zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy

Tianyi Liu
Texas A&M University and Shanghai Key Laboratory of Privacy-Preserving Computation
tianyi@tamu.edu

Xiang Xie
Shanghai Key Laboratory of Privacy-Preserving Computation
xiexiang@matrixelements.com

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

ABSTRACT

Deep learning techniques with neural networks are developing prominently in recent years and have been deployed in numerous applications. Despite their great success, in many scenarios it is important for the users to validate that the inferences are truly computed by legitimate neural networks with high accuracy, which is referred to as the integrity of machine learning predictions. To address this issue, in this paper, we propose zkCNN, a zero knowledge proof scheme for convolutional neural networks (CNN). The scheme allows the owner of the CNN model to prove to others that the prediction of a data sample is indeed calculated by the model, without leaking any information about the model itself. Our scheme can also be generalized to prove the accuracy of a secret CNN model on a public dataset.

Underlying zkCNN is a new sumcheck protocol for proving fast Fourier transforms and convolutions with a linear prover time, which is even faster than computing the result asymptotically. We also introduce several improvements and generalizations on the interactive proofs for CNN predictions, including verifying the convolutional layer, the activation function of ReLU and the max pooling. Our scheme is highly efficient in practice. It can support the large CNN of VGG16 with 15 million parameters and 16 layers. It only takes 88.3 seconds to generate the proof, which is 1264× faster than existing schemes. The proof size is 341 kilobytes, and the verifier time is only 59.3 milliseconds. Our scheme can further scale to prove the accuracy of the same CNN on 20 images.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

Zero knowledge proofs; Machine learning; Convolutional Neural Networks

ACM Reference Format:
Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3460120.3485379

1 INTRODUCTION

Deep learning with neural networks has seen a great success in many machine learning tasks in recent years, being deployed in various applications and products in practice. In deep learning, convolutional neural networks (CNN) are particularly useful in the domain of computer vision for image classifications and recognition. Despite their excellent performance, there are many security issues when applying CNN models. In many scenarios, we need to guarantee that the results are indeed predictions of a particular CNN model, which is referred to as the *integrity* of machine learning. A naïve solution to address the integrity issue is to publish the CNN models. However, as the models are usually trained on valuable and sensitive datasets and are important intellectual properties of the owners, it is often impossible in practice to share the models.

In this paper, we propose zero knowledge CNN predictions and accuracy schemes to address the issues above. The cryptographic primitive of *zero knowledge proof* (ZKP) allows a *prover* to convince a *verifier* that a computation on the prover's secret input is correctly calculated through a short proof. Zero knowledge proofs guarantee that if the prover sends a wrong result of the computation, it can only pass the verification with a negligible probability, which is the soundness property. At the same time, the proof leaks no information about the prover's secret input, which is the zero knowledge property. Prover's secret input is usually referred to as the witness or auxiliary input. In the scenario of zero knowledge CNN, the witness is the parameters of the CNN model. The owner of the secret model can prove to the users that the predictions are correctly computed using her CNN model, while preserving the privacy of the model.

Applications of zero knowledge machine learning. With the strong guarantees on the privacy and integrity, zero knowledge machine learning has the potential to enable many new applications in practice. First, machine-learning-as-a-service (MLaaS) such as Amazon Forecast and Amazon Fraud Detector [1] offers cloud-based platforms for predictive data analytics through machine learning as a paid service. However, the clients do not have access to the machine learning models, which are intellectual properties of the companies, and have to trust that the predictions are computed as promised. Using zero knowledge machine learning, the service provider can prove that the models are of high quality and accuracy, and the predictions are indeed computed by the same models. It

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3485379

provides the integrity for MLaaS, while preserving the privacy of the models. Second, reproducibility is a challenging problem in machine learning. Some machine learning models or algorithms are claimed to achieve high accuracy, yet it is challenging to validate these claims in many cases. Zero knowledge machine learning offers a partial solution to this issue. By attaching a ZKP for the claimed accuracy, it at least shows that there exists such a machine learning model known by the owner, due to the knowledge soundness of the ZKP. Moreover, in some ZKP schemes, including our schemes proposed in this paper, the time to verify the result is much faster than computing it. This property further reduces the burden of the verifier to validate the accuracy of the machine learning models.

In a different setting, zero knowledge machine learning can also support predictions of public models on private datasets. As pointed out by [46], in this setting, zero knowledge machine learning can be applied to test the accuracy of a model on private benchmarks. The data owner can evaluate the model on her sensitive data to help the trainer improve the quality of the model, while preserving the privacy of the data. Zero knowledge machine learning can also be used to prove that there are adversary examples for a public model where two inputs are "close" to each other but are classified into two different categories. Our scheme can be modified to support this setting with minimal changes.

Our Contributions. While there are general-purpose zero knowledge proof schemes that work for any computations modeled as arithmetic circuits, they introduce a high overhead on the prover and existing schemes do not scale to the size of CNN models and predictions. In this paper, we propose efficient zero knowledge proof schemes for CNN predictions and accuracy that scale to large CNN models in practice. In particular, our contributions include:

- **Sumcheck protocol for FFT and convolutions.** First, we propose a new sumcheck protocol for two-dimensional (2-D) convolutions. The key ingredient of the protocol is an efficient sumcheck for the fast Fourier transform (FFT)[1] where the additional time to generate the proof is $O(N)$ for a vector of size N. This is indeed asymptotically better than the complexity to compute the FFT in $O(N \log N)$. This is another example of common computations in the literature besides the matrix multiplication where the prover time of the sumcheck is sublinear in the time to compute the result [41]. Using the protocol as a building block, we propose the sumcheck protocol for 2-D convolutions and the prover time is $O(n^2)$ for two inputs of $n \times n$ and $w \times w$. The prover time is equal to the size of the input and the output asymptotically and thus is optimal. The proof size is $O(\log n)$. We further propose a protocol to achieve a sublinear verifier time of $O(\log^2 n)$ with a proof size of $O(\log^2 n)$.

  Our new sumcheck protocol for FFT may be of independent interest as a stand-alone primitive. For example, in [52], the authors applied the interactive proofs to delegate the computation of polynomial evaluations from the verifier to the prover. The scheme uses an FFT circuit of size $O(N \log N)$ and depth $O(\log N)$. With our new sumcheck protocol, we are able to reduce the prover time of this delegation from $O(N \log N)$ to $O(N)$. In [42], the authors proposed to verify the computation of a high-performing but untrusted ASIC (application specific integrated circuit) by a relatively slower but trusted ASIC through interactive proofs, and the FFT circuit is an important application used in the benchmark. With our new sumcheck protocol, the area and energy of the untrusted ASIC devoted to generating the proof can be even smaller than those for computing FFT, and the cost of the verifier ASIC is also reduced by a logarithmic factor, making the verifiable ASICs faster than the baseline of computing FFT using the slow but trused ASIC [42, Figure 11].
- **Efficient interactive proofs for CNN predictions.** Second, we propose several improvements and generalizations of the sumcheck protocol and the doubly efficient interactive proofs (usually referred as the *GKR* protocol) for CNN predictions. (1) We introduce generalized addition gates and multiplications gates so that additions with fan-in larger than 2 and inner products can be implemented with a single sumcheck. (2) With the techniques in [51], we extend the generalized gates to take inputs from any layers without any overhead on the prover time. (3) We further improve the sumcheck protocol for the convolutional layer in CNN to reduce the prover time of IFFT by a factor of ch_{in} for convolutions on inputs with ch_{in} channels. (4) We design an efficient gadget of circuit to compute the ReLU activation function and the max pooling together with a single bit-decomposition per number.
- **Implementation and evaluations.** We fully implement our zero knowledge convolutional neural network system, named zkCNN. We test it on large CNNs such as VGG16 with 15 million parameters on the CIFAR-10 dataset [31]. It takes 88.3s to generate a proof of prediction, and 59.3ms to verify the proof. The proof size is 341KB. The prover time is 1264× faster than the existing scheme in [34]. Our system can also scale to prove the accuracy of the same model on 20 images.

1.1 Related Work

Zero knowledge machine learning. The most relevant work to our paper is vCNN [34] and ZEN [23] on zero knowledge neural network predictions. vCNN supports convolutional neural networks and verifies the convolutions through polynomial multiplications. It combines the regular quadratic arithmetic program (QAP) and the polynomial QAP in pairing-based zero knowledge proofs using commit-and-prove. We verify the convolutions directly using the sumcheck protocol and our approach is 34× faster than vCNN (see Section 5.1). In [23], Feng et al. proposed quantization techniques that are friendly to zero knowledge proofs to support real numbers. We use the quantization approach proposed in [28] and we believe the techniques in [23] are compatible with our scheme and integrating them is left as future work. We compare our zkCNN with vCNN and ZEN in Section 5.2 and show that the prover time of zkCNN is orders of magnitude faster than vCNN and ZEN.

Ghodsi et al. proposed SafetyNets [25], a scheme to prove the correctness of neural network inferences using the GKR protocol. The scheme assumes that the model and the data are known to both the prover and the verifier, and it is essentially a verifiable computation scheme where the verifier time is faster than computing the inference locally. The scheme does not guarantee the privacy of the

[1] Sometimes discrete Fourier transform (DFT) or number theoretic transform (NTT) are used for the transform on finite fields. In this paper, we use the general name of FFT and our protocols work on a finite field.

model or the input. The scheme in [56] verifies the correctness of predictions of several machine learning models including shallow neural network, and provides privacy and integrity simultaneously. Keuffer et al. [29] propose a scheme to verify machine learning models by combining the GKR protocol and QAP-based protocols to handle linear layers and non-linear layers, respectively. Other than neural networks, Zhang et al. [50] proposed an efficient zero knowledge proof scheme for decision tree predictions and accuracy. The techniques are dedicated to decision tree models. Wu et al. [47] designed a distributed zero knowledge proof system and proved the integrity of training a linear regression model as a benchmark.

Interactive proofs. In the seminal work of [26], Goldwasser et al. proposed the doubly efficient interactive proof for layered arithmetic circuits. Later, Cormode et al. improved the prover time of the GKR protocol from $O(|C|^3)$ to $O(|C| \log |C|)$ for circuits of size $|C|$ using multilinear extensions in [20]. Several follow-up papers further reduced the prover time for circuits with special structures [41, 43, 55]. Notably Justin Thaler proposed a highly-efficient sumcheck protocol for matrix multiplications between $n \times n$ matrices where the additional prover time is $O(n^2)$, which is faster than computing the result in $O(n^3)$. Our sumcheck protocol for FFT provides the another example where the prover time is even faster than computing the result. Recently Xie et al. [48] proposed a variant of the GKR protocol with $O(|C|)$ prover time for arbitrary layered arithmetic circuits, and Zhang et al. [51] generalized the GKR protocol to general circuits instead of layered circuit without any overhead on the prover time.

In [53], Zhang et al. extended the GKR protocol to an argument system using polynomial commitments. Subsequent works [39, 44, 48, 52, 54] followed the framework and constructed efficient zero knowledge argument schemes based on interactive proofs. We follow the same framework and constructs zkCNN schemes using interactive proofs and polynomial commitments.

Zero knowledge proofs. Other than interactive proofs, there are many recent schemes of zero knowledge succinct argument of knowledge (zk-SNARK) [6, 8–10, 12, 13, 15, 18, 27, 30, 36, 37] based on pairing, discrete-log, MPC-in-the-head and interactive oracle proofs. They provide succinct proof size on the size of the statement, but incur a high overhead on the running time and the memory usage of the prover. Therefore, these systems are not able to scale to the size of CNN predictions (see Section 5). Recent zero knowledge protocols based on vector oblivious linear evaluation [7, 22, 45, 49] are efficient in the execution time and the memory overhead, but the communication is linear in the size of the statement (several GBs in practice), and the schemes cannot be made non-interactive and are not publicly verifiable.

2 PRELIMINARIES

We use $\mathsf{negl}(\cdot) : \mathbb{N} \to \mathbb{N}$ for the negligible function, where for any positive polynomial $f(\cdot)$, $\mathsf{negl}(k) < \frac{1}{f(k)}$ for sufficiently large integer k. Let λ denote the security parameter. "PPT" stands for probabilistic polynomial time. We use $f(), g()$ for polynomials, x, y, z for vectors of variables and g, u, v for vectors of values. x_i denotes the i-th variable in x. For a multivariate polynomial f, its "variable-degree" is the maximum degree of f in any of its variables. We denote $\{0, 1, \dots, m-1\}$ as $[m]$.

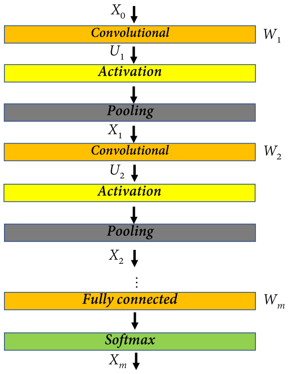


Figure 1: An example of convolutional neural network.

2.1 Convolutional Neural Networks

A convolutional neural network consists of a pipeline of layers, typically including a *convolutional* layer, an *activation* layer and a *pooling* layer. A series of the three previous layers is followed by several *fully connected* layers and an *output* layer. Each layer receives input and processes it to produce an output that serves as input to the next layer. Figure 1 shows a simple example of a convolutional neural network.

Convolutional layer. A convolutional layer computes the dot product between a small weight matrix and the neighborhood of an element in the input data; this process is repeated by sliding the weight matrix step by step. The computation can be viewed as a two-dimensional (2-D) convolution, and each weight matrix is usually referred to as a *kernel* or *filter*. Formally, we define the result of a 2-D convolution between two matrices X and W of size $n \times n$ and $w \times w$ as a $(n - w + 1) \times (n - w + 1)$ matrix $U = X * W$ such that

$$U_{j,k} = \sum_{t=0,l=0}^{w-1,w-1} X_{j+t,k+l} \cdot W_{t,l} \tag{1}$$

for $j, k = 0, \dots, n - w$. [2] In convolutional neural networks, the data samples are represented as matrices and 2-D convolutions are applied in each layer. The input and the output of each layer usually have an additional dimension, referred to as the *channels*. For example, a colored image has 3 channels: red, green and blue. After a convolution, the number of channels in the output is the same as the number of kernels in a convolutional layer. We use $\mathsf{ch}_{in}, \mathsf{ch}_{out}$ to denote the number of input and output channels in a convolutional layer, respectively. Let W_i be the model parameters in layer i, which consists of $\mathsf{ch}_{out,i}$ matrices of size $w_i \times w_i \times \mathsf{ch}_{in,i}$. The input X_i is represented as $\mathsf{ch}_{in,i}$ matrices of size $n_i \times n_i$. We use the notation $W_i[\tau, \sigma, t, l]$ to represent W_i's value at index (τ, σ, t, l), and $X_i[\sigma, j, k]$ to represent X_i's value at index (σ, j, k). Then the convolutional layer i for each data sample computes

$$U_i[\tau, j, k] = \sum_{\sigma=0}^{\mathsf{ch}_{in,i}-1} \sum_{t=0,l=0}^{(w_i-1),(w_i-1)} X_i[\sigma, j, k] \cdot W_i[\tau, \sigma, t, l]. \tag{2}$$

Where $0 \le \tau < \mathsf{ch}_{out,i}$, $0 \le j, k < n_i - w_i + 1$. This is $\mathsf{ch}_{in,i} \cdot \mathsf{ch}_{out,i}$ 2-D convolutions in the form of Equation 1.

[2] Throughout this paper, the size of stride is 1 for simplicity, and thus no padding is needed. Our result could be easily extended to other stride size.

Activation layer. After the convolutional layer, an activation function f is then applied to U_i element-wise to build the nonlinear relationships between input and output. Piecewise linear functions such as ReLU $f(x) = \max(x, 0)$ and Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ are the most commonly used activation functions due to their outstanding performance in the training phase.

Pooling layer. Pooling layers are then used to reduce the dimensions of the feature maps, thus reducing the number of parameters to learn and the amount of computation performed in the CNN. Two common pooling methods are average pooling and max pooling, where AvgPool$(x_0, ..., x_{k-1}) = (x_0+, ..., +x_{k-1})/k$ and MaxPool$(x_0, ..., x_{k-1}) = \max_k(x_0, ..., x_{k-1})$. The pooling layer slides a kernel over the result of the activation layer $f(U_i)$ and do the above two operations within the region covered by the kernel. The result of pooling, denoted as X_{i+1}, is then fed into the next convolutional layer as the input.

Fully connected layer. Finally, at the end of a series of convolutions, activations and poolings, several fully connected layers are applied in CNN. In each fully connected layer, the input is multiplied by a *weight matrix* and added with a *bias vector.*

Output layer. The output layer typically applies a Softmax function to compute a probability distribution for categorical classification problems. In the inference phase, it is enough to calculate the maximal value over the output of the last fully connected layer as the prediction of the most likely outcome. Therefore, in our construction, we omit the computation of Softmax.

Finally, to classify multiple input data samples, an additional dimension is introduced to the input of each layer and the computations are performed independently on each data sample with the same kernels, activations and pooling.

2.2 Interactive Proofs

An interactive proof is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The protocol runs in several rounds, allowing $\mathcal{V}$ to ask questions in each round based on $\mathcal{P}$'s answers in previous rounds. We formalize this in terms of $\mathcal{P}$ trying to convince $\mathcal{V}$ that $C(x) = y$, and give the formal definitions below.

Definition 2.1. Let C be a function. A pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an interactive proof for C with soundness ϵ if the following holds:

- **Completeness.** For every x such that $C(x) = y$ it holds that $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = accept] = 1$.
- **ϵ-Soundness.** For any x with $C(x) \neq y$ and any $\mathcal{P}^*$ it holds that $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle = accept] \leq \epsilon$

We say an interactive proof scheme has succinct proof size and verifier time if they are $O(\text{polylog}(|C|, |x|))$.

2.2.1 Sumcheck Protocol. Sumcheck protocol is one of the most important interactive proofs in the literature. The sumcheck problem is to sum a multivariate polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$ on all binary inputs: $\sum_{b_1, b_2, ..., b_\ell \in \{0,1\}} f(b_1, b_2, ..., b_\ell)$. Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \dots, b_\ell$. Lund et al. [35] proposed a *sumcheck* protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ that H is the correct sum. We provide a description of the sumcheck protocol in Protocol 1.

PROTOCOL 1 (SUMCHECK). *The protocol proceeds in ℓ rounds.*

- *In the first round, $\mathcal{P}$ sends a univariate polynomial*
$$f_1(x_1) \overset{def}{=} \sum_{b_2, \dots, b_\ell \in \{0,1\}} f(x_1, b_2, \dots, b_\ell),$$
$\mathcal{V}$ checks $H = f_1(0) + f_1(1)$. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.
- *In the i-th round, where $2 \leq i \leq \ell - 1$, $\mathcal{P}$ sends a univariate polynomial*
$$f_i(x_i) \overset{def}{=} \sum_{b_{i+1}, \dots, b_\ell \in \{0,1\}} f(r_1, \dots, r_{i-1}, x_i, b_{i+1}, \dots, b_\ell),$$
$\mathcal{V}$ checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$, and sends a random challenge $r_i \in \mathbb{F}$ to $\mathcal{P}$.
- *In the ℓ-th round, $\mathcal{P}$ sends a univariate polynomial*
$$f_\ell(x_\ell) \overset{def}{=} f(r_1, r_2, \dots, r_{l-1}, x_\ell),$$
$\mathcal{V}$ checks $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. The verifier generates a random challenge $r_\ell \in \mathbb{F}$. Given oracle access to an evaluation $f(r_1, r_2, \dots, r_\ell)$ of f, $\mathcal{V}$ will accept if and only if $f_\ell(r_\ell) = f(r_1, r_2, \dots, r_\ell)$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

LEMMA 1. *Protocol 1 is an interactive proof for $H = \sum_{b_1, b_2, ..., b_\ell \in \{0,1\}} f(b_1, b_2, ..., b_\ell)$ that is complete and sound with $\epsilon = \frac{\ell}{|\mathbb{F}|}$ by Definition 2.1.*

The proof size of the sumcheck protocol is $O(\ell)$ if the variable degree of f is constant, which is the case in our protocols. This is because in each round, $\mathcal{P}$ sends a univariate polynomial of one variable in f, which is of constant size. The verifier time of the protocol is $O(\ell)$. The prover time depends on the degree and the sparsity of f, and we will give the complexity later in our scheme.

Definition 2.2 (Identity function). Let $\beta : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be the identity function such that $\beta(x, y) = 1$ if $x = y$, and $\beta(x, y) = 0$ otherwise. Suppose $\tilde{\beta}$ is the multilinear extension of β. Then $\tilde{\beta}$ can be expressed as: $\tilde{\beta}(x, y) = \prod_{i=1}^{\ell}((1 - x_i)(1 - y_i) + x_i y_i)$.

Definition 2.3 (Multilinear Extension [21]). 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 $x_1, x_2, \dots, x_\ell \in \{0,1\}$. $\tilde{V}$ can be expressed as:

$$\tilde{V}(x_1, x_2, ..., x_\ell) = \sum_{b \in \{0,1\}^\ell} \tilde{\beta}(x, b) \cdot V(b)$$
$$= \sum_{b \in \{0,1\}^\ell} \prod_{i=1}^{\ell}((1 - x_i)(1 - b_i) + x_i b_i)) \cdot V(b),$$

where b_i is i-th bit of b.

Multilinear extensions of arrays and matrices. Inspired by the closed-form equation of the multilinear extension given above, we can view an array $\mathrm{a} = (a_0, a_1, \dots, a_{N-1})$ as a function $a : \{0,1\}^{\log N} \to \mathbb{F}$ such that $\forall i \in [0, N-1], a(i_1, \dots, i_{\log N}) = a_i$. Here we assume N is a power of 2. Therefore, in this paper, we abuse the use of multilinear extension on an array as the multilinear extension $\tilde{a}$ of a. Similarly, we use the multilinear extension on an $N \times M$ matrix A as the multilinear extension of the function $A : \{0,1\}^{\log N + \log M} \to \mathbb{F}$ defined by the matrix.

2.2.2 GKR Protocol. Using the sumcheck protocol as a building block, Goldwasser et al. [26] showed an interactive proof protocol for layered arithmetic circuits. Let C be a layered arithmetic circuit with depth d over a finite field $\mathbb{F}$. Each gate in the i-th layer takes inputs from two gates in the $(i+1)$-th layer; layer 0 is the output layer and layer d is the input layer. Following the convention in prior work [20, 41, 48, 52, 53], we denote the number of gates in the i-th layer as S_i and let $s_i = \lceil \log S_i \rceil$. (For simplicity, we assume S_i is a power of 2, and we can pad the layer with dummy gates otherwise.) We then define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes a binary string $b \in \{0,1\}^{s_i}$ and returns the output of gate b in layer i, where b is called the gate label. With this definition, V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. Finally, we define two additional functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, referred to as *wiring predicates* in the literature. add_i ($mult_i$) takes one gate label $z \in \{0,1\}^{s_{i-1}}$ in layer $i-1$ and two gate labels $x, y \in \{0,1\}^{s_i}$ in layer i, and outputs 1 if and only if gate z is an addition (multiplication) gate that takes the output of gate x, y as input. Taking the multilinear extensions of V_i, add_i and $mult_i$, for any $g \in \mathbb{F}^{s_i}$,

$$\begin{aligned}\tilde{V}_i(g) &= \sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} f_i(g,x,y) \\ &= \sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ &\quad + \tilde{mult}_{i+1}(g,x,y)\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)), \end{aligned} \tag{3}$$

With Equation 3, the GKR protocol proceeds as follows. The prover $\mathcal{P}$ first sends the claimed output of the circuit to $\mathcal{V}$. From the claimed output, $\mathcal{V}$ defines polynomial $\tilde{V}_0$ and computes $\tilde{V}_0(g)$ for a random $g \in \mathbb{F}^{s_0}$. $\mathcal{V}$ and $\mathcal{P}$ then invoke a sumcheck protocol on Equation 3 with $i = 0$. As described in Section 2.2.1, at the end of the sumcheck, $\mathcal{V}$ needs an oracle access to $f_i(g,u,v)$, where u, v are randomly selected in $\mathbb{F}^{s_{i+1}}$. To compute $f_i(g,u,v)$, $\mathcal{V}$ computes $\tilde{add}_{i+1}(g,u,v)$ and $\tilde{mult}_{i+1}(g,u,v)$ locally (they only depend on the wiring pattern of the circuit, not on the values), asks $\mathcal{P}$ to send $\tilde{V}_1(u)$ and $\tilde{V}_1(v)$ and computes $f_i(g,u,v)$ to complete the sumcheck protocol. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduce a claim about the output to two claims about values in layer 1. $\mathcal{V}$ and $\mathcal{P}$ then combines the two claims into one through a random linear combination, and run a sumcheck protocol on Equation 3 for layer $i+1$, and then recursively all the way to the input layer. The formal GKR protocol and its properties are presented in Protocol 2 in Appendix A.

2.3 Zero Knowledge Arguments

A zero knowledge argument scheme is a protocol between a PPT prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where at the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that the result of a computation C on a public input x and prover's secret witness w is $y = C(x,w)$. A zero knowledge argument has (1) correctness: $\mathcal{V}$ always accepts if the result and the proof are honestly computed by $\mathcal{P}$; (2) soundness: $\mathcal{V}$ rejects with all but negligible probability if the result is not correctly computed; (3) zero knowledge: the proof leaks no information about the witness w beyond the fact the $C(x,w) = y$. We give the formal definitions of zero knowledge arguments in Definition B.1 of Appendix B.

Following the framework in [44, 48, 52, 53], the GKR protocol can be lifted to a zero knowledge argument scheme using *zero knowledge polynomial commitments*. The observation is that in the last round of the GKR protocol 2, the verifier needs the multilinear extension of the input of the circuit evaluated at two random points. To allow secret witness from the prover, it suffices for the prover to commit to the multilinear extension of the witness, and later opens the polynomial evaluations to complete the reduction of the GKR protocol. We follow the same framework to build our zero knowledge CNN, and we give the formal definitions in Appendix B.

In our implementation, we use the polynomial commitment scheme in [39, 44]. The security is based on the discrete-log assumption and the scheme does not require trusted setup. For a polynomial of size N, the prover time is $O(N)$ modular exponentiation, and the proof size and the verifier time are $O(\sqrt{N})$.

3 NEW SUMCHECK FOR CONVOLUTIONS

Convolution is undoubtedly the most important layer of CNNs and takes the most computational resources in CNN predictions. There are three existing approaches to support convolutions in zero knowledge proof schemes. The first one is to implement convolutions naively using addition and multiplication gates. Though the circuit, and thus the ZKP backend, is very simple, the size of the circuit is big, which is $O(n^2 \cdot w^2)$ for a 2-D convolution between two inputs of $n \times n$ and $w \times w$. The second approach is to compute convolutions using FFT implemented as a circuit. The circuit is of $O(n^2 \log n)$ size and $O(\log n)$ depth (assuming $w < n$), and there are candidates of ZKP on the butterfly circuit of FFT (e.g., [52]). Though asymptotically better, for typical convolutions in CNNs usually $w << n$ and the circuit size of FFT is comparable or even larger than the first naive approach, with an overhead on the depth. The third approach relies on the fact that convolution is equivalent to the polynomial multiplication between the two polynomials represented by the inputs. Instead of computing the convolution, given the result of the convolution we can test the equality of the polynomial multiplication at a random point and the security is guaranteed by the Schwartz-Zipppel lemma [38, 57]. The circuit or ZKP to evaluate polynomials at a random point is of size $O(n^2 + w^2)$. vCNN [34] took this approach and further improves the check by combining a regular QAP and a polynomial-QAP. However, in this approach the prover has to additionally commit to the result of the convolution. As the commitments are usually the bottleneck of ZKP schemes, the overhead of this approach is still high in practice.

In this section, we propose a new protocol to verify the correctness of convolutions. The additional prover time is $O(n^2)$, which is asymptotically optimal and is even faster than computing the convolution. The protocol does not involve additional commitments from the prover and can be embedded in general-purpose ZKP schemes based on the GKR protocols. The key ingredient of our scheme is a new sumcheck protocol for FFT with linear prover time.

3.1 New Sumcheck for Fast Fourier Transform

FFT transforms a polynomial from its coefficients to its evaluations at powers of the root of unity. Formally speaking, let $\mathbf{c} = (c_0, c_1, \ldots, c_{N-1})$ be the vector of coefficients of a polynomial, $\mathbf{a} = (a_0, a_1, \ldots, a_{M-1})$ be the vector of evaluations at $(\omega^0, \omega^1, \ldots, \omega^{M-1})$, where ω is the M-th root of unity such that $\omega^M = 1 \mod p$. We always work in a finite field and we omit mod p in the following. Here the length of $\mathbf{c}$ and $\mathbf{a}$ are padded to the nearest powers of 2. By the definition of polynomial evaluations, $a_j = \sum_{i=0}^{N-1} c_i \omega^{ji}$ for $j = 0, 1, \ldots, M-1$, which can also be written as a matrix-vector multiplication $\mathbf{a} = F \cdot \mathbf{c}$, where F is the standard Fourier matrix:

$$F = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^1 & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{M-1} & \omega^{2(M-1)} & \ldots & \omega^{(M-1)(N-1)} \end{pmatrix} \tag{4}$$

The key property of the FFT algorithm is that as $\omega^M = 1$, there are only M distinct values in the Fourier matrix F and $\mathbf{a}$ can be computed in quasi-linear time using the divide-and-conquer technique [19]. We omit the algorithm of FFT, but will utilize the same property in the design of our sumcheck protocol.

In our setting, given the multilinear extension of $\mathbf{a}$ evaluated at a random point, we want to reduce its correctness to the evaluation of the multilinear extension of c. The evaluations can either be computed directly on $\mathbf{c}$ and $\mathbf{a}$, or be given by the prover during the GKR protocols. To do so, we first turn the equation of polynomial evaluation to the form of multivariate polynomials:

$$\tilde{a}(y) = \sum_{x \in \{0,1\}^{\log N}} \tilde{c}(x)\tilde{F}(y,x), \tag{5}$$

for $y \in \{0,1\}^{\log M}$. Here $\tilde{a}(\cdot)$ and $\tilde{c}(\cdot)$ are multilinear extensions of $\mathbf{a}$ and $\mathbf{c}$, and $\tilde{F}(\cdot,\cdot)$ is the multilinear extension defined by the Fourier matrix F such that $\tilde{F}(y,x)$ is the (y,x)-th entry in F. As x, y are binary strings, we further denote the values represented by y, x as $\mathcal{Y}, \mathcal{X} \in \mathbb{F}$, and thus $\tilde{F}(y,x) = \omega^{\mathcal{Y}\mathcal{X}}$. The equation basically replaces the univariate indices $i \in [N], j \in [M]$ by $x \in \{0,1\}^{\log N}, y \in \{0,1\}^{\log M}$. To run the sumcheck protocol on Equation 5, we rely on the algorithm proposed in [41, 48]. Given the evaluation $\tilde{a}(u)$ of $\tilde{a}(\cdot)$ at a random point $u \in \mathbb{F}^{\log M}$, if the prover can initialize the values of $\tilde{c}(x)$ and $\tilde{F}(u,x)$ on all $x \in \{0,1\}^{\log N}$, there is an algorithm for the prover to generate all messages in the sumcheck protocol in $O(N)$ time. The algorithm applies dynamic programming [41] and the initialization is referred as the bookkeeping tables in [48]. We give the algorithm for our particular sumcheck on Equation 5 in Algorithm 1 for completeness.

Algorithm 1 Sumcheck$(\tilde{c}, \mathbf{A}_c, \tilde{F}, \mathbf{A}_F, r_1, \ldots, r_{\log N})$

Input: Arrays $\mathbf{A}_c$ and $\mathbf{A}_F$ storing $\tilde{c}(x)$ and $\tilde{F}(u,x)$ on all $x \in \{0,1\}^{\log N}$, random $r_1, \ldots, r_{\log N}$;

Output: $\log N$ sumcheck messages for $\sum_{x \in \{0,1\}^{\log M}} \tilde{c}(x)\tilde{F}(u,x)$. Each message consists of 3 elements;

```
for $i = 1, \ldots, \log N$ do
    for $b \in \{0,1\}^{\ell-i}$ do                // B is the number represented by b.
        for $t = 0, 1, 2$ do
            $\tilde{c}(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_c[B] \cdot (1-t) + \mathbf{A}_c[B+2^{\ell-i}] \cdot t$
            $\tilde{F}(r_1, \ldots, r_{i-1}, t, b) = \mathbf{A}_F[B] \cdot (1-t) + \mathbf{A}_F[B+2^{\ell-i}] \cdot t$
    for $t \in \{0,1,2\}$ do                    // Aggregate messages in round i.
        Send $\sum_{b \in \{0,1\}^{\ell-i}} \tilde{c}(r_1, \ldots, r_{i-1}, t, b) \cdot \tilde{F}(r_1, \ldots, r_{i-1}, t, b)$
    for $b \in \{0,1\}^{\ell-i}$ do                // Update the arrays.
        $\mathbf{A}_c[B] = \mathbf{A}_c[B] \cdot (1 - r_i) + \mathbf{A}_c[B + 2^{\ell-i}] \cdot r_i$
        $\mathbf{A}_F[B] = \mathbf{A}_F[B] \cdot (1 - r_i) + \mathbf{A}_F[B + 2^{\ell-i}] \cdot r_i$
```

In the input of Algorithm 1, the array $\mathbf{A}_c$ is simply $\mathbf{c}$ itself by the definition of the multilinear extension. The challenging part is to calculate $\mathbf{A}_F$, i.e., $\tilde{F}(u,x)\ \forall x \in \{0,1\}^{\log N}$. Existing techniques in [48, 51] cannot be applied here, as $\tilde{F}(y,x)$ is not sparse. It is the multilinear extension defined by the Fourier matrix F in Equation 4 with $O(MN)$ nonzero values. Computing $\mathbf{A}_F$ naively would take $O(MN)$ time in total.

In order to reduce the prover time, we write $\tilde{F}(u,x)$ as:

$$\begin{aligned} \tilde{F}(u,x) &= \sum_{z \in \{0,1\}^{\log M}} \tilde{\beta}(u,z)\tilde{F}(z,x) \\ &= \sum_{z \in \{0,1\}^{\log M}} \tilde{\beta}(u,z)\omega^{\mathcal{X}\mathcal{Z}} \\ &= \sum_{z \in \{0,1\}^{\log M}} \tilde{\beta}(u,z)\omega^{\mathcal{X}(z_0 \cdot 2^{\log M-1} + z_1 \cdot 2^{\log M-2} + \cdots + z_{\log M-1})}, \end{aligned} \tag{6}$$

where $\mathcal{Z} = z_0 \cdot 2^{\log M-1} + z_1 \cdot 2^{\log M-2} + \cdots + z_{\log M-1}$ is the number represented by the binary string z with z_0 being the most significant bit. By the closed-form of $\tilde{\beta}$ given in Section 2.2.1, the equation above is equal to

$$\begin{aligned} &\sum_{z \in \{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i) + u_i z_i) \\ &\qquad \cdot \omega^{\mathcal{X} \cdot \sum_{j=0}^{\log M-1} 2^{\log M-1-j} z_j} \\ &= \sum_{z \in \{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i) + u_i z_i) \\ &\qquad \cdot \omega^{\sum_{j=0}^{\log M-1} 2^{\log M-1-j} \cdot (\mathcal{X} \cdot z_j)} \\ &= \sum_{z \in \{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i) + u_i z_i) \\ &\qquad \cdot \prod_{j=0}^{\log M-1} (\omega^{2^{\log M-1-j}})^{\mathcal{X} \cdot z_j}. \end{aligned} \tag{7}$$

Note that $\omega^{2^{\log M-1-j}} = \omega^{\frac{M}{2^{j+1}}}$ above is the 2^{j+1}-th root of unity. We use the same notation as in [19] to denote it as $\omega_{2^{j+1}}$. Then the equation above is

$$\begin{aligned} &= \sum_{z \in \{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i) + u_i z_i) \cdot \prod_{j=0}^{\log M-1} \omega_{2^{j+1}}^{\mathcal{X} \cdot z_j} \\ &= \sum_{z \in \{0,1\}^{\log M}} \prod_{i=0}^{\log M-1} ((1-u_i)(1-z_i) + u_i z_i) \cdot \omega_{2^{i+1}}^{\mathcal{X} \cdot z_i} \\ &= \prod_{i=0}^{\log M-1} \sum_{z_i \in \{0,1\}} ((1-u_i)(1-z_i) + u_i z_i) \cdot \omega_{2^{i+1}}^{\mathcal{X} \cdot z_i} \\ &= \prod_{i=0}^{\log M-1} \left((1-u_i) + u_i \cdot \omega_{2^{i+1}}^{\mathcal{X}}\right). \end{aligned} \tag{8}$$

An easy way to check the correctness of the equation above is that both $\tilde{F}(u,x)$ and Equation 8 are multilinear extensions of matrix F. By the uniqueness of multilinear extensions, they must be equal as long as they agree on all binary inputs. Therefore, by substituting u with any binary string, it is not hard to see that they are the same, because u_i is a selector to choose the right $\omega_{2^{i+1}}^{\mathcal{X}}$ to multiply together in Equation 8. Moreover, if we take a closer look at each parenthesis,

Algorithm 2 $\mathbf{A}_F \leftarrow \mathsf{Initialize}(\omega, u, N)$

```
Input: M-th root of unity $\omega$, random point $u \in \mathbb{F}^{\log M}$ and the degree $N$;
Output: $\mathbf{A}_F$ storing $\tilde{F}(u, x)$ for all $x \in \{0,1\}^{\log N}$.
1: $\mathbf{A}_F[0] = 1$;
2: for $i = 0, \ldots, \log N - 1$ do
3:     for $j = 2^{i+1} - 1, \ldots, 0$ do
4:         $\mathbf{A}_F[j] = \mathbf{A}_F[j \mod 2^i] \cdot \left((1 - u_i) + u_i \cdot \omega_{2^{i+1}}^j\right)$
   // In round i, $(\omega_{2^{i+1}})^{\mathcal{X}}$ has $2^{i+1}$ possible values $\forall \mathcal{X} \in [N]$, indexed by $j = \mathcal{X} \mod 2^{i+1}$.
5: return $\mathbf{A}_F$;
```

$\omega_{2^{i+1}}$ is the 2^{i+1}-th root of unity, and $\omega_{2^{i+1}}^{\mathcal{X}}$ only has 2^{i+1} distinct values for all $\mathcal{X} \in [N]$, which is exactly the property used in the standard FFT algorithm. Therefore, instead of computing $\tilde{F}(u, x)$ for every $x \in \{0,1\}^{\log N}$ one by one, we divide the computation in $\log M$ iterations. In each iteration i, the prover computes a running product for each x with the first i-th parenthesis in Equation 8 from the last iteration. Specifically, the prover precomputes all M distinct values of $\omega_{2^{i+1}}^j$ for $0 \le i < \log N - 1, 0 \le j < 2^{i+1}$ (which are the points to evaluate anyway), calculates all 2^{i+1} different values of $((1 - u_i) + u_i \omega_{2^{i+1}}^j)$ in iteration i and multiplies them to 2^i distinct running products in iteration $i-1$. In the last iteration, the algorithm outputs N values for $\tilde{F}(u, x)\ \forall x \in \{0,1\}^{\log N}$, and the total running time is $O(M + N)$. The algorithm is presented in Algorithm 2.

Together with Algorithm 1, we are able to construct an algorithm for the prover to generate all proofs in the sumcheck protocol on Equation 5 in time $O(M + N)$. The proof size is $O(\log N)$ and the verifier time is $O(\log N)$, given oracle accesses of $\tilde{c}(\cdot)$ and $\tilde{F}(\cdot)$.

Reducing the verifier time. Though our new protocol has optimal prover time and good proof size, it introduces an overhead on the verifier time. In particular, the oracle accesses of $\tilde{c}(\cdot)$ and $\tilde{a}(\cdot)$ are usually provided by the prover or computed on verifier's input as in existing approaches mentioned above, but our protocol requires an additional evaluation of $\tilde{F}(\cdot)$ at a random point. It takes linear time if the verifier evaluates it on her own using a similar algorithm as the prover in Algorithm 2. We further show an approach to delegate this computation through a sequence of sumcheck protocols.

Our approach follows exactly the same algorithm to compute $\mathbf{A}_F$, the bookkeeping table, in Algorithm 2. Recall that $\mathbf{A}_F$ stores $\tilde{F}(u, x)\ \forall x \in \{0,1\}^{\log N}$, thus $\tilde{F}(u, v)$ is the multilinear extension of $\mathbf{A}_F$ evaluated at v. Moreover, in Algorithm 2, the values in $\mathbf{A}_F$ in the i-th round are computed from the values in the $(i-1)$-th round by the equation in Step 4. Therefore, we abuse the notation and use $A_F^{(i)}(\cdot) : \{0,1\}^{i+1} \to \mathbb{F}$ to denote the array $\mathbf{A}_F$ in the i-th round for $i = 0, \ldots, \log N - 1$, and $\tilde{A}_F^{(i)}(\cdot) : \mathbb{F}^{i+1} \to \mathbb{F}$ to denote its multilinear extension. Then $\tilde{F}(u, v) = \tilde{A}_F^{(\log N - 1)}(v)$, and we can write $A_F^{(i)}(\cdot)$ as an equation of $A_F^{(i-1)}(\cdot)$:

$$A_F^{(i)}(x, b) = A_F^{(i-1)}(x)((1 - u_i) + u_i \cdot \omega_{i+1}(x, b)), \quad (9)$$

for all $x \in \{0,1\}^i, b \in \{0,1\}$, where $\omega_{i+1}(x, b) = \omega_{2^{i+1}}^j$ for $j = \sum_{k=0}^{i} x_k 2^{k+1} + b$, the number in $\mathbb{F}$ represented by (x, b) in binary. Equation 9 is exactly the same as Step 4 in Algorithm 2 with binary indices. Then by Definition 2.3,

$$\tilde{A}_F^{(i)}(x, b) = \sum_{z \in \{0,1\}^i} \tilde{\beta}(x, z) \tilde{A}_F^{(i-1)}(z)((1 - u_i) + u_i \cdot \tilde{\omega}_{i+1}(z, b)), \quad (10)$$

for all $x \in \mathbb{F}^i, b \in \mathbb{F}$, as both sides agree on the Boolean hypercube by Equation 9, and are both multilinear in x and b.

Starting from $\tilde{F}(u, v) = \tilde{A}_F^{(\log N - 1)}(v)$, the verifier and the prover can reduce its correctness to the evaluation of $\tilde{A}_F^{(i)}(\cdot)$ at a random point through a sumcheck protocol on Equation 10 for $i = \log N - 1, \ldots, 0$. In the last round, as defined in Step 1 of Algorithm 2, $\tilde{A}_F^{(0)}(\cdot)$ is simply the constant 1. As the size of $\tilde{\beta}(\cdot), \tilde{A}_F^{(i)}(\cdot)$ and $\tilde{\omega}_{i+1}\cdot$ in the i-th sumcheck are $O(2^i)$, the prover time is $O(2^i)$ using the dynamic programming technique in Algorithm 1. The proof size in the i-th sumcheck is $O(i)$. It remains to show that the verifier time is also logarithmic. The verifier time during the i-th sumcheck is $O(i)$. However, at the end of each sumcheck, the verifier has to evaluate $\tilde{\beta}(\cdot)$ and $((1 - u_i) + u_i \tilde{\omega}_{i+1}(\cdot))$ at a random point to obtain $\tilde{A}_F^{(i-1)}(\cdot)$ at the random point for the next sumcheck. By the closed-form of $\tilde{\beta}$ in Definition 2.2, it can be evaluated at a random point in time $O(i)$. By the closed-form of multilinear extension in Definition 2.3, $\tilde{\omega}_{i+1}(r) = \sum_{x \in \{0,1\}^{i+1}} \beta(r, x) \omega_{2^{i+1}}^j$ for $j = \sum_{k=0}^{i+1} x_k 2^k$, which equals to $\prod_{k=0}^{i+1}((1 - r_k) + r_k \omega_{2^{i+1}}^{2^k})$ and can also be evaluated in time $O(i)$. Therefore, with this approach, the total prover time remains $O(N)$ and the total verifier time is reduced to $O(\log^2 N)$, while the total proof size increases to $O(\log^2 N)$.

3.2 Two-Dimensional Convolutions

With our new sumcheck protocol for FFT as a building block, we construct a protocol to validate 2-D convolutions.

Inverse FFT. Inverse FFT (IFFT) can be viewed as FFT with a different root of unity [19],

$$a_j = \sum_{i=0}^{N-1} c_i \omega^{ji} \Leftrightarrow c_i = \frac{1}{M} \sum_{j=0}^{M-1} a_j \omega^{-ji},$$

for $i \in [N], j \in [M]$. As M is known and its inverse exists in a finite field, we can just apply the same sumcheck protocol with linear prover time to validate the result of IFFT.

2-D convolution to 1-D convolution. As introduced in Section 2.1, a convolutional layer in CNN computes the 2-D convolution between the input and the kernel. Here we show that the computation can be reduced to a 1-D convolution. Following Equation 1 in Section 2.1, let $\bar{X}, \bar{W} \in \mathbb{F}^{n^2}$ be

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

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

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

Equation 1 is

$$\begin{aligned} U_{j,k} &= \sum\nolimits_{t=0,l=0}^{(w-1),(w-1)} X_{j+t,k+l} W_{t,l} \\ &= \sum\nolimits_{t=0,l=0}^{(w-1),(w-1)} \bar{X}_{(n-1-j-t)\cdot n+(n-1-k-l)} \bar{W}_{t\cdot n+l} \\ &= \sum\nolimits_{t=0,l=0}^{(n-1),(n-1)} \bar{X}_{(n-1-j-t)\cdot n+(n-1-k-l)} \bar{W}_{t\cdot n+l} \\ &= \sum\nolimits_{i=0}^{n^2-1-j\cdot n-k} \bar{X}_{n^2-1-j\cdot n-k-i} \bar{W}_i \\ &= \bar{U}_{n^2-1-j\cdot n-k} \end{aligned} \tag{11}$$

Thus U can be computed through 1-D convolution between $\bar{X}, \bar{W}$, vectors defined by the input and the kernel of a convolutional layer.

Computing 1-D convolution using FFT. It is well-known that 1-D convolution is the same as multiplications between two univariate polynomials. We abuse the notation to denote the univariate polynomials with coefficients $\bar{X}, \bar{W}$ as $\bar{X}(\eta), \bar{W}(\eta)$, then $\bar{U}(\eta) = \bar{X}(\eta)\bar{W}(\eta) \Leftrightarrow \bar{U}_j = \sum_{i=0}^{j} \bar{X}_{j-i}\bar{W}_i$ by taking $\bar{U}$ as the first n^2 coefficients of $\bar{U}(\eta)$.

Finally, polynomial multiplications can be calculated using FFT and IFFT in three steps. First, we transform $\bar{X}(\eta)$ and $\bar{W}(\eta)$ from coefficients to evaluations at powers of the root of unity, denoted by $\texttt{FFT}(\bar{X})$ and $\texttt{FFT}(\bar{W})$. Here we implicitly assume that W is padded to n^2 and both are evaluated at $2n^2$ points. Then we compute the Hadamard product (element-wise product) of the vectors, and finally transform the result back to the coefficients through IFFT. The algorithm is given as:

$$\bar{U} = \bar{X} * \bar{W} = \texttt{IFFT}(\texttt{FFT}(\bar{X}) \odot \texttt{FFT}(\bar{W})) \tag{12}$$

where "$\odot$" denotes Hadamard product. With the equation above, we are able to verify the computation of 2-D convolutions using three sumcheck protocols, one for FFT, one for Hadamard product and one for IFFT. The real protocol also deals with the indexing and padding, but they do not introduce any major overhead.

Complexity. The prover time of our protocol is $O(n^2)$, which is asymptotically optimal and is faster than computing the convolution. The proof size is $O(\log^2 n)$ and the verifier time is $O(\log^2 n)$, given oracle accesses to the multilinear extensions of the input and the output.

4 ZERO KNOWLEGE CONVOLUTIONAL NEURAL NETWORKS

We present our zero knowledge CNN scheme in this section. We start with the formal definitions of zkCNN, and then introduce several improvements on the sumcheck and the GKR protocol tailored for CNN predictions, and describe our design for activation functions and pooling that lead to concrete efficiency in practice.

4.1 Definitions

In our setting, the prover owns a pre-trained CNN model that is sensitive, and proves to the public that an input data sample is correctly classified using the CNN model. The prover commits to the parameters of the CNN first, and then later the verifier queries for the prediction of the data sample. The prover generates a proof together with the prediction to convince the verifier of its validity. Similar to existing schemes [23, 34], we assume that the structure of the CNN (e.g., number of layers, dimensions of kernels and activation fuctions) is known to the verifier. Admittedly the structure of CNN also leaks information in some scenarios. The structure can further be hidden by introducing upper bounds on the depth and dimensions and selectors from a set of activation functions, or through proof compositions of zero knowledge proofs. Our scheme in this paper only protects the privacy of the parameters while ensuring the integrity of predictions, which is the first step for zero knowledge CNN and the extensions are left as future work.

Formally speaking, let $\mathbf{W}$ be the parameters of a CNN model where the dimensions are given in Section 2.1, and $\mathbf{X} \in \mathbb{F}^{n_1 \times n_1 \times \mathsf{ch}_{in,1}}$ be a data sample. Let $y = \mathsf{pred}(\mathbf{W}, \mathbf{X})$ be the prediction of $\mathbf{X}$ using the CNN as described in Section 2.1. A zero knowledge CNN scheme (zkCNN) consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkCNN.KeyGen}(1^\lambda)$: Given the security parameter, the algorithm generates the public parameters pp.
- $\mathsf{com}_\mathbf{W} \leftarrow \mathsf{zkCNN.Commit}(\mathbf{W}, \mathsf{pp}, r)$: The algorithm commits the parameters $\mathbf{W}$ of the CNN model using the randomness r.
- $(y, \pi) \leftarrow \mathsf{zkCNN.Prove}(\mathbf{W}, \mathbf{X}, \mathsf{pp}, r)$: Given a data sample $\mathbf{X}$, the algorithm runs CNN prediction algorithm to get $y = \mathsf{pred}(\mathbf{W}, \mathbf{X})$ and generates the proof π.
- $\{0, 1\} \leftarrow \mathsf{zkCNN.Verify}(\mathsf{com}_\mathbf{W}, \mathbf{X}, y, \pi, \mathsf{pp})$: The algorithm verifies the prediction y with the commitment $\mathsf{com}_\mathbf{W}$, the proof π and the input $\mathbf{X}$.

A zkCNN scheme is sound, where the probability that the prover returns a wrong prediction and passes the verification is negligible; it is also zero knowledge, where the proof leaks no information about the prover's model $\mathbf{W}$. We give the formal definitions in Appendix C. A Zero knowledge CNN accuracy scheme simply repeats the zkCNN predictions on multiple data samples and compares the predictions with the labels to calculate the accuracy. The definitions can be modified slightly to accommodate zkCNN accuracy and we omit the formal definitions. Moreover, our constructions can also support zero knowledge predictions for secret input data with public CNN models, and both secret input and secret models in a straight forward way, which may be useful in other applications. This is because our scheme is a commit-and-prove SNARK [16]. This is in contrast to zero knowledge proofs based on MPC techniques [7, 22, 45, 49], where there are different trade-offs on the public and private data and models.

4.2 Generalizations of GKR for CNN

In this section, we introduce several improvements and generalizations for the sumcheck and the GKR protocol, which lead to better performance for CNN predictions.

4.2.1 Generalized addition and multiplication gates. As described in the preliminaries, the GKR protocol reduces layer i to layer $i + 1$ through Equation 3 in Section 2.2.2. Because of the definition of $add_i(z, x, y)$, each addition gate can only take two inputs and it takes $\log n$ layers to sum n values in the circuit. Justin Thaler partially addressed this issue in [41] by observing that the circuit of an addition tree can be represented as a single sumcheck. Here we consider the more general case of addition gates with multiple inputs, as well as the sum of multiple products. We define

$$X\tilde{add}_i(z,x) = \begin{cases} 1, & \text{if } V_{i+1}(x) \text{ is added to } V_i(z) \\ 0, & \text{otherwise} \end{cases}$$

$$X\tilde{mult}_i(z,x,y) = \begin{cases} 1, & \text{if } V_{i+1}(x) \cdot V_{i+1}(y) \text{ is added to } V_i(z) \\ 0, & \text{otherwise} \end{cases}$$

for all $x, y \in \{0,1\}^{s_{i+1}}$ and $z \in \{0,1\}^{s_i}$. With the new definitions, we can write the multilinear extensions of layer i as:

$$\begin{aligned} \tilde{V}_i(z) &= \sum\nolimits_{x\in\{0,1\}^{s_{i+1}}} X\tilde{add}_i(z,x) \cdot \tilde{V}_{i+1}(x) \\ &+ \sum\nolimits_{x,y\in\{0,1\}^{s_{i+1}}} X\tilde{mult}_i(z,x,y) \cdot \tilde{V}_{i+1}(x) \cdot \tilde{V}_{i+1}(y) \\ &= \sum\nolimits_{x,y\in\{0,1\}^{s_{i+1}}} \Big(\tilde{\beta}(y,\vec{0}) \cdot X\tilde{add}_i(z,x) \cdot \tilde{V}_{i+1}(x) \\ &+ X\tilde{mult}(z,x,y) \cdot \tilde{V}_{i+1}(x) \cdot \tilde{V}_{i+1}(y)\Big) \end{aligned} \tag{13}$$

With the equation above, we can compute common functions such as additions with fan-in ≥ 2 and inner products with a single sumcheck[3]. Note that for inner products this is better than using the sumcheck for addition trees in [41], which takes 2 layers of the circuit. The prover time remains linear by generalizing the algorithms for the prover in [48], and we omit the formal algorithms. In practice, this generalization reduces the proof size by a logarithmic factor for proving CNN predictions, and improves the concrete efficiency of the prover. Furthermore, we can also supports scalar multiplications with constants for free by replacing the 1 in $X\tilde{add}$ and $X\tilde{mult}$ with the scalars.

4.2.2 Taking inputs from arbitrary layers. Recently Zhang *et al.* [51] proposed a variant of the GKR protocol where a gate can take input from arbitrary layers above, instead of only the previous layer, without introducing any overhead on the prover time. We show that our generalization above is compatible the techniques in [51]. The motivation is that CNN consists of multiple convolutional layers and fully-connected layers. The kernels and the weight-matrices of these layers are the witness from the prover in our zkCNN. When provided at the input layer, they have to be relayed all the way to the corresponding convolutional or fully-connected layers to perform the real computation, which introduces a considerable overhead on the size of the circuit and thus the prover time. Instead, we design an efficient circuit where each convolutional or fully-connected layer connects directly to the witness. See Figure 2 for the structure of our circuit. In this circuit, a generalized addition gate or multiplication gate takes input from either the layer above or from the input layer. To support such a structure, we further extend our protocol above by applying the same techniques in [51].

Following the ideas in [51], we denote the subset of values in the input layer connecting to the i-th layer as $V_{i,\text{in}}$ of size $S_{i,\text{in}}$ and $s_{i,\text{in}} = \lceil \log S_{i,\text{in}} \rceil$, and its multilinear extension as $\tilde{V}_{i,\text{in}}(\cdot)$. We also separately define the generalized addition gates between the i-th and the $(i+1)$-th, the i-th and the input layer as $X\tilde{add}_{i,i+1}(z,x)$, $X\tilde{add}_{i,\text{in}}(z,x)$. Similarly, we define the generalized multiplication gates respectively as $X\tilde{mult}_{i,i+1,i+1}(z,x,y)$, $X\tilde{mult}_{i,\text{in},\text{in}}(z,x,y)$ and $X\tilde{mult}_{i,i+1,\text{in}}(z,x,y)$ for inputs both from layer $i+1$, both from input layer and one from layer $i+1$ one from input. With these definitions, it suffices to write the multilinear extension for layer i in Figure 2 as:

$$\begin{aligned} \tilde{V}_i(z) &= \sum\nolimits_{x\in\{0,1\}^{s_{i+1}}} X\tilde{add}_{i,i+1}(z,x) \cdot \tilde{V}_{i+1}(x) \\ &+ \sum\nolimits_{x\in\{0,1\}^{s_{i,\text{in}}}} X\tilde{add}_{i,\text{in}}(z,x) \cdot \tilde{V}_{i,\text{in}}(x) \\ &+ \sum\nolimits_{x,y\in\{0,1\}^{s_{i+1}}} X\tilde{mult}_{i,i+1,i+1}(z,x,y) \cdot \tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y) \\ &+ \sum\nolimits_{x,y\in\{0,1\}^{s_{i,\text{in}}}} X\tilde{mult}_{i,\text{in},\text{in}}(z,x,y) \cdot \tilde{V}_{i,\text{in}}(x)\tilde{V}_{i,\text{in}}(y) \\ &+ \sum\nolimits_{\substack{x\in\{0,1\}^{s_{i+1}}, \\ y\in\{0,1\}^{s_{i,\text{in}}}}} X\tilde{mult}_{i,i+1,\text{in}}(z,x,y) \cdot \tilde{V}_{i+1}(x)\tilde{V}_{i,\text{in}}(y). \end{aligned}$$

By executing the sumcheck protocol on the equation above, the verifier and the prover can directly reduce $\tilde{V}_i(z)$ to two evaluations of $\tilde{V}_{i+1}(\cdot)$ and two evaluations of $\tilde{V}_{i,\text{in}}(\cdot)$. The prover time is $O(S_i + S_{i+1} + S_{i,\text{in}})$ as there are a constant number of sums in the equation.

Reducing to a single evaluation of the input. After the sumcheck of layer i, the verifier and the prover can proceed to layer $i+1$ in the same way as the GKR protocol 2. However, when reaching to the input layer, the verifier has received two evaluations about the input per layer. Moreover, they are evaluations of $\tilde{V}_{i,\text{in}}(\cdot)$, the subset of V_{in} connected to layer i. In order to combine them to a single evaluation of the multilinear extension of the input $\tilde{V}_{\text{in}}(\cdot)$, we take the approach in [51].

Suppose the evaluations received from layer i are $\tilde{V}_{i,\text{in}}(z_{i,0})$ and $\tilde{V}_{i,\text{in}}(z_{i,1})$, the verifier generates $r_{i,0}, r_{i,1} \in \mathbb{F}$ for layer i and combines all the evaluations through a random linear combination:

$$\begin{aligned} &\sum_i \Big(r_{i,0}\tilde{V}_{i,\text{in}}(z_{i,0}) + r_{i,1}\tilde{V}_{i,\text{in}}(z_{i,1})\Big) \\ &= \sum_i \left(r_{i,0} \sum_{z\in\{0,1\}^{s_{\text{in}}}} C_i(z_{i,0},z)\tilde{V}_{\text{in}}(z) + r_{i,1} \sum_{z\in\{0,1\}^{s_{\text{in}}}} C_i(z_{i,1},z)\tilde{V}_{\text{in}}(z) \right) \\ &= \sum\nolimits_{z\in\{0,1\}^{s_{\text{in}}}} \tilde{V}_{\text{in}}(z) \left(\sum_i \big(r_{i,0}C_i(z_{i,0},z) + r_{i,1}C_i(z_{i,1},z)\big) \right) \end{aligned} \tag{14}$$

where $C_i(z_i, z)$ is defined as:

$$C_i(z_i,z) = \begin{cases} 1, & \text{if the } z_i\text{-th value in } V_{i,\text{in}} \text{ is the } z\text{-th value in } V_{\text{in}} \\ 0, & \text{otherwise} \end{cases}$$

By running the sumcheck protocol on the equation above, the verifier reduces multiple evaluations on $\tilde{V}_{i,\text{in}}(\cdot)$ to a single evaluation of $\tilde{V}_{\text{in}}(\cdot)$. The prover time is linear in S_{in} and the size of the circuit.

4.2.3 Convolutional layer. In Section 3.2, we proposed an efficient protocol to verify the result of the 2-D convolution between one input and one kernel. However, in practice, there are multiple channels and kernels in each convolutional layer of a CNN, as described in Section 2.1. It turns out that we can do better than naively repeating our protocol for a single convolution multiple times. We present our improved protocol in this section.

Formally speaking, we represent the computation of an entire convolutional layer given by Equation 2 by FFT, IFFT and Hadamard

[3] The technique also works for matrix multiplications. However, Justin Thaler [41] proposed a better sumcheck for matrix multiplication with a quadratic prover time in the dimension, and we take his approach in our implementation.

product. Recall that the input data to a convolutional layer is $X \in \mathbb{F}^{\mathsf{ch}_{in} \times n \times n}$ and the kernel is $W \in \mathbb{F}^{\mathsf{ch}_{out} \times \mathsf{ch}_{in} \times w \times w}$. Here with omit the subscript of layer i for the ease of notations. The convolutional layer computes $U \in \mathbb{F}^{\mathsf{ch}_{out} \times (n-w+1) \times (n-w+1)}$ where for each $0 \leq \tau < \mathsf{ch}_{out}$, $0 \leq j, k < n - w + 1$,

$$U[\tau, j, k] = \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \sum_{t=0,l=0}^{(w-1),(w-1)} X[\sigma, j, k] \cdot W[\tau, \sigma, t, l]$$
$$= \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \sum_{i=0}^{n^2-1-jn-k} \bar{X}_\sigma[n^2 - 1 - jn - k - i] \cdot \bar{W}_{\tau,\sigma}[i].$$

This is a generalization of Equation 11, where $\bar{X}_\sigma$ is the vector defined by the σ-th channel of data X, and $\bar{W}_{\tau,\sigma}$ is the vector defined by the (τ, σ)-th kernel. If we apply the algorithm in Equation 12 naively, there are $\mathsf{ch}_{in} \cdot \mathsf{ch}_{out}$ FFTs and IFFTs and the prover time is $O(\mathsf{ch}_{in} \cdot \mathsf{ch}_{out} \cdot n^2)$. Instead, we utilize the linearity of the FFT algorithm. Let $\bar{U}_\tau$ be the vector defined by the τ-th channel of the output U, as we show in Section 3.2, we have

$$\begin{aligned} \bar{U}_\tau &= \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \bar{X}_\sigma * \bar{W}_{\tau,\sigma} \\ &= \sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \mathtt{IFFT}(\mathtt{FFT}(\bar{X}_\sigma) \odot \mathtt{FFT}(\bar{W}_{\tau,\sigma})) \\ &= \mathtt{IFFT}\left(\sum_{\sigma=0}^{\mathsf{ch}_{in}-1} \mathtt{FFT}(\bar{X}_\sigma) \odot \mathtt{FFT}(\bar{W}_{\tau,\sigma})\right). \end{aligned} \tag{15}$$

Note that the total number of IFFTs in Equation 15 is only ch_{out} for $\tau \in [\mathsf{ch}_{out}]$. By running our sumcheck protocols in Section 3, the prover time of the IFFT is reduced to $O(\mathsf{ch}_{out} \cdot n^2)$. Though the total complexity remains the same, the efficiency in practice is improved. Moreover, by applying the GKR protocol with our generalized addition and multiplication gates, the sum of Hadamard products in Equation 15 can also be validated with a single sumcheck.

4.3 Design of Zero Knowledge CNN

In this section, we present the full design of our zero knowledge CNN scheme. The structure of our zkCNN is shown in Figure 2. As shown in the figure, the input consists of the data sample X for CNN prediction, the secret witness of the CNN model W from the prover, and the additional auxiliary inputs from the prover for computing functions such as ReLU and max pooling efficiently. Each convolutional layer takes the input from the previous layer, takes the kernels from W and executes our new sumcheck protocol in Section 3.2 and 4.2. The fully-connected layer takes the input from the previous layer and the weight matrix from W and executes the sumcheck protocol for matrix multiplication in [41]. The activation layer and the pooling layer takes the input from the previous layer and the auxiliary input, and we explain the details of our design for these layers below. Such connections are supported by our generalized GKR protocols in Section 4.2 without any overhead.

Converting real numbers. In practice, the parameters of the CNN model and the data samples are often represented as real numbers. In our scheme, we use the existing technique of quantization in [28] to encode them as integers in the finite field. The quantization scheme is an affine mapping of integers q to real numbers a. In particular, $a = L(q - Z)$, where quantization parameter L is a real number called the *scale* of the quantization and Z is an integer called the *zero-point* of the quantization. Using the quantization, we represent each value of the data samples and the model parameters as a Q-bit integer q. For the input matrix to each layer and each kernel, there is a single zero-point Z, represented by a Q bit integer. The representation of the scale is explained below.

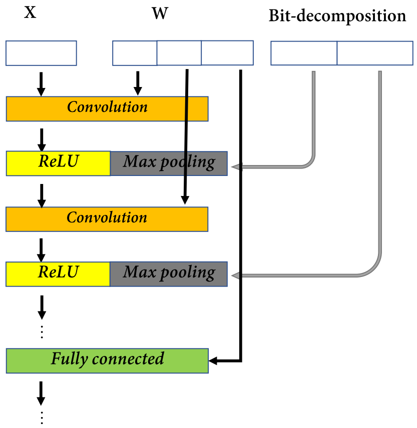


Figure 2: The design of our zkCNN structure.

With this representation, the addition of two real numbers with the same scale can naturally be expressed as integer addition. In particular, for $a_1 = L(q_1 - Z_1)$ and $a_2 = L(q_2 - Z_2)$, $a_1 + a_2 = L(q_1 + q_2 - Z_1 - Z_2)$. To perform real-number multiplications with different scales, i.e., $a_3 = a_1 \cdot a_2$, we have

$$L_3(q_3 - Z_3) = L_1(q_1 - Z_1) \cdot L_2(q_2 - Z_2)$$
$$\Leftrightarrow q_3 = Z_3 + \frac{L_1 L_2}{L_3}(q_1 - Z_1) \cdot (q_2 - Z_2).$$

Everything except $\frac{L_1 L_2}{L_3}$ is an integer and can be computed directly by the arithmetic circuit. Following the approach in [28], let $e = \frac{L_1 L_2}{L_3}$ be a real number, we normalize it as $2^{-E} \cdot \bar{e}$, where $\bar{e}$ is an integer called the *normalized scale*. Therefore, $q_3 = Z_3 + 2^{-E} \cdot \bar{e} \cdot (q_1 - Z_1) \cdot (q_2 - Z_2)$ where the multiplications are over integers in the finite field and 2^{-E} can be computed using a bit decomposition and shift in the arithmetic circuit. In this way, similar to the zero-point Z, the normalized scale $\bar{e}$ for the entire layer is also provided by the prover as part of the model.

The equations above naturally generalizes to the convolutions and matrix multiplications, as they consist of multiplications and additions with the same scale. To verify these computations in our protocol, the sumcheck protocols can be executed directly on $(q - Z)$ in the finite field. The normalized scale and the zero-point will be incorporated at the end. Moreover, because of the properties of ReLU and max pooling which will be presented below, they can also be computed on $(q - Z)$ and the scaling can be deferred further to the output of the pooling.

Computing ReLU. The ReLU function $\mathrm{ReLU}(x) = \max(x, 0)$ is applied element-wise after each convolutional layer. In our design, we denote a negative value x as $p - |x|$ in the finite field, where $|x|$ is the absolute value of x. Suppose $|x|$ is in the range $[0, 2^Q - 1]$, i.e. $|x|$ can be represented by Q bits (the same as the quantization q above), then we ask the prover to provide the bit decomposition $(b_0, \ldots, b_{Q-1})$ of $|x|$, as well as an additional bit b_Q denoting negative (0) or positive (1), as the auxiliary input to compute ReLU. Following the techniques in the literature [37], the protocol checks

(1) The auxiliary inputs are binary: $b_i(b_i - 1) = 0\ \forall i = 0, \ldots, Q$;

(2) They are the bit decomposition of $|x|$: $b_Q(x - \sum_{i=0}^{Q-1} b_i \cdot 2^i) + (1 - b_Q)(x + \sum_{i=0}^{Q-1} b_i \cdot 2^i) = 0$.
(3) With the bit-decomposition of x, the protocol computes the result of ReLU together with the truncation by keeping only Q' most significant bits to avoid overflow: $\mathrm{ReLU}(x) = b_Q \cdot \sum_{i=0}^{Q'-1} b_{i+Q-Q'} 2^i$.

Computing composition of max pooling and ReLU efficiently. A pooling layer is applied after an activation layer to reduce the dimension of the data. Max pooling works better than average pooling in practice for computer vision tasks such as image classifications [40]. However, due to efficiency considerations, existing schemes [23, 25] usually use average pooling instead of max pooling, as the former is a linear function while the latter requires comparisons in the circuit. In this paper, we propose a simple approach to compute the composition of max pooling and ReLU with only a small overhead.

The composition of ReLU and max pooling layers compute $\max\{\mathrm{ReLU}(x_0), ..., \mathrm{ReLU}(x_{k-1})\}$ with size k. The prover is required to provide the result of the above function $\bar{x}_{\max}$ as an auxiliary input. Then by the property of ReLU and maximum,

(1) $\bar{x}_{\max} - x_j \geq 0 \; \forall j \in [k]$.
(2) If x_is are not all negative numbers, then $\exists j \in [k]$, such that $\bar{x}_{\max} - x_j = 0$; otherwise $\bar{x}_{\max} = 0$.

The first condition can be checked by bit-decomposing each $\bar{x}_{\max} - x_j$ with Q bits as the auxiliary input from the prover (the $Q+1$-th bit denoting the sign of the number is not necessary, as it is always non-negative). The checks are exactly the same as the first two checks in the computation of ReLU above. The second condition is equivalent to $\bar{x}_{\max} \cdot \prod_{j=0}^{k-1}(\bar{x}_{\max} - x_j) = 0$. Finally, to avoid overflow, the prover also provides the bits of $\bar{x}_{\max}$ and the circuit validates the bit decomposition. Overall, comparing to computing ReLU above, the prover only additionally provides $\bar{x}_{\max}$ and its bits, and the protocol checks one additional bit decomposition.

4.4 Putting Everything Together

With the building blocks presented in Section 3 and 4, we construct a scheme of zero knowledge CNN predictions. The prover commits to the parameters $\mathbf{W}$ of a CNN model using a polynomial commitment scheme. Given an input data $\mathbf{X}$, the prover computes the prediction of $\mathrm{pred}(\mathbf{W}, \mathbf{X})$ together with the auxiliary input shown in Figure 2. The prover further commits to the additional auxiliary input using the polynomial commitment scheme. The prover and the verifier then invoke the sumcheck protocols and our generalized GKR protocols for matrix multiplications ([41]), convolutional layers (Section 3.2 and 4.2) and pooling and activation functions (Section 4.3) to reduce the correctness of the prediction to an evaluation of the multilinear extension of the input in Figure 2. Finally, the prover opens the polynomial commitments of the witness and auxiliary input at the evaluation point and completes the proof. The full protocol is presented in Protocol 3 in Appendix C.

Theorem 4.1. *Protocol 3 is correct and sound by Definition C.1.*

We present a proof sketch in Appendix C. We then take existing approaches in [17, 48] to turn the protocol to a zero knowledge argument. In particular, we use the zero knowledge sumcheck and the low degree extensions together with polynomial commitments to achieve zero knowledge. As shown in [48], the overhead is small in practice compared to the plain version without zero knowledge. We omit the formal protocol and the proof for the zero knowledge version of zkCNN in this paper. Finally, we remove the interactions in our zero knowledge CNN scheme using the Fiat-Shamir Heuristic [24] in the random oracle model. The transformation only incurs a negligible soundness loss [11].

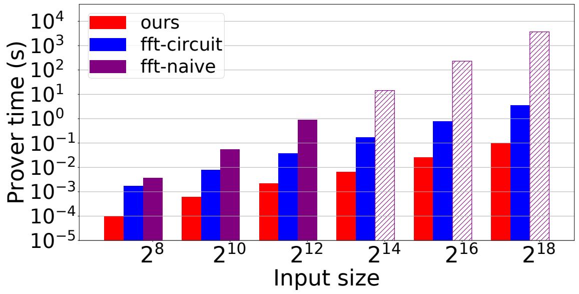


Figure 3: Sumcheck for FFT.

Complexity. The prover time of the interactive proof in our scheme is $O(\sum_{i=0}^{m}(n_i^2 \mathsf{ch}_{in,i} \mathsf{ch}_{out,i} + n_i^2 \mathsf{ch}_{out,i} Q))$, where Q is the maximum bit-length for bit decomposition in ReLU and max pooling. The proof size and verifier time are $O(\sum_{i=0}^{m}(\log^2(n_i^2 \mathsf{ch}_{in,i} \mathsf{ch}_{out,i}) + \log(n_i^2 \mathsf{ch}_{out,i} Q)))$. In addition, our scheme involves a polynomial commitment of size $S_{\mathsf{in}} = \sum_{i=0}^{m}(w_i^2 \mathsf{ch}_{in,i} \mathsf{ch}_{out,i} + n_i^2 \mathsf{ch}_{out,i} Q)$. Using the polynomial commitment scheme in [44], the prover time of this part is $O(S_{\mathsf{in}})$, the proof size and the verifier time are $O(\sqrt{S_{\mathsf{in}}})$.

Our scheme can be modified to a zero knowledge CNN accuracy scheme. The protocol is executed on the circuit in Figure 2 for multiple input samples, followed by a circuit to compare the results with the labels and computes the accuracy. Finally, our schemes can be made *non-interactive* using the Fiat-Shamir heuristic [24] with a negligible soundness loss [11].

5 IMPLEMENTATION AND EVALUATIONS

We implemented our zero knowledge proof scheme for CNN, zkCNN, and we present the experimental results in this section.

Software. The scheme is implemented in C++ and there are around 5000 lines of code. Some of our algorithms on the sumcheck protocol and the GKR protocol are based on the open-source implementation of [48, 51, 52]. We use the polynomial commitment scheme in [39, 44] because of its good prover time and reasonable proof size for the witness in our experiments. The security of the scheme relies on the discrete-log assumption. The prover time is $O(N)$ and the proof size and the verifier time are $O(\sqrt{N})$ for a polynomial of size N. We replace the Curve-25519 in the implementation of [39] with Curve BLS12-381 [14], as the order of Curve-25519 does not have a root of unity of a large power of 2 and thus is not FFT friendly. Curve BLS12-381 offers 128-bits of security and we use the mcl library [4] for its field and curve operations.

Hardware. We run all of the experiments on a machine with AMD EPYC 7R32 64-Core Processor and 128GB of RAM. Our current implementation is not parallelized and we only utilize a single CPU core. The large memory is used to run experiments on large CNNs (VGG16 with 15 million parameters) and multiple images. On one hand the memory usage is actually the bottleneck to further scale zkCNN and it is an interesting future work to improve it. On the other hand the memory usage and the scalability are already orders of magnitude better than existing SNARKs (See Section 5.2). We report the average running time of 10 executions.

5.1 New Sumcheck for FFT and Convolution

We first benchmark the performance of our new sumcheck protocol for FFT and 2-D convolutions. We exclude the running time and the proof size of the protocol to delegate the verifier's computation via Equation 10. This is because in applications such as CNN predictions in Section 5.2, the same Fourier matrix is used in many FFTs/convolutions and it is actually faster to compute the evaluation of its multilinear extension at a random point.

FFT. Figure 3 shows the prover time of our new sumcheck protocol for FFT in Section 3.1. As shown in the figure, the prover time of our new protocol is very fast in practice. It only takes 0.6ms to generate the proof for a vector of size 2^{10}, and 0.1s for a vector of size 2^{18}. The prover time grows strictly linearly with the size of the vector, as indicated by the complexity of our protocol. We compare the performance of our protocol with two baseline approaches: the GKR protocol on the FFT circuit and the naive sumcheck on Equation 5. Comparing to the baseline of the FFT circuit, our prover time is 17× faster for $n = 2^8$ and 33.2× faster for $n = 2^{18}$. The gap increases as the prover time of the baseline is $O(n \log n)$. The proof size of our protocol is 15.4–35.4× smaller than the baseline, as our protocol consists of a single sumcheck, while the depth of the circuit in the baseline is $\log n$. The verifier time of the two schemes are similar.

Comparing to the second baseline of naive sumcheck on Equation 5, our new protocol is significantly faster. The prover time is already 80× faster than the naive sumcheck for a vector of size 2^8, and the gap grows dramatically with the size as the complexity of the naive sumcheck is quadratic. The naive approach runs out of memory for 2^{14} and the shaded bars in the figure denote estimations. The proof size and the verifier time of the two schemes are exactly the same, as they are different algorithms for the same sumcheck on Equation 5.

Convolution. Figure 4 shows the prover time of the our protocol for convolutions and compares it with the GKR protocol on a circuit computing 2-D convolutions naively using multiplications and additions. The experiment is for a single convolution, and we vary the size of the input from 32×32 to 256×256, and the kernel size from 4×4 to half of the dimension of the input (this is the maximum kernel size for convolutions in CNN without padding).

As shown in the figure, the prover time is improved significantly over the baseline. It only takes 4.7ms to generate the proof for a convolution on 32×32 and 4×4 matrices, which is already 1.6× faster than the 7.7ms in the naive approach. The speedup grows dramatically with the size of the kernel. For a convolution between input 32×32 and kernel 16×16, our prover time is 8.5× faster than the baseline; for a convolution on the largest instance of input 256×256 and kernel 64×64, our speedup is 291×. Moreover, the prover time almost remains the same for the same input size. This is because the kernel has to be padded to the size of the input anyway to perform FFT, thus different kernel sizes do not make a difference for the same input size in our protocol.

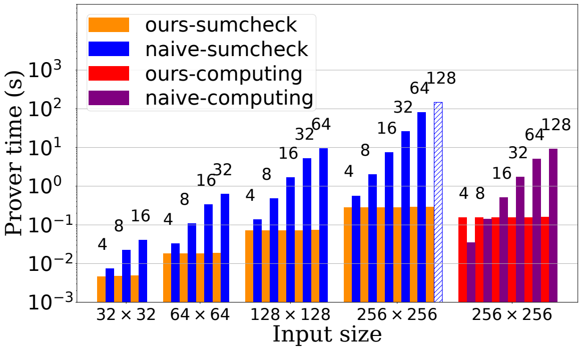


Figure 4: Sumcheck for a single convolution.

Proof size and verifier time. Our proof size and verifier time are slightly worse than the baseline. The proof size of our protocol ranges from 5.6KB to 8.4KB, while it is 3.9KB to 7.1KB in the naive approach. This is because our protocol has three sumchecks for FFT, Hadamard product and IFFT to compute the convolution, and the naive approach has two sumchecks, one for all multiplications and one for addition trees. The proof size is linear in the number of sumchecks and logarithmic in the size of each sumcheck. The verifier time in both protocols are extremely fast. Here we do not count the time to compute the multilinear extensions of the input and the output, as they are given by the prover during the reductions of the GKR protocols from other layers. The verifier time is only logarithmic, and ranges from 0.1ms (32×32 and 4×4) to 0.3ms (256×256 and 128×128) in our protocol, and ranges from 0.1ms to 0.2ms in the baseline.

Comparing to computing the result. As another benchmark, we further measure the time to compute the result of the convolutions for the input size of 256×256 using FFT and naive multiplications and additions. As shown in Figure 4, the additional prover time of our sumcheck protocol is only 1.8× slower than computing the result using FFT. The prover time is slower than the naive computation by 8× on the small kernel of 4×4, but is 31× faster on the large kernel of 128×128. The result agrees with the optimal complexity of our sumcheck protocol and shows that the overhead to generate the proof is very small in practice.

Comparing to other approaches. To further demonstrate the efficiency of our protocol, we compare the running time with the approach of verifying convolutions in [34]. In [34, Figure 6], it takes around 2.5s to generate the proof for a convolution between an input of size 10,000 and a kernel of size 10. On a larger instance of input 128×128 =16,384 and kernel $4 \times 4 = 16$ in our scheme, the prover time is only 0.072s, which is 1–2 orders of magnitude faster.

5.2 Performance of zkCNN

In this section, we evaluate the performance of our zkCNN system.

Datasets and CNNs. We use two datasets: MNIST [33] and CIFAR-10 [31]. MNIST is a dataset of hand-written digits. The images are of size $28 \times 28 \times 1$. There are 50,000 training data samples and 10,000 testing data sample, classified into 10 categories of digits 0–9. The CIFAR-10 dataset consists of 60,000 images in 10 classes including airplane, automobile and so on. The images are of size $32 \times 32 \times 3$. There are 50,000 training images and 10,000 testing images.

	LeNet	VGG11	VGG16
sumcheck	0.280s	28.9s	57.7s
poly commit	0.161s	19.0s	30.6s
Total prover	**0.441**s	**47.8**s	**88.3**s
sumcheck	0.900ms	2.70ms	3.80ms
poly commit	4.90ms	36.5ms	55.5ms
Total verifier	**5.80**ms	**39.3**ms	**59.3**ms
sumcheck	45.9KB	110KB	147KB
poly commit	25.4KB	194KB	194KB
Total proof	**71.3**KB	**304**KB	**341**KB

Table 1: Performance of zkCNN.

LeNet (average pooling)			
	prover	proof	verifier
Ours	0.49s	63.6KB	5.5ms
vCNN [34]	5.49s	0.34KB	84ms
LeNet (CIFAR-10 & average pooling)			
	prover	proof	verifier
Ours	0.56s	68.4KB	5.6ms
ZEN [23]	119.5s	0.28KB	18.6ms
VGG16			
Ours	88.3s	341KB	59.3ms
vCNN [34]	31 hours*	0.34KB	20s*

Table 2: Comparison to existing schemes. * means estimated.

We test both small CNNs and relatively large CNNs in this paper. For the small CNN, we use the LeNet [32] with 61,706 parameters, consisting of 2 convolutional layers, 2 pooling layers and 3 fully-connected layers. For large CNNs, we use the VGG11 and VGG16 [40] with 9.7 million and 15.2 million parameters respectively. In VGG11, there are 8 convolution layers, 5 pooling layers and 3 fully connected layers. VGG16 has 5 more convolutional layers than VGG11. LeNet, VGG11 and VGG16 can use both average and max pooling in the pooling layers.

Performance of zkCNN. Table 1 summarizes the performance of our zkCNN system on various CNNs, with the breakdown of the sumcheck protocols and the polynomial commitment. As shown in the table, the performance of zkCNN is very efficient in practice. It only takes 0.44s to generate a proof for a prediction of LeNet on an MNIST data sample. For a large CNN of VGG16 with 15 million parameters and 16 layers on CIFAR-10, the prover time is only 88 seconds. The proof size is 71.3KB for LeNet and 341KB for VGG16, both significantly smaller than the size of the parameters of the models. The verifier time of zkCNN is extremely fast in practice, thanks to the sublinear verifier of the GKR protocol on highly structured computations including our zkCNN in Figure 2. It only takes 5.8ms to verify a prediction of LeNet, and 59.3ms to verify a prediction of VGG16. They are even faster than computing the CNN predictions locally, thus our zkCNN both protects the privacy of the CNN models and improves the efficiency of the verifier. Finally, the maximum memory usage of our zkCNN on VGG16 is 24GB, which is reasonable for a personal computer.

The breakdown of the prover time further shows the improvement of our new sumcheck and GKR protocols. Though the computation of CNN predictions is significantly larger than the size of the input and the model, our scheme is able to bring down the cost of this part to around half of the total prover time.

Our models are trained using TensorFlow [5] with the post-training quantization to be compatible with the quantization technique we use to encode real numbers. We use 8-bit integer for the quantized number q and 32-bit integer for the normalized scale $\bar{e}$ as in [28]. Our LeNet achieves 98.85% accuracy on MNIST, and our VGG11 and VGG16 achieves 88.7% and 90.3% accuracy on CIFAR-10.

Comparison to existing schemes. We then compare the performance of zkCNN with existing schemes in Table 2. We change the pooling in LeNet to average pooling to be consistent with [23, 34]. We executed their open-source implementations [2] and [3] on the same machine to obtain their performance. As shown in the table, the prover time of zkCNN is 11.2× faster than vCNN on LeNet (with average pooling as in [34]) and 1264× faster than vCNN on VGG16. In fact vCNN cannot scale to VGG16 and the numbers are estimated. As ZEN only supports LeNet on CIFAR-10, we further test our zkCNN on the same CNN and dataset and our prover time is 0.56s, 213× faster than ZEN. The result dramatically improves the state-of-the-art on zero knowledge neural network predictions and makes it possible to prove large CNN predictions in minutes. The proof size of zkCNN is worse than vCNN and ZEN, as they are using the pairing-based SNARK with a constant size proof. However, zkCNN does not have a trusted setup and a common reference string of tens of GBs as in [23, 34]. The SNARK in [23] could be replaced by others to remove the trusted setup and the large common reference string, but doing so would increase the proof size. vCNN [34] has to use the SNARK in [27] as the protocol heavily relies on the conversion between the QAP and the polynomial-QAP.

Zero knowledge proofs for CNN accuracy. Finally, we demonstrate our zkCNN for proving the accuracy of the CNN models on multiple input samples, which has not been explored in existing works due to the issue of scalability. Figure 5 in the appendix shows the prover time, proof size and verifier time of proving the accuracy of VGG16. It takes 520s to prove the accuracy on a dataset of 20 images, which is faster than repeating the single prediction 20 times, as the convolutions and matrix multiplications are performed on the same parameters. For 20 images, the proof size is only 635KB and the verifier time is only 121ms. The sudden jumps in the proof size and the verifier time are due to the polynomial commitment scheme. The size of the witness is padded to the nearest power of 2, which increases to 2^{26} from 2^{25} at 3 images and 2^{27} at 10 images. It significantly affects the proof size and the verifier time, but not the prover time as the prover time mostly depends on the number of nonzero elements in the witness. Moreover, the proof size does not increase significantly at 10 images as the witness is arranged as a $\sqrt{N} \times \sqrt{N}$ matrix in the polynomial commitment scheme, and the number of rows is set to 2^{13} for both $N = 2^{26}$ and $N = 2^{27}$.

ACKNOWLEDGMENTS

We would like to thank Gaofeng Huang and Junjie Shi for helping train plain CNN models. This material is based upon work supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. The third author is generously supported by a research award from Latticex Foundation.

REFERENCES

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

[2] [n.d.]. Implementation of vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs. https://github.com/snp-labs/VCNN.

[3] [n.d.]. Implementation of ZEN: Efficient Zero-Knowledge Proof for Neural Networks. https://github.com/UCSB-TDS/ZEN.

[4] [n.d.]. mcl. https://github.com/herumi/mcl/.

[5] 2021. TensorFlow. https://www.tensorflow.org/.

[6] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[7] Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. 2020. Mac'n'Cheese: Zero-Knowledge Proofs for Arithmetic Circuits with Nested Disjunctions. Cryptology ePrint Archive, Report 2020/1410. https://eprint.iacr.org/2020/1410.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*. Springer, 701–732.

[9] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. [n.d.]. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*.

[10] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019*. Springer International Publishing, 103–128. https://doi.org/10.1007/978-3-030-17653-2_4

[11] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive oracle proofs. In *Theory of Cryptography Conference*. Springer, 31–60.

[12] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. [n.d.]. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *Proceedings of the USENIX Security Symposium, 2014*.

[13] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. 2020. Ligero++: A New Optimized Sublinear IOP. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2025–2038.

[14] Sean Bowe. [n.d.]. BLS12-381: New zk-SNARK Elliptic Curve Construction.

[15] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. [n.d.]. Bulletproofs: Short Proofs for Confidential Transactions and More. In *Proceedings of the Symposium on Security and Privacy (SP), 2018*, Vol. 00. 319–338.

[16] Matteo Campanelli, Dario Fiore, and Anaïs Querol. [n.d.]. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs.. In *CCS 2019*.

[17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. 2017. A Zero Knowledge Sumcheck and its Applications. *CoRR* abs/1704.02086 (2017). arXiv:1704.02086 http://arxiv.org/abs/1704.02086

[18] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 738–768.

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[20] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. [n.d.]. Practical Verified Computation with Streaming Interactive Proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*.

[21] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*. 90–112.

[22] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. 2020. Line-Point Zero Knowledge and Its Applications. Cryptology ePrint Archive, Report 2020/1446. https://eprint.iacr.org/2020/1446.

[23] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: Efficient Zero-Knowledge Proofs for Neural Networks. Cryptology ePrint Archive, Report 2021/087. https://eprint.iacr.org/2021/087.

[24] Amos Fiat and Adi Shamir. [n.d.]. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto 1986*.

[25] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 4672–4681.

[26] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4, Article 27 (Sept. 2015), 64 pages.

[27] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 305–326.

[28] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877 [cs.LG]

[29] Julien Keuffer, Refik Molva, and Hervé Chabanne. 2018. Efficient Proof Composition for Verifiable Computation. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11098)*. Springer, 152–171.

[30] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. 2020. MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs. 2129–2146.

[31] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2012).

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[33] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[34] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs. Cryptology ePrint Archive, Report 2020/584. https://eprint.iacr.org/2020/584.

[35] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (Oct. 1992), 859–868.

[36] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings. Cryptology ePrint Archive, Report 2019/099. https://eprint.iacr.org/2019/099.

[37] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *S&P 2013*. 238–252.

[38] Jacob T Schwartz. 1979. Probabilistic algorithms for verification of polynomial identities. In *International Symposium on Symbolic and Algebraic Manipulation*. Springer, 200–215.

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

[40] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[41] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.).

[42] Riad S Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. 2016. Verifiable asics. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 759–778.

[43] Riad S Wahby, Ye Ji, Andrew J Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. 2017. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

[44] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 926–943.

[45] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2020. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. Cryptology ePrint Archive, Report 2020/925. https://eprint.iacr.org/2020/925.

[46] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. Cryptology ePrint Archive, Report 2021/730. https://ia.cr/2021/730.

[47] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. 2018. DIZK: A Distributed Zero Knowledge Proof System. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 675–692.

[48] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology (CRYPTO)*.

[49] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. Cryptology ePrint Archive, Report 2021/076. https://eprint.iacr.org/2021/076.

[50] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero Knowledge Proofs for Decision Tree Predictions and Accuracy. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. ACM, 2039–2053.

[51] Jiaheng Zhang, Weijie Wang, Yinuo Zhang, and Yupeng Zhang. 2020. Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time. Cryptology ePrint Archive, Report 2020/1247. https://eprint.iacr.org/2020/1247.

[52] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. [n.d.]. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *S&P 2020*.

[53] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying arbitrary SQL queries over

dynamic outsourced databases. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 863–880.

[54] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology ePrint.

[55] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. vRAM: Faster verifiable RAM with program-independent preprocessing. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)*.

[56] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, Xiaodong Lin, Shengshan Hu, and Minxin Du. 2019. VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service. arXiv:1909.06961 [cs.CR]

[57] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Manipulation*. Springer, 216–226.

A GKR PROTOCOL

Theorem A.1. *[48]. Let $C\colon \mathbb{F}^{S_{\text{in}}} \rightarrow \mathbb{F}^{S_{\text{out}}}$ be a layered arithmetic circuit of depth d. Protocol 2 is an interactive proof for the function computed by C with soundness $O(d \log |C|/|\mathbb{F}|)$. It uses $O(d \log |C|)$ rounds of interaction and the running time of the prover $\mathcal{P}$ is $O(|C|)$. Let T be the time to evaluate all $\tilde{add}_i$ and $\tilde{mult}_i$ at the corresponding random points, the running time of $\mathcal{V}$ is $O(S_{\text{in}} + S_{\text{out}} + d \log |C| + T)$.*

B ZERO KNOWLEDGE ARGUMENTS

An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in R$ for some input x. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w. We use $\mathcal{G}$ to represent the generation phase of the public parameters pp. Formally, consider the definition below, where we assume R is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition B.1. Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero knowledge argument of knowledge for $\mathcal{R}$ if the following holds.

- **Correctness**. For every pp output by $\mathcal{G}(1^\lambda)$ and $(x, w) \in R$,
$$\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}) \rangle(x) = 1$$
- **Knowledge Soundness**. For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that given the access to the entire executing process and the randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $\pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp})$ and $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, x, \pi^*)$, the following probability is $\mathsf{negl}(\lambda)$:
$$\Pr[(x; w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \mathsf{pp}) = 1]$$
- **Zero knowledge**. There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0, 1\}^*$, $(x; w) \in \mathcal{R}$, pp output by $\mathcal{G}(1^\lambda)$, it holds that
$$\mathsf{View}(\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}^*(z, \mathsf{pp}) \rangle(x)) \approx \mathcal{S}^{\mathcal{V}^*}(x, z)$$

We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a **succinct** argument system if the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) are $\mathsf{poly}(\lambda, |x|, \log |w|)$.

In the definition of zero knowledge, $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ is given the randomness of $\mathcal{V}^*$ sampled from polynomial-size space. This definition is commonly used in existing transparent zero knowledge proof schemes [6, 10, 15, 44, 52].

Zero knowledge polynomial commitment. Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$, and D be a variable-degree parameter. We use $\mathcal{W}_{\ell,d}$ to denote the collection of all monomials in $\mathcal{F}$ and $N = |\mathcal{W}_{\ell,D}| = (D+1)^\ell$. A zero knowledge verifiable polynomial commitment (zkPC) for $f \in \mathcal{F}$ and $t \in \mathbb{F}^\ell$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$,
- $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$,
- $((y, \pi); \{0, 1\}) \leftarrow \langle \mathsf{zkPC.Open}(f, r_f), \mathsf{zkPC.Verify}(\mathsf{com}) \rangle(t, \mathsf{pp})$

Definition B.2. A zkPC scheme satisfies the following properties:

- **Completeness.** For any polynomial $f \in \mathcal{F}$ and value $t \in \mathbb{F}^\ell$, $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$, it holds that
$$\Pr\left[\langle \mathsf{zkPC.Open}(f, r_f), \mathsf{zkPC.Verify}(\mathsf{com}) \rangle(t, \mathsf{pp}) = 1\right] = 1$$
- **Knowledge Soundness.** For any PPT adversary $\mathcal{A}$ and $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, there exists a PPT extractor $\mathcal{E}$. Given any tuple (pp, com^*) and the executing process of $\mathcal{A}$, $\mathcal{E}$ can extract a function $f^* \in \mathcal{F}$ and the randomness r_{f^*} such that:
$$\Pr\begin{bmatrix} ((y^*, \pi^*); 1) \leftarrow \langle \mathcal{A}(), \mathsf{zkPC.Verify}(\mathsf{com}^*) \rangle(t, \mathsf{pp}) \\ \wedge (f^*, r_{f^*}) \leftarrow \mathcal{E} || \mathcal{A}(\mathsf{pp}, \mathsf{com}^*) \\ \wedge \mathsf{com}^* = \mathsf{zkPC.Commit}(f^*, r_{f^*}, \mathsf{pp}) \\ \wedge f^*(t) \neq y^* \end{bmatrix} \leq \mathsf{negl}(\lambda).$$
- **Zero Knowledge.** For security parameter λ, polynomial $f \in \mathcal{F}$, $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:

$\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp})$:
(1) $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f)$
(2) $t \leftarrow \mathcal{A}(\mathsf{com})$
(3) $(y, \pi) \leftarrow \langle \mathsf{zkPC.Open}(f, r_f), \mathcal{A} \rangle(t)$
(4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi)$
(5) Output b

$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp})$:
(1) $\mathsf{com} \leftarrow \mathcal{S}_1(1^\lambda)$
(2) $t \leftarrow \mathcal{A}(\mathsf{com})$
(3) $(y, \pi) \leftarrow \langle \mathcal{S}_2, \mathcal{A} \rangle(t_i)$, given oracle access to $y = f(t)$.
(4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi)$
(5) Output b

For any PPT algorithm $\mathcal{A}$ and all polynomial $f \in \mathbb{F}$, there exists simulator $\mathcal{S}$ such that
$$|\Pr[\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp}) = 1]| \leq \mathsf{negl}(\lambda).$$

C DEFINITION, PROTOCOL AND PROOFS OF ZERO KNOWLEDGE CNN

Definition C.1. We say that a scheme is a zero knowledge convolutional neural network if the following holds:

- **Completeness.** For any CNN parameters $\mathbf{W}$ and data sample $\mathbf{X}$, $\mathsf{pp} \leftarrow \mathsf{zkCNN.KeyGen}(1^\lambda)$, $\mathsf{com}_{\mathbf{W}} \leftarrow \mathsf{zkCNN.Commit}(\mathbf{W}, \mathsf{pp}, r)$, $(y, \pi) \leftarrow \mathsf{zkCNN.Prove}(\mathbf{W}, \mathbf{X}, \mathsf{pp}, r)$, it holds that
$$\Pr\left[\mathsf{zkCNN.Verify}(\mathsf{com}_{\mathbf{W}}, \mathbf{X}, y, \pi, \mathsf{pp}) = 1\right] = 1$$
- **Soundness.** For any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:
$$\Pr\begin{bmatrix} \mathsf{pp} \leftarrow \mathsf{zkCNN.KeyGen}(1^\lambda) \\ (\mathbf{W}^*, \mathsf{com}_{\mathbf{W}^*}, \mathbf{X}, y^*, \pi^*, r) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \\ \mathsf{com}_{\mathbf{W}^*} = \mathsf{zkCNN.Commit}(\mathbf{W}^*, \mathsf{pp}, r) \\ \mathsf{zkCNN.Verify}(\mathsf{com}_{\mathbf{W}^*}, \mathbf{X}, y^*, \pi^*, \mathsf{pp}) = 1 \\ y^* \neq \mathsf{pred}(\mathbf{W}^*, \mathbf{X}) \end{bmatrix}$$

Protocol 2 (GKR). *Let $\mathbb{F}$ be a finite field. Let $C\colon \mathbb{F}^{S_{\text{in}}} \rightarrow \mathbb{F}^{S_{\text{out}}}$ be a layered arithmetic circuit of depth d. $\mathcal{P}$ wants to convince that* ***out*** $= C($***in***$)$ *where* ***in*** *is the input from $\mathcal{V}$, and* ***out*** *is the output. Without loss of generality, we pad S_{in} and S_{out} to powers of 2.*

1. *Define the multilinear extension of array* ***out*** *as $\tilde{V}_0$. $\mathcal{V}$ chooses a random $g \in \mathbb{F}^{s_0}$ and sends it to $\mathcal{P}$. Both parties compute $\tilde{V}_0(g)$.*
2. *$\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on*

$$\tilde{V}_0(g^{(0)}) = \sum_{x,y\in\{0,1\}^{s_1}} (\tilde{add}_1(g^{(0)},x,y)(\tilde{V}_1(x)+\tilde{V}_1(y)) + \tilde{mult}_1(g^{(0)},x,y)\tilde{V}_1(x)\tilde{V}_1(y))$$

At the end of the protocol, $\mathcal{V}$ receives $\tilde{V}_1(u^{(1)})$ and $\tilde{V}_1(v^{(1)})$. $\mathcal{V}$ computes $\tilde{mult}_1(g^{(0)},u^{(1)},v^{(1)})$, $\tilde{add}_1(g^{(0)},u^{(1)},v^{(1)})$ and checks that $\tilde{add}_1(g^{(0)},u^{(1)},v^{(1)})\ (\tilde{V}_1(u^{(1)})+\tilde{V}_1(v^{(1)})) + \tilde{mult}_1(g^{(0)},u^{(1)},v^{(1)})\ \tilde{V}_1(u^{(1)})\tilde{V}_1(v^{(1)})$ equals to the last message of the sumcheck.

3. *For $i = 1, \ldots, d-1$:*
   - *$\mathcal{V}$ randomly selects $r_{i,1}, r_{i,2} \in \mathbb{F}$ and sends them to $\mathcal{P}$.*
   - *$\mathcal{P}$ and $\mathcal{V}$ run the sumcheck on the equation*

$$\begin{aligned} r_{i,1}\tilde{V}_i(u^{(i)}) + r_{i,2}\tilde{V}_i(v^{(i)}) = \sum_{x,y\in\{0,1\}^{s_{i+1}}} & ((r_{i,1}\tilde{add}_{i+1}(u^{(i)},x,y) + r_{i,2}\tilde{add}_{i+1}(v^{(i)},x,y)) \cdot (\tilde{V}_{i+1}(x)+\tilde{V}_{i+1}(y)) \\ & +(r_{i,1}\tilde{mult}_{i+1}(u^{(i)},x,y) + r_{i,2}\tilde{mult}_{i+1}(v^{(i)},x,y)) \cdot \tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)) \end{aligned}$$

   - *At the end of the sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ $\tilde{V}_{i+1}(u^{(i+1)})$ and $\tilde{V}_{i+1}(v^{(i+1)})$.*
   - *$\mathcal{V}$ computes the following and checks if it equals to the last message of the sumcheck.*

$$\begin{gathered} (r_{i,1}\tilde{mult}_{i+1}(u^{(i)},u^{(i+1)},v^{(i+1)}) + r_{i,2}\tilde{mult}_{i+1}(v^{(i)},u^{(i+1)},v^{(i+1)}) \cdot (\tilde{V}_{i+1}(u^{(i+1)})\tilde{V}_{i+1}(v^{(i+1)}))+ \\ (r_{i,1}\tilde{add}_{i+1}(u^{(i)},u^{(i+1)},v^{(i+1)}) + r_{i,2}\tilde{add}_{i+1}(v^{(i)},u^{(i+1)},v^{(i+1)}) \cdot (\tilde{V}_{i+1}(u^{(i+1)}) + \tilde{V}_{i+1}(v^{(i+1)})) \end{gathered}$$

If all checks in the sumcheck pass, V uses $\tilde{V}_{i+1}(u^{(i+1)})$ and $\tilde{V}_{i+1}(v^{(i+1)})$ to proceed to the $(i+1)$-th layer. Otherwise, $\mathcal{V}$ outputs `0`.

4. *At the input layer d, $\mathcal{V}$ has two claims $\tilde{V}_d(u^{(d)})$ and $\tilde{V}_d(v^{(d)})$. $\mathcal{V}$ evaluates $\tilde{V}_d$ at $u^{(d)}$ and $v^{(d)}$ using the input and checks that they are the same as the two claims. If yes, output* `1`*; otherwise, output* `0`.

- **Zero Knowledge.** For security parameter λ, pp $\leftarrow$ zkCNN.KeyGen(1^λ), for CNN parameters $\mathbf{W}$, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:

Real$_{\mathcal{A},W}$(pp):

1. com$_{\mathbf{W}}$ $\leftarrow$ zkCNN.Commit $(\mathbf{W}, \text{pp}, r)$
2. $\mathbf{X} \leftarrow \mathcal{A}(\text{com}_{\mathbf{W}}, \text{pp})$
3. $(y, \pi) \leftarrow$ zkCNN.Prove$(\mathbf{W}, \mathbf{X}, \text{pp}, r)$
4. $b \leftarrow \mathcal{A}(\text{com}_{\mathbf{W}}, \mathbf{X}, y, \pi, \text{pp})$
5. Output b

Ideal$_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}$(pp, h):

1. com $\leftarrow \mathcal{S}_1(1^\lambda, \text{pp}, r)$
2. $\mathbf{X} \leftarrow \mathcal{A}(\text{com}, \text{pp})$
3. $(y, \pi) \leftarrow \mathcal{S}_2^{\mathcal{A}}(\text{com}, \mathbf{X}, \text{pp}, r)$, given oracle access to $y = \text{pred}(\mathbf{W}, \mathbf{X})$.
4. $b \leftarrow \mathcal{A}(\text{com}, \mathbf{X}, y, \pi, \text{pp})$
5. Output b

For any PPT algorithm $\mathcal{A}$ and all CNN models $\mathbf{W}$, there exists simulator $\mathcal{S}$ such that

$$|\Pr[\text{Real}_{\mathcal{A},\mathbf{W}}(\text{pp}) = 1] - \Pr[\text{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\text{pp}) = 1]| \le \text{negl}(\lambda).$$

Our zkCNN scheme further satisfies the stronger notion of knowledge soundness, where there exists an extractor to extract the CNN parameters from a valid proof and prediction with overwhelming probability, when we use the polynomial commitment scheme in [39] with knowledge soundness.

Proof sketch of Theorem 4.1. The correctness of our zkCNN follows the correctness of the sumcheck protocols for convolutions and matrix multiplications, our generalized GKR protocol on ReLU and max pooling, and the polynomial commitments.

We prove soundness following the same ideas as the proofs of the GKR protocol and the GKR-based zero knowledge arguments in [26, 48, 52]. Suppose the prediction sent by $\mathcal{P}$ is not correctly computed, i.e., $y^* \neq \text{pred}(\mathbf{W}, \mathbf{X})$, but still passes the verification. Let $\tilde{X}^*_m()$ be the multilinear extension of $X^*_m = y \neq X_m$, by the Schwartz-Zippel lemma [38, 57], in Step (2) of Protocol 3, $\tilde{X}^*_m(r^{(m)}) \neq \tilde{X}_m(r^{(m)})$ with all but probability $\frac{1}{|\mathbb{F}|}$. In Step (3) of Protocol 3, for layer $i = m, m-1, m, \ldots, 2$, suppose $\tilde{X}^*_i(r^{(i)}) \neq \tilde{X}_i(r^{(i)})$, where $\tilde{X}^*_i(r^{(i)})$ denotes the message sent by the adversary as the evaluation of the multilinear extension of the output of layer i, and $\tilde{X}_i(r^{(i)})$ denotes the corresponding message of an honest prover. We differentiate the following cases:

- If layer i is a fully connected layer: Case 1: if $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}^*_{i-1}(r^{(i-1)}) = \tilde{W}_{i-1}(r^{(i-1)})$, then as $\tilde{X}^*_i(r^{(i)}) \neq \tilde{X}_i(r^{(i)})$, by Lemma 1 of the sumcheck protocol, $\mathcal{V}$ rejects with all but negligible probability. Case 2: $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}^*_{i-1}(r^{(i-1)}) \neq \tilde{W}_{i-1}(r^{(i-1)})$, proceed to the input layer. Case 3: $\tilde{X}^*_{i-1}(r^{(i-1)}) \neq \tilde{X}_{i-1}(r^{(i-1)})$, then proceed to layer $i-1$.
- If layer i is a convolutional layer: Case 1: if there is $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}^*_{i-1}(r^{(i-1)}) = \tilde{W}_{i-1}(r^{(i-1)})$, then as $\tilde{X}^*_i(r^{(i)}) \neq \tilde{X}_i(r^{(i)})$, by Lemma 1 of the sumcheck protocol, $\mathcal{V}$ rejects with all but negligible probability. Note that our new protocols in Section 3 for the sumcheck of FFT and convolution improve the prover efficiency with new algorithms. The messages sent by the prover remain the same as the original sumcheck protocols on these equations and thus correctness and the soundness of the

Protocol 3 (zkCNN Protocol). *Let C be a CNN of m layers with model parameters $\mathbf{W}$. Let $X_1 = \mathbf{X}$ be a data sample as the input, $X_m = y$ be the prediction* $\mathsf{pred}(\mathbf{W}, \mathbf{X})$ *of C on $\mathbf{X}$, and X_i be the output of the i-th layer of C.*

- $\mathsf{zkCNN.KeyGen}(1^\lambda)$*: Run* $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$ *and outputs* pp.
- $\mathsf{zkCNN.Commit}(\mathbf{W}, \mathsf{pp}, r)$*: Define the multilinear extension of $\mathbf{W}$, viewed as an array by concatenating the parameters of each layer, as $\tilde{W}$. $\mathcal{P}$ commits to $\tilde{W}$ by running* $\mathsf{com}_W \leftarrow \mathsf{zkPC.Commit}(\tilde{W}, \mathsf{pp}, r_W)$ *and sends $\mathcal{V}$* com_W.
- $\langle \mathsf{zkCNN.Prove}(\mathbf{W}, r_W), \mathsf{zkCNN.Verify}(\mathsf{com}_{\mathbf{W}}) \rangle(\mathbf{X}, \mathsf{pp})$*:*
  1. *Upon receiving $\mathbf{X}$ from $\mathcal{V}$, $\mathcal{P}$ evaluates the CNN to compute the prediction $y = X_m$, the values in each layer X_i and the auxiliary input* **aux** *for bit-decomposition. $\mathcal{P}$ commits to the multilinear extension of* aux *by running* $\mathsf{com}_{aux} \leftarrow \mathsf{zkPC.Commit}(a\tilde{u}x, \mathsf{pp}, r_{aux})$ *and sends $\mathcal{V}$* com_{aux} *and $y = X_m$. Without loss of generality, we pad* $\mathbf{X}, \mathbf{W}, \mathbf{aux}$ *to the maximum length of the three, N, and arrange the input of the circuit as* $\mathbf{X}||\mathbf{W}||\mathbf{X}||\vec{0}$ *of size $4N$.*
  2. *$\mathcal{V}$ defines the multilinear extension of $y = X_m$ as $\tilde{X}_m$. $\mathcal{V}$ chooses a random $r^{(m)}$ and sends it to $\mathcal{P}$. Both parties compute $\tilde{X}_m(r^{(m)})$.*
  3. *For $i = m, m-1, \ldots, 2$, with $\tilde{X}_i(r^{(i)})$,*
     - *If layer i is a fully connected layer, $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on*

       $$\tilde{X}_i(x, y) = \sum_z \tilde{X}_{i-1}(x, z) \cdot \tilde{W}_{i-1}(z, y)$$

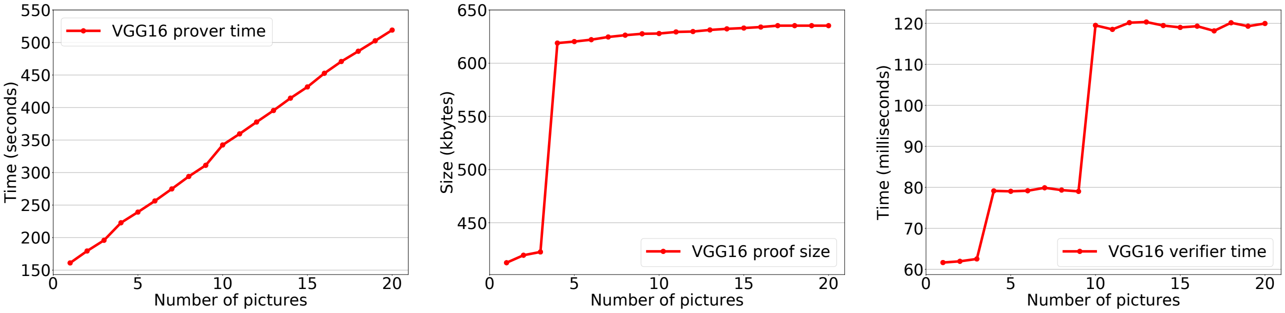
       using the sumcheck algorithm for matrix multiplication proposed in [41]. At the end of the sumcheck, $\mathcal{V}$ receives $\tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}_{i-1}(r^{(i-1)})$, where $\tilde{W}_{i-1}$ denotes the multilinear extension defined by the model parameters W_i used in layer i.
     - *If layer i is a convolutional layer, $\mathcal{P}$ and $\mathcal{V}$ run sumcheck protocols on Equation 15, which consists of two sumchecks for FFT and IFFT using the algorithms in Section 3 and one sumcheck for Hadamard product. At the end of the sumcheck, $\mathcal{V}$ receives $\tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}_{i-1}(r^{(i-1)})$.*
     - *If layer i is an activation and max pooling layer, $\mathcal{P}$ and $\mathcal{V}$ run the GKR protocol on our optimized circuit for ReLU and max pooling described in Section 4.3. At the end of the sumcheck, $\mathcal{V}$ receives $\tilde{X}_{i-1}(r^{(i-1)})$ and $a\tilde{u}x_{i-1}(r^{(i-1)})$, where $a\tilde{u}x_{i-1}$ denotes the multilinear extension defined by auxiliary input aux_i used in layer i.*
  4. *At the input layer, $\mathcal{V}$ has received $\tilde{X}_1(r^{(1)})$ and $\tilde{W}_{i-1}(r^{(i-1)})$, $a\tilde{u}x_{i-1}(r^{(i-1)})$ for $i = m, \ldots, 1$. $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on Equation 14 to combine them into a single evaluation. At the end of the sumcheck, $\mathcal{V}$ receives $\tilde{in}(r)$, where $\tilde{in}$ denotes the multilinear extension defined by the entire input of the circuit.*
  5. *Finally, $\mathcal{V}$ validates $\tilde{in}(r_{in})$ by opening the polynomial commitments. In particular, as the size of the input is $4N$, let $r = (r_1, \ldots, r_{\log N+2})$ and $r^- = (r_1, \ldots, r_{\log N})$, $\mathcal{P}$ and $\mathcal{V}$ run* $\langle \mathsf{zkPC.Open}(\tilde{W}, r_W), \mathsf{zkPC.Verify}(\mathsf{com}_W) \rangle(r^-, \mathsf{pp})$ *and* $\langle \mathsf{zkPC.Open}(a\tilde{u}x, r_{aux}), \mathsf{zkPC.Verify}(\mathsf{com}_{aux}) \rangle(r^-, \mathsf{pp})$. *$\mathcal{V}$ receives $\tilde{W}(r^-)$ and $a\tilde{u}x(r^-)$, and evaluates $\tilde{X}_1(r^-)$ locally. $\mathcal{V}$ checks that $\tilde{in}(r) = \tilde{X}_1(r^-) \cdot (1 - r_{\log N+1})(1 - r_{\log N+2}) + \tilde{W}(r^-) \cdot (1 - r_{\log N+1}) r_{\log N+2} + a\tilde{u}x(r^-) \cdot r_{\log N+1}(1 - r_{\log N+2})$. If the check and all the verifications of the polynomial commitments and sumchecks pass, $\mathcal{V}$ outputs* `1`*; otherwise, $\mathcal{V}$ outputs* `0`.

protocols remain the same. Case 2: $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $\tilde{W}^*_{i-1}(r^{(i-1)}) \neq \tilde{W}_{i-1}(r^{(i-1)})$, proceed to the input layer. Case 3: $\tilde{X}^*_{i-1}(r^{(i-1)}) \neq \tilde{X}_{i-1}(r^{(i-1)})$, then proceed to layer $i-1$.

- If layer i is an activation and max pooling layer: Case 1: if there is $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $a\tilde{u}x^*_{i-1}(r^{(i-1)}) = a\tilde{u}x_{i-1}(r^{(i-1)})$, then as $\tilde{X}^*_i(r^{(i)}) \neq \tilde{X}_i(r^{(i)})$, by Theorem A.1 of the GKR protocol, $\mathcal{V}$ rejects with all but negligible probability. Case 2: $\tilde{X}^*_{i-1}(r^{(i-1)}) = \tilde{X}_{i-1}(r^{(i-1)})$ and $a\tilde{u}x^*_{i-1}(r^{(i-1)}) \neq a\tilde{u}x_{i-1}(r^{(i-1)})$, proceed to the input layer. Case 3: $\tilde{X}^*_{i-1}(r^{(i-1)}) \neq \tilde{X}_{i-1}(r^{(i-1)})$, then proceed to layer $i-1$.

Note that in the cases above, at the beginning of each layer i, it is always true that $\tilde{X}^*_i(r^{(i)}) \neq \tilde{X}_i(r^{(i)})$ because of Case 2. In Case 1, $\mathcal{V}$ rejects with overwhelming probability and in Case 3, we proceed directly to the input layer. By the reduction above, at layer 1, either $\tilde{X}^*_1(r^{(i)}) \neq \tilde{X}_1(r^{(i)})$, or $\exists i$ s.t. $\tilde{W}^*_{i-1}(r^{(i-1)}) \neq \tilde{W}_{i-1}(r^{(i-1)})$, or $\exists i$ s.t. $a\tilde{u}x^*_{i-1}(r^{(i-1)}) \neq a\tilde{u}x_{i-1}(r^{(i-1)})$. In Step (4) of Protocol 3, Case 1: $\tilde{in}^*(r_{in}) = \tilde{in}(r_{in})$, then by Lemma 1 of the sumcheck on Equation 14, $\mathcal{V}$ rejects with all but negligible probability. Case 2: $\tilde{in}^*(r_{in}) \neq \tilde{in}(r_{in})$, then as $\tilde{X}_1(r^-)$ is computed by $\mathcal{V}$, either $\tilde{W}^*(r^-) \neq \tilde{W}(r^-)$, or $a\tilde{u}x^*(r^-) \neq a\tilde{u}x(r^-)$. By the soundness of the polynomial commitment in Definition B.2, $\mathcal{V}$ rejects with overwhelming probability in zkPC.Verify. Finally, by the union bound, the probability that $\mathcal{V}$ accepts in all cases above is negligible.

To prove knowledge soundness of Protocol 3, we first extract **W** and **aux** using the extractor of the zkPC. The rest of the proof stays the same.



(a) Prover time

(b) Proof size

(c) Verifier time

Figure 5: Performance of zkCNN on proving accuracy.