VerfCNN, Optimal Complexity zkSNARK for Convolutional Neural Networks

Wenjie Qu*, Yanpei Guo*, Yue Ying, Jiaheng Zhang
National University of Singapore

***Abstract*—With the widespread deployment of machine learning services, concerns about potential misconduct by service providers have emerged. Providers may deviate from their promised methodologies when delivering their services, undermining customer trust. Zero-knowledge proofs (ZKPs) offer a promising solution for customers to verify service integrity while preserving the intellectual property of the model weights. However, existing ZKP systems for convolutional neural networks (CNNs) impose significant computational overhead on the prover, hindering their practical deployment.**

To address this challenge and facilitate real-world deployment of ZKPs for CNNs, we introduce VerfCNN, a novel and efficient ZKP system for CNN inference. The core innovation of VerfCNN lies in a specialized protocol for proving multi-channel convolutions, achieving optimal prover complexity that matches the I/O size of the convolution. Our design significantly reduces the prover overhead for verifiable CNN inference. Experiments on VGG-16 demonstrate that our system achieves a prover time of just 12.6 seconds, offering a $6.7\times$ improvement over zkCNN (CCS'21). Remarkably, VerfCNN incurs only a $10\times$ overhead compared to plaintext inference on CPU, whereas general-purpose zkSNARKs typically impose overheads exceeding $1000\times$. These results underscore VerfCNN's strong potential to enhance the integrity and transparency of real-world ML services.

1. Introduction

Since the advent of AlexNet, machine learning (ML) technologies—particularly deep neural networks—have achieved remarkable success and are increasingly integrated into our daily lives. Today, machine learning models are widely deployed to provide product recommendations [1], offer medical advice [2], enable video surveillance [3], and determine creditworthiness for banks [4].

With the increasing adoption of ML services, concerns about potential misconduct by ML service providers have emerged. These service providers may not always adhere to expected behaviors when delivering their services [5]. For example, ML service providers may have financial incentives to cut costs by deploying smaller, less capable models, potentially compromising service quality [6].

A fundamental requirement for ML services is *integrity*, ensuring that predictions are generated by the specified model based solely on the customer's input data [7], [8], [9]. Integrity ensures fair treatment by maintaining the model's consistency across all users. It is also mandated by legal frameworks that prohibit discriminatory service provision [10]. Integrity also guarantees that the service provider cannot substitute the model with an alternative one.

However, the inherent opacity of ML systems makes it difficult for customers to verify compliance with integrity requirements. Even if a provider discloses technical details, users may still question their accuracy and honesty.

One potential solution to this trust issue is to disclose the model weights and input data for each inference, allowing external parties to rerun the model and verify the prediction. However, this approach compromises the service provider's intellectual property, as training ML models requires substantial investment.

Zero-knowledge proof (ZKP) [11] offers a promising solution to ensure that the execution of a machine learning (ML) model aligns with the model owner's claims without disclosing the model's proprietary weights. In ZKP, a prover generates a proof π to convince a verifier that the output of a public function f—given public input x and the prover's secret input w—is indeed $y = f(x, w)$. The secret input w, often referred to as the witness, remains confidential throughout the process. ZKP ensures that if the prover attempts to cheat, the verifier will detect it with overwhelming probability. Importantly, the proof π reveals no additional information about the secret w. By representing f as the model's computation, x as the input data, and w as the model weights, a service provider can use ZKP to prove the correct execution of the claimed ML model inference without compromising model privacy.

In recent years, extensive research has been conducted on building ZKP for ML, covering a range of models from classic decision trees to modern neural networks like Convolutional Neural Networks (CNN) [9], [7], [12] and Large Language Models (LLM) [13], [14]. Since CNNs are widely deployed in security-critical scenarios where transparency is of great importance, including medical image analysis, face identification, autonomous driving, and automated grading, in this paper, we primarily focus on building ZKP for CNNs.

Although various studies have explored building ZKP for CNN, there are still significant gaps in the performance of existing systems for real-world deployment. For instance, the well-known work zkCNN [9] (CCS'21) achieved a proof generation time of 88 seconds for the inference of VGG16 [15] model. ZKML [14] (Eurosys'2024) reported a prover time of about 50 seconds on the ResNet-18 [16]

* The first two authors contributed equally.

model. However, in comparison, plaintext inference for these CNN models on the same CPU hardware takes about only a second. These results indicate that even state-of-the-art proof systems for CNNs introduce at least $40\times$ overhead compared to plaintext inference. This substantial prover overhead poses a significant barrier to the deployment of these systems in the real world.

Convolutional layers dominate the computational cost in plain-text CNN inference and constitute the main prover time bottleneck in existing ZKP systems for CNN. zkCNN [9] (CCS'21) was the first to design specialized protocols for convolutional layers. Their method proves a single convolution by proving a Fast Fourier Transform (FFT). Their design of zkFFT enables proving a single convolution in strictly linear time. However, a convolutional layer consists of multiple input and output channels, revealing a significant drawback in their approach. Specifically, their method extends each convolutional kernel to the input size using FFT. Since convolutional kernels are typically much smaller than the input size and a convolutional layer contains numerous different kernels, this extension results in considerable prover time overhead [7].

Balbás et al. [7] (CCS'23) proposed an alternative protocol that unrolls the convolutional layer into matrix multiplications, which is easy to prove in ZKP. However, this method also has a notable limitation: unrolling convolutions into matrices roughly replicates each element in the input matrix approximately m^2 times, where m is the kernel size. This redundancy leads to considerable computational overhead for the prover.

The above analysis indicates that existing protocol designs for convolutional layers do not achieve a prover complexity that is *linear with respect to the I/O size* (the sum of the input and output sizes of the convolution). Since the proof of convolutional layers is the performance bottleneck of most ZK for CNN systems, this limitation significantly hinders overall efficiency. This leads to the following research question:

Research Question: *Can we design a protocol to prove convolutional layers with optimal prover complexity?*

1.1. Our Contributions

We affirmatively answer this question by introducing a novel and efficient protocol for proving convolutional layers. Our contributions are listed as follows:

- **Efficient protocol for convolutional layers.** Our core contribution is a novel polynomial interactive oracle proof (PIOP) [17] for proving convolutional layers with multiple input and output channels. For the convolution between a $n \times n$ image with C input channels and a $m \times m$ kernel with C input channels and D output channels, our protocol achieves prover complexity $O(Cn^2 + Dn^2 + CDm^2)$, which is exactly the same with I/O size of a multi-channel convolution statement. This indicates that our protocol achieves theoretical optimal complexity for a multi-channel convolution. In contrast, zkCNN [9] achieved $O(CDn^2)$ prover complexity, and Balbás et al. [7] achieved $O(Cm^2n^2 + Dn^2 + CDm^2)$.
- **Efficient proof system for CNN inference.** Based on our efficient PIOP for proving multi-channel convolution, we build an efficient proof system that supports end-to-end proving convolution neural networks. To achieve low cost on proving ReLU and max-pooling layer, we adopt the logUp [18], [19] lookup argument. The design of our system supports proving prevalent CNN architectures like VGG16 [15].
- **Optimized proof for 1-D convolution.** The proof of 1-D convolutions is a key building block in our protocol for proving convolutional layers. Although zkCNN has already achieved optimal complexity in proving 1-D convolutions, we propose a scheme that is concretely faster. Based on paper-and-pencil operation counts, our scheme has $4\times$ fewer field multiplications than zkFFT, and may be of independent interest.
- **Implementation and evaluations.** We fully implement our zero knowledge convolutional neural network system, named VerfCNN. We test it on VGG-11 and VGG-16 on the CIFAR-10 dataset [20]. For VGG-16, it takes 12.6s to generate a proof of prediction, which is $6.7\times$ more efficient than zkCNN [9] (CCS'21) and $2.2\times$ more efficient than Balbás et al. [7] (CCS'23). Remarkably, our system achieves only about $10\times$ overhead compared with the inference on CPU in plaintext. These results highlight the significant potential of our system to enhance the integrity and transparency of real-world ML services.

1.2. Related works

There has been extensive research on developing ZKPs for machine learning. Existing works focus on three types of ML models: CNNs, decision trees, and LLMs. For CNNs, early works such as [21], [22], and [23] explored proving CNN inference. These approaches incur substantial overhead, which require several hours or even days to generate proofs for CNN inference. Mystique [12] (CCS'21) utilized the VOLE [24] backend to prove the correctness of CNNs. A significant milestone in this field is zkCNN [9] (CCS'21), which was the first to develop a specialized protocol for proving convolutional layers, leveraging efficient proofs based on FFT. More recently, Balbás et al. [7] (CCS'23) unrolled convolution operations into matrix multiplications to enable efficient proof generation for convolutional layers. Aside from designing specialized protocols for convolutional layers, ZKML [14] (EuroSys'24) explored building ZKPs for CNNs using general-purpose zkSNARKs. It represents CNN computations as arithmetic circuits and introduce various optimization techniques to enhance efficiency.

Decision trees have also been a key focus in the ZKP for ML literature. A pioneering study, zkDT [8] (CCS'20), introduced a scheme for verifiable decision tree inference. Subsequent research has built upon this foundation: [25] (PKC'24) improved efficiency through an optimized matrix lookup protocol, while Sparrow [26] (CCS'24) introduced a memory-efficient SNARK to further reduce the cost of

[8]. More recently, proving inference for LLMs has gained significant attention. Prominent works in this area include zkLLM [13] (CCS'24), Hao et al.[27] (Security'24), and zkGPT [6] (Security'25).

In this section, we focus on the main techniques introduced in zkCNN [9] and Balbás et al. [7], which developed efficient, dedicated protocols for proving convolutional layers. These works are most closely related to our research.

1.2.1. zkCNN.

zkFFT. The design of zkCNN [9] capitalizes on the intrinsic relationship between convolution and FFT. Their approach begins by designing an efficient ZKP protocol for FFT. By definition, FFT converts a polynomial from its coefficient representation to its evaluation at powers of a root of unity. Formally speaking, let $c \in \mathbb{F}^N$ be the vector of coefficients of a polynomial, and let $a \in \mathbb{F}^M$ be the vector of evaluations at $\left(\omega^0, \omega^1, \ldots, \omega^{M-1}\right)$, $a_j = \sum_{i=0}^{N-1} c_i \omega^{ji}$ for $j = 0, 1, \ldots, M-1$. In matrix-vector form, the FFT relation becomes $a = F \cdot c$. Here, F denotes the standard Fourier matrix, as defined in Equation 4 of[9].

Leveraging the above relation and the classical matrix multiplication protocol [28], zkCNN transforms proving FFT into proving the following relation:

$$\tilde{a}(u) = \sum_{x \in \{0,1\}^{\log N}} \tilde{c}(x)\tilde{F}(u,x) \tag{1}$$

where vector $u \in \mathbb{F}^{\log M}$ is the challenge vector sent by the verifier. zkCNN proves Equation 1 using the sumcheck protocol. The challenge becomes how to efficiently compute the bookkeeping table $\tilde{F}(u, x), x \in \{0,1\}^{\log N}$ in $O(N)$ time. zkCNN derived that $\tilde{F}(u,x) = \prod_{i=0}^{\log M-1} \left((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{\mathcal{X}}\right)$, where $\mathcal{X} \in \mathbb{F}$ is the value represented by bit-string x. Then zkCNN designed an efficient method to compute $\tilde{F}(u,x), x \in \{0,1\}^{\log N}$ in $O(M+N)$ time. After computing $\tilde{F}(u,\cdot)$, Equation 1 is proved in $O(N)$ time by the sumcheck protocol.

Proving convolution using zkFFT. After deriving zkFFT, zkCNN proceeds to design a method based on zkFFT to prove 1-D convolution. Consider the convolution of $\vec{a} \in \mathbb{F}^n$ and $\vec{b} \in \mathbb{F}^m$, where $n \gg m$, resulting in $\vec{c} = \vec{a} * \vec{b}$. The procedure is as follows: First, we define the univariate polynomials $a(\cdot)$ and $b(\cdot)$ with coefficients corresponding to $\vec{a}$ and $\vec{b}$. $a(\cdot)$ and $b(\cdot)$ are evaluated at powers of a root of unity. The results are denoted as $\mathsf{FFT}(\vec{a})$ and $\mathsf{FFT}(\vec{b})$. In this approach, zkCNN implicitly assumes that b is *padded* to length n and both are evaluated at $2n$ points. Then, we compute the Hadamard product of $\mathsf{FFT}(\vec{a})$ and $\mathsf{FFT}(\vec{b})$. Finally, the result is transformed back into its coefficient representation using IFFT. The algorithm is given as:

$$\vec{c} = \mathsf{IFFT}(\mathsf{FFT}(\vec{a}) \circ \mathsf{FFT}(\vec{b}))$$

where "$\circ$" denotes the Hadamard product. Using the equation above, the computation of 1-D convolutions can be verified with four sumcheck protocols. These include two invocations for proving FFT, one for proving Hadamard product, and one for proving IFFT.

Proof of 2-D convolution. The next step in zkCNN is to prove 2-D convolution, a fundamental operation in CNNs Section 2.3 provides a detailed explanation of 2-D convolution. The conversion of the 2-D convolution of $X \in \mathbb{F}^{n \times n}$ and $W \in \mathbb{F}^{m \times m}$ to a 1-D convolution is as follows: Let $Y \in \mathbb{F}^{n \times n}$ be the result of the 2-D convolution, padded to the shape of $n \times n$. Define $\bar{X}, \bar{W}, \bar{Y} \in \mathbb{F}^{n^2}$ as

$$\bar{X}_{tn+l} = X_{n-1-t,n-1-l}, \quad 0 \le t < n, 0 \le l < n$$

$$\bar{W}_{tn+l} = \begin{cases} W_{t,l}, & 0 \le t, l < m \\ 0, & \text{otherwise} \end{cases}$$

$$\bar{Y}_j = \sum_{i=0}^{j} \bar{X}_{j-i}\bar{W}_i$$

zkCNN derived in their paper that $Y_{j,k} = \bar{Y}_{n^2-1-j \cdot n-k}$. According to the above equation, Y can be computed through a 1-D convolution between $\bar{X}$ and $\bar{W}$ vectors. So far, zkCNN has demonstrated how to prove a single 2-D convolution. However, in a CNN, multi-channel 2-D convolution is performed at each convolutional layer. For a convolutional layer with C input channels and D output channels, zkCNN runs its protocol $C \cdot D$ times, resulting in a prover cost of $O(CDn^2)$ for a convolution between $X \in \mathbb{F}^{C \times n \times n}$ and $W \in \mathbb{F}^{C \times D \times m \times m}$. The inefficiency of zkCNN stems from performing FFT for all $C \cdot D$ different kernels, which involves extending small kernels to large, input-sized non-zero vectors.

1.2.2. Balbás et al. Balbás et al. [7] introduced an alternative technique to prove the correctness of 2-D convolution by unrolling it as a matrix multiplication. For instance, given the 2-D convolution between $X * W$:

$$X = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{bmatrix}, W = \begin{bmatrix} w_0 & w_1 \\ w_2 & w_3 \end{bmatrix} \tag{2}$$

Balbás et al. [7] unrolled $Y = X * W$ into matrix multiplication as:

$$\begin{bmatrix} x_0 & x_1 & x_3 & x_4 \\ x_1 & x_2 & x_4 & x_5 \\ x_3 & x_4 & x_6 & x_7 \\ x_4 & x_5 & x_7 & x_8 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{3}$$

For an input $X \in \mathbb{F}^{n \times n}$ and a kernel $W \in \mathbb{F}^{m \times m}$, X is transformed into a matrix $F \in \mathbb{F}^{(n-m+1)^2 \times m^2}$. W is reshaped into a vector of length m^2. The convolution result Y is reshaped into a vector of length $(n-m+1)^2$.

This approach can be further extended to support multiple channels in a convolutional neural network. Consider the case where there are C input channels and D output channels. We extend the definition of the unrolled input F for the multi-channel case as $F \in \mathbb{F}^{C \times (n-m+1)^2 \times m^2}$. where the weight kernel is

denoted as $W \in \mathbb{F}^{C\times D\times m^2}$. The convolution's output $Y \in \mathbb{F}^{D\times(n-m+1)^2}$ is computed by:

$$Y = [Y_0|\cdots|Y_{D-1}] = \sum_{c\in[C]} F_c \cdot [W_{c,0}|\cdots|W_{c,D-1}] \quad (4)$$

For each input channel c, there is a matrix multiplication between a $(n-m+1)^2\times m^2$ matrix and a $D\times m^2$ matrix; the resulting Y has a shape of $D\times(n-m+1)^2$.

$$Y_{d,x} = \sum_{c\in[C]}\sum_{y\in[m^2]} F_{c,x,y}\cdot W_{c,d,y} \quad (5)$$

Proving the above equation is very similar to proving matrix multiplication using the sumcheck protocol in [28]. The prover complexity is $O(D(n-m+1)^2 + C((n-m+1)^2m^2 + Dm^2))$, which simplifies to $O(Cn^2m^2 + Dn^2 + CDm^2)$, since in most real-world scenarios, $n \gg m$. The inefficiency of Balbás et al. stems from constructing the matrix F of size $O(Cn^2m^2)$, which results in each element of the input matrix X being replicated $O(m^2)$ times on average.

2. Preliminary

2.1. Notations

Let λ and $\mathsf{negl}(\lambda)$ be the security parameter and a negligible function. We denote a prime field $\mathbb{F}_p = \mathbb{F}$, cycle groups $\mathbb{G}$ of order p. We represent all groups additively.

For $n \in \mathbb{N}$, we denote $\{0,\cdots,n-1\}$ by $[n]$. By default, the indices of vectors, matrices, and tensors start at 0 unless stated otherwise. For any two vectors v_1, v_2, we denote their concatenation by $v_1||v_2$. We denote the natural injection mapping an integer to its binary representation by $bin(\cdot)$.

2.2. Multilinear extension

Define the identity function $eq(x,y)$, which takes two bit-strings x and y of the same length as input. The function $eq(\cdot,\cdot)$ returns 1 if $x = y$; and 0 otherwise. Suppose $\tilde{eq}(\cdot,\cdot)$ is the multilinear extension of $eq(\cdot,\cdot)$. $\tilde{eq}(x,y) = \prod_{i=1}^{\ell}((1-x_i)(1-y_i)+x_iy_i)$. We present the definition of multilinear Extension as follows:

Definition 1 (Multilinear Extension [29], [30]). *Let $V : \{0,1\}^\ell \to \mathbb{F}$ be a function. The multilinear extension of V is the unique polynomial $\tilde{V} : \mathbb{F}^\ell \to \mathbb{F}$ such that $\tilde{V}(x_1,x_2,...,x_\ell) = V(x_1,x_2,...,x_\ell)$ for all $x_1,x_2,\ldots,x_\ell \in \{0,1\}^l$. $\tilde{V}$ can be expressed as $\tilde{V}(x_1,x_2,...,x_\ell) = \sum_{b\in\{0,1\}^\ell}\tilde{eq}(x,b)\cdot V(b)$.*

For a vector Z of length 2^ℓ, the multilinear extension of $\tilde{Z}$ is defined as: $\tilde{Z}(x_1,\cdots,x_\ell) = \sum_{b\in[2^\ell]}\tilde{eq}(x,bin(b))\cdot Z_b$.

For a public vector in the form of tensor-product [31], [32], [33], its multilinear extension (MLE) can be efficiently evaluated by the verifier at a random point. Formally, we have the following lemma:

Lemma 2.1. *[32] Let the vector $Z \in \mathbb{F}^{2^\ell} = \langle a_1,b_1\rangle \otimes \langle a_2,b_2\rangle\cdots\otimes\langle a_\ell,b_\ell\rangle$. $\forall x \in \mathbb{F}^\ell$, $\tilde{Z}(x) = \prod_{i=1}^{\ell}((1-x_i)\cdot a_i + x_i\cdot b_i)$ (the indices of x start from 1).*

The proof is presented in Appendix B.

2.3. Convolutional neural network

1-D convolution. The convolution of vectors $\vec{a} \in \mathbb{F}^n$ and $\vec{b} \in \mathbb{F}^m$, denoted as $\vec{c} = \vec{a} * \vec{b}$, is defined as $\vec{c}_j = \sum_{i=0}^{j}\vec{a}_i\vec{b}_{j-i}$ for $j \in [n+m-1]$. The convolution of vectors has an intrinsic link to the multiplication of univariate polynomials. Specifically, the relationship can be expressed as:

$$\begin{aligned} c(X) &= \sum_{j\in[n+m-1]} \vec{c}_j X^j \\ &= \sum_{j\in[n+m-1]}\sum_{i=0}^{j}(\vec{a}_iX^i)\cdot(\vec{b}_{j-i}X^{j-i}) \\ &= (\sum_{i\in[n]}\vec{a}_iX^i)\cdot(\sum_{i\in[m]}\vec{b}_iX^i) \\ &= a(X)\cdot b(X) \end{aligned} \quad (6)$$

This demonstrates that the output of 1-D convolution corresponds to the coefficients of the product of the polynomial forms of the input vectors.

A CNN consists of a pipeline of distinct layer types, typically including convolutional layers, ReLU layers, max-pooling layers, and fully-connected layers. The first four types of layers are usually arranged in sequence and followed by one or more fully connected layers. Each layer type serves a specific function in transforming the input data and producing an output that feeds into the next layer in the network.

Convolutional layer. A convolutional layer computes the two-dimensional (2-D) convolution between its input data and its weight matrix. The weight matrix is usually referred to as a kernel or filter. Formally, the result of a 2-D convolution between two matrices, X of size $n\times n$ and W of size $m\times m$, is a matrix Y with dimensions $(n-m+1)\times(n-m+1)$. The 2-D convolution $Y = X * W$ is computed as:

$$\begin{aligned} Y_{j,k} = \sum_{t=0,\ell=0}^{m-1,m-1} X_{j+t,k+\ell}\cdot W_{t,\ell} \\ \text{for } j,k = 0,\ldots,n-m \end{aligned} \quad (7)$$

Both the input and output of each convolutional layer usually have an additional channel dimension in addition to width and height. Each convolutional layer applies distinct kernels for different input channels and output channels. We use C and D to denote the number of input and output channels, respectively. Let W represent the set of model parameters in this convolutional layer, consisting of $C\cdot D$ matrices of size $m\times m$. The input X has shape $C\times n\times n$, weight tensor W has shape $C\times D\times m\times m$, and output Y has shape $D\times(n-m+1)\times(n-m+1)$. Then, the convolutional layer for each data sample computes

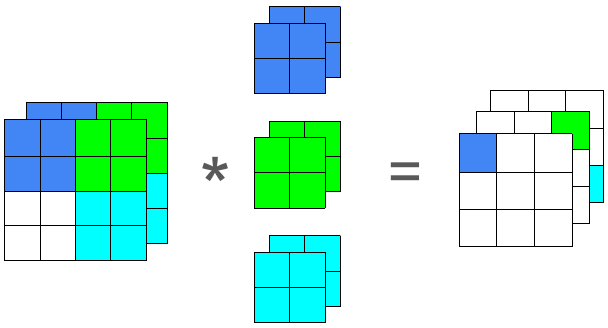


Figure 1: Example illustrating a multi-channel convolution with an input of $C = 2$ channels and size $n = 4$, an output of $D = 3$ channels, a kernel size of $m = 2$, and an output size of $n - m + 1 = 3$.

$$Y_d = \sum_{c \in [C]} X_c * W_{c,d}, d \in [D] \tag{8}$$

From the equation above, we observe that $C \cdot D$ 2-D convolutions are computed in a convolutional layer. In practical CNN architectures, the most widely used kernel size m is 3. Consequently, for most convolutional layers, the input size n is significantly larger than the kernel size, i.e., $n \gg m$. Furthermore, for most convolutional layers in modern CNNs, the number of input channels C and output channels D are typically of the same order of magnitude. For instance, in the VGG-16 [15] network, most convolutional layers follow the pattern $C = D$ or $2 \cdot C = D$. C and D are typically in the range of a few hundred, making $C, D \gg m$.

ReLU layer. In the ReLU layer, the ReLU function [34] is applied element-wise to its input vector. The ReLU function is defined as $\mathsf{ReLU}(x) = \max(x, 0)$.

Max-pooling layer. The max-pooling layer slides a kernel over its input matrix and computes the maximum value within the region covered by the kernel. Let the input vector of the max-pooling layer be $Y \in \mathbb{F}^{s \cdot k}$, and the output vector be $Z \in \mathbb{F}^k$. The output is defined as $Z_i = \max_{j \in K_i}\{Y_j\}$, where $|K_i| = s$ represents the set of indices of Y corresponding to Z_i.

Fully connected layer. In the fully connected layer, the input is processed by multiplying it with a weight matrix and adding a bias vector.

Output layer. During CNN inference, the predicted label can be determined by finding the maximum value from the output vector of the last layer. Therefore, in our system, following existing works [9], [21], [22], we omit the proof of the Softmax [35] function computation; instead, we prove that the output logits for each class are correct.

2.4. Quantization

Real number calculations in ML models pose significant challenges for ZKP. Most ZKP for ML approaches [21], [14], [23], [9], [13] utilize quantization to convert real numbers to integers, making computations much easier to prove. We adopt the quantization approach adapted from previous works [36], [13], [37], in which a real number x is mapped to an signed integer q using the formula $q = \lfloor x \cdot 2^Q \rfloor$. Here, Q is a positive integer that determines the quantization scale, represented by 2^Q. Under this scheme, all model parameters and intermediate results in our computation are represented as B-bit integers. The value of B is the smallest integer that ensures all floating-point values x during inference satisfy $-2^{B-1} \leq x \cdot 2^Q < 2^{B-1}$.

When two numbers are added, we add their integer parts: $x_1 + x_2 = \frac{q_{x_1} + q_{x_2}}{2^Q}$. However, when multiplying two numbers, *rescaling* is necessary to avoid overflow. Specifically, when multiplying $x = \frac{q_x}{2^Q}$ and $y = \frac{q_y}{2^Q}$ to obtain $z = \frac{q_z}{2^Q}$, we compute $q_z = \lfloor \frac{q_x}{2^Q} \cdot \frac{q_y}{2^Q} \cdot 2^Q \rfloor = \lfloor \frac{q_x \cdot q_y}{2^Q} \rfloor$. This approach allows floating-point addition and multiplication to be approximated using integer arithmetic. Most ZKP constructions work in finite fields. For a negative integer $q < 0$, we represent it in $\mathbb{F}$ as $p - (-q)$, and for a non-negative integer $q \geq 0$, we map it to its corresponding field element directly. Combined with the fixed-point encoding above, we map any real number x to a field element in $\mathbb{F}_p$.

3. Technical Overview

In this section, we introduce our core technique.

Verifiable one-dimensional convolution. Proving 1-D convolutions serve as a fundamental building block in our protocol for proving convolutional layers. We first introduce a protocol for proving 1-D convolution that is more efficient than zkFFT. Our design leverages the connection between convolution and univariate polynomial multiplication. It avoids the need for proving FFT in zkCNN [9]'s approach, which incurs high overhead due to large constant factors. Given the multilinear commitments to vectors $\vec{a}$, $\vec{b}$, $\vec{c}$, the verifier can verify $\vec{a} * \vec{b} = \vec{c}$ by sampling a random $\beta \in \mathbb{F}$ and requiring the prover to prove $a(\beta) \cdot b(\beta) = c(\beta)$, where $a(\cdot), b(\cdot), c(\cdot)$ are univariate polynomials whose coefficients correspond to the elements of vectors $\vec{a}$, $\vec{b}$, $\vec{c}$, respectively. In our proposed protocol, the prover proves $(\sum_{i \in [|\vec{a}|]} \beta^i a_i) \cdot (\sum_{i \in [|\vec{b}|]} \beta^i b_i) = \sum_{i \in [|\vec{c}|]} \beta^i c_i$ to the verifier. Specifically, the prover proves the correctness of $\sum_{i \in [|\vec{a}|]} \beta^i a_i$, $\sum_{i \in [|\vec{b}|]} \beta^i b_i$, and $\sum_{i \in [|\vec{c}|]} \beta^i c_i$ individually using the sumcheck protocol.

Although the construction appears straightforward, an important challenge remains. At the end of the sumcheck protocol for each term, the verifier must efficiently evaluate the multilinear extension of the "monomial vector" with the form $\{1, \beta^1, \cdots, \beta^u, 0, \cdots, 0\}$ at a random point. This results in a non-succinct verifier. To overcome this challenge, we propose the *twisted evaluation* protocol for multilinear polynomials to prove $\sum_{i \in [u]} \beta^i \tilde{M}(\mathrm{bin}(i))$ given the commitment to the multilinear extension of M.

As analyzed in detail in Section 4.1, our method achieves optimal prover complexity for proving 1-D convolution while significantly improving the constant factors compared to zkCNN [9].

Our 1-D convolution proving technique can be extended to proving the sum of multiple convolutions. More precisely,

	zkCNN [9] (CCS'21)	Balbás et al. [7] (CCS'23)	Ours
Prover Time	$O(CDn^2)$	$O(Cm^2n^2 + Dn^2 + CDm^2)$	$O(Cn^2 + Dn^2 + CDm^2)$
Proof size	$O(\log^2(CDn^2))$	$O(\log(Cm^2))$	$O(\log(Cn^2))$
Verifier Time	$O(\log^2(CDn^2))$	$O(\log(Cm^2))$	$O(\log^2 n + \log(Cn^2))$

TABLE 1: Performance of existing convolution protocols, where C denotes input channel number, D denotes output channel number, n is input size, and m is kernel size. The proof size and verifier time do not account for the opening cost of a polynomial commitment, which is scheme-dependent. Detailed analysis could be found at the end of Section 4.3.

let the input be $a \in \mathbb{F}^{C\times N}$ and the kernel be $b \in \mathbb{F}^{C\times K}$, where the sum of convolutions is $Y \in \mathbb{F}^{N+K-1}$. The statement we now want to prove is $\sum_{c\in[C]} a_c * b_c = Y$. The details of our technique for proving the sum of multiple convolutions are in Section 4.2.

Accelerating multi-channel convolution by random combination. In this section, we construct an efficient proof of multi-channel convolution on top of the previous techniques. For multi-channel convolution, we aim to prove that: $\sum_{c\in[C]} X_c * W_{c,d} = Y_d, d = 0, \cdots, D-1$, where X has shape $C \times n \times n$, W has shape $C \times D \times m \times m$, and Y is the convolution result.

A strawman protocol to prove this relation is to directly run our single-convolution protocol $C \cdot D$ times. However, this approach results in a high prover complexity of $O(CDn^2)$, which is the same as zkCNN. In this strawman protocol, we observe that across the $C \cdot D$ invocations of convolutions, different kernels are used each time, but each input channel is reused in D convolutions. This redundancy provides an opportunity to reduce prover computation.

Our high-level idea is to merge the checks of these $C \cdot D$ convolutions by randomly linearly combining the $C \cdot D$ kernels into C kernels. By the linearity of convolution, the result of convolving the input with the randomly combined kernel is equivalent to the original output combined in the same manner. Roughly speaking, this reduces the proof of $C \cdot D$ convolutions into proving C convolutions of the same shape.

Specifically, the verifier samples $r_D \in \mathbb{F}^{\log D}$ as the random linear combination factor. After combination, we check the following relation:

$$\begin{aligned} &\sum_{d\in[D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot \left(\sum_{c\in[C]} X_c * W_{c,d}\right) \\ &= \sum_{d\in[D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot Y_d \end{aligned} \tag{9}$$

By the linearity of convolution, we can transform the left-hand side of Equation 9 and obtain the following expression:

$$\begin{aligned} &\sum_{c\in[C]} \left(X_c * (\sum_{d\in[D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot W_{c,d})\right) \\ &= \sum_{d\in[D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot Y_d \end{aligned} \tag{10}$$

Equation 10 takes the form of the sum of C convolutions, which can be proved using previously proposed techniques. A more detailed illustration of these techniques can be found in Section 4.3. In conclusion, our method achieves prover complexity $O((C+D)n^2+CDm^2)$, which matches the I/O size of multi-channel convolution (the sum of input size and output size). This indicates that our method is *the first to achieve optimal prover complexity in proving multi-channel convolution*.

Comparison with previous work. Table 1 presents a comparison of our method, zkCNN [9], and Balbás et al. [7]. For the prover time, our method is not only asymptotically more efficient, but also concretely much faster. Refer to the properties of C, D, n, m described in Section 2.3. Compared to zkCNN [9], given that $n \gg m$ and $C \cdot D \gg (C + D)$, it follows that $CDn^2 \gg (C + D)n^2 + CDm^2$. For Balbás et al. [7], since C and D are of the same order of magnitude, it follows that $Cm^2n^2 \gg Cn^2, Dn^2$. Since n typically ranges from thousands to tens of thousands for CNNs processing moderate-sized images, while D is usually only a few hundred, the condition $n^2 \gg D$ holds. Consequently, we can deduce that $Cm^2n^2 \gg CDm^2$. Therefore, we establish the relationship $Cm^2n^2+Dn^2+CDm^2 \gg (C+D)n^2+CDm^2$. Empirical results presented in Section 5.2 demonstrate our concrete prover time advantage over previous schemes.

The core intuition behind our complexity reduction, compared to prior works, is twofold: (1) Compared to zkCNN [9], which proves each of the $C \cdot D$ convolutions independently—resulting in an overall cost of $O(CDn^2)$—our random linear combination reduces it into proving C convolutions, cutting an $O(D)$ factor down. (2) Compared to Balbás et al. [7], their approach transforms a multi-channel convolution into a matrix multiplication between two matrices of size $O(Cn^2m^2)$ and $O(Dm^2)$, respectively. The dominant $O(Cn^2m^2)$ term arises from an $O(m^2)$ times redundancy introduced by unrolling the input image into the matrix form. In contrast, our method proves convolution via univariate polynomial multiplication, avoiding this unrolling step, and thus eliminating the $O(m^2)$ times overhead.

For proof size, our method achieves much smaller proof size compared with zkCNN [9] and at the same order of magnitude with Balbás et al. [7]. Our verifier time is $O(\log^2 n)$ due to the twisted evaluation PIOP, larger than that of Balbás et al.[7]. Notably, this complexity has little impact on practical applications, as the $O(\log^2 n)$ overhead arises purely from field operations, which are relatively inexpensive. For convolutions in typical scenarios, our verifier performs fewer than 1000 field operations for checking a single convolutional layer, which suggests that the verifier's time for a single layer is less than 0.1 milliseconds (estimated based on profiling results in zk-Bench[38]). There-

fore, the verifier time of our protocol will not become a bottleneck in an end-to-end proof system for CNN inference in practice.

4. Methodology

4.1. Efficient proof of one-dimensional convolution

Verifiable one-dimensional convolution addresses the following question: given committed vectors $\vec{a}$, $\vec{b}$, $\vec{c}$, the prover tries to prove that $\vec{c} = \vec{a} * \vec{b}$. zkCNN achieves proving the convolution relation in strictly linear time by introducing a novel primitive called zkFFT, which enables the proof of FFT and IFFT operations with linear complexity. In conventional plaintext computation, 1-D convolution is typically performed by applying FFT twice followed by a single IFFT. zkCNN leverages the zkFFT primitive to efficiently prove this entire process.

While zkCNN achieves optimal asymptotic complexity for proving 1-D convolution, its concrete prover time still has significant room for improvement. Rather than proving the intricate process of FFT-based convolution, we adopt a more streamlined approach by directly leveraging the relationship between the convolution inputs and outputs.

Building on the connection between convolution and univariate polynomial multiplication, as stated in Equation 6, we construct a more efficient convolution proof: the verifier samples a random challenge $\beta \in \mathbb{F}$, and the prover shows that $\vec{a} * \vec{b} = \vec{c}$ by proving the following equation:

$$(\sum_{i\in[|\vec{a}|]} \beta^i a_i) \cdot (\sum_{i\in[|\vec{b}|]} \beta^i b_i) = \sum_{i\in[|\vec{c}|]} \beta^i c_i \tag{11}$$

This approach significantly simplifies the proof process and benefits prover efficiency. Due to the efficiency of sumcheck protocol, following existing works in ZKP for CNN [9], [7], we commit $\vec{a}, \vec{b}, \vec{c}$ by a multilinear polynomial commitment scheme. Each term in $\sum_{i\in[|\vec{a}|]} \beta^i a_i$, $\sum_{i\in[|\vec{b}|]} \beta^i b_i$, and $\sum_{i\in[|\vec{c}|]} \beta^i c_i$ is proven by the sumcheck protocol.

Although the above construction may appear simple, a significant challenge remains. At the end of the sumcheck protocol on proving the three terms, the verifier has to efficiently evaluate the multilinear extension of the "monomial vector" with the form $\{1, \beta^1, \cdots, \beta^u, 0, \cdots, 0\}$ at a random point. In our protocol, u takes value $|\vec{a}|-1, |\vec{b}|-1, |\vec{a}|+|\vec{b}|-2$, respectively. A straightforward evaluation of the multilinear extension of the "monomial vector" results in an $O(|\vec{a}|+|\vec{b}|)$ verifier complexity, which is non-succinct. To deal with this challenge, we extend the evaluation of multilinear polynomial commitment into a new functionality called *twisted evaluation*, denoted as TwEval. Roughly speaking, for $u \in \{1, 2, 3, \cdots, 2^\ell\}, \beta \in \mathbb{F}$, twisted evaluation enables efficient verification of the sum $\sum_{i\in[u]} M(\mathrm{bin}(i)) \cdot \beta^i$, given the multilinear commitment to $\tilde{M}$. It is formally defined as: $\mathsf{TwEval}(\beta, u, \tilde{M}) \to (S, \pi)$, takes input evaluation point β and polynomial $\tilde{M}$, outputs evaluation result S and evaluation proof π. It is an argument of the following relation:

$$\left\{ \mathbb{x} = (\mathcal{C}, \beta, u, S); \mathbb{w} = \tilde{M} : \begin{array}{r} \sum_{i\in[u]} \tilde{M}(\mathrm{bin}(i)) \cdot \beta^i = S \\ \mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{M}) = 1 \end{array} \right\}$$

Next, we present how to construct the twisted evaluation PIOP of a committed multilinear polynomial $\tilde{M}$. We define multi-variate function $B : \{0,1\}^\ell \to \mathbb{F}$ as:

$$B(\mathrm{bin}(x)) = \begin{cases} \beta^x, & x \in [u] \\ 0, & x \in [2^\ell] \setminus [u] \end{cases}$$

$\mathcal{P}$ runs the sumcheck protocol with $\mathcal{V}$ on $S = \sum_{b\in\{0,1\}^\ell} M(b) \cdot B(b)$. Let the final message of the sumcheck protocol be $m_\ell = \tilde{M}(\vec{r}) \cdot \tilde{B}(\vec{r})$, where $\vec{r}$ is the randomness used in sumcheck. The key challenge is that $\mathcal{V}$ needs to compute the evaluation of $\tilde{B}(\vec{r})$ on its own. We observe although $\vec{B}$ does not have the structure of tensor product, but it can be decomposed as the sum of $O(\ell)$ tensor products. As an example, for $u = 6$, $\ell = 3$, the monomial vector is $\{1, \beta^1, \cdots, \beta^5, 0, 0\}$. It is "split" into 2 vectors of length 8: $\{1, \beta^1, \beta^2, \beta^3, 0, 0, 0, 0\}$ and $\{0, 0, 0, 0, \beta^4, \beta^5, 0, 0\}$, these two vectors can be represented in tensor-product form as: $\langle 1, \beta\rangle \otimes \langle 1, \beta^2\rangle \otimes \langle 1, 0\rangle$ and $\langle 1, \beta\rangle \otimes \langle 1, 0\rangle \otimes \langle 0, \beta^4\rangle$.

By Lemma 2.1, since each splitted vector satisfy the tensor-product structure, $\mathcal{V}$ could compute the MLE evaluation of each tensor at the random point $\vec{r}$, and obtain $\tilde{B}(\vec{r})$ by summing these evaluations.

We outline the concrete protocol for twisted evaluation in Protocol 1. The following theorem gives the completeness and soundness of the twisted evaluation PIOP.

Theorem 4.1. *The PIOP for twisted evaluation is perfectly complete and has soundness error* $\frac{\ell}{|\mathbb{F}|}$.

The proof of Theorem 4.1 is detailed in Appendix C.

Using the twisted evaluation protocol, given the multilinear commitments of $\vec{a}, \vec{b}$ and $\vec{c}$, $\mathcal{P}$ can succinctly prove to $\mathcal{V}$ the values of $\sum_{i\in[|\vec{a}|]} \beta^i a_i, \sum_{i\in[|\vec{b}|]} \beta^i b_i$ and $\sum_{i\in[|\vec{c}|]} \beta^i c_i$.

4.1.1. Analysis of our verifiable convolution scheme. We now analyze the cost of our new protocol. For simplicity, we compare zkFFT in the context of proving 1-D convolution. The convolution takes an input of length $n = 2^u$, convolved with another input of length $m = 2^v$. In real-world scenarios, it is typically the case that $n \gg m$. For zkCNN, there are four procedures to analyze (1) $\mathsf{FFT}(\vec{a})$, (2) $\mathsf{FFT}(\vec{b})$, (3) the hadamard product (4) IFFT. In our proposed method, the prover's computation consists of $\mathsf{TwEval}(\vec{a})$, $\mathsf{TwEval}(\vec{b})$, and $\mathsf{TwEval}(\vec{c})$ respectively. The comparison results are presented in Table 2.

From Table 2, we conclude that our method reduces the number of field multiplications by approximately $4\times$, given that $n \gg m$. The efficiency of our approach in handling constants stems from several key factors: (1) zkFFT incurs a high cost due to evaluating $b(\cdot)$ on $2n$ points, whereas our method effectively exploits the fact that $\vec{b}$ is significantly

Protocol 1 (**Twisted evaluation**). • *Input: Multilinear commitment to vector v, random point β, degree limit u, value S claimed to equal $\sum_{i\in[u]} \tilde{v}(\text{bin}(i)) \cdot \beta^i$.*
Define $\tilde{B} : \mathbb{F}^{\log n} \to \mathbb{F}$, $\tilde{B}(\text{bin}(j)) = \beta^j, j \in [u], \tilde{B}(\text{bin}(j)) = 0, j \in [n] \setminus [u]$.
• *$\mathcal{P} \to \mathcal{V}$: claim on S.*
• *$\mathcal{P} \leftrightarrow \mathcal{V}$: Run sumcheck protocol together on $S = \sum_{i\in\{0,1\}^{\log n}} \tilde{v}(i) \cdot \tilde{B}(i)$, let the last message be m. m is claimed to equal $\tilde{v}(r') \cdot \tilde{B}(r')$, $r' \in \mathbb{F}^{\log n}$.*
• *$\mathcal{P} \leftrightarrow \mathcal{V}$: Run the* PCS.open *protocol on $\tilde{v}$, verifying the claim on $\tilde{v}(r')$.*
• *$\mathcal{V}$: ① Decompose u into $u = \sum_{i=0}^{\log n-1} 2^i \cdot b_i$, define set $E = \{i | b_i = 1\}$. Define $B'_k \in \mathbb{F}^{2^\ell}$ as:*

$$B'_k = \langle 1, \beta\rangle \otimes \cdots \otimes \langle 1, \beta^{2^{E_k-1}}\rangle \otimes < 1, 0 > \otimes < 1 - b_{E_k+1}, \beta^{2^{E_k+1}} \cdot b_{E_k+1} > \otimes \cdots \otimes < 1 - b_n, \beta^{2^{n-1}} \cdot b_n > \quad k = 1, \cdots, |E| \tag{12}$$

② For $k = 1, \cdots, |E|$,
compute $\tilde{B}'_k(r')$ leveraging Equation 12 and Lemma 2.1.
③ Check $m \stackrel{?}{=} \tilde{v}(r') \cdot \sum_{k=1}^{|E|} \tilde{B}'_k(r')$.

Figure 2: The protocol for proving twisted evaluation.

	Procedure	Bookkeeping Table	Sumcheck	Total
zkFFT	$\mathsf{FFT}(\vec{a})$	$6n$	$\frac{9}{2}n$	$62n$
	$\mathsf{FFT}(\vec{b})$	$6n$	$\frac{9}{2}n$	
	Hadamard	$4n$	$20n$	
	IFFT	$8n$	$9n$	
Ours	$\mathsf{TwEval}(\vec{a})$	n	$\frac{9}{2}n$	$\frac{31}{2}n + \frac{13}{2}m$
	$\mathsf{TwEval}(\vec{b})$	m	$\frac{9}{2}m$	
	$\mathsf{TwEval}(\vec{c})$	$n + m$	$9n$	

TABLE 2: Prover cost comparison between zkFFT and our method, measured in terms of the number of field multiplications. A detailed cost analysis for each step is provided in Appendix D.

shorter than the input $\vec{a}$. (2) The bookkeeping table in zkFFT is considerably more complex than ours, leading to a higher computational overhead. (3) zkFFT requires proving the Hadamard product for two vectors of length $2n$, which is costly as it involves invoking the sumcheck protocol on a polynomial of degree 3.

4.2. Proof of sum of multiple 1-D convolutions

Proving multi-channel convolution is more complicated compared with one-dimensional convolution, due to introducing multiple input channels and output channels. We can start from a simple case where output channel $D = 1$. In this case, it is essentially proving the sum result of C 1-D convolutions.

We denote the input $a \in \mathbb{F}^{C\times N}$ and kernel $b \in \mathbb{F}^{C\times K}$, the sum of convolution is $Y \in \mathbb{F}^{N+K-1}$. The statement we now want to prove is:

$$\sum_{c\in[C]} a_c * b_c = Y \tag{13}$$

By the linearity of polynomial addition, adapting techniques in Section 4.1, the sum of convolutions could be checked by:

$$\begin{aligned} &\sum_{c\in[C]} (\sum_{i\in[N]} a_{c,i} \cdot \beta^i) \cdot (\sum_{i\in[K]} b_{c,i} \cdot \beta^i) \\ &= \sum_{i\in[N+K-1]} Y_i \cdot \beta^i \end{aligned} \tag{14}$$

We now analyze the soundness of verifying the sum of convolutions using the approach described above. If $\sum_{c\in[C]} a_c * b_c \neq Y$, this indicates that $\exists j \in [N + K - 1]$:

$$\sum_{c\in[C]} \left(\sum_{i=0}^{j} a_{c,i} \cdot b_{c,j-i}\right) \neq Y_j$$

Then, by the Schwartz-Zippel Lemma [39], we have:

$$\sum_{j\in[N+K-1]} \beta^j \cdot \sum_{c\in[C]} \left(\sum_{i=0}^{j} a_{c,i} \cdot b_{c,j-i}\right) \neq \sum_{i\in[N+K-1]} \beta^j Y_i$$

holds with probability no less than $1 - O(\frac{N+K}{|\mathbb{F}|})$ for $\beta \in \mathbb{F}$. For the LHS of the previous inequality relation, it equals to:

$$\begin{aligned} &\sum_{c\in[C]} \sum_{j\in[N+K-1]} \beta^j \left(\sum_{i=0}^{j} a_{c,i} \cdot b_{c,j-i}\right) \\ &= \sum_{c\in[C]} \sum_{j\in[N+K-1]} \left(\sum_{i=0}^{j} a_{c,i} \cdot \beta^i \cdot b_{c,j-i} \cdot \beta^{j-i}\right) \\ &= \sum_{c\in[C]} \left((\sum_{i\in[N]} a_{c,i} \cdot \beta^i) \cdot (\sum_{i\in[K]} b_{c,i} \cdot \beta^i)\right) \end{aligned} \tag{15}$$

Thus, we have shown that if $\sum_{c\in[C]} a_c * b_c \neq Y$, Equation 14 holds with a probability of at most $O(\frac{N+K}{|\mathbb{F}|})$ for a randomly chosen $\beta \in \mathbb{F}$, which is negligible. This confirms the soundness of our approach for verifying the sum of convolutions.

4.3. Efficiently proving multi-channel convolution

Using the previously proposed tools, we now present our time-optimal proof for multi-channel convolution. Recall the definition of multi-channel convolution in Equation 8. To prove this relation, we first describe the conversion from a single 2-D convolution to a 1-D convolution. The design of our conversion method is adapted from the method used in zkCNN [9]. Given an input $\hat{a} \in \mathbb{F}^{n \times n}$ and a kernel $\hat{b} \in \mathbb{F}^{m \times m}$, the 2-D convolution $\hat{c} = \hat{a} * \hat{b}$, we define $a \in \mathbb{F}^{n^2}, b \in \mathbb{F}^{n \cdot m}, c \in \mathbb{F}^{n^2 + n \cdot m - 1}$ such that:

$$
\begin{aligned}
& a_{tn+\ell} = \hat{a}_{t,\ell}, \quad 0 \le t < n, 0 \le \ell < n \\
& b_{(m-1-t)n+n-1-\ell} = \begin{cases} \hat{b}_{t,\ell}, & 0 \le t, \ell < m \\ 0, & \text{otherwise} \end{cases} \\
& c = a * b
\end{aligned}
$$

The result of the 2-D convolution $\hat{c}$ can be derived from the 1-D convolution result c:

$$
\begin{aligned}
\hat{c}_{j,k} &= \sum_{t=0,l=0}^{(m-1),(m-1)} \hat{a}_{j+t,k+\ell} \cdot \hat{b}_{t,\ell} \\
&= \sum_{t=0,\ell=0}^{(m-1),(n-1)} a_{nj+k+nt+\ell} \cdot b_{n \cdot m-1-(t \cdot n+\ell)} \\
&= \sum_{i=0}^{n \cdot m-1} a_{n \cdot m-1+nj+k-i} b_i
\end{aligned} \tag{16}
$$

Therefore, we have $c_{n \cdot m-1+tn+\ell} = \hat{c}_{t,\ell}, \quad 0 \le t \le n-m, 0 \le \ell \le n-m$. The derivation shows that a 2-D convolution of shape $n \times n$ and $m \times m$ can be represented in the form of a 1-D convolution of a vector of length $n \cdot n$ and a kernel with length $n \cdot m$. Our 2-D to 1-D transformation provides an advantage over the method used in zkCNN, as our padded kernel has a size of $n \cdot m$, which is significantly smaller than the padded kernel size of n^2 in zkCNN. This also helps our method reduce constant factors.

There is a discrepancy between 2-D convolution and the transformed 1-D convolution result: The result of a 1-D convolution $c = a * b$ has a length of $n^2 + n \cdot m - 1$, whereas the result of a 2-D convolution $\hat{c} = \hat{a} * \hat{b}$ contains only $(n-m+1)^2$ elements. In fact, all elements in $\hat{c}$ can be viewed as copied from fixed indices of c. This can be handled by using a copy constraint similar to Plonk [40] or zkCNN [9], incurring only a minor overhead.

In our subsequent derivation, we primarily focus on proving the transformed 1-D convolution. We denote the padded length of each weight kernel as $K = n \cdot m$, and the length of each flattened 2-D input size as $N = n^2$. Here, X has a shape of $C \times N$, W has a shape of $C \times D \times K$ and the multi-channel convolution result Y has a shape of $D \times (N+K-1)$. Now we check the following statement:

$$
\sum_{c \in [C]} X_c * W_{c,d} = Y_d, d \in [D] \tag{17}
$$

A naive approach for proving the above equation is to execute our efficient protocol for a single convolution operation $C \cdot D$ times. However, this results in a complexity of $O(CDn^2)$, which incurs a high computational cost. Our key idea for accelerating this procedure is to reduce the number of convolutions that must be proven. We observe that, although there are $C \cdot D$ distinct small weight kernels, there are only C unique input channels. Thus, each input channel is reused in D different convolutions.

Consider the simple case where $C = 1$. In this scenario, the single input channel is processed using D different kernels, producing D different outputs. Instead of verifying each convolution individually, we can combine the kernels randomly and verify the convolution between the input and the combined kernel. Due to the linearity of convolution, the combined convolution result should match the combined original convolution outputs accordingly. This technique reduces the number of proofs required from D convolutions to a single one.

For the general case where $C > 1$, this approach naturally extends. By combining the convolution kernels randomly across different output channels, we effectively reduce the $C \cdot D$ kernels to C combined kernels. Similarly, the D output channels can be combined in a corresponding manner. As a result, the problem of proving $C \cdot D$ convolutions in a multi-channel setting is reduced to proving only the sum of C convolutions.

Specifically, we can check the Equation 17 by random combination with verifier's random challenge $r_D \in \mathbb{F}^{\log D}$.

$$
\sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot \sum_{c \in [C]} X_c * W_{c,d} = \sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot Y_d \tag{18}
$$

By the linearity of convolution, we transform the LHS of Equation 18 and derive the following statement

$$
\begin{aligned}
& \sum_{c \in [C]} \left(X_c * \left(\sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot W_{c,d} \right) \right) \\
&= \sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot Y_d
\end{aligned} \tag{19}
$$

For notation simplicity, we define $W' \in \mathbb{F}^{C \times K}$ such that $W'_c = \sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot W_{c,d}$. We observe that the LHS of Equation 19 also has a form of the sum of C 1-D convolutions, between X_c and W'_c for $c \in [C]$. We apply the techniques in Section 4.2 to check Equation 19 holds. Specifically, we obtain the following statement:

$$
\begin{aligned}
& \sum_{c \in [C]} \left(\sum_{i \in [N]} X_{c,i} \cdot \beta^i \right) \cdot \left(\sum_{j \in [K]} W'_{c,j} \cdot \beta^j \right) \\
&= \sum_{d \in [D]} \tilde{eq}(r_D, \text{bin}(d)) \cdot \left(\sum_{i \in [N+K-1]} Y_{d,i} \cdot \beta^i \right)
\end{aligned} \tag{20}
$$

Denote the multilinear extension of W and Y as $\tilde{W}(\cdot)$ and $\tilde{Y}(\cdot)$ respectively. By the property of multilinear exten-

sions, we have: $W'_{c,j} = \tilde{W}(\text{bin}(c)||r_D||\,\text{bin}(j))$. Now we can rewrite the LHS and RHS of Equation 20 into:

$$\mathsf{LHS} = \sum_{c\in\{0,1\}^{\log C}} (\sum_{i\in[N]} \tilde{X}(c||\,\text{bin}(i)) \cdot \beta^i) \cdot (\sum_{j\in[K]} \tilde{W}(c||r_D||\,\text{bin}(j)) \cdot \beta^j) \tag{21}$$

$$\mathsf{RHS} = \sum_{i\in[N+K-1]} \tilde{Y}(r_D||\,\text{bin}(i)) \cdot \beta^i \tag{22}$$

Therefore, verifying the correctness of multi-channel convolution reduces to proving the validity of the two sums above. The result of Equation 22 can be directly proved using a twisted evaluation protocol. Thus, it remains to provide the proof for LHS.

Define S_{XW} as the LHS. To prove the result of S_{XW}, prover first defines $X' : \{0,1\}^{\log C} \to \mathbb{F}$ as $X'(c) = \sum_{i\in[N]} \tilde{X}(c||\,\text{bin}(i)) \cdot \beta^i, c \in \{0,1\}^{\log C}$. $W'' : \{0,1\}^{\log C} \to \mathbb{F}$, $W''(c) = \sum_{j\in[K]} \tilde{W}(c||r_D||\,\text{bin}(j)) \cdot \beta^j, c \in \{0,1\}^{\log C}$. The prover first runs the sumcheck protocol on the following Equation:

$$S_{XW} = \sum_{c\in\{0,1\}^{\log C}} X'(c)W''(c) \tag{23}$$

The sumcheck protocol reduces the correctness of S_{XW} to oracle access to $\tilde{X}'(r_C)$ and $\tilde{W}''(r_C)$, $r_C \in \mathbb{F}^{\log C}$. Then by the property of MLE, the prover can prove $\tilde{X}'(r_C)$ and $\tilde{W}''(r_C)$ by running the twisted evaluation protocol on:

$$\begin{aligned} \tilde{X}'(r_C) &= \sum_{i\in[N]} \tilde{X}(r_C||\,\text{bin}(i)) \cdot \beta^i \\ \tilde{W}''(r_C) &= \sum_{j\in[K]} \tilde{W}(r_C||r_D||\,\text{bin}(j)) \cdot \beta^j \end{aligned} \tag{24}$$

The remaining challenge is the prover has to efficiently compute the evaluations of $\tilde{X}'(\cdot), \tilde{W}''(\cdot), \tilde{X}(r_C||\cdot)$ and $\tilde{W}(r_C||r_D||\cdot)$ on the corresponding hypercubes.

Apparently, the evaluations of $X'(\cdot)$ and $\tilde{X}(r_C||\cdot)$ could be each computed in $O(CN)$ field multiplications.

For $W''(\cdot)$, computing it by the definition requires $O(C \cdot D \cdot K) = O(CDnm)$ complexity. However, this can be optimized by leveraging the fact that W is padded from CDm^2 elements to $CDnm$ due to the conversion from 2-D convolution to 1-D convolution. Specifically, let Z represent the subset of non-padded elements within $[K]$, $|Z| = m^2$. It follows that for all $c \in \{0,1\}^{\log C}, d \in \{0,1\}^{\log D}$ and $j \in [K] \setminus Z$, $\tilde{W}(c||d||\,\text{bin}(j)) = 0$. Consequently, we have $W''(c) = \sum_{d\in\{0,1\}^{\log D}} \tilde{eq}(r_D, d) \sum_{j\in Z} \tilde{W}(c||d||\,\text{bin}(j)) \cdot \beta^j$. Thus, the final complexity of computing the bookkeeping table of $W''(\cdot)$ is optimized to $O(CDm^2)$.

Similarly, considering the bookkeeping table of $\tilde{W}(r_C||r_D||\,\text{bin}(j))$, for $j \in [K] \setminus Z$, $\tilde{W}(r_C||r_D||\,\text{bin}(j)) = 0$. Thus, the total cost of computing $\tilde{W}(r_C||r_D||\cdot)$ is also only $O(CDm^2)$ field multiplications.

Our complete protocol for proving multi-channel convolution is presented in Protocol 2. The soundness of this protocol can be analyzed in a similar manner to the protocol for verifying the sum of convolutions, as discussed in Section 4.2. In conclusion, the total prover time for multi-channel convolution is $O((C+D)n^2 + CDm^2)$. Given that the input size of the statement is $O(Cn^2 + CDm^2)$ and the output size is $O(Dn^2)$, our prover complexity aligns with the sum of input and output sizes. This clearly demonstrates that our protocol achieves optimal time complexity for proving multi-channel convolution.

The proof size of our multi-channel convolution protocol is $O(\log C + \log N + \log K) = O(\log(Cn^2))$. The verifier complexity of our protocol is $O(\log C + \log^2 N + \log^2 K) = O(\log^2 n + \log C)$, dominated by the twisted evaluation protocol.

4.4. Proof of other layers

Aside from the convolutional layer, we also need to prove the ReLU layer, the max-pooling layer, and the fully connected layer.

ReLU. The ReLU function $\mathsf{ReLU}(x) = \max(x, 0)$ is applied element-wise after each convolutional layer. In quantized computation, we incorporate the *rescaling* on the input of ReLU. This can be expressed as:

$$Y'' \leftarrow \mathsf{ReLU}(\lfloor \frac{Y}{2^Q} \rfloor) \tag{25}$$

where 2^Q represents the scaling factor used in our system. Our proof of ReLU is based on lookup arguments [41], [18], [42], which prove that all elements in a committed query vector $\{f_i\}_{i\in[m]}$ belong to a predefined table $\{t_i\}_{i\in[n]}$.

Let $Y' = \lfloor \frac{Y}{2^Q} \rfloor$. We first prove the consistency between Y' and Y. We can see that this is equivalent to:

$$\begin{aligned} Y &= 2^Q \cdot Y' + R \\ R &\in [0, 2^Q - 1] \end{aligned}$$

We can leverage the lookup protocol to prove $R \subseteq \{0, 1, \cdots, 2^Q - 1\}$. We have to further prove $Y'' = \mathsf{ReLU}(Y')$, $Y' \subseteq \{-2^{B-1}, \cdots, -1, 0, \cdots, 2^{B-1}-1\}$, recall that B satisfies $-2^{B-1} \le Y < 2^{B-1}$. To prove this relation, we construct a lookup table containing all possible input-output pairs. The input column of the lookup table is defined as $T^{(1)} := \{-2^{B-1}, \cdots, -1, 0, \cdots, 2^{B-1} - 1\}$. The output column of the lookup table is defined as $T^{(2)} = \{0, \cdots, 0, 1, \cdots, 2^{B-1} - 1\}$. In conclusion, $\mathcal{P}$ proves the following statements:

$$Y' + \gamma \cdot Y'' \subseteq T^{(1)} + \gamma \cdot T^{(2)} \tag{26}$$

$$Y = Y' \cdot 2^Q + R \tag{27}$$

$$R \subseteq T^{(3)} \tag{28}$$

where γ is a random challenge specified by $\mathcal{V}$, $T^{(3)}$ is an additional lookup table defined as $T^{(3)} = \{0, \cdots, 2^Q - 1\}$. Equation 26 and Equation 28 can be proven by the logUp lookup protocol [18].

Protocol 2 (zk-MultiConv). *Consider the multi-channel 2-D convolution between $C \times n^2$ and $C \times D \times m^2$, C, D, n, m are all padded to powers of 2, denote $N = n^2$, $K = n \cdot m$. In this proposed protocol, input X is reshaped to $X \in \mathbb{F}^{C \times N}$, weight W is padded to $W \in \mathbb{F}^{C \times D \times K}$, and result Y has the form $\mathbb{F}^{D \times (N+K-1)}$, following the transformation described in Section 4.3.*

- *Input: Polynomial commitment to $\tilde{X}$, $\tilde{W}$, $\tilde{Y}$.*
- *$\mathcal{V} \to \mathcal{P}$: Random challenge $\beta \in \mathbb{F}, r_D \in \mathbb{F}^{\log D}$.*
- *$\mathcal{P}$: Compute $S_{XW} = \sum_{c \in \{0,1\}^{\log C}} (\sum_{i \in [N]} \tilde{X}(c||\operatorname{bin}(i)) \cdot \beta^i) \cdot (\sum_{j \in [K]} \tilde{W}(c||r_D||\operatorname{bin}(j)) \cdot \beta^j)$.*
- *$\mathcal{P} \to \mathcal{V}$: Claim on S_{XW}.*
- *$\mathcal{P}$: For each $c \in \{0,1\}^{\log C}$, compute $X'(c)$ and $W''(c)$ defined in Section 4.3.*
- *Prove claim on S_{XW}*

(1): $\mathcal{P} \leftrightarrow \mathcal{V}$ run sumcheck protocol together to prove that $\sum_{c \in \{0,1\}^{\log C}} X'(c)W''(c) = S_{XW}$, reducing to the claim on $\tilde{X}'(r_C)$ and $\tilde{W}''(r_C)$, $r_C \in \mathbb{F}^{\log C}$.

(2): $\mathcal{P} \leftrightarrow \mathcal{V}$ run the twisted evaluation protocol together to prove that $\tilde{X}'(r_C) = \sum_{i \in [N]} \tilde{X}(r_C||\operatorname{bin}(i)) \cdot \beta^i$. The protocol reduces to opening $\tilde{X}$ at $\tilde{X}(r_C||r_i)$.

(3): $\mathcal{P} \leftrightarrow \mathcal{V}$ run the twisted evaluation protocol together to prove that $\tilde{W}''(r_C) = \sum_{j \in [K]} \tilde{W}(r_C||r_D||\operatorname{bin}(j)) \cdot \beta^j$. The protocol reduces to opening $\tilde{W}$ at $\tilde{W}(r_C||r_D||r_j)$.

- *Prove twisted evaluation on Y: $\mathcal{P} \leftrightarrow \mathcal{V}$ run the twisted evaluation protocol together to prove that $S_{XW} = \sum_{i=0}^{N-K} \tilde{Y}(r_D||\operatorname{bin}(i)) \cdot \beta^i$. The protocol reduces to opening $\tilde{Y}$ at $\tilde{Y}(r_D||r'_i)$.*

Figure 3: Our efficient protocol for proving multi-channel convolution.

Max-Pooling. The max-pooling layer is applied after the ReLU layer. First, denote the input of max-pooling layer as vector $Y'' \in \mathbb{F}^{s \cdot k}$ (max-pooling is always performed after ReLU), and denote the output as $Z \in \mathbb{F}^k$. Denote the max-pooling layer takes the max on s elements. The max-pooling layer computes $Z_i = \max_{j \in [K_i]}\{Y''_j\}$, where K_i is the set of indices of Y'' that are aggregated to indice i of Z, $|K_i| = s$. The following constraints must hold:

$$Z_i - Y''_j \geq 0, \forall i \in [k], j \in K_i \tag{29}$$

$$\prod_{j \in K_i} (Z_i - Y''_j) = 0, \forall i \in [k] \tag{30}$$

Considering that max-pooling is always applied after the ReLU layer, we have $Z_i \in \{0, \cdots, 2^{B-1} - 1\}$, consequently, $Z_i - Y''_j \in \{0, \cdots, 2^{B-1} - 1\}$. Therefore, we can check Equation 29 by querying a lookup table containing $\{0, \cdots, 2^{B-1} - 1\}$. Equation 30 can be checked by constructing an arithmetic circuit representing the product, and running a GKR protocol [43] on it.

Fully connected layer. Fully connected layers in CNNs are essentially performing matrix multiplication. We can leverage the well known Thaler13 protocol [28] to prove matrix multiplication efficiently based on the sumcheck protocol.

4.5. Putting everything together

Based on the building blocks presented in the previous sections, we construct an efficient zero-knowledge proof (ZKP) protocol for CNN inference. The public inputs to our verifiable CNN inference protocol consist of the input image and the predicted logits vector, while the witness includes the weight values and intermediate values computed during the forward pass of the CNN. Our protocol follows a "commit-and-prove" structure, a widely adopted approach in proof systems such as Plonk [40]. Notably, the commitment to the weights is performed once during preprocessing and can be reused across all users, as it is independent of the input image. The proof generation begins after the prover computes all intermediate values of the CNN inference in plaintext.

During the commitment phase, the prover sequentially commits to the computed outputs of each layer and transmits the corresponding commitments to the verifier. Intermediate vectors that serve as both the output of one layer and the input to the next are committed only once. This approach improves efficiency by saving prover time and ensuring consistent data flow across layers.

The protocol then enters the proof phase. Here, the prover demonstrates the correctness of each layer's execution using the commitments sent earlier. Specifically:

- For a convolutional layer, the prover verifies its correctness using the protocol described in Section 4.3.
- For a ReLU or max-pooling layer, the prover proves correctness using techniques outlined in Section 4.4.
- For a fully connected layer, the prover verifies correctness using the matrix multiplication proof protocol from [28].

The full protocol is detailed in Protocol 3. The soundness of the protocol is rooted in the soundness of the sub-protocols used to prove the correctness of individual layers. To add zero-knowledge to our system, we need to ensure that both the polynomial commitment scheme and the sumcheck protocol are zero-knowledge. For the polynomial commitment scheme, we adopt Hyrax polynomial commitment scheme (PCS), which is proven to be perfectly zero-knowledge (see [44] Lemma 5). The sumcheck protocol can also be made zero-knowledge following the standard masking technique from Libra [45].

Protocol 3 (CNN-Verf end-to-end). • *Public Input: A convolutional neural network with H layers, an input image $X^{(1)} \in \mathbb{F}^{C_1 \cdot n_1^2}$, output logits vector $Y^{(H)} \in \mathbb{F}^{D_H}$, quantized number bit length B.*
• *Preprocessing: $\mathcal{P}$ commits to the following three tables: $T^{(1)} := \{-2^{B-1}, \cdots, 0, \cdots, 2^{B-1}-1\}$, $T^{(2)} = \{ReLU(-2^{B-1}), \cdots, ReLU(0), \cdots, ReLU(2^{B-1}-1)\}$, $T^{(3)} = \{0, \cdots, 2^Q - 1\}$.*
For $\ell = 1, 2, \cdots, H$:
If layer ℓ is a convolutional layer, $\mathcal{P}$ commits to $\tilde{W}^{(\ell)}$, $W^{(\ell)} \in \mathbb{F}^{C_\ell \cdot D_\ell \cdot K_\ell}$, where C_ℓ, D_ℓ is the input channel number and output channel number; $K_\ell = n_\ell \cdot m_\ell$, n_ℓ is the input size at this layer, m_ℓ is the kernel size. $C_\ell, D_\ell, n_\ell, m_\ell$ are all padded to the nearest power of 2.
If layer ℓ is a fully-connected layer, $\mathcal{P}$ commits to $\tilde{W}^{(\ell)}$, $W^{(\ell)} \in \mathbb{F}^{C_\ell \cdot D_\ell \cdot K_\ell}$, where C_ℓ, D_ℓ is the input dimension number and output dimension number, C_ℓ, D_ℓ are all padded to the power of 2.
• *Commitment phase:*
$\mathcal{P} \to \mathcal{V}$:
For $\ell = 1, 2, \cdots, H$:
(1) If layer ℓ is a convolutional layer,
$\mathcal{P}$ computes and commits to $\tilde{X}^{(\ell)}, \tilde{Y}^{(\ell)}$, where $X^{(\ell)} \in \mathbb{F}^{C_\ell \cdot N_\ell}$, $Y^{(\ell)} \in \mathbb{F}^{D_\ell \cdot N_\ell}$, $N_\ell = n_\ell^2$.
(2) If layer ℓ is a ReLU layer,
$\mathcal{P}$ computes and commits to $\tilde{Y}'^{(\ell)}, \tilde{Y}''^{(\ell)}, \tilde{R}^{(\ell)}$, where $Y'^{(\ell)}, Y''^{(\ell)}, R^{(\ell)} \in \mathbb{F}^{C_\ell \cdot N_\ell}$.
(3) If layer ℓ is a max-pooling layer,
$\mathcal{P}$ computes and commits to $\tilde{Z}^{(\ell)}$, where $Z^{(\ell)} \in \mathbb{F}^{C_\ell \cdot N_\ell}$.
(4) If layer ℓ is a fully-connected layer,
$\mathcal{P}$ computes and commits to $\tilde{X}^{(\ell)}, \tilde{Y}^{(\ell)}$, where $X^{(\ell)} \in \mathbb{F}^{C_\ell}$, $Y^{(\ell)} \in \mathbb{F}^{D_\ell}$.
• *Proving phase:*
$\mathcal{V}$: Check the commitment to $\tilde{X}^{(1)}$ and $\tilde{Y}^{(H)}$ are consistent with public input.
$\mathcal{P} \leftrightarrow \mathcal{V}$:
For $\ell = 1, 2, \cdots, H$:
(1) If layer ℓ is a convolutional layer,
$\mathcal{P}$ executes Protocol 2 with $\mathcal{V}$, using the commitments to $\tilde{X}^{(\ell)}, \tilde{W}^{(\ell)}$, and $\tilde{Y}^{(\ell)}$ as input.
(2) If layer ℓ is a ReLU layer,
① $\mathcal{P}$ run the sumcheck protocol with $\mathcal{V}$ to check that: $Y^{(\ell-1)} = Y^{(\ell)} \cdot 2^Q + R^{(\ell)}$.
② $\mathcal{V} \to \mathcal{P}$: random challenge $\delta \in \mathbb{F}$.
$\mathcal{P}, \mathcal{V}$ run the logUp protocol together to prove that $Y''^{(\ell)} + \delta \cdot Y'^{(\ell)} \subseteq T^{(1)} + \delta \cdot T^{(2)}$.
③ $\mathcal{P}, \mathcal{V}$ run the logUp protocol together to prove that $R^{(\ell)} \subseteq T^{(3)}$.
(3) If layer ℓ is a max-pooling layer,
① $\mathcal{P}, \mathcal{V}$ run the logUp protocol together to prove that $Z_i^{(\ell)} - Y_j''^{(\ell-1)} \geq 0$, $i \in [N_\ell]$, $j \in K_i$, K_i is the set of positions in $Y''^{(\ell-1)}$ that is aggregated into position i in layer ℓ.
② $\mathcal{P}$ proves to $\mathcal{V}$ that $\prod_{j \in K_i} \left(Z_i^{(\ell)} - Y_j''^{(\ell-1)} \right) = 0$, $i \in [N_\ell]$ using the GKR protocol.
(4) If layer ℓ is a fully-connected layer, $\mathcal{P}, \mathcal{V}$ run the dedicated matrix multiplication protocol in [28] together, to prove $Y^{(\ell)} = W^{(\ell)} \cdot X^{(\ell)}$.
If all checks of $\mathcal{V}$ passes, $\mathcal{V}$ outputs 1, else $\mathcal{V}$ outputs 0.

Figure 4: The protocol of end-to-end proving CNN inference.

5. Experiment

In this section, we present the experimental results of our method for proving multi-channel convolution, as well as our VerfCNN system for end-to-end CNN inference. We also compare our system with state-of-the-art schemes.

5.1. Implementations

Software. In our experiments, we utilized the BN254 elliptic curve group, which provides a security level of 100 bits. We implemented our system using C++ and the mcl library [46]. To achieve efficient proof generation, following zkCNN [9] and zkLLM [13], we incorporated the Hyrax PCS [44].

Hardware. All our experiments are conducted on a server equipped with an Intel Xeon 6126 2.60GHz 16-core CPU and 200 GB of memory. We always use a single thread to run the experiments.

Dataset and Models. Following previous work [9], [14], we evaluate our system using the popular CIFAR-10 dataset [20], widely used in deep learning. It is a classification dataset containing 10 classes, such as cats and dogs, with 50,000 training images and 10,000 testing images of size $32 \times 32 \times 3$. We evaluate on VGG-11 and VGG-16 [15].

Baselines. For VGG-11 and VGG-16, we compare against several baselines, including zkCNN [9] (CCS'21), Balbás et

TABLE 3: Comparison of our system's performance in proving single convolutional layers against baseline methods. The experiment is conducted with fixed parameters of $n = 64$ and $m = 4$, matching the settings of Balbás et al. [7].

		Input channel number c, output channel number d									
		(4,32)	(8,32)	(16,32)	(32,32)	(64,32)	(4,128)	(8,128)	(16,128)	(32,128)	(64,128)
Prover time (ms)	zkCNN	42.5	84.9	169.9	339.7	679.5	169.8	339.7	679.4	1358.9	2717.9
	Balbás et al.	30.2	50.4	90.8	171.7	333.4	60.7	81.4	122.8	205.5	370.9
	Ours	6.7	7.5	9.2	12.5	19.1	24.6	25.8	28.0	32.4	41.3
Proof size (KB)	zkCNN	22.6	25.0	27.6	30.3	33.1	27.6	30.3	33.1	36.0	39.1
	Balbás et al.	0.6	0.7	0.8	0.8	0.9	0.6	0.7	0.8	0.8	0.9
	Ours	1.3	1.4	1.5	1.6	1.7	1.3	1.4	1.5	1.6	1.7
Verifier time (ms)	zkCNN	1.083	1.200	1.323	1.452	1.587	1.323	1.452	1.587	1.728	1.875
	Balbás et al.	0.02	0.02	0.02	0.03	0.03	0.02	0.02	0.02	0.03	0.03
	Ours	0.11	0.12	0.12	0.12	0.13	0.11	0.12	0.12	0.12	0.13

TABLE 4: End-to-end comparison of our method with baseline approaches on VGG models.

	VGG-11			VGG-16		
Schemes	Prover Time	Verifier Time	Proof Size	Prover Time	Verifier Time	Proof Size
zkCNN [9] (CCS'21)	51.6 s	243 ms	284 KB	84.6s	346 ms	316 KB
Balbás et al. [7] (CCS'23)	20.1 s	138 ms	150 KB	27.4 s	179 ms	193 KB
ZKML [14] (EuroSys'24)	378.9 s	9.8 ms	11.2 KB	621.2s	10.6 ms	11.8 KB
Ours	8.6 s	145 ms	152 KB	12.6 s	187 ms	196 KB

al. [7] (CCS'23), and ZKML [14] (EuroSys'24).

5.2. Pure convolution evaluation

We first benchmark the efficiency of our system in proving a single multi-channel convolution operation. Following the configurations in Balbás et al. [7], we fix the input image size $n = 64$, kernel size $m = 4$, the input channel C takes value in $\{4, 8, 16, 32, 64\}$ and the output channel D takes value in $\{32, 128\}$.

Table 3 presents the experimental results, demonstrating that the prover time of VerfCNN outperforms zkCNN by a factor of $6.3 \sim 65.8\times$ across various settings. Additionally, VerfCNN outperforms the prover time of Balbás et al. [7] by a factor of $2.5 \sim 17.4\times$ across various settings. These results indicate that VerfCNN consistently surpasses both Balbás et al. [7] and zkCNN [9] across different configurations, highlighting the exceptional efficiency of our approach in proving multi-channel convolution operations.

We also observe that the advantage of our method over Balbás et al. [7] varies for different values of C and D. This variation stems from the complexity difference between our method and [7] shown in Table 1. When C is larger than D, the term $O(Cm^2n^2)$ is more dominating in the prover cost of [7], making the advantage of our approach more significant. In real-world CNNs, C and D are typically of the same order of magnitude across most layers. Under this scenario, our method already demonstrates a noticeable performance improvement over [7].

Balbás et al.[7] only support proving a single convolutional layer; for a fair end-to-end comparison, we re-implement their convolutional layer protocol within our framework and integrate it with our own proof constructions for the remaining layers.

TABLE 5: Comparison of model accuracy before and after quantization, and the performance comparison between plaintext inference (single-threaded CPU) and prover time.

Model	Original Accuracy	Our Accuracy	Plain-text Time	Prover Time
VGG-11	91.2%	91.1%	0.9s	8.6s
VGG-16	93.1%	93.0%	1.4s	12.6s

5.3. End-to-end evaluation

Finally, we evaluate the end-to-end efficiency of our system on CNN inference tasks using VGG models, and compare it against existing baselines. The experimental results are shown in Table 4. From Table 4 we can see that, on VGG-16 network, compared with the most famous work in this field, zkCNN [9], our system achieves a groundbreaking $5.7\times$ acceleration. Compared with Balbás et al. [7] on prover time, our method achieves about $2.2\times$ prover time acceleration. The high efficiency of our system mainly comes from the design of our primitive for proving multi-channel convolution with theoretically optimal complexity.

When comparing verifier cost and proof size, we can see that our proof size outperforms zkCNN [9]. This is mainly because zkCNN leverages bit-decomposition to evaluate ReLU and max-pooling, which results in the commitment of more values and larger proof size. Our verifier time and proof size is comparable with the most prover efficient baseline Balbás et al. [7]. This is not surprising since the proof size of the convolution part of the two works is at the same order of magnitude as analyzed in Table 1, and our implementation for non-linear layers is the same. For verifier time, as analyzed at the end of Section 3, although

our method's verifier time for convolutional layers is asymptotically higher, this part does not become the bottleneck of verifier time. The bottleneck of the verifier time is the $O(\sqrt{N})$ group operations imposed by the Hyrax PCS.

Our system strikes a tradeoff with ZKML [14] between prover time and proof size/verifier time. We argue that this tradeoff is favorable for most ZKP-based ML auditing applications. When proofs are hosted online by the ML service provider, the difference between downloading a 20KB versus a 200KB proof is negligible given modern internet bandwidth. In contrast, our method significantly reduces the prover time of ZKML—often by orders of magnitude—which is crucial for scalability. In blockchain-based settings, where proof size and verifier time directly impact on-chain cost, our Hyrax PCS can be replaced with PST PCS [47], which achieves $O(\log N)$ proof size at the expense of higher prover time.

We also compare our system with running the CNN in plaintext on CPU single thread in Table 5. The results show that, the numerical precision error introduced by quantization is negligible, with about only 0.1% accuracy drop. Deploying our system for verifiable CNN inference incurs an overhead of approximately $10\times$ compared to plaintext inference. Notably, this overhead is remarkably low, as existing general-purpose zkSNARK systems typically impose an overhead ranging from 10^3 to $10^6\times$ [48], [49], [50].

6. Discussion

Twisted Evaluation. Twisted Evaluation enables evaluating a multilinear polynomial in a univariate-style manner, providing functionality similar to univariate PCS. We adopt the multilinear PCS Hyrax rather than univariate PCS such as KZG [51], since Hyrax offers significantly lower opening costs.

Recent works, PolyFrim [52] and DualPCS [53], also explored the connection between univariate and multilinear polynomials. PolyFrim [52] observed that a univariate evaluation of degree $2^\mu - 1$ can be mapped to a multilinear evaluation, but their approach fails for arbitrary degrees d, as the vector $\langle \beta^0, \beta^1, \cdots, \beta^d, 0, \cdots, 0 \rangle$ lacks the tensor-product structure. DualPCS [53] supports evaluating the same witness in both polynomial forms, but incurs substantially higher commitment and opening costs. In contrast, our twisted evaluation protocol is the first to achieve univariate-style evaluation for arbitrary degrees while relying solely on multilinear polynomial commitments.

Vision Transformers. CNNs no longer deliver state-of-the-art accuracy on many tasks compared with vision transformers [54]. However, under limited computational budgets or strict real-time requirements, CNNs remain highly effective and often preferable. Such scenarios are common in practice—for instance, a recent report [55] highlighted that CNN-based applications accounted for a substantial share of global AI market value in 2025. Given the growing prominence of vision transformers, we leave the design of efficient proof systems for them to future work.

Parallelized Prover. Our paper mainly adopted a single-threaded CPU implementation to directly reflect the amount of prover work, making it easier to validate protocol-level improvments. Many recent work [6], [13], [56], [57] adopted parallelization for prover acceleration, which is highly beneficial for real-world deployment. We leave GPU-accelerated CNN proof generation as future work.

References

[1] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[2] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, no. 1, pp. 221–248, 2017.

[3] G. Sreenu and S. Durai, "Intelligent video surveillance: a review through deep learning techniques for crowd analysis," *Journal of Big Data*, vol. 6, no. 1, pp. 1–27, 2019.

[4] Y. Wang, Y. Zhang, Y. Lu, and X. Yu, "A comparative assessment of credit risk model based on machine learning——a case study of bank loan data," *Procedia Computer Science*, vol. 174, pp. 141–149, 2020.

[5] W. Cai, T. Shi, X. Zhao, and D. Song, "Are you getting what you pay for? auditing model substitution in llm apis," *arXiv preprint arXiv:2504.04715*, 2025.

[6] W. Qu, Y. Sun, X. Liu, T. Lu, Y. Guo, K. Chen, and J. Zhang, "zkgpt: An efficient non-interactive zero-knowledge proof framework for llm inference," in *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

[7] D. Balbás, D. Fiore, M. I. González Vasco, D. Robissout, and C. Soriente, "Modular sumcheck proofs with applications to machine learning and image processing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1437–1451.

[8] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero knowledge proofs for decision tree predictions and accuracy," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 2039–2053.

[9] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[10] E. U. law, "Council directive 2000/43/ec of 29 june 2000 implementing the principle of equal treatment between persons irrespective of racial or ethnic origin," 2024. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2000/43/oj/eng

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, 1985, pp. 291–304.

[12] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 501–518.

[13] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," *arXiv preprint arXiv:2404.16109*, 2024.

[14] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 560–574.

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

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

[17] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 677–706.

[18] U. Haböck, "Multivariate lookups based on logarithmic derivatives," *Cryptology ePrint Archive*, 2022.

[19] L. Eagen, "Bulletproofs++." *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 510, 2022.

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

[21] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences," *Cryptology ePrint Archive*, 2021.

[22] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," *Cryptology ePrint Archive*, 2020.

[23] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "Scaling up trustless dnn inference with zero-knowledge proofs," *arXiv preprint arXiv:2210.08674*, 2022.

[24] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1074–1091.

[25] M. Campanelli, A. Faonio, D. Fiore, T. Li, and H. Lipmaa, "Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees," in *IACR International Conference on Public-Key Cryptography*. Springer, 2024, pp. 337–369.

[26] C. Pappas and D. Papadopoulos, "Sparrow: Space-efficient zksnark for data-parallel circuits and applications to zero-knowledge decision trees," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3110–3124.

[27] M. Hao, H. Chen, H. Li, C. Weng, Y. Zhang, H. Yang, and T. Zhang, "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3819–3836.

[28] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology – CRYPTO 2013*, 2013, pp. 71–89.

[29] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 90–112.

[30] Y. Guo, X. Liu, K. Huang, W. Qu, T. Tao, and J. Zhang, "{DeepFold}: Efficient multilinear polynomial commitment from {Reed-Solomon} code and its application to zero-knowledge proofs," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 3497–3516.

[31] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "Hyperplonk: Plonk with linear-time prover and high-degree custom gates," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 499–530.

[32] J. Bootle, A. Chiesa, Y. Hu, and M. Orru, "Gemini: Elastic snarks for diverse environments," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 427–457.

[33] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[35] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, architectures and applications*. Springer, 1990, pp. 227–236.

[36] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.

[37] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[38] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù, "zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks," in *International Conference on Security and Cryptography for Networks*. Springer, 2024, pp. 46–72.

[39] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[40] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[41] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," *Cryptology ePrint Archive*, 2020.

[42] L. Eagen, D. Fiore, and A. Gabizon, "cq: Cached quotients for fast lookups," *Cryptology ePrint Archive*, 2022.

[43] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," *Journal of the ACM (JACM)*, vol. 62, no. 4, pp. 1–64, 2015.

[44] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 926–943.

[45] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "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, 2019, pp. 733–764.

[46] S. Mitsunari, "MCL: a portable and fast pairing-based cryptography library," https://github.com/herumi/mcl, 2018, accessed: 2024-04-10.

[47] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 222–242.

[48] J. Thaler, "Measuring snark performance: Frontends, backends, and the future." 2024. [Online]. Available: https://a16zcrypto.com/posts/article/measuring-snark-performance-frontends-backends-and-the-future/

[49] Glaze, "How small teams can pick cloud zk proof infrastructure." 2024. [Online]. Available: https://hackernoon.com/how-small-teams-can-pick-cloud-zk-proof-infrastructure

[50] Yiping, "The next battleground: Generating a zk proof market." 2024. [Online]. Available: https://www.binance.com/en-IN/square/post/2472772899570/

[51] A. Kate, G. M. Zaverucha, and I. Goldberg, "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, 2010, pp. 177–194.

[52] Z. Zhang, W. Li, Y. Guo, K. Shi, S. S. Chow, X. Liu, and J. Dong, "Fast {RS-IOP} multivariate polynomial commitments and verifiable secret sharing," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3187–3204.

[53] C. Ganesh, V. Nair, and A. Sharma, "Dual polynomial commitment schemes and applications to commit-and-prove snarks," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 884–898.

[54] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[55] Vinova, "Convolutional neural networks in 2025: Architecture applications," 2025. [Online]. Available: https://vinova.sg/convolutional-neural-networks-architecture-applications/#Where_Are_CNNs_Used

[56] C. Li, P. Zhu, Y. Li, C. Hong, W. Qu, and J. Zhang, "Hyperpianist: Pianist with linear-time prover and logarithmic communication cost," *Cryptology ePrint Archive*, 2024.

[57] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang, "Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 39–39.

[58] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with lasso," in *Advances in Cryptology – EUROCRYPT 2024*, 2024, pp. 180–209.

[59] J. Brewer, "Kronecker products and matrix calculus in system theory," *IEEE Transactions on circuits and systems*, vol. 25, no. 9, pp. 772–781, 1978.

[60] K. Schacke, "On the kronecker product," *Master's thesis, University of Waterloo*, 2004.

[61] S. Bagad, Y. Domb, and J. Thaler, "The sum-check protocol over fields of small characteristic," *Cryptology ePrint Archive*, 2024.

[62] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish, "A hybrid architecture for interactive verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 223–237.

Appendix A. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This work provides concretely improved zero-knowledge proofs for convolutional neural network inference. The core technique is to provide an asymptotically optimal and concretely efficient proof of multi-channel convolution. A corresponding implementation shows that this work is 6.7x faster than zkCNN [9], and 2.3x faster than Balbas et al. [7] (two prior state-of-the-arts). Moreover the presented techniques have only a 10x overhead over native inference on a CPU.

A.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) Provides a Valuable Step Forward in an Established Field: This work makes considerable progress in providing efficient proofs for inference. The target improvement—convolutions—is specialized, but for a very important subproblem. The core idea of avoiding FFTs and using polynomial equality testing instead is a natural and novel observation. On the other hand, the "twisted evaluation" is less obvious and central to the overall technique.

Appendix B. Proof of Lemma 2.1

Proof. By the definition of multilinear extension, we have:

$$\tilde{Z}(x) = \langle \tilde{eq}(x, \text{bin}(0)), \cdots, \tilde{eq}(x, \text{bin}(2^\ell - 1)) \rangle \cdot Z \quad (31)$$

By the tensor-product structure [33], [58], [31] of $\tilde{eq}(\cdot)$, we have:

$$\begin{aligned} &\langle \tilde{eq}(x, \text{bin}(0)), \cdots, \tilde{eq}(x, \text{bin}(2^\ell - 1)) \rangle \\ &= \langle 1 - x_1, x_1 \rangle \otimes \langle 1 - x_2, x_2 \rangle \cdots \otimes \langle 1 - x_\ell, x_\ell \rangle \end{aligned} \quad (32)$$

Note that, for $\forall d$, $\forall x, y, z, t \in F^d$, we have:

$$\begin{aligned} &(x \otimes y) \cdot (z \otimes t) \\ &= \sum_{i=1}^{d} ((x \otimes y) \circ (z \otimes t))_i \\ &= \sum_{i=1}^{d} ((x \circ z) \otimes (y \circ t))_i \\ &= (\sum_{i=1}^{d} (x \circ z)_i) \cdot (\sum_{i=1}^{d} (y \circ t)_i) \\ &= (x \cdot z) \cdot (y \cdot t) \end{aligned} \quad (33)$$

where $x \circ z$ denotes the Hadamard product between vector x and z. This can also be derived from the mixed-product property of the Kronecker product [59], [60].

By Equation 33, we can further derive that, $\forall A_1, \cdots, A_\ell, B_1, \cdots, B_\ell \in \mathbb{F}^d$, we have:

$$\begin{aligned} &(A_1 \otimes A_2 \otimes \cdots \otimes A_\ell) \cdot (B_1 \otimes B_2 \otimes \cdots \otimes B_\ell) \\ &= ((A_1 \otimes A_2 \otimes \cdots \otimes A_{\ell-1}) \otimes A_\ell) \cdot ((B_1 \otimes B_2 \otimes \cdots \otimes B_{\ell-1}) \otimes B_\ell) \\ &= ((A_1 \otimes A_2 \otimes \cdots \otimes A_{\ell-1}) \cdot (B_1 \otimes B_2 \otimes \cdots \otimes B_{\ell-1})) \cdot (A_\ell \cdot B_\ell)) \end{aligned} \quad (34)$$

By recursively applying Equation 34 to $A_1, \cdots, A_{\ell-1}, B_1, \cdots, B_{\ell-1}$ and so on, we have:

$$\begin{aligned} &(A_1 \otimes A_2 \otimes \cdots \otimes A_\ell) \cdot (B_1 \otimes B_2 \otimes \cdots \otimes B_\ell) \\ &= (A_1 \cdot B_1) \cdot (A_2 \cdot B_2) \cdot \cdots \cdot (A_\ell \cdot B_\ell) \end{aligned} \quad (35)$$

Leveraging the definition of vector Z and Equation 32, we can rewrite Equation 31 as:

$$\begin{aligned} \tilde{Z}(x) = &(\langle 1 - x_1, x_1 \rangle \otimes \cdots \otimes \langle 1 - x_\ell, x_\ell \rangle) \cdot \\ &(\langle a_1, b_1 \rangle \otimes \cdot \otimes \langle a_\ell, b_\ell \rangle) \end{aligned} \quad (36)$$

Thus, leveraging Equation 35, we have:

$$\tilde{Z}(x) = \prod_{i=1}^{\ell} ((1 - x_i) \cdot a_i + x_i \cdot b_i) \quad (37)$$

This ends our proof.

Appendix C. Security Proof of Twisted evaluation

We present the proof of Theorem 4.1, demonstrating the completeness and soundness of the twisted evaluation PIOP.

Proof. • **Completeness.** The completeness of the PIOP roots in $B = \sum_{k=1}^{T} B'_k$. If $B = \sum_{k=1}^{T} B'_k$ holds, $\tilde{B}(r_b) = \sum_{k=1}^{T} \tilde{B}'_k(r_b)$. We prove $B = \sum_{k=1}^{T} B'_k$ as follows:

For $x \in \{0,1\}^\ell$, we consider the $x-$th element of B'_k.

Let $d = E_k$. By Lemma 2.1, we have:

$$B'_k(x) = \prod_{i=0}^{d-1}(1 - x_i + \beta^{2^i} \cdot x_i) \cdot (1 - x_d) \cdot \prod_{i=d+1}^{\ell-1}((1 - x_i) \cdot (1 - b_i) + x_i \cdot b_i \cdot \beta^{2^i}) \tag{38}$$

From the above equation, we can infer that, if $\exists i \in \{d+1, \cdots, \ell-1\}$, $x_i \neq b_i$, then $B'_k(x) = 0$. Next, we consider x that satisfy $x_d = 0$ and $i \in \{d+1, \cdots, \ell-1\}, x_i = b_i$. There are only 2^{d-1} points that satisfy this constraint, $x_1, \cdots, x_{d-1} \in \{0,1\}^d$. For these points, since $b_i \in \{0,1\}$, we have:

$$\begin{aligned}
&\prod_{i=d+1}^{\ell-1}((1 - x_i) \cdot (1 - b_i) + x_i \cdot b_i \cdot \beta^{2^i}) \\
=& \prod_{i=d+1}^{\ell-1}((1 - b_i)^2 + b_i^2 \cdot \beta^{2^i}) \\
=& \prod_{i=d+1}^{\ell-1}(1 - b_i + b_i \cdot \beta^{2^i}) \\
=& \prod_{i=d+1}^{\ell-1}\beta^{2^i \cdot b_i} = \beta^{\sum_{i=d+1}^{\ell-1} 2^i \cdot b_i}
\end{aligned}$$

Similarly, we have $\prod_{i=0}^{d-1}(1 - x_i + \beta^{2^i} \cdot x_i) = \beta^{\sum_{i=0}^{d-1} 2^i \cdot x_i}$. Thus, for $x_1, \cdots, x_{d-1} \in \{0,1\}^{d-1}$, $x_d = 0$, $x_{d+1} = b_{d+1}, \cdots, x_\ell = b_\ell$, we have:

$$\begin{aligned}
B'_k(x) &= \beta^{\sum_{i=0}^{d-1} 2^i \cdot x_i} \cdot \beta^{\sum_{i=d+1}^{\ell-1} 2^i \cdot b_i} \\
&= \beta^{\sum_{i=0}^{d-1} 2^i \cdot x_i + \sum_{i=d+1}^{\ell-1} 2^i \cdot b_i} \\
int(x) &= \sum_{i=0}^{\ell-1} 2^i \cdot x_i \\
&= \sum_{i=1}^{d-1} 2^i \cdot x_i + \sum_{i=d+1}^{\ell-1} 2^i \cdot b_i
\end{aligned} \tag{39}$$

Therefore, we can derive that $B'_k(x) = \beta^{int(x)}$. Let $L_k = \sum_{i=d+1}^{\ell-1} 2^i \cdot b_i, R_k = \sum_{i=d+1}^{\ell-1} 2^i \cdot b_i + 2^d - 1$. The above equation indicates that $int(x)$ covers each integer in the range of $[L_k, R_k]$. Thus, for $i \notin [L_k, R_k]$, $B'_k(\text{bin}(i)) = 0$, for $i \in [L, R]$, $B'_k(\text{bin}(i)) = \beta^i$. By the definition of set S, $L_k = \sum_{j=k+1}^{T} 2^{S_j}$. $R_k = L_k + 2^{E_k} - 1$. We have $L_{k-1} = L_k + 2^{E_k} = R_k + 1, k = 2, \cdots, T$. We have $L_T = 0$. $R_1 = \sum_{j=1}^{T} 2^{S_j} - 1 = u - 1$. Thus, we have: $\bigcup_{i=1}^{T}[L_i, R_i] = [0, u-1]$. Thus, we have $B = \sum_{k=1}^{T} B'_k$.

- **Soundness** If $S \neq \sum_{i=0}^{u-1} M(\text{bin}(i)) \cdot \beta^i$, then $S \neq \sum_{b \in \{0,1\}^\ell} M(b) \cdot B(b)$. Under this condition, due to the soundness of the sumcheck protocol, with probability at least $1 - \frac{\ell}{|\mathbb{F}|}$, $\mathcal{V}$ will output rejection.

Appendix D.
Cost analysis of zkFFT and our protocol for 1-D convolution

We start from analyzing the prover cost of zkFFT on 1-D convolution. For proving $\mathsf{FFT}(a)$ and $\mathsf{FFT}(b)$, Algorithm 2 in [9] computes the bookkeeping table in $u+1$ rounds. For round $i = 1, \cdots, u$, in round i, 2^i bookkeeping table entries are updated. In the last round, 2^u bookkeeping table entries are updated. Each update $\mathbf{A}_F[j] = \mathbf{A}_F\left[j \bmod 2^i\right] \cdot \left((1 - u_i) + u_i \cdot \omega_{2^{i+1}}^j\right)$ requires 2 field multiplications. The total number of field multiplications for computing the bookkeeping table is: $2 \cdot (\sum_{i=1}^{u} 2^i + 2^u) \approx 6n$.

For the proof of IFFT, Algorithm 2 in [9] computes the bookkeeping table in $u+1$ rounds. For round $i = 1, \cdots, n+1$, in round i, 2^i bookkeeping table entries are updated. The total number of field multiplications for computing the bookkeeping table is: $2 \cdot \sum_{i=1}^{u+1} 2^i \approx 8n$.

For a sumcheck protocol on a hypercube with size n over polynomial $d = 2$, according to the conclusion in [61], the cost of field multiplications is $\frac{9}{2}n$. Similarly, for a sumcheck on a $2n$ hypercube, the cost is $9n$. This gives out the cost of sumcheck for steps (1) (2) (4) of zkFFT on proving 1-D convolution.

For the bookkeeping table of proving the Hadamard product in zkFFT, the evaluation of $\tilde{eq}$ on the $2n$ size hypercube is computed by the memory method in [62]. This procedure requires $4n$ field multiplications. For the sumcheck of proving the Hadamard product, it is invoked over a polynomial with $d = 3$. The cost of field multiplications is $10 \cdot 2n = 20n$, according to the conclusion in [61].

Next, we analyze the prover cost of our approach. The computation pattern for the bookkeeping table in our method is significantly simpler. For a degree limit p, it is basically just computing all values in $\{\beta^0, \beta^1, \cdots, \beta^{p-1}\}$. Thus, the cost of computing the bookkeeping table for S_a, S_b, S_Y in our method is $n, m, n+m$ field multiplications, respectively.

For the sumcheck protocol applied in the three steps of our method, the involved polynomials all have degree $d = 2$. They all work on a hypercube with size n, m and $2n$, respectively. Thus, similar with previous analysis, their cost is $\frac{9}{2}n$, $\frac{9}{2}m$, and $9n$, respectively.