-prove paradigm. These protocols are basic blocks of our ZKP framework for neural networks.

3.4 Tools for Range Proof

3.4.1 Three-Square Decomposition. Legendre's three-square theorem states that a natural number x can be represented as the sum of three squares of integers ($x = y_1^2 + y_2^2 + y_3^2$) if and only if x is not of the form $x = 4^m(8n + 7)$ for non-negative integers m and n. This theorem infers the following lemma, showing the necessary and sufficient condition for an integer x to be in the range $[0, B]$.

Lemma 3.1. *Let $x \in \mathbb{Z}$ be an integer. It holds that $x \in [0, B]$ if and only if there exists three integers y_1, y_2, y_3 satisfying $4x(B - x) + 1 = y_1^2 + y_2^2 + y_3^2$.*

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

$[x] = (x, \mathsf{m}, \mathsf{k})$ is denoted as an authentication code, where the prover $\mathcal{P}$ holds (x, m) and the verifier $\mathcal{V}$ holds k.
Init: On input (init) from the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, sample $\Delta \leftarrow \mathbb{F}_p$ if $\mathcal{V}$ is honest. Otherwise, receive $\Delta \in \mathbb{F}_p$ from adversary. Store Δ and send it to $\mathcal{V}$, and then ignore all subsequent (init) commands.
Auth: This procedure can be run multiple times. On input (auth, id, x) from $\mathcal{P}$ and (auth, id) from $\mathcal{V}$, perform $\mathsf{Auth}(x)$ if id is fresh identifier and $x \in \mathbb{F}_p$. The functionality of $\mathsf{Auth}(x)$ is shown in Figure 3. Finally, store (id, x).

Commands for zero-knowledge proof

Polynomial: On the input (poly, p, $\{\mathsf{id}_i\}_{i\in[1,n]}$) from $\mathcal{P}$ and $\mathcal{V}$, retrieve $\{(\mathsf{id}_i, x_i)\}_{i\in[1,n]}$ if each id_i is present in memory, or send false to $\mathcal{V}$ and abort. Next, send true to $\mathcal{V}$ if $p(x_1, x_2, ..., x_n) = 0$, or send false to $\mathcal{V}$ otherwise. Here, p represents a polynomial.
Range: On the input (range, B, $\{\mathsf{id}_i\}_{i\in[1,n]}$) from $\mathcal{P}$ and $\mathcal{V}$, retrieve $\{(\mathsf{id}_i, x_i)\}_{i\in[1,n]}$ if each id_i is present in memory, or send false to $\mathcal{V}$ and abort. Send true to $\mathcal{V}$ if all $x_i \in [0, B]$, or send false to $\mathcal{V}$. Here, B represents a range bound.
Lookup: Set the table $\mathbf{T} := \{\boldsymbol{t}_j\}_{j\in[1,d]}$ and $\boldsymbol{t}_j \in (\mathbb{F}_p)^w$. Let $\boldsymbol{id}_i := (\mathsf{id}_{i1}, ..., \mathsf{id}_{iw})$ and $\boldsymbol{x}_i \in (\mathbb{F}_p)^w$. On the input (lookup, $\mathbf{T}$, $\{\boldsymbol{id}_i\}_{i\in[1,n]}$) from $\mathcal{P}$ and $\mathcal{V}$, retrieve $\{(\boldsymbol{id}_i, \boldsymbol{x}_i)\}_{i\in[1,n]}$ if each $\boldsymbol{id}_i$ is present in memory, or send false to $\mathcal{V}$ and abort. Next, send true to $\mathcal{V}$ if all $\boldsymbol{x}_i \in \mathbf{T}$, or send false to $\mathcal{V}$.

Figure 4: The ideal functionality for zero-knowledge proof with the authentication codes.

Furthermore, the three integers $\{y_i\}_{i\in[1,3]}$ can be identified via the searching method proposed in [47], with an expected time complexity of $O(b^2)$, where b denotes the bit length of x.

3.4.2 Masking Scheme. We use an additive masking method to prevent information leakage in our range proof. The masking algorithm for value v in range $[0, V]$ works as follows: (1) Randomly pick a value r from $[0, VL]$. We call r the mask and $L \geq 1$ the masking overhead. (2) Perform $\mathsf{mask}(v, r)$ that outputs $v + r$ if $v + r \in [V, VL]$ or $\perp$ otherwise. This masking algorithm has the properties shown in the following lemma.

Lemma 3.2. *Let $L \geq 1$ be the masking overhead and r is uniformly chosen from $[0, VL]$. For any value $v \in [0, V]$, it holds that the probability of $\mathsf{mask}(v, r)$ outputting $\perp$ is $\frac{V}{1+VL}$ which is less than $1/L$. The distribution of the output of $\mathsf{mask}(v, r)$ is uniform over $[V, VL]$ if $\mathsf{mask}(v, r)$ does not outputs $\perp$.*

3.4.3 Shortness Test. We use the binary affine shortness test [13] to check whether all elements of a vector $\boldsymbol{x} \in (\mathbb{F}_p)^n$ are inside a specified range $[-K, K]_{\mathbb{F}_p}$. Specifically, for any vector $\boldsymbol{x} \in (\mathbb{F}_p)^n$ and any affine value $\mu \in \mathbb{F}_p$, the binary affine shortness test $\mathsf{BAST}(\boldsymbol{x}, \mu, K)$ works as follows: (1) Randomly pick a binary vector $\boldsymbol{\gamma} \in \{0, 1\}^n$. (2) Compute $\zeta = \mu + \sum_{i=1}^n \gamma_i x_i$, and then output true if $\zeta \in [0, K]_{\mathbb{F}_p}$

or false otherwise. This shortness test method has the soundness property shown in the following lemma.

Lemma 3.3. *Let* BAST *be the binary affine shortness test for any vector* $\boldsymbol{x} \in (\mathbb{F}_p)^n$. *Let* K *be the range bound and* $K \leq (p-1)/2$. *For any affine value* $\mu \in \mathbb{F}_p$, *if there is* $x_i \in \boldsymbol{x}$ *not in the range* $[-K, K]_{\mathbb{F}_p}$, *the probability of* $\mathsf{BAST}(\boldsymbol{x}, \mu, K)$ *outputs* true *is at most* $\frac{1}{2}$.

Note that Lemma 3.3 does not guarantee the correctness of the shortness test, that is, if all elements of the vector $\boldsymbol{x} \in (\mathbb{F}_p)^n$ are in the range $[-K, K]_{\mathbb{F}_p}$, we cannot imply the output of BAST is true.

3.5 Tools for Lookup Proof

3.5.1 Subset Lemma. We use the subset lemma proposed in [19] to prove an ordered set of vectors $\mathbf{X} := \{\boldsymbol{x}_i\}_{i\in[1,n]}$ being the subset of the table $\mathbf{T} := \{\boldsymbol{t}_j\}_{j\in[1,d]}$. Let $\mathbf{S} := \{\boldsymbol{s}_i\}_{i\in[1,n+d]}$. We define two polynomials g and h with α, β, γ as variables:

$$\begin{aligned} g(\alpha,\beta,\gamma) := (1+\beta)^n \cdot \prod_{i=1}^{n} (\gamma + \sum_{k=1}^{w} \alpha^{k-1} x_{ik}) \\ \cdot \prod_{i=1}^{d-1} (\gamma(1+\beta) + \sum_{k=1}^{w} \alpha^{k-1}(t_{ik} + \beta t_{(i+1)k})) \end{aligned} \tag{1}$$

$$h(\alpha,\beta,\gamma) := \prod_{i=1}^{n+d-1} (\gamma(1+\beta) + \sum_{k=1}^{w} \alpha^{k-1}(s_{ik} + \beta s_{(i+1)k})) \tag{2}$$

, where x_{ik}, t_{ik}, s_{ik} are the k-th elements of the vectors $\boldsymbol{x}_i, \boldsymbol{t}_i, \boldsymbol{s}_i$, respectively. The following lemma shows the necessary and sufficient condition for $\mathbf{X}$ to be a subset of $\mathbf{T}$.

Lemma 3.4. *Given the ordered sets* $\mathbf{X}$, $\mathbf{T}$, *and* $\mathbf{S}$, *let* g *and* h *are two polynomials described by Equation (1) and Equation (2) with* α, β, γ *as variables. It holds* $g(\alpha,\beta,\gamma) \equiv h(\alpha,\beta,\gamma)$ *if and only if* $\mathbf{X}$ *is the subset of* $\mathbf{T}$ *and* $\mathbf{S}$ *is the union set* $\{\mathbf{X}, \mathbf{T}\}$ *sorted by* $\mathbf{T}$.

Here, the ordered set $\mathbf{S}$ is sorted by $\mathbf{T}$ when all elements in $\mathbf{S}$ are in the same order as they appear in $\mathbf{T}$. Formally, for any $i < i'$ such that $\boldsymbol{s}_i \neq \boldsymbol{s}_{i'}$, if there exists j, j' that $\boldsymbol{s}_i = \boldsymbol{t}_j, \boldsymbol{s}_{i'} = \boldsymbol{t}_{j'}$, then $j < j'$.

4 ZERO-KNOWLEDGE RANGE PROOF

Zero-knowledge range proof enables the prover to convince the verifier that a secret value is in a certain range. It serves as a fundamental component in numerous private-preserving applications [7, 22] as well as the basic block of our ZKP framework for neural networks. In this section, we present a VOLE-based range proof, taking inspiration from prior works [13, 14] that leverage the three-square decomposition technique based on Lemma 3.1. Our approach enhances efficiency significantly by improving the search algorithm [47] for identifying three-square decompositions.

4.1 Scheme

In our range proof, two parties are given public inputs that include a range bound B and identifiers $\{\mathsf{id}_i^x\}_{i\in[1,n]}$ corresponding to authentication codes $\{[x_i]\}_{i\in[1,n]}$. It guarantees that the verifier only outputs true when each x_i is in the range $[0, B]$ for $i \in [1, n]$. Below, we outline the core ideas behind our range proof, with the full details described in Figure 5.

Initially, the prover checks that each x_i is in the range $[0, B]$ by its own. If all checks pass, the prover can identify three integers $\{y_{ij}\}_{j\in[1,3]}$ such that $4x_i(B-x_i)+1 = \sum_{j=1}^{3} y_{ij}^2$, which infers the corresponding integer x_i is in the range $[0, B]$ based on Lemma 3.1.

To efficiently find these three integers, we improve the searching method in [47]. First, we identify the largest even integer y_{i1} such that $4x_i(B-x_i)+1 = y_{i1}^2 + q_i$, where q_i is a prime. This step is achieved by trying multiple y_{i1} from large to small, and the probability of each attempt being successful is $O(1/b)$ according to the distribution of primes appearing among integers, where b denotes the bit length of x_i. Second, Fermat's two-square theorem states that the prime q_i can be expressed as the sum of two squares when $q_i \equiv 1 \pmod 4$. We determine these two integers y_{i2} and y_{i3} from a precomputed table that contains two-square decompositions of primes. This table does not need to be large because the prime q_i is small in most our cases. Querying two-square decompositions from the table can be completed in $O(1)$ time complexity. Thus, the expected time complexity of our approach is $O(b)$.

Protocol Π_{Range}

Let R be the repetition number of the shortness tests shown in Section 3.4.3, L be the masking overhead shown in Section 3.4.2, and n be the scale of input.

Input: Two parties $\mathcal{P}$ and $\mathcal{V}$ receive public inputs including a range bound B and identifiers $\{\mathsf{id}_i^x\}_{i\in[1,n]}$, which correspond to authentication codes $\{[x_i]\}_{i\in[1,n]}$, where each $x_i \in \mathbb{F}_p$.

Range Prove: $\mathcal{P}$ convinces $\mathcal{V}$ that each x_i is in the range $[0, B]$ for $i \in [1, n]$ as follows:

(1) If any $x_i \notin [0, B]$ for $i \in [1, n]$, $\mathcal{P}$ sends $\perp$ to $\mathcal{V}$, and $\mathcal{V}$ outputs false and aborts. Next, $\mathcal{P}$ identifies y_{i1}, y_{i2}, y_{i3} that satisfy $4x_i(B-x_i)+1 = \sum_{j=1}^{3} y_{ij}^2$.
(2) For $i \in [1, n]$ and $j \in [1, 3]$, $\mathcal{P}$ and $\mathcal{V}$ select identifiers id_{ij}^y. Next, they invoke the (auth) command of $\mathcal{F}_{\mathsf{authZK}}$, which returns authentication codes $[y_{ij}]$ to the parties. For $k \in [1, R]$, $\mathcal{P}$ samples $m_k \leftarrow [0, 4nBL]$. $\mathcal{P}$ and $\mathcal{V}$ select identifiers id_k^m. They invoke the (auth) command of $\mathcal{F}_{\mathsf{authZK}}$, which returns $[m_k]$ to the parties.
(3) $\mathcal{V}$ samples shortness testing challenges $\gamma_{ijk} \leftarrow [0, 1]$ for $i \in [1, n]$, $j \in [0, 3]$, $k \in [1, R]$ and sends them to $\mathcal{P}$. Next, $\mathcal{P}$ computes $\zeta_k = \mathsf{mask}\ (\sum_{i=1}^{n}\sum_{j=0}^{3} \gamma_{ijk} y_{ij}, m_k)$ and responds ζ_k to $\mathcal{V}$, where $y_{i0} := x_i$.
(4) $\mathcal{V}$ checks $\zeta_k \neq \perp$ and $\zeta_k \in [4nB, 4nBL]$ for all $k \in [1, R]$. If any check fails, $\mathcal{V}$ outputs false and aborts.
(5) Set $f(x, y_1, y_2, y_3) := \sum_{j=1}^{3} y_i^2 - 4x(B-x) - 1$. $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{poly}, f, \{\mathsf{id}_i^x, \mathsf{id}_{i1}^y, \mathsf{id}_{i2}^y, \mathsf{id}_{i3}^y\})$ to $\mathcal{F}_{\mathsf{authZK}}$ for all $i \in [1, n]$. $\mathcal{V}$ outputs false once $\mathcal{F}_{\mathsf{authZK}}$ returns false.
(6) For $k \in [1, R]$, $\mathcal{P}$ and $\mathcal{V}$ locally compute $[z_k] := [m_k] + \sum_{i=1}^{n}\sum_{j=0}^{3} \gamma_{ijk}[y_{ij}] - \zeta_k$, which is achieved using the additive homomorphic property. Finally, they execute CheckZero on $[z_k]$, where $\mathcal{P}$ sends the tag of $[z_k]$ to $\mathcal{V}$ who checks if it equals the local key of $[z_k]$. If any check fails, $\mathcal{V}$ outputs false. Otherwise, $\mathcal{V}$ outputs true.

Figure 5: Protocol for the zero-knowledge range proof in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.

Next, the prover would employ the polynomial proof to prove the satisfaction of the equation $4x_i(B-x_i)+1 = \sum_{j=1}^{3} y_{ij}^2$. However, an issue remains. Our ZKP framework works on the finite field $\mathbb{F}_p$. The phenomenon of wrapping around in the finite field could cause this equation to hold over the finite field, but not hold over integers.

We use the shortness test shown in Section 3.4.3 to ensure that x_i and $\{y_{ij}\}_{j\in[1,3]}$ are small enough that wrapping around does not occur. Specifically, the verifier picks $4nR$ random binary challenges and sends these challenges to the prover, where R is the repeat number to reduce the soundness error in the shortness test. The prover responds with the linear combination of these binary challenges and $\{y_{ij}\}_{j\in[0,3]}$ with $y_{i0} := x_i$. To avoid leaking information about the secret x_i, we mask the linear combination value using the additive masking method shown in Section 3.4.2.

Finally, the following three points are checked by the verifier to ensure that x_i is in the range $[0, B]$ for $i \in [1, n]$:

- The challenge responses from the prover pass all shortness tests. It is achieved by checking the response values are in the range $[4nB, 4nBL]$, where L is the masking overhead.
- The decomposed three elements satisfy the equation $4x_i(B-x_i)+1 = \sum_{j=1}^{3} y_{ij}^2$. It is achieved by invoking the (poly) command of $\mathcal{F}_{\mathsf{authZK}}$ to verify the evaluation of polynomial $f(x_i, y_{i1}, y_{i2}, y_{i3}) = \sum_{j=1}^{3} y_{ij}^2 - 4x_i(B-x_i) - 1$ equals zero for all $i \in [1, n]$.
- The shortness test challenge responses from the prover are indeed masked linear combinations. It is achieved by locally subtracting the authentication codes of these linear combinations from the response values using the additive homomorphic property. Subsequently, they execute CheckZero, where the prover forwards tags of the authenticated results to the verifier, who checks that these tags are equal to the corresponding local keys.

Theorem 4.1. *Let B be the range bound, R be the repetition number of shortness tests in Section 3.4.3, L be the masking overhead shown in Section 3.4.2, n be the scale of input, and $K := 4nBL$. If $4K^2 + 4BK + 1 \le \frac{p-1}{2}$, the protocol Π_{Range} shown in Figure 5 UC-realizes the* (range) *command of functionality $\mathcal{F}_{\mathsf{authZK}}$ in the presence of a static, malicious adversary with correctness error at most $\frac{R}{L}$ and statistical soundness error at most $\max(\frac{R}{L}, \frac{1}{p} + \frac{1}{2^R})$ in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.*

The security proof of this theorem can be found in Appendix B. Given the range bound B, the number n, and a finite field $\mathbb{F}_p$, we can choose suitable parameters L and R such that $\frac{R}{L} \le 2^{-\rho}$ and $\frac{1}{p} + \frac{1}{2^R} \le 2^{-\rho}$. For example, when $B = 2^{32}$, $n = 2^{20}$, and $p \approx 2^{254}$, we can choose $L = 2^{46}$ and $R = 42$ to achieve at least 40-bit statistical security with $4K^2 + 4BK + 1 \le \frac{p-1}{2}$.

5 ZERO-KNOWLEDGE LOOKUP PROOF

Zero-knowledge lookup proof allows the prover to convince the verifier that a secret vector is in a public table. We utilize lookup proof to prove exponent relations between two numbers by showing that they appear in the same row of the exponent table, as shown in Figure 2. In our setting, we only need a small exponent table with a few hundred elements. Therefore, we employ the subset lemma [20] shown in Section 3.5 to design a VOLE-based lookup proof tailored to this setting.

Protocol Π_{Lookup}

Let d be the table size, w be the dimensions of each element in the table, and n be the number of lookups.

Input: The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ are given public inputs that include a table $\mathbf{T} := \{\boldsymbol{t}_j\}_{j\in[1,d]}$ and identifiers $\{\boldsymbol{id}_i^x\}_{i\in[1,n]}$, where $\boldsymbol{t}_j \in (\mathbb{F}_p)^w$ and $\boldsymbol{id}_i^x := (\mathrm{id}_{i1}^x, ..., \mathrm{id}_{iw}^x)$. These identifiers are corresponding to authentication codes $\{[\boldsymbol{x}_i]\}_{i\in[1,n]}$ with $\boldsymbol{x}_i \in (\mathbb{F}_p)^w$.

Lookup Prove: $\mathcal{P}$ convinces $\mathcal{V}$ that all elements of $\mathbf{X}$ are in the table $\mathbf{T}$ as follows:

(1) If any $\boldsymbol{x}_i \notin \mathbf{T}$ for $i \in [1, n]$, $\mathcal{P}$ sends $\perp$ to $\mathcal{V}$, and $\mathcal{V}$ outputs false and aborts. Otherwise, $\mathcal{P}$ calculate the set $\mathbf{S} := \{\boldsymbol{s}_i\}_{i\in[1,n+d]}$ that is the union set $\{\mathbf{X}, \mathbf{T}\}$ sorted by $\mathbf{T}$.
(2) For $i \in [1, n+d]$, $\mathcal{P}$ and $\mathcal{V}$ select identifiers $\boldsymbol{id}_i^s$. Next, they invoke the (auth) command of $\mathcal{F}_{\mathsf{authZK}}$, which returns authentication codes $[\boldsymbol{s}_i]$ to the parties.
(3) $\mathcal{V}$ samples three random elements (r_1, r_2, r_3) from the finite field $\mathbb{F}_p$, and sends them to $\mathcal{P}$.
(4) $\mathcal{P}$ and $\mathcal{V}$ define two polynomial $g'(\mathbf{X})$ and $h'(\mathbf{S})$ that are same as $g(\alpha, \beta, \gamma)$ and $h(\alpha, \beta, \gamma)$ in Section 3.5 but with $\mathbf{X}, \mathbf{S}$ as variables and $(\alpha, \beta, \gamma) = (r_1, r_2, r_3)$ as coefficients. $\mathcal{P}$ and $\mathcal{V}$ define the polynomial $l'(\mathbf{X}, \mathbf{S}) := g'(\mathbf{X}) - h'(\mathbf{S})$, and send $(\mathrm{poly}, l', (\{\boldsymbol{id}_i^x\}_{i\in[1,n]}, \{\boldsymbol{id}_i^s\}_{i\in[1,n+d]}))$ to $\mathcal{F}_{\mathsf{authZK}}$. Finally, $\mathcal{V}$ outputs what $\mathcal{F}_{\mathsf{authZK}}$ outputs.

Figure 6: Protocol for the zero-knowledge lookup proof in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.

5.1 Scheme

In our lookup proof, two parties are given public inputs that include a table denoted as $\mathbf{T} := \{\boldsymbol{t}_j\}_{j\in[1,d]}$ and identifiers $\{\boldsymbol{id}_i^x\}_{i\in[1,n]}$, where $\boldsymbol{t}_j \in (\mathbb{F}_p)^w$ and $\boldsymbol{id}_i^x := (\mathrm{id}_{i1}^x, ..., \mathrm{id}_{iw}^x)$. These identifiers are corresponding to authentication codes $\{[\boldsymbol{x}_i]\}_{i\in[1,n]}$ with $\boldsymbol{x}_i \in (\mathbb{F}_p)^w$. The protocol guarantees that the verifier only outputs true when $\boldsymbol{x}_i$ are in the table $\mathbf{T}$ for all $i \in [1, n]$. Below, we outline the key ideas of our scheme that implements the lookup proof. The details of our scheme are described in Figure 6.

First, the prover checks that each $\boldsymbol{x}_i$ is in the table $\mathbf{T}$ by its own. If all checks pass, the prover prove to the verifier the equivalence of two polynomial $g(\alpha, \beta, \gamma)$ and $h(\alpha, \beta, \gamma)$ shown in Section 3.5, which infers $\boldsymbol{x}_i$ is in the table $\mathbf{T}$ using Lamma 3.4.

The coefficients of $g(\alpha, \beta, \gamma)$ are related with the public table $\mathbf{T}$ and the set $\mathbf{X} = \{\boldsymbol{x}_i\}_{i\in[1,n]}$ known by the prover, while the coefficients of $h(\alpha, \beta, \gamma)$ correlate with $\mathbf{S}$, the union set $\{\mathbf{X}, \mathbf{T}\}$ sorted by $\mathbf{T}$. The set $\mathbf{S}$ is not readily available to the prover. Therefore, the prover must first obtain $\mathbf{S}$ before proving the equivalence of two polynomials. Although using the quick sort algorithm to compute $\mathbf{S}$ has a time complexity of $O(n \log n)$, but this part is still the performance bottleneck in the protocol. We perform the sorting process by grouping together all elements of the union set $\{\mathbf{X}, \mathbf{T}\}$ with identical values, which is the optimal sorting method for our setting where the elements exhibit only a few distinct values. This adjustment allows us to obtain $\mathbf{S}$ in $O(n)$ time complexity.

Following the above steps, we define a polynomial $l(\alpha, \beta, \gamma) := g(\alpha, \beta, \gamma) - h(\alpha, \beta, \gamma)$. Based on Schwartz-Zippel Lemma, when the verifier randomly samples $(r_1, r_2, r_3) \leftarrow (\mathbb{F}_p)^3$ as the inputs to the polynomial l, the probability of $l(r_1, r_2, r_3) = 0$ is negligible if two polynomials g and h are not equivalent. Thus, the final step is to prove that the evaluation of polynomial l equals zero at a random point (r_1, r_2, r_3). Two parties invoke the (poly) command of $\mathcal{F}_{\text{authZK}}$, which utilizes the authentication codes of $\mathbf{X}$ and $\mathbf{S}$ to check the evaluation of the polynomial $l'(\mathbf{X}, \mathbf{S})$ equals zero, where l' is same as l but with $\mathbf{X}$, $\mathbf{S}$ as variables and $(\alpha, \beta, \gamma) := (r_1, r_2, r_3)$ as coefficients. Finally, the verifier outputs what $\mathcal{F}_{\text{authZK}}$ outputs.

Theorem 5.1. *Let d be the table size, w be the dimensions of the elements in the table, n be the lookup number. The protocol Π_{Lookup} shown in Figure 6 UC-realizes the* (lookup) *command of functionality $\mathcal{F}_{\text{authZK}}$ in the presence of a static, malicious adversary with statistical soundness error at most $(n+d)(w+1)/p$ in the $\mathcal{F}_{\text{authZK}}$-hybrid model.*

The security proof of this theorem is in Appendix C.

6 ZKP FOR NON-LINEAR OPERATIONS

In this section, we present our ZKPs for non-linear operations used in neural networks. The core idea of our work is to constrain the correct calculation of these non-linear operations by range and exponent relations, whose satisfaction can be further proved using range and lookup proofs discussed in the above two sections.

We complete the constraining process with two steps. In Section 6.1, we employ the range and exponent relations to constrain the primitive operations, such as round and bit-shift operations. In Section 6.2, we obtain the constraints for the non-linear layers in neural networks using the constraints for these primitive operations.

Note that some non-linear operations, such as the max operation, exhibit both non-linear and linear characteristics. These operations are additionally subject to constraints via polynomial relations that can be proved by polynomial proofs studied in previous work [63].

6.1 Constraints for Primitive Operations

Table 1 outlines the constraints for primitive non-linear operations. We carefully design these constraints to guarantee that the calculation results of these primitive operations are correct and unique. All constrains are on the finite field, where the elements are effectively treated as integers by employing additional shortness tests, shown in Section 3.4.3, to ensure the elements are small enough so that the phenomenon of wrapping around in the finite field cannot occur. In the following, we describe the underlying logic of these constraints for each primitive operation.

Max Operation. The max operation identifies the maximum value within a given set, donated as $q_{max} = \max(q_1, ..., q_n)$. Obviously, each element in the set maintains a non-negative difference with the maximum value q_{max}, satisfying the range relation: $0 \le q_{max} - q_i$ for all $i \in [1, n]$. In addition, q_{max} is equal to at least one element in the set. This constraint is described using a polynomial relation: $\prod_{i=1}^{n} (q_{max} - q_i) = 0$. If q_{max} is not equal to any q_i, the value on the left side of the equation would not be 0.

Next, we argue that q_{max}, satisfying the above two relations, is unique. If there is another $\bar{q}_{max}$ such that $\bar{q}_{max} = q_{max} + t$, where t is a non-zero element. It follows that $0 \le q_{max} - q_i + t$ for all $i \in [1, n]$. By choosing $q_i = q_{max}$, we get t must be positive, which is conflicted with q_{max} being the maximum value.

Overall, the above two relations ensure that the result of the max operation is correct and unique.

Min Operation. The min operation gets the minimum value within a given set, donated as $q_{min} = \min(q_1, ..., q_n)$. The result q_{min} is constrained by the range relation: $0 \le q_i - q_{min}$ for all $i \in [1, n]$, and the polynomial relation: $\prod_{i=1}^{n} (q_i - q_{min}) = 0$. The underlying logic of constraints for the min operation is the same as the constraints for the max operation.

Sign Operation. The sign operation, donated as $s = \text{sign}(q)$, identifies the sign of a given element. It returns +1 for positive elements, −1 for negative ones, and 0 for zero. First, the sign operation implies that s inherently shares the same sign as the element q, establishing the range relation: $0 \le s(q - s)$. It is worth noting that this relation also guarantees that s must be zero when q is zero. Next, we need to constrain s within the values $\{+1, 0, -1\}$ through the polynomial relation: $(s-1)(s+1)(s^2+q^2) = 0$, which guarantees that s must be -1 or $+1$ when q is not zero.

We argue that s, satisfying the above two relations, is unique. When q is zero, the inequality $0 \le -s^2$ from the range relation implies $s = 0$. When q is not zero, the polynomial relation restricts s to be either $+1$ or -1, and the range relation ensures s matches the sign of q, thereby confirming the uniqueness of s.

Absolute Value Operation. This operation yields the absolute value of a given element, donated as $|q| = \text{abs}(q)$. We notice that the constraints for the result $|q|$ can be easily established based on the sign operation. Initially, we constrain s to be the sign of q. Following this, the polynomial relation $|q| = sq$ ensures the result $|q|$ is correct. It also guarantees $|q|$ being unique due to the uniqueness of s.

Right Shift Operation. The right shift operation, denoted as $m = q \gg b$, produces the result by discarding the last b bits of the binary form of q. Therefore, the difference between q and mE_b is the value of the last b bits, which is less than $E_b = 2^b$. Thus, we constrain the result m with the range relation: $0 \le q - mE_b \le E_b - 1$.

When q is zero, the relation becomes $0 \le -mE_b \le E_b - 1$, which infers $m = 0$. When q is not zero, it ensures the difference between q and mE_b is less then E_b, and the result m is correct and unique.

Round Operation. This operation, donated as $z = \text{round}(x)$, approximates a given element to the nearest integer, where the given element x can be fractions, floating-point numbers, and square roots. Obviously, the difference between the result z and the input x is at most $\frac{1}{2}$, satisfying the relation: $-\frac{1}{2} \le x - z < \frac{1}{2}$. We employ it to build constraints for the round operation.

For the round operation involving fractions, donated as $z_d = \text{round}(q_1/q_2)$, we multiply the round constraint $-\frac{1}{2} \le \frac{q_1}{q_2} - z_d < \frac{1}{2}$ by $2sq_2$ to eliminate fractions, where $s = \text{sign}(q_2)$. It yields the range relation: $-sq_2 \le 2s(q_1 - z_d q_2) \le sq_2 - 1$. The term -1 is used to convert the inequality sign from $<$ to $\le$. This range relation is equivalent to the two corresponding range relations in Table 1.

For the round operation involving floating-point numbers, donated as $z_e = \text{round}(c \cdot 2^e)$, where c is the coefficient and e is the exponent of the floating-point number, we require two auxiliary elements E^+ and E^- that have the properties: $E^+ = 2^{|e|+1}$ when e is positive, $E^- = 2^{|e|+1}$ when e is negative, and $E^+ = 2^e E^-$. Next, we

Table 1: The constraints for primitive non-linear operations

Primitive Operations	Constraints
1) $q_{max} = \max(q_1, ..., q_n)$	• $\forall i \in [1, n],\ 0 \le q_{max} - q_i$ • $\prod_{i=1}^{n} (q_{max} - q_i) = 0$
2) $q_{min} = \min(q_1, ..., q_n)$	• $\forall i \in [1, n],\ 0 \le q_i - q_{min}$ • $\prod_{i=1}^{n} (q_i - q_{min}) = 0$
3) $s = \text{sign}(q)$	• $0 \le s(q - s)$ • $(s-1)(s+1)(s^2 + q^2) = 0$
4) $\|q\| = \text{abs}(q)$	• $s = \text{sign}(q)$ • $\|q\| = sq$
5) $m = q \gg b\ (b \ge 0)$	• $E_b = 2^b$ • $0 \le q - mE_b \le E_b - 1$
6) $z_d = \text{round}(q_1/q_2)$	• $s = \text{sign}(q_2)$ • $0 \le 2sq_1 - 2sz_dq_2 + sq_2$ • $0 \le sq_2 + 2sz_dq_2 - 2sq_1 - 1$
7) $z_e = \text{round}(c \cdot 2^e)$	• $s = \text{sign}(e)$ • $E = 2^{se}$ • $E^+ = s(s+1)E + (2+s)(1-s)$ • $E^- = s(s-1)E + (2-s)(1+s)$ • $0 \le 2cE^+ - 2z_eE^- + E^- \le 2E^- - 1$
8) $z_s = \text{round}(\sqrt{q})\ (q \ge 0)$	• $s = \text{sign}(q)$ • $0 \le 4q - s(2z_s - 1)^2 + (s-1)z_s^2$ • $0 \le (2z_s + 1)^2 - 4q - 1$
9) $z_f = \text{floor}(q_1/q_2)$	• $s = \text{sign}(q_2)$ • $0 \le sq_1 - sq_2z_f$ • $0 \le sq_2(z_f + 1) - sq_1 - 1$

multiple the round constraint $-\frac{1}{2} \le c \cdot 2^e - z_e < \frac{1}{2}$ by $2E^-$ to remove fractions. It yields the range relation: $-E^- \le 2cE^+ - 2z_eE^- \le E^- - 1$.

For the round operation involving square roots, donated as $z_s = \text{round}(\sqrt{q})$. We square the round constraint $-\frac{1}{2} + z_s \le \sqrt{q} < \frac{1}{2} + z_s$, and then multiple it by 4. It yields the range relation: $s(2z_s - 1)^2 + (1-s)z_s^2 \le 4q \le (2z_s + 1)^2 - 1$, where $s = \text{sign}(q)$ and the term $(1-s)z_s^2$ is used to guarantee $z_s = 0$ when $q = 0$. This range relation is equivalent to the two corresponding range relations in Table 1.

The uniqueness of all rounding results is guaranteed by the initial constraint: $-\frac{1}{2} \le x - z < \frac{1}{2}$, because there is only a single integer z for which $x - z$ is in the range $[-\frac{1}{2}, \frac{1}{2})$, regardless of x.

Floor Operation. The floor operation, donated as $z = \text{floor}(x)$, approximates a given element x to the largest integer less than or equal to x. It constrains the difference between the result z and the input x being at most 1, satisfying the relation: $0 \le x - z < 1$. We give an example for the floor operation on fractions, donated as $z_f = \text{floor}(q_1/q_2)$. It is constrained by the range relations: $0 \le sq_1 - sq_2z_f$ and $0 \le sq_2(z_f + 1) - sq_1 - 1$. The logic of constraints for the floor operation is the same as that for the round operation.

Remark: The Upper Bounds of Range Relations. Most range relations listed in Table 1 are defined only by lower bounds. To align with the requirement of our range proof, we need to determine their upper bounds. In our setting, the elements we use have an inherent range, as shown in the approximation method part of Section 6.2, where each integer q has $|q| \le B_Q$, and each floating-point number $c \cdot 2^e$ has $|c| \le B_C$ and $2^{|e|} \le B_E$. We set the upper bounds of these range relations using the maximum values of their associated elements. For example, the upper bound of $2sq_1 - 2sz_dq_2 + sq_2$ in the round operation on fraction is $2B_Q$ because $\frac{q_1}{q_2} - z_d < \frac{1}{2}$, which infers $2sq_1 - 2sz_dq_2 \le sq_2 = |q_2| \le B_Q$, where $s = \text{sign}(q_2)$.

6.2 Constraints for Non-linear Layers

We employ the primitive operations to build the constraints for non-linear layers, including ReLU, MaxPooling, and AvgPooling layers of convolutional neural networks as well as Normalization, Softmax, and GELU layers of transformer neural networks. Furthermore, our framework offers the flexibility to extend to additional non-linear layers, as the primitive operations serve as generic components for constructing non-linear layers in neural networks.

Performing non-linear layers involves floating-point operations, but floating-point operations are not ZKP-friendly. As a solution, existing ZKP schemes [16, 38, 40, 60], including ours, opt for approximating floating-point numbers to integers or fixed-point numbers. In this section, we first present the approximation method and then the constraints for each non-linear layer.

Approximation Method. The approximation method [24] maps floating-point numbers to Q-bit integers with the formula $x = S(q - Z)$, where x is the original floating-point number, q is the Q-bit approximated integer, S is a floating-point number called scale, and Z is an integer called zero point. Every Q-bit integer is in the inherent range $[-B_Q, B_Q]$. For example, $B_Q = 127$ when $Q = 8$. The scale S is adjustable to achieve the mapping from float-point numbers to Q-bit integers with minimal inaccuracy. In our setting, all floating-point numbers are represented as the form $c \cdot 2^e$ with $|c| \le B_C$ and $2^{|e|} \le B_E$. To simplify, we omit the discussion about zero point Z. This is because, in our ZKP scheme, the prover and the verifier can locally calculate the authentication code of $q - Z$ by themselves, using the homomorphic property of VOLE correlations.

ReLU Layer. ReLU layers apply an element-wise operation on their input x to produce the output y with the formula $y = \max(0, x)$. In the approximation setting, both the input q_x and the output q_y share the same scale S. The constraints for ReLU layers can be built on the constraints for the max operation with the formula $q_y = \max(0, q_x)$. The scale S appears on both sides of the equation and thus has been eliminated.

MaxPooling Layer. MaxPooling layers use a sliding kernel to calculate the maximum value of the input covered by the kernel with the formula $y = \max(x_1, ..., x_k)$, where k is the kernel size. In the approximation setting, the input $\{q_{x1}, ..., q_{xk}\}$ and the output q_y share the same scale S. With the same logic of constraints for ReLU layers, the constraints for MaxPooling layers are built on the constraints for the max operation with the formula $q_y = \max(q_{x1}, ..., q_{xk})$.

AvgPooling Layer. AvgPooling layers calculate the average value of the input covered by a sliding kernel with the formula $y = (\sum_{i=1}^{k} x_i)/k$, where k is the kernel size. In the approximation setting, we are required to prove the relation between the input and the output as $q_y = \text{round}(\sum_{i=1}^{k} q_{xi}/k)$. It is consistent with the constraints for the round operation on fractions. The difference is that the divisor k is a public positive integer. Thus, we directly assign the sign $s = +1$ to the constraints for the round operation, obtaining the range relation: $0 \le (\sum_{i=1}^{k} 2q_{xi}) - 2kq_y + k \le 2k - 1$.

Normalization Layer. Normalization layers first calculates the input's mean $\mu = (\sum_{i=1}^{n} x_i)/n$ and variance $v = (\sum_{i=1}^{n} (x_i - \mu)^2)/n$, where n is the input size. Then, it applies an element-wise operation on its input with the formula $y_i = (x_i - \mu)/\sigma$, where $\sigma = \sqrt{v}$. In the approximation setting, the three elements $\{q_{xi}, q_\mu, q_\sigma\}$ share the same scale S_x, and the output q_{yi} has an independent scale S_y. The normalization formula can be represented as $S_y q_{yi} = (q_{xi} - q_\mu)/q_\sigma$. The output q_{yi} is constrained by the following relations:
(1) $q_\mu = \mathsf{round}(\sum_{i=1}^{n} q_{xi}/n)$; (2) $q_v = \mathsf{round}(\sum_{i=1}^{n} (q_{xi} - q_\mu)^2/n)$;
(3) $q_\sigma = \mathsf{round}(\sqrt{q_v})$; (4) $q_{ti} = \mathsf{round}((1/S_y)(q_{xi} - q_\mu))$;
(5) $q_{yi} = \mathsf{round}(q_{ti}/q_\sigma)$, where q_{ti} is an intermediate result.

The relations (1)(2)(5) can be converted to the constraints for the round operations on fractions. The relation (3) is unfolded as constraints for the round operation on square roots, and the relation (4) involves the round operation on floating-point numbers.

Softmax Layer. Softmax layers perform an element-wise operation on its input with the formula $y_i = \exp(x_i)/\sum_{j=1}^{n} \exp(x_j)$, where n is the input size and exp denotes the exponent function with the natural logarithm e as the base. This formula is same as $y_i = \exp(\widetilde{x_i})/\sum_{j=1}^{n} \exp(\widetilde{x_j})$, where each $\widetilde{x_i} = x_i - \max(x_1, ..., x_n)$.

We decompose $\widetilde{x_i}$ with $\widetilde{x_i} = (-\ln 2)z_i + p_i$, where $p_i \in (-\ln 2, 0]$ and z is a non-negative integer. Next, the approach in [37] is used to approximate the function exp for the input $p_i \in (-\ln 2, 0]$ with a second-order polynomial $L(p_i) = A(p_i + B)^2 + C$, where $A = 0.3585$, $B = 1.353$, and $C = 0.344$. Each exponent result is calculated by the formula $t_i = \exp(\widetilde{x_i}) = 2^{-z_i}\exp(p_i) \approx L(p_i) \gg z_i$.

In the approximation setting, the scales of p_i and t_i are constant floating-point numbers with $S_p = \ln 2/B_Q$ and $S_t = (AB^2 + C)/B_Q$, where B_Q is the maximum value of Q-bit approximated integers. The output q_{yi} is constrained by the six relations:
(1) $\widetilde{q_{xi}} = q_{xi} - \max(q_{x1}, ..., q_{xk})$; (2) $z_i = \mathsf{floor}((-S_x/\ln 2)\widetilde{q_{xi}})$;
(3) $q_{pi} = \mathsf{rd}((S_x/S_p)\widetilde{q_{xi}}) + B_Q z_i$; (4) $t_i = \mathsf{rd}(L(S_p q_{pi})/S_t) \gg z_i$;
(5) $q_{mi} = \mathsf{rd}((1/S_y)t_i)$; (6) $q_{yi} = \mathsf{rd}(q_{mi}/\sum_{j=1}^{n} t_j)$,
where rd is abbreviation of round and q_{mi} is an intermediate result.

The above relations all involve primitive non-linear operations in Table 1 and can be converted to the corresponding constraints.

GELU Layer. GELU layers apply an element-wise operation on its input with the formula $y = \frac{1}{2}x(1 + \mathsf{erf}(x/\sqrt{2}))$, where erf denotes the Gauss error function [32]. The function erf involves the integration operation, which is not computationally efficient.

We employ the approach in [37] to approximate the function erf by the formula $t = \mathsf{erf}(x) \approx \mathsf{sign}(x) \cdot L(\min(|x|, -B))$, where $L(x) = A(x+B)^2+1$ is a second-order polynomial with $A = -0.2888$ and $B = -1.769$. In the approximation setting, using above equation, the output q_y is constrained by the relations:
(1) $s_x = \mathsf{sign}(q_x)$; (2) $|q_x| = \mathsf{abs}(q_x)$; (3) $z_B = \mathsf{floor}(-\sqrt{2}B/S_x)$;
(4) $q_{min} = \min(|q_x|, z_B)$; (5) $t = s_x \cdot \mathsf{round}(L((S_x/\sqrt{2})q_{min}))$;
(6) $q_y = \mathsf{round}((S_x/2S_y)q_x(1 + t))$.

Similarly, the above relations can be converted to the constraints for the corresponding primitive non-linear operations in Table 1.

7 ZERO-KNOWLEDGE NEURAL NETWORKS

In this section, we put everything together to provide our ZKPs for neural networks. For an intuitive description, we present it in the Machine-Learning-as-a-Service (MLaaS) setting, where our work enables the service provider (prover) to convince the customer (verifier) that its prediction outcomes are correctly calculated from a particular neural network model, while preserving the model's privacy. Below, we introduce the crucial parts of our work.

Overall Workflow. Our work consists of three steps to provide the service of verifiable prediction. First, we follow the commit-and-prove method with the authentication code as a form of commitment. It requires the service provider to authenticate a well-trained neural network model for its customer. Second, upon receiving input, the service provider employs the authenticated model to compute the prediction result and return it to the customer. Third, the service provider proves to its customer that the prediction result is indeed generated from the authenticated model.

Basic Blocks. Our work adopts the VOLE-based proving backend with the ideal functionality shown in Figure 4. Specifically, we employ the recent LPN-based VOLE protocols [5, 6, 59] as the foundation to authenticate the parameters of neural network models. We utilize previous works [60, 63] to provide the polynomial proof. The range and lookup proofs are designed by ourselves, shown in Section 4 and Section 5, respectively. These components are compatible with each other under our proving framework so that we can put them together to build our ZKPs for neural networks.

Neural Networks. Our work provides the service of verifiable prediction for convolutional and transformer neural networks [45, 56]. In addition, the modular design in our framework offers the flexibility for expansion to more neural networks.

Existing ZKP schemes [16, 38, 40, 60], including ours, opt for approximating floating-point numbers to integers or fixed-point numbers, thereby avoiding floating-point operations that are not ZKP-friendly. To minimize the loss of accuracy due to approximation, our work is designed to be compatible with existing approximated neural networks that have high accuracy.

For convolutional neural networks, we employ multiple classic approximated networks in PyTorch [46], a prominent library for machine learning, where the method proposed in [34] is used to approximate models. For transformer neural networks, we employ the GPT-2 model [49] provided by OpenAI [42] and utilize the method proposed in IBert [37] to approximate models. These approximated networks maintain similar accuracy comparable to the original networks, with evaluation details outlined in Section 8.4.

Proving Procedure. Neural networks comprise a sequence of layers, where each layer processes its input to produce the output. Except for the first layer's input and the last layer's output, intermediate results of other layers are secret to verifiers. Therefore, the prover should authenticate these intermediate results, which enables the subsequent proving procedure to process without leaking these intermediate results. Next, we aim to prove the correct calculation of each layer in neural networks. Our proving framework works under the universal composition security model [9], which helps us to construct our ZKPs for the entire network by compositing the proving procedure for each layer.

The proving procedure of linear layers, such as convolutional and fully-connected layers, has been explored in previous works [16, 17, 38, 40]. In essence, the calculation of these linear layers is fundamentally matrix multiplication. Therefore, we adopt the optimized polynomial proof in Mystique [60], designed for matrix multiplication, to prove the correct calculation of linear layers.

The proving procedure of non-linear layers is performed by constraining these non-linear layers with range and exponent relations, as detailed in Section 6. Next, we prove the satisfaction of these relations using our range and lookup proofs, presented in Section 4 and Section 5, respectively. Some non-linear layers, such as MaxPooling, exhibit non-linear and linear characteristics. Thus, they are additionally subject to constraints via polynomial relations. We prove the satisfaction of these polynomial relations using the polynomial proof proposed in Quicksilver [63].

8 EVALUATION

In this section, we present a comprehensive evaluation of our ZKP framework for neural networks.

8.1 Experimental Setup

Software. Our work is implemented in C++ and the source code can be found in [69].. We utilize the libff library [39] to provide the finite field operations and employ a 254-bit field with the prime order $p = 21888242871839275222246405745257275088548364400416034343698204186575808495617$. We use OpenMP [44] to implement multi-core parallelism. For evaluations performed in this paper, we achieve the computational security parameter $\lambda = 128$ and the statistical security parameter $\rho = 40$.

Hardware. Our evaluations are performed in the machine, which is equipped with an AMD Ryzen 3700X 8-Core CPU and 32GB of memory. The memory resource is not the bottleneck for our scheme, even when evaluating large-scale neural networks. All experiments are performed in a network environment with a bandwidth of 500 Mbps and a latency of around 50ms.

8.2 Evaluating Range and Lookup Proofs

We benchmark the performance of our range and lookup proofs, where prover time and verifier time refer to the local computational time of two parties. The time for data transfer between two parties is listed separately. We specify that the total runtime includes prover time, verifier time, and the time for data transfer.

Range proof. Table 2 gives the evaluation results of range proofs. The experiment benchmarks the performance for proving that n elements are in the range $[0, 2^{24}]$, where n varying across 2^{10} to 2^{18}. We compare our scheme with two different types of range proofs. Bulletproofs [8] is the range proof based on the technology of bit decomposition, and Lasso [51] is a state-of-the-art work for the lookup proof that can also be applied to range checking.

As shown in Table 2, the computational costs of our range proof are small. It only takes 287ms for the prover and 142ms for the verifier to complete proving 2^{18} elements. The prover time and verifier time of our scheme are over 847.0× and 75.3× shorter than Bulletproofs, respectively. The advantage comes from two perspectives. On the one hand, our scheme employs the three-square decomposition technology, which introduces fewer constraints than bit decomposition in Bulletproofs. On the other hand, we optimize the searching method [47] to efficiently identify three squares required in our scheme, removing the performance bottleneck when three-square decomposition is used in the VOLE proving backend.

Lasso [51] is a very recent work. The results show that the computational time of our scheme is up to 22.2× shorter than that of Lasso. However, our scheme introduces more communication overheads. This is because the commitments used in our scheme are VOLE correlations, whose communication complexity is linear to the number of committing elements. It is greater than the polynomial commitments [50] with sublinear communication complexity. Overall, the total runtime of ours is still competitive with Lasso. We can achieve at most 1.87× speedup over Lasso.

Table 2: The performance of range proofs

Num.	Schemes	Prover	Verifier	Proof Size	Transfer
2^{10}	Bulletproofs	5.37s	0.39s	0.033MB	101ms
	Lasso	141ms	23.6ms	0.062MB	152ms
	Ours	6.34ms	5.18ms	0.117MB	202ms
2^{14}	Bulletproofs	85.8s	6.11s	0.513MB	307ms
	Lasso	484ms	49.6ms	0.165MB	203ms
	Ours	29.2ms	12.7ms	1.831MB	415ms
2^{18}	Bulletproofs	1358s	96.7s	8.005MB	540ms
	Lasso	2175ms	108ms	0.558MB	308ms
	Ours	287ms	142ms	29.33MB	955ms

Table 3: The performance of lookup proofs

Num.	Schemes	Prover	Verifier	Proof Size	Transfer
2^{10}	Plookup	78.2ms	4.47ms	960B	54.6ms
	Ours	4.00ms	1.23ms	0.096MB	153ms
2^{14}	Plookup	1034ms	6.40ms	960B	55.9ms
	Ours	63.7ms	20.4ms	1.501MB	367ms
2^{18}	Plookup	123.1s	31.4ms	960B	58.3ms
	Ours	1066ms	379ms	24.43MB	872ms

Lookup proof. Table 3 shows the evaluation results of lookup proofs. In our work, they serve to prove the relation between two field elements, e and E, with the constraint that E is the e-th power of 2. The pair of two elements appearing in the exponent table, shown in Figure 2, infers their exponent relation. Our experiment benchmarks the performance for proving that n pairs are in the exponent table, where n varies across 2^{10} to 2^{18}.

We compare our work with Plookup [19], a lookup scheme suitable for small lookup tables. The proof size of Plookup is constant because it employs KZG commitments [36] with constant communication complexity. However, KZG commitments would introduce significant computational overheads. As shown in Table 3, the prover time of ours is at most 115.5× shorter than that of Plookup. Although our work has larger communication overheads, the total runtime of ours still achieves up to 53.2× speedup over Plookup.

8.3 Evaluating Non-linear Layer Proofs

This section presents the evaluation results of our ZKPs for non-linear layers. We compare our work with another two VOLE-based ZKP schemes. [29] is a concurrent work with ours. As their implementation has not been open-sourced, we use the evaluation results from their literature to perform comparisons. For fairness,

Table 4: The performance of non-linear layer proofs

Layers	Schemes	Prover	Verifier	Proof Size	Transfer
ReLU	Mystique	194s†		58.2MB	1.54s
	Concurrent	1.91s		30.1MB	0.99s
	Ours	0.28s	0.14s	18.3MB	0.74s
Softmax	Mystique	14050s		3972MB	77.29s
	Concurrent	78.3s		816MB	16.16s
	Ours	24.1s	11.9s	1166MB	22.95s
GELU	Mystique	2712s		655MB	13.07s
	Concurrent	32.7s		338MB	6.92s
	Ours	2.31s	1.10s	99.6MB	2.30s
Norm.	Mystique	9643s		2220MB	43.32s
	Concurrent	176s		1787MB	34.98s
	Ours	9.69s	4.76s	394MB	8.02s

† The complex synchronization in their implementations makes it difficult to timekeep the prover and the verifier separately.

we configure the same setting as in their literature to execute our implementation on AWS c5.9xlarge instance with Intel Xeon 8000 series CPUs. Mystique [60] is used as the common baseline of the concurrent work and ours. In our experiments, we demonstrated the performance of ZKP schemes on four classic non-linear layers: ReLU, Softmax, GELU, and Normalization Layers.

As shown in Table 4, for separated non-linear layers, our work significantly reduces computational and communication costs, achieving over 168.6× (up to 477.2×) speedup compared to Mystique, which employs the traditional bit decomposition method to generate non-linear layer proofs. Our work uses range and exponent relations to constrain the correct calculation of non-linear layers, significantly reducing the number of required relations compared to the relations introduced by bit decomposition. Moreover, our work is competitive to the concurrent work [29]. Especially for Normalization layers, we achieve 12.2× lower computational costs and 4.54× lower communication costs. Note that the concurrent work [29] cannot directly deploy their non-linear layer proofs into neural networks due to the lack of compatibility with existing networks.

8.4 Evaluating Neural Network Proofs

In this section, we evaluate the performance of our ZKP framework for convolutional and transformer neural networks.

Setup. For convolutional neural networks, we evaluate our ZKPs on three well-known architectures. They are VGG-11 [53], ResNet-50 [30], and ResNet-101 [30]. Among them, ResNet-101 is the largest model with around 44.5 million parameters. We train these models using Pytorch [46] with the 8-bit approximation technique [34]. These well-trained models are fully compatible with our ZKP framework. In our experiments, we use the CIFAR-10 dataset [12], which contains 10 classes of images, and the image size is $32 \times 32 \times 3$.

For transformer neural networks, we evaluate our ZKP scheme on GPT-2 [49], which is a large language model with around 117 million parameters. In our experiments, we utilize the GPT-2 model provided by OpenAI [42] and employ the method proposed in IBert [37] to approximate the model using 16-bit integers. We use data from the dataset [43] provided by OpenAI as the input. This dataset contains various English sentences.

Table 5: The performance of ZKP schemes for convolutional and transformer neural networks

Models	Schemes	Prover	Verifier	Proof Size	Transfer
VGG-11	zkCNN	52.8s	8.33s	45.9KB	0.15s
	Ours	1.62s	0.57s	65.9MB	1.66s
ResNet-50	Mystique	403s†		1.27GB	25.59s
	Ours	4.22s	1.75s	205MB	4.35s
ResNet-101	Mystique	617s		1.98GB	39.69s
	Ours	6.65s	2.85s	310MB	6.37s
GPT-2	ZKML	2.84h	24.43s	15.6KB	0.10s
	Ours	136.1s	62.85s	4.37GB	88.17s

† The complex synchronization in their implementation makes it difficult to timekeep the prover and the verifier separately.

Table 6: The performance of our ZKP scheme for GPT-2

Layers	Prover	Verifier	Proof Size	Transfer
Norm.	7.87s	4.29s	298MB	6.15s
Softmax	16.45s	8.23s	667MB	13.28s
GELU	74.56s	36.39s	2281MB	44.49s
Linear	37.26s	13.94s	1235MB	24.25s
Total	**136.1s**	**62.85s**	**4481MB**	**88.17s**

Efficiency. For convolutional neural networks, we compare our work with two state-of-the-art ZKP schemes. zkCNN [40] is a scheme based on the GKR protocol [25]. Its open-source implementation provides the proof for VGG-11. Mystique [60] is a VOLE-based ZKP scheme like ours, and its open-source implementation provides the proofs for ResNet-50 and ResNet-101. Other previous works [16, 35, 38] have greater computational costs than the above two schemes, so we skip the comparisons with them.

As shown in Table 5, the computational costs of our scheme are small in practice. It only takes 1.62s for the prover and 0.57s for the verifier to complete the proof for VGG-11. Compared to zkCNN, our scheme has larger communication overheads but smaller computational costs. The total runtime of our scheme achieves 15.9× speedup. Compared to Mystique, a VOLE-based ZKP scheme like ours, we have advantages in both computational and communication costs. The overall runtime of our scheme achieves 41.5× speedup for ResNet-50 and achieves 41.4× for ResNet-101. We count the prover and verifier time together for Mystique in Table 5 because the complex synchronization operations in its implementation make it difficult to count them individually.

For transformer neural networks, we compare our work with the concurrent work ZKML [10], a ZKP scheme that also supports transformer neural networks. Executing ZKML requires over 500GB of memory. Therefore, we perform ZKML and our implementation on Alibaba Cloud ecs.re4.20xlarge instance [1] with 960GB memory and 80 vCPU cores. Moreover, in order to be consistent with the experimental setting of ZKML, we use English sentences of length 64 as input to GPT-2. zkLLM [54] is another concurrent work that

Table 7: The accuracy of convolutional neural networks

Models	Original	Mystique	zkCNN	Ours
VGG-11	91.73%	/	88.70%	91.70%
ResNet-50	93.96%	93.94%	/	94.00%
ResNet-101	93.83%	93.81%	/	93.79%

requires GPU acceleration. Due to the different requirements in hardware devices, we cannot make a fair comparison with zkLLM.

Table 5 shows that our work can generate a proof for GPT-2 in a total of 287.1 seconds, achieving 35.7× speedup over ZKML. Table 6 gives evaluation results of our ZKP scheme for each layer type, including GELU, Softmax, Normalization, and other linear layers. We notice that the proof for the linear layers only takes 26.3% of the total runtime. This reflects that optimizing the proof of non-linear layers in complex neural networks is an important step towards improving the overall performance.

Accuracy. For convolutional neural networks, we evaluate the accuracy of the neural network models used in these ZKP schemes on the CIFAR-10 dataset [12]. As shown in Table 7, both Mysitique [60] and ours are compatible with the high-accuracy approximated neural networks, and we almost achieve the same accuracy as the original model without approximation. The accuracy difference between our scheme and the original networks is within 0.04%.

For transformer neural networks, we evaluate the cosine similarity between the outputs of the original GPT-2 model provided by OpenAI [42] and the outputs of the approximated GPT-2 model that our ZKP scheme is compatible with. We execute both models with the same inputs. The evaluation results show that the average cosine similarity of the outputs from two models reaches 99.95%.

REFERENCES

[1] Alibaba. 2024. Alibaba cloud service. https://www.alibabacloud.com

[2] Amazon. 2024. Amazon Machine Learning Services. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/machine-learning.html

[3] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. 2023. Sok: vector OLE-based zero-knowledge protocols. *Designs, Codes and Cryptography* 91, 11 (2023), 3527–3561.

[4] Carsten Baum, Alex J Malozemoff, Marc B Rosen, and Peter Scholl. 2021. Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*. Springer, 92–122.

[5] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 896–912.

[6] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 291–308.

[7] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*. Springer, 423–443.

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

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

[10] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 560–574.

[11] Miranda Christ, Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Deepak Maram, Arnab Roy, and Joy Wang. 2024. SoK: Zero-Knowledge Range Proofs. *Cryptology ePrint Archive* (2024).

[12] CIFAR-10. 2024. A collection of images that are commonly used to train machine learning and computer vision algorithms. https://www.cs.toronto.edu/~kriz/cifar.html

[13] Geoffroy Couteau, Dahmun Goudarzi, Michael Klooß, and Michael Reichle. 2022. Sharp: Short relaxed range proofs. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 609–622.

[14] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. 2021. Efficient range proofs with transparent setup from bounded integer commitments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 247–277.

[15] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive* (2022).

[16] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive* (2021).

[17] Boyuan Feng, Zheng Wang, Yuke Wang, Shu Yang, and Yufei Ding. 2024. ZENO: A Type-based Optimization Framework for Zero Knowledge Neural Network Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 450–464.

[18] Ariel Gabizon and Dmitry Khovratovich. 2022. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. *Cryptology ePrint Archive* (2022).

[19] Ariel Gabizon and Zachary J Williamson. 2020. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive* (2020).

[20] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).

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

[22] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. 2022. Succinct Zero Knowledge for Floating Point Computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1203–1216.

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

[24] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*. Chapman and Hall/CRC, 291–326.

[25] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)* 62, 4 (2015), 1–64.

[26] Google. 2024. Innovative AI and machine learning products, solutions, and services powered by Google's research and technology. https://cloud.google.com/products/ai

[27] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 305–326.

[28] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. 2019. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine* 25, 1 (2019), 65–69.

[29] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. 2024. Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning. (2024).

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[31] Miao He and David He. 2017. Deep learning based approach for bearing fault diagnosis. *IEEE Transactions on Industry Applications* 53, 3 (2017), 3057–3065.

[32] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[33] Rahat Iqbal, Tomasz Maniak, Faiyaz Doctor, and Charalampos Karyotis. 2019. Fault detection and isolation in industrial processes using deep learning approaches. *IEEE Transactions on Industrial Informatics* 15, 5 (2019), 3077–3084.

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

[35] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. Scaling up Trustless DNN Inference with Zero-Knowledge Proofs. *arXiv preprint arXiv:2210.08674* (2022).

[36] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 177–194.

[37] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR, 5506–5518.

[38] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vcnn: Verifiable convolutional neural network based on zk-snarks. *Cryptology ePrint Archive* (2020).

[39] Libff. 2024. A C++ library for finite fields and elliptic curves. https://github.com/scipr-lab/libff

[40] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[41] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

[42] Openai. 2024. An AI research and deployment company whose mission is to ensure that artificial general intelligence benefits all of humanity. https://openai.com

[43] Openai. 2024. Dataset of GPT-2 outputs for research in detection, biases, and more. https://github.com/openai/gpt-2-output-dataset/tree/master

[44] OpenMP. 2024. OpenMP: A framework is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable. https://www.openmp.org

[45] Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[46] Pytorch. 2024. A fast, flexible experimentation and efficient production through a user-friendly front-end, distributed training, and ecosystem of tools and libraries. https://pytorch.org

[47] Michael O Rabin and Jeffery O Shallit. 1986. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics* 39, S1 (1986), S239–S256.

[48] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[50] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.

[51] Srinath Setty, Justin Thaler, and Riad Wahby. 2023. Unlocking the lookup singularity with Lasso. *Cryptology ePrint Archive* (2023).

[52] Ali Shahin Shamsabadi, Sierra Calanda Wyllie, Nicholas Franzese, Natalie Dullerud, Sébastien Gambs, Nicolas Papernot, Xiao Wang, and Adrian Weller. 2022. Confidential-PROFITT: confidential PROof of fair training of trees. In *The Eleventh International Conference on Learning Representations*.

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

[54] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkLLM: Zero Knowledge Proofs for Large Language Models. (2024).

[55] Haochen Sun and Hongyang Zhang. 2023. zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training. *arXiv preprint arXiv:2307.16273* (2023).

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[57] Nan Wang and Sid Chi-Kin Chau. 2022. Flashproofs: Efficient zero-knowledge arguments of range and polynomial evaluation with transparent setup. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 219–248.

[58] Nan Wang, Sid Chi-Kin Chau, and Dongxi Liu. 2023. SwiftRange: A Short and Efficient Zero-Knowledge Range Argument For Confidential Transactions and More. *Cryptology ePrint Archive* (2023).

[59] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1074–1091.

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

[61] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 38–45.

[62] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 733–764.

[63] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2986–3001.

[64] Yibin Yang and David Heath. 2023. Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM. *Cryptology ePrint Archive* (2023).

[65] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3121–3134.

[66] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Rafols. 2022. Baloo: Nearly optimal lookup arguments. *Cryptology ePrint Archive* (2022).

[67] zcash. 2024. The Halo2 zero-knowledge proving system. https://github.com/zcash/halo2

[68] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. 2021. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems* 32, 10 (2021), 2524–2540.

[69] zkNN. 2024. A Efficient and Extensible ZKP framework for neural networks. https://anonymous.4open.science/r/zknn-D209

A UNIVERSAL COMPOSITION

In this paper, all our protocols are proven in the Universal Composability (UC) framework [9], which is a general-purpose security model for the analysis of cryptographic protocols. It uses ideal functionality as a benchmark for describing how a cryptographic protocol should behave.

Formally, we say a protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$, the output distribution of $\mathcal{Z}$ is indistinguishable in two worlds where the parties execute protocol Π with the adversary $\mathcal{A}$ and where the parties perform the ideal functionality $\mathcal{F}$ with the simulator $\mathcal{S}$. Moreover, we prove the security of our protocols in the $\mathcal{G}$-hybrid model, where it allows two parties that execute the protocol Π in the real world to additionally access the command in another ideal functionality $\mathcal{G}$.

B SECURITY PROOF OF THEOREM 4.1

Proof. First, we consider the case where both the prover and the verifier execute Π_{Range} honestly with all x_i being in the range $[0, B]$ for $i \in [1, n]$. In this case, we show that the verifier would output true except for the probability at most $\frac{R}{L}$, which infers the correction error of Π_{Range} is at most $\frac{R}{L}$.

Correctness: Due to each $x_i \in [0, B]$, we can identify $\{y_{ij}\}_{j\in[1,3]}$ that satisfy $4x_i(B-x_i)+1 = \sum_{j=1}^{3} {y_{ij}}^2$ for $i \in [1, n]$, and y_{ij} are all in the range $[0, B]$. Therefore, following the protocol Π_{Range}, it passes the first check of verifier with all prover's responses $\{\zeta_k\}_{k\in[1,R]}$ being in the range $[4nB, 4nBL]$, unless at least one mask procedure performed by prover outputs $\perp$. Based on Lemma 3.2, the event that each mask procedure outputs $\perp$ is independent and occurs with probability at most $\frac{1}{L}$. Thus, the probability that there exists $\zeta_k = \perp$ is at most $1 - (1 - \frac{1}{L})^R \leq \frac{R}{L}$. For the second check and the third check, it succeeds obviously when the prover performs honestly. Overall, the honest verifier would output true except for the probability at most $\frac{R}{L}$. □

Next, we consider two cases where the prover is malicious or the verifier is malicious. In each case, we construct a simulator $\mathcal{S}$, which is given access to the (range) command of $\mathcal{F}_{\mathsf{authZK}}$ and runs an adversary $\mathcal{A}$ as a subroutine when emulating other commands of $\mathcal{F}_{\mathsf{authZK}}$. In both cases, we show that it cannot distinguish the real-world execution and the ideal-world execution with statistical error at most $\max(\frac{R}{L}, \frac{1}{p} + \frac{1}{2^R})$.

Before the protocol Π_{Range} begins, $\mathcal{P}$ and $\mathcal{V}$ have authenticated $x_i \in \mathbb{F}_p$ by performing the (auth) command of $\mathcal{F}_{\mathsf{authZK}}$. During this procedure, $\mathcal{S}$ records x_i and its tags sent by the malicious prover or records the local keys sent by the malicious verifier.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ and the (range) command of $\mathcal{F}_{\mathsf{authZK}}$ as follows:

(1) Whenever $\mathcal{S}$ receives $\perp$ from $\mathcal{A}$, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$ which sends false to the honest verifier and aborts.
(2) $\mathcal{S}$ emulates (auth) command of $\mathcal{F}_{\mathsf{authZK}}$ in Step (2) of the protocol Π_{Range}. $\mathcal{S}$ records $\{y_{ij}\}_{i\in[1,n],j\in[1,3]}$, $\{m_k\}_{k\in[1,R]}$, and their tags that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.
(3) $\mathcal{S}$ randomly samples challenges of the shortness tests $\gamma_{ijk} \leftarrow [0,1]$ for $i \in [1,n]$, $j \in [0,3]$, $k \in [1,R]$, and send them to $\mathcal{A}$.
(4) $\mathcal{S}$ receives the responses ζ_k from $\mathcal{A}$, and if any $\zeta_k \notin [4nB, 4nBL]$ for $i \in [1,R]$, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$.
(5) $\mathcal{S}$ emulates (poly) command of $\mathcal{F}_{\mathsf{authZK}}$ in Π_{Range}, where $\mathcal{S}$ checks $\sum_{j=1}^{3} y_{ij}^2 - 4x_i(B-x_i) - 1 = 0$ for $i \in [1,n]$. If any above check fails, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$.
(6) $\mathcal{S}$ computes the tags of $[z_k] := \sum_{i=1}^{n}\sum_{j=0}^{3} \gamma_{ijk}[y_{ij}] + [m_k] - \zeta_k$ for $k \in [1,R]$ with $y_{i0} := x_i$. Next, $\mathcal{S}$ uses them to check the correctness of the tag sent by $\mathcal{A}$ in the CheckZero procedure. If any check fails, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$.
(7) $\mathcal{S}$ sends the $(\mathsf{range}, \{\mathsf{id}_i^x\}_{i\in[1,n]}, B)$ to $\mathcal{F}_{\mathsf{authZK}}$, which sends true to the honest verifier if x_i is in the range $[0,B]$ for $i \in [1,n]$, or sends false to the honest verifier otherwise.

Clearly, the view of the adversary $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$, except for the CheckZero procedure. In CheckZero procedure, if the adversary $\mathcal{A}$ performs honestly and outputs the tag on zero, then the event that $\mathcal{S}$ check the correctness in the ideal world is equivalent to the event that the honest verifier check the correctness in the real world. Otherwise, the honest verifier outputs true only when the adversary can randomly hit the global key $\Delta \in \mathbb{F}_p$ with probability at most $\frac{1}{p}$ in the real world. In the ideal world, the honest verifier definitely outputs true.

Conditioned on the same view of $\mathcal{A}$, we remain to get the probability that the honest verifier outputs true in the real world when there exists $x_i \notin [0,B]$. Based on Lemma 3.3, after performing R repetitions of the shortness tests, all elements of $\{x_i\}_{i\in[1,n]}$ and $\{y_{ij}\}_{i\in[1,n],j\in[1,3]}$ are in the range $[-4nBL, 4nBL]$, except for the probability at most $\frac{1}{2^R}$. This ensures that x_i and $\{y_{ij}\}_{j\in[1,3]}$ are small enough that wrapping around, in the equation $4x_i(B-x_i)+1 = \sum_{j=1}^{3} {y_{ij}}^2$, does not occur if $4K^2+4BK+1 \le \frac{p-1}{2}$ and $K = (4nB-1)L$. This is because $|4x_i(B-x_i)+1| \le 4K^2+4BK+1$ and $\sum_{j=1}^{3} y_{ij}^2 \le 3K^2$. Therefore, $4x_i(B-x_i)+1 = \sum_{j=1}^{3} {y_{ij}}^2$ hold over the integer setting, which infers $x_i \in [0,B]$ based on Lemma 3.1.

Overall, it cannot distinguish the real-world execution and the ideal-world execution with statistical error at most $\frac{1}{p} + \frac{1}{2^R}$. □

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ and the (range) command of $\mathcal{F}_{\mathsf{authZK}}$ as follows:

(1) $\mathcal{S}$ sends $(\mathsf{range}, \{\mathsf{id}_i^x\}_{i\in[1,n]}, B)$ to $\mathcal{F}_{\mathsf{authZK}}$, which in turn replies with either true or false based on whether all $\{x_i\}_{i\in[1,n]}$ are in the range $[0,B]$. If $\mathcal{F}_{\mathsf{authZK}}$ responds false, $\mathcal{S}$ sends $\perp$ to $\mathcal{A}$, and then outputs false.
(2) $\mathcal{S}$ sets $\bar{y}_{i1} := 0$, $\bar{y}_{i2} := 0$ and $\bar{y}_{i3} := 0$ for $i \in [1,n]$, and randomly samples $m_k \leftarrow [0, 4nBL]$ for $k \in [1,R]$. $\mathcal{S}$ emulate (auth) command of $\mathcal{F}_{\mathsf{authZK}}$ in Step (2) of Π_{Range}, and records the identifiers and the local keys of $\{[\bar{y}_{ij}]\}_{i\in[1,n],j\in[1,3]}$ that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.
(3) $\mathcal{S}$ interacts with $\mathcal{A}$, and receives the challenges $\gamma_{ijk} \leftarrow [0,1]$ for $i \in [1,n]$, $j \in [0,3]$, $k \in [1,R]$ from $\mathcal{A}$. Next, $\mathcal{S}$ computes $\zeta_k := \mathsf{mask}(0, m_k)$ and sends ζ_k to $\mathcal{A}$ for $k \in [1,R]$.
(4) $\mathcal{S}$ emulates (poly) command of $\mathcal{F}_{\mathsf{authZK}}$, where $\mathcal{S}$ checks the identifiers that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$ are consistent with the previously recorded identifiers. If the check fails, $\mathcal{S}$ sends false to $\mathcal{A}$, or sends true to $\mathcal{A}$ otherwise.
(5) During the CheckZero procedure, $\mathcal{S}$ computes the local keys of $[z_k] := \sum_{i=1}^{n} \gamma_{i0k}[x_i] + \sum_{j=1}^{3} \gamma_{ijk}[\bar{y}_{ij}] + [m_k] - \zeta_k$ for $k \in [1,R]$ using the corresponding local keys that $\mathcal{S}$ has recorded, and sends the local keys of $[z_k]$ to $\mathcal{A}$.
(6) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Clearly, the view of adversary $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$, except for the responses $\{\zeta_k\}_{k\in[1,R]}$ from the honest prover in the real world and from S in the ideal world. The distribution of $\{\zeta_k\}_{k\in[1,R]}$ is uniform over $[4nB, 4nBL]$ based on Lemma 3.2, unless some of ζ_k are equal to $\perp$. The event that ζ_k is $\perp$ is independent and occurs with probability at most $\frac{1}{L}$ for $k \in [1,R]$. Thus, the probability that there exists $\zeta_k = \perp$ is at most $1 - (1-\frac{1}{L})^R \le \frac{R}{L}$. In conclusion, it cannot distinguish the real-world execution and the ideal-world execution with statistical error at most $\frac{R}{L}$. □

C SECURITY PROOF OF THEOREM 5.1

Proof. The correctness of this protocol is obviously satisfied in the case where both the prover and the verifier execute Π_{Lookup} honestly with each $\boldsymbol{x_i}$ being in the table $\mathbf{T}$ for $i \in [1,n]$.

Next, we consider two cases where the prover is malicious or the verifier is malicious. In each case, we construct a simulator $\mathcal{S}$, which is given access to the (lookup) command of $\mathcal{F}_{\mathsf{authZK}}$ and runs an adversary $\mathcal{A}$ as a subroutine when emulating other commands of $\mathcal{F}_{\mathsf{authZK}}$. In both cases, we show that it cannot distinguish the real-world execution and the ideal-world execution with statistical error at most $(n+d)(w+1)/p$.

Before the protocol Π_{Lookup} begins, $\mathcal{P}$ and $\mathcal{V}$ have authenticated $\{\boldsymbol{x_i}\}_{i\in[1,n]}$ by performing the (auth) command of $\mathcal{F}_{\mathsf{authZK}}$. During this procedure, $\mathcal{S}$ records $\boldsymbol{x_i}$ and its tags sent by the malicious prover, or it records the local keys sent by the malicious verifier.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ and the (lookup) command of $\mathcal{F}_{\mathsf{authZK}}$ as follows:

(1) Whenever $\mathcal{S}$ receives $\perp$ from $\mathcal{A}$, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$ which sends false to the honest verifier and aborts.

(2) $\mathcal{S}$ emulates (auth) command of $\mathcal{F}_{\mathsf{authZK}}$ in Step (2) of the protocol Π_{Lookup}. $\mathcal{S}$ records $\{\boldsymbol{s}_i\}_{i\in[1,n+d]}$ and their tags that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.
(3) $\mathcal{S}$ randomly samples $(r_1, r_2, r_3) \leftarrow (\mathbb{F}_p)^3$ and sends them to $\mathcal{A}$.
(4) $\mathcal{S}$ emulates (poly) command of $\mathcal{F}_{\mathsf{authZK}}$, where $\mathcal{S}$ checks $g'(\mathbf{X}) = h'(\mathbf{S})$. If the above check fails, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$.
(5) $\mathcal{S}$ sends $(\mathsf{lookup}, \mathbf{T}, \{\boldsymbol{id}_i^x\}_{i\in[1,n]})$ to $\mathcal{F}_{\mathsf{authZK}}$, which sends true to the honest verifier if all $\boldsymbol{x}_i$ is in the table $\mathbf{T}$ for $i \in [1, n]$, or sends false to the honest verifier otherwise.

Clearly, the view of $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$. Only the case that $\mathbf{X}$ is not the subset of the table $\mathbf{T}$ would cause a different distribution between two worlds. In this case, $g(\alpha, \beta, \gamma)$ and $h(\alpha, \beta, \gamma)$ are definitely not equivalent due to the converse of Lemma 3.4. Therefore, for randomly sampled $(r_1, r_2, r_3) \leftarrow (\mathbb{F}_p)^3$, the probability that $l(r_1, r_2, r_3) := g(r_1, r_2, r_3) - h(r_1, r_2, r_3)$ equals zero is at most $(n + d)(w + 1)/p$, which is same as the probability that $\mathcal{V}$ outputs true in the real world because the evaluation value of $l'(\mathbf{X}, \mathbf{S}) := g'(\mathbf{X}) - h'(\mathbf{S})$ is the same as l when the authenticated sets $\mathbf{X}$ and $\mathbf{S}$ are fixed. Therefore, it cannot distinguish the real world and the ideal world with statistical error at most $(n + d)(w + 1)/p$. □

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ and the (lookup) command of $\mathcal{F}_{\mathsf{authZK}}$ as follows:

(1) $\mathcal{S}$ sends $(\mathsf{lookup}, \mathbf{T}, \{\boldsymbol{id}_i^x\}_{i\in[1,n]})$ to $\mathcal{F}_{\mathsf{authZK}}$, which in turn replies with either true or false based on whether $\{\boldsymbol{x}_i\}_{[i\in n]}$ are in the table $\mathbf{T}$ or not. If $\mathcal{F}_{\mathsf{authZK}}$ responds false, $\mathcal{S}$ sends $\perp$ to $\mathcal{A}$, and then outputs false.
(2) $\mathcal{S}$ sets $\bar{\mathbf{S}} := \{\bar{\boldsymbol{s}}_i\}_{[1,n+d]}$ with all elements of $\bar{\boldsymbol{s}}_i$ being zero for $i \in [1, n + d]$. $\mathcal{S}$ emulate (auth) command of $\mathcal{F}_{\mathsf{authZK}}$ in Step (2) of Π_{Lookup}, and records the identifiers of $[\bar{\mathbf{S}}]$ that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.
(3) $\mathcal{S}$ interacts with $\mathcal{A}$ and receives (r_1, r_2, r_3) from $\mathcal{A}$. Then, $\mathcal{S}$ emulates (poly) command of $\mathcal{F}_{\mathsf{authZK}}$, where $\mathcal{S}$ checks the identifiers that are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$ are consistent with the previously recorded identifiers. If the check fails, $\mathcal{S}$ sends false to $\mathcal{A}$, or sends true to $\mathcal{A}$ otherwise.
(4) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

When there exists $\boldsymbol{x}_i \notin \mathbf{T}$, $\mathcal{A}$ receives $\perp$ in both the real world and the ideal world. Otherwise, $\mathcal{A}$ receives true in both worlds when the identifiers, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$, are consistent. Therefore, the view of $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$, and it cannot distinguish the real-world execution and the ideal-world execution. □