vCNN: Verifiable Convolutional Neural Network

Seunghwa Lee
Kookmin University
ttyhgo@kookmin.ac.kr

Hankyung Ko
Hanyang University
hankyungko@hanyang.ac.kr

Jihye Kim
Kookmin University
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University
hoh@hanyang.ac.kr

ABSTRACT

Inference using convolutional neural networks (CNNs) is often outsourced to the cloud for various applications. Hence it is crucial to detect the malfunction or manipulation of the inference results. To provide trustful services, the cloud services should prove that the inference results are correctly calculated with valid input data according to a legitimate model. Particularly, a resource-constrained client would prefer a small proof and fast verification. A pairing-based zero-knowledge Succinct Non-interactive ARgument of Knowledge(zk-SNARK) scheme is a useful cryptographic primitive that satisfies both the short-proof and quick-verification requirements with only black-box access to the models, irrespective of the function complexity. However, they require tremendous efforts for the proof generation. It is impractical to build a proof using traditional zk-SNARK approaches due to many (multiplication) operations in CNNs.

This paper proposes a new efficient verifiable convolution neural network (vCNN) framework, which allows a client to verify the correctness of the inference result rapidly with short evidence provided by an untrusted server. Notably, the proposed vCNNs framework is the first practical pairing-based zk-SNARK scheme for CNNs, and it significantly reduces space and time complexities to generate a proof with providing perfect zero-knowledge and computational knowledge soundness. The experimental results validate the practicality of vCNN with improving VGG16 performance and key size by 18000 fold compared with the existing zk-SNARKs approach (reducing the key size from 1400 TB to 80 GB, and proving time from 10 years to 8 hours).

KEYWORDS

Convolutional Neural Networks, Verifiable Computation, zk-SNARKs

1 INTRODUCTION

Machine learning and neural networks have greatly expanded our understanding of data and the insights it carries. Among these, convolutional neural networks (CNNs), based on the convolution operation, are particularly useful tools for classification and recognition, as compared with standard neural networks, CNNs are easily trained with considerably fewer connections and parameters while providing a better recognition rate. Thus, CNNs generate various business opportunities such as those based on law, banking, insurance, document digitization, healthcare predictive analytics, etc. However, extra caution is required when applying CNNs to real-world applications since they are vulnerable to malfunction or manipulation. For example, a sentence made by an AI based judge may be deliberately altered by an attacker, causing an innocent person to be convicted, or a guilty person to be acquitted.[1] Incorrect results in healthcare prediction and precision medicine using CNNs are even more catastrophic, as the lives of many users depend on them.

This paper focuses on verifying CNN inference, which is often outsourced to the cloud. CNN applications are vulnerable to malicious data inputs, physical attacks, and misconfigurations [10]. Therefore, it is crucial to verify that CNN-inference results were correctly performed on the given input and model. The most straightforward approach to verify a CNN is to re-execute the same operation; however, neither does it save computational burden for the verifier, nor does it help outsource its computation. Even if the verifier has sufficient resources to compute the CNN inference, there is a privacy issue to consider; the verifier will unnecessarily learn potentially important secrets, e.g., weights associated with the model, which are a company's important IP in many applications. To ensure efficient verification while retaining model information confidentiality, we adopt the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), which is a nearly practical cryptographic proof system for achieving computational integrity and privacy protection [14, 17–19, 25, 30]. From the view point of a verifier, pairing-based zk-SNARKs [14, 19, 25] are considered the most practical verifiable computation systems when verifying computation of a general function; the proof size is constant, and verification only requires constant-time group operations irrespective of the size of the function.

Pairing-based zk-SNARKs, however, require significant number of computations on the prover's side. In zk-SNARKs, a function is translated to an arithmetic circuit comprising addition and multiplication gates to be represented as quadratic arithmetic programs (QAPs). Although proving computation of addition gates is almost free, proving computation of multiplication gates requires non-negligible overhead. Thus, zk-SNARKs based on QAPs are inappropriate for multiplication-intensive functions, because the prover may be overwhelmed by the huge amount of computations required. In addition, the size of public parameters containing common reference string (CRS) linearly increases with the number of multiplications. Thus, it is challenging to effectively apply zk-SNARKs to CNNs that include a tremendous number of multiplications. The convolution operation, the core in CNNs, is the dot product of the input vector $\vec{x}$ and the kernel vector $\vec{a}$. If we express the convolution as an arithmetic circuit, the number of multiplications becomes $O(|\vec{x}| \times |\vec{a}|)$. Considering VGG16 [27] as an example, which is a

[1] The Estonian Ministry of Justice plans to build a robot judge authorized to adjudicate small claims disputes of less than 7,000 Euros.

model commonly used for image classification, it is too heavy to be deployed in practice; only for the convolution and pooling layers of VGG16 (excluding fully connected layers), the circuit is more than 6 TB in size, and 90 TB or more memory is required to generate a CRS of size approximately 1400 TB; in addition, the proof computation takes 10 years using the state-of-the-art zk-SNARK in [19]. Because 90% of CNNs resources are used for convolution in CNNs, optimizing the convolution circuit is crucial to devise a practical zk-SNARK scheme for CNNs.

1.1 Main Idea

Optimizing Convolutional Relation: We propose a new efficient QAP formula for convolution that significantly curtails the number of multiplications. Consider the sum of products as a general convolution expression. Since this generates considerable multiplication operations when converted into arithmetic circuits, we adopt a different expression, *a product of sums*, to represent the convolution, with additional refitting to preserve the equality of equations. For instance, consider the following convolution: $y_i = \sum_{j=0}^{l-1} a_j \cdot x_{i+l-1-j}$, where $\vec{x_i} = (x_i, \cdots, x_{i+l-1})$ denotes the i-th input vector, y_i the ith-output, and $\vec{a} = (a_0, \cdots, a_{l-1})$ the kernel vector for $0 \leq i \leq n-l$. Notably, the original equation contains $O(nl)$ number of multiplications. First, we reconstruct the equation into a product of sums with only one multiplication gate as follows: $(\sum_{i=0}^{n-1} x_i) \cdot (\sum_{i=0}^{l-1} a_i) = \sum_{i=0}^{n+l-2} y_i$. Still, this transformation (combining multiple equations into one equation) is not sufficient, as it covers surplus relations; i.e., verification can include the incorrect values of input, kernel, and output.

To safeguard each convolution equation, we must enforce independence in equations. Therefore, we rearrange the equation using *identities* by combining indeterminate Z as follows: $(\sum_{i=0}^{n-1} x_i Z^i) \cdot (\sum_{i=0}^{l-1} a_i Z^i) = \sum_{i=0}^{n+l-2} y_i Z^i$. Consequently, there are $O(n+l)$ identities, and each identity comprises $O(n+l)$ multiplications. Notably, the number of output y_i's increases from $O(n-l)$ to $O(n+l)$. To guarantee the equation correctness using the arithmetic circuit, we need to evaluate $d+1$ different point evaluations for a polynomial of degree d. Because the polynomial evaluation at a point requests $O(n+l)$ operations and there are $O(n+l)$ points, the total computation becomes $O((n+l)^2)$, which is even more expensive than proving the original equation naively. We resolve this problem by adopting a polynomial circuit using the quadratic polynomial program (QPP) in [21], in which a wire is represented as a polynomial. Thus, we can express the revised equation as a single multiplication gate with two input polynomials and one output polynomial.

Connection with ReLU and Pooling: Our newly proposed formula using QPP minimizes a prover's computation only when convolution is verified; however, it is inefficient when the prover proves computation of the whole CNN with other operations such as ReLU or pooling. Polynomial circuits are represented using a single bivariate equation in QPP. Since the division (required to generate a proof) is slow when QPP is expressed as a bivariate polynomial, we convert it to a univariate polynomial by increasing the polynomial degree to utilize the fast division algorithm based on number theoretic transform (NTT). To eliminate one variable, we change it into the form of another variable with a higher degree. However, the substitution of one variable by another incurs excessive overheads in non-convolution operations, such as ReLU and Pooling, thereby amplifying the degree of the equation to $O((|\vec{x}| + |\vec{a}|)^2)$.

Since the intermediates of convolutions and non-convolution operations are independent, it is better to treat those operations separately to avoid mutual effects. In particular, to alleviate the degree increments involving ReLU and Pooling, we apply the polynomial circuit only to the convolution and the arithmetic circuit to the rest part of CNN, and build a connecting proof between QPP and QAP using the commit and prove SNARK (CP-SNARK) technique [8]. The CP-SNARK technique guarantees that QPP and QAP are interconnected with inputs for one component corresponding to outputs from the other. To use this technique, we adopt commit and carry SNARK (cc-SNARK) [8] rather than traditional SNARK for QPP and QAP, as commitments are required for interconnected values with proofs. Figure 1 illustrates the overview of our verifiable convolutional neural network scheme called vCNN. As shown in Figure 1, CNNs are proved by generating (cm_{qpp}, π_{qpp}) from QPP cc-SNARK and (cm_{qap}, π_{qap}) from QAP cc-SNARK, respectively, and then interconnecting the commitments through π_{cp}. Hence, the final proof for our proposed scheme is a tuple of two commitments and three proofs $(cm_{qap}, cm_{qpp}, \pi_{qap}, \pi_{qpp}, \pi_{cp})$. The proposed scheme generates a single proof for QAP and QPP circuits even for multiple layer CNNs, as all the convolution layers are collected and QPP is applied to the collected convolution layer, and QAP is applied to the other collected circuit in a similar manner. See Section 4 for details.

1.2 Contributions

This paper provides several significant contributions as follows.

(1) We propose a new QPP relation optimized for the convolutions and construct an efficient zk-SNARK scheme of which CRS size and proving time are linear with the input size and the kernel size, i.e., $O(n+l)$. The proposed scheme is a verifier-friendly zk-SNARK with constant proof size, and its verification time complexity linearly depends on the input and output only, regardless of convolution intricacy.
(2) We propose vCNN as a practical construction to verify the evaluation of the whole CNN. vCNN combines QPP-based zk-SNARK optimized for convolutions and QAP-based zk-SNARK that effectively works for Pooling and ReLU, and interconnecting them using CP-SNARK.
(3) We prove that vCNN comprising QAP-based SNARK, QPP-based SNARK, and CP-SNARK provides computational knowledge soundness and perfect zero-knowledge properties.
(4) We implement vCNN and compare it with the existing zk-SNARK in terms of size and computation. The proposed scheme improves the key generation/proving time 25 fold and the CRS size 30 fold compared with the state-of-art zk-SNARK scheme [19] for a small example of MNIST (2-layer model) comprising a single convolution layer with ReLU and a single pooling layer. For the realistic application of VGG16, the proposed scheme improves the performance at least 18000 fold, compared with [19]; the proving time is reduced to 8 hours from 10 years, and the CRS size is shortened to 80 GB from 1400 TB. Thus, we provide the first

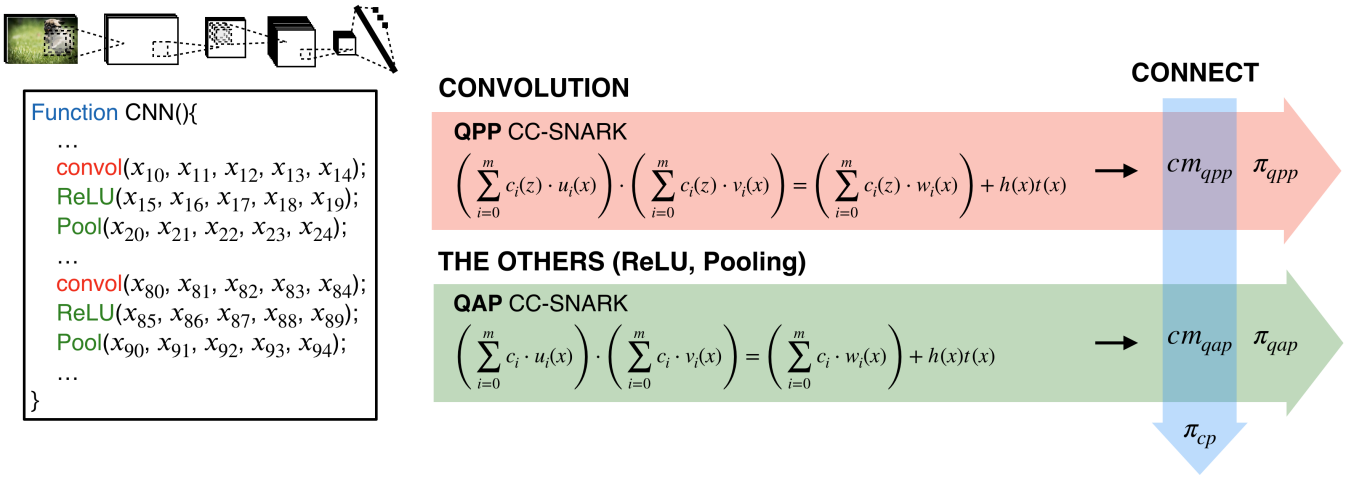


Figure 1: Proposed vCNN overview

practical verifiable convolutional neural network, which has been nearly impossible to realize so far.

1.3 Organization

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 describes preliminaries for the proposed schemes. Section 4 constructs a verifiable CNN scheme using zk-SNARKs and Section 5 represents experiment results. Finally, Section 6 summarizes and concludes the paper. Security proofs are presented in the Appendix.

2 RELATED WORKS

Verifiable Computation. Various cryptographic proof systems [4–6, 11, 14, 18, 19, 21, 25, 30] have been proposed to provide the privacy and computational integrity. These systems have been improved into many forms for the efficiency of their provers and verifiers, and the expressiveness of the statement being proven. Each scheme supports a general function, but it tends to be efficient only for a specific function, so performance issues may occur when applied to an application composed of functions with multiple characteristic.

Goldwasser et al. [18] proposed the GKR protocol, an interactive proof protocol for a general function, where the function was represented as a layered arithmetic circuit, and the circuit was proved using the sum-check protocol. GKR takes $O(S \log S)$ computations for proof generation and $O(d \log S)$ computations for verifying the proof, where S denotes the circuit size and d the circuit depth. Cormode et al. [11] and Thaler [28] subsequently optimized GKR, and Wahby et al. [30] added zero-knowledge property, producing zk-SNARK in the ROM.

In contrast, Gennaro et al. [14] proposed a quadratic arithmetic program (QAP) based zk-SNARK, where QAP is the representation of an arithmetic circuit as a polynomial equation, and the circuit satisfiability is checked using polynomial division. Parno et al. [25] proposed Pinocchio, the first nearly practical QAP-based zk-SNARK scheme with eight group elements for its proof, and implemented zk-SNARK tools. Groth [19] improved Pinocchio with a shorter proof comprising only three group elements.

Other than theoretical developments, many studies have investigated practical zk-SNARK implementations. Libsnark [5, 7] implemented QAP-based zk-SNARKs. Privacy preserving cryptocurrency Zcash [3] utilizes libsnark as a real-world case, and other systems, such as Zokrates and ZSL [2, 12], have also been proposed by implementing zk-SNARKs using libsnark. The zk-SNARK system also requires a front-end compiler that converts a function into a arithmetic circuit. Pinocchio [25] provides a C-compiler that produces arithmetic circuits for its own scheme. Kosba built Jsnark [1] which generates the arithmetic circuit for zk-SNARKs using java language. It provides gadgets that can easily convert conditional statements, loops, and cryptographic schemes such as hashes and encryptions into the arithmetic circuits that are difficult to perform in Pinocchio compiler. He also proposed xjsnark [22] to convert their own high-level language to an arithmetic circuit and optimized it.

Verifiable Neural Networks. To protect the privacy of the input data and model of deep neural networks, Dowlin et al. proposed CryptoNets [16] based on using the fully homomorphic encryption. Juvekar et al. accelerates the overall performance through homomorphic matrix multiplication technique by proposing Gazelle [20]. These schemes based on homomorphic encryption focused on privacy and did not consider execution integrity. Slalom [29] was proposed as a verifiable neural network scheme using a trusted hardware, Intel SGX. It uses Freivalds' algorithm [13] on SGX which verifies the correctness of matrix multiplication. Since the inputs and outputs are exposed to use the algorithm, Slalom adds random values to protect the privacy of the inputs and outputs. However, Slalom aims to provide the privacy of the inputs and outputs, and it does not focus on the privacy of the model.

Even though zk-SNARKs are generally applicable for CNNs, they are not very efficient for some functions, particularly convolutions. Ghodsi et al. [15] proposed SafetyNet, the first SNARK-based scheme supporting neural networks specifically. SafetyNet is based on the GKR protocol [18], which is suitable for linear functions.

Table 1: Verifiable neural network scheme security coverage and performance , where $\vec{x}$ denotes the input, $\vec{a}$ the kernel, and $\vec{y}$ the output.

Approach	Privacy	Integrity	Activation function	Proving time	Proof size	Verifying time
Gazelle [20]	O	X	ReLU	-	-	-
SafetyNet [15]	X	O	Quadratic	$\|\vec{a}\| \cdot \|\vec{x}\| + \|\vec{y}\|$	$\|\vec{a}\| \cdot \|\vec{x}\| + \|\vec{y}\|$	$\|\vec{a}\| \cdot \|\vec{x}\| + \|\vec{y}\|$
VeriML [31]	O	O	Quadratic	$\|\vec{a}\| \cdot \|\vec{x}\| + \|\vec{y}\|$	1	$\|\vec{x}\| + \|\vec{y}\|$
Embedded proof [9]	O	O	ReLU	$\|\vec{a}\| \cdot \|\vec{x}\| + \|\vec{y}\|$	1	$\|\vec{x}\| + \|\vec{y}\|$
vCNN (ours)	O	O	ReLU	$\|\vec{a}\| + \|\vec{x}\| + \|\vec{y}\|$	1	$\|\vec{x}\| + \|\vec{y}\|$

However, to effectively use this advantage, it adopts a quadratic activation function (x^2) rather than ReLU, which reduces the neural network accuracy. Thus, it is difficult to apply SafetyNet to actual models, since most modern neural networks use ReLU. Zhao et al. proposed VeriML [31] to verify neural networks using QAP-based zk-SNARK for machine learning as a service (MLaaS). Although VeriML ensures both privacy and integrity, it requires a long proving time, ($O(|\vec{a}| \cdot |\vec{x}| + |\vec{y}|)$), where $\vec{x}$ denotes the input, $\vec{a}$ the kernel, and $\vec{y}$ the output.

Chabanne [9] proposed an embedded proofs protocol that combines the GKR and QAP schemes, using GKR for linear and QAP for non-linear functions. To combine them, the verifying process of GKR is verified in the QAP circuit. However, it still has large computation complexity of ($O(|\vec{a}| \cdot |\vec{x}| + |\vec{y}|)$), as the input($\vec{x}$) and kernel($\vec{a}$) sizes are significantly large in real applications. So far, to the best of our knowledge, there is no practical solution which supports the model privacy and the integrity of execution.

3 PRELIMINARIES

First, we define some notations to avoid duplicate words. The term $[n]$ denotes the set of indices $\{0, 1, \ldots, n-1\}$. The input of convolution is represented as $\{x_i\}_{i\in[n]}$ where the input size is n and the kernel of convolution is represented as $\{a_i\}_{i\in[l]}$ where the kernel size is l.

3.1 Bilinear groups

We use a Type III bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$ with the following properties:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generator $G_1 \in \mathbb{G}_1, G_2 \in \mathbb{G}_2$.
- The pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map.
- $e(G_1, G_2)$ generates $\mathbb{G}_T$.

3.2 Quadratic Arithmetic Program

Gennaro et al. [14] defined QAP as an efficient encoding method for circuit satisfiability. QAP represents an arithmetic circuit that encodes the constraints into the multiplication gates. The correctness of the computation can be tested using QAP by performing a divisibility check between polynomials. A cryptographic protocol enables to check divisibility for a single polynomial and prevents a cheating prover from building a proof for a false statement that might be accepted.

Definition 3.1. **Quadratic Arithmetic Program (QAP)** A QAP comprises three sets of polynomials $\{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}$ and a target polynomial $t(X)$. The QAP computes an arithmetic circuit if $(c_1, \ldots, c_{l-1})$ are valid assignments of both the inputs and outputs for the circuit iff there exist coefficients $(c_l, \ldots, c_m)$ such that $t(X)$ divides $p(X)$, as follows:

$$p(X) = (\Sigma_{i=1}^{m} c_i \cdot u_i(X)) \cdot (\Sigma_{i=1}^{m} c_i \cdot v_i(X)) - (\Sigma_{i=1}^{m} c_i \cdot w_k(X))$$

A QAP that satisfies the aforementioned definition computes an arithmetic circuit. The size of QAP is m and its degree is the degree of $t(X)$.

In the above-mentioned definition, $t(X) = \prod_{i\in mul}(x - r_i)$, where mul is the set of multiplication gates of the arithmetic circuit and each r_j is a random labeling for corresponding multiplication gate. The polynomial $u_i(X)$ encodes the left inputs, $v_i(X)$ encodes the right inputs, and $w_i(X)$ encodes the gate outputs. By definition, if r_j is a root for polynomial $p(X)$, $p(r_j)$ represents the relation between inputs and outputs for the corresponding multiplicative gate g.

3.3 Quadratic Polynomial Program

QAP verifies wires that are represented as an arithmetic value in an arithmetic circuit. Kosba et al. [21] subsequently defined the quadratic polynomial program (QPP), similar to QAP, except circuit wires that can be represented as a univariate polynomial.

Definition 3.2. **Quadratic Polynomial Program(QPP)** A QPP for a polynomial circuit comprises three sets of polynomials $\{u_i(X), v_i(X), w_i(X)\}_{i=1}^{m}$ and a target polynomial $t(X)$. The QPP computes the circuit if $(c_1(Z), \ldots, c_l(Z))$ are valid assignments of both the inputs and outputs iff there exist coefficients $(c_{l+1}, \ldots, c_m)$ such that $t(X)$ divides $p(X, Z)$:

$$\begin{aligned} p(X, Z) = (\Sigma_{i=1}^{m} c_i(Z) \cdot u_i(X)) \cdot (\Sigma_{i=1}^{m} c_i(Z) \cdot v_i(X)) \\ - (\Sigma_{i=1}^{m} c_i(Z) \cdot w_k(X)) \end{aligned} \tag{1}$$

A QPP that satisfies this definition computes the circuit. The size of QPP is m and its degree is the degree of $t(X)$.

Similarly to the QAP definition, $u_i(X)$, $v_i(X)$, and $w_i(X)$ represent a gate, where $u_i(X)$ encodes a left input, $v_i(X)$ a right input, and $w_i(X)$ an output. If the left input wire of a multiplication gate r_j is $c_l(Z)$, then the right wire is $c_r(Z)$ and the output is $c_o(Z)$; hence $c_l(Z) \cdot c_r(Z) = c_o(Z)$ and it can be represented as $(\sum_{i=1}^{m} c_i(Z) \cdot u_i(r_j))(\sum_{i=1}^{m} c_i(Z) \cdot v_i(r_j)) = (\sum_{i=1}^{m} c_i(Z) \cdot w_i(r_j))$.

3.4 Zero-Knowledge Succinct Non-interactive Arguments of Knowledge

In this section, we recall the zk-SNARKs definition [19, 25].

Definition 3.3. A zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) scheme for a relation R is the quadruple of PPT algorithms (KeyGen, Prove, Verify, Sim) as follows.

- $(crs, \tau) \leftarrow \mathsf{Setup}(R)$: The setup algorithm takes a relation $R \in \mathcal{R}_\lambda$ as input, and returns a common reference string crs and a simulation trapdoor td.
- $\pi \leftarrow \mathsf{Prove}(crs, \phi, w)$: The prover algorithm takes a crs for a relation R and $(\phi, w) \in R$ as input, and returns a proof π.
- $0/1 \leftarrow \mathsf{Verify}(crs, \phi, \pi)$: the verifier algorithm takes a crs, a statement ϕ, and a proof π as input, and returns 0(reject) or 1(accept).
- $\pi \leftarrow \mathsf{Sim}(crs, td, \phi)$: The simulator algorithm takes a crs, a simulation trapdoor td, and a statement ϕ as input, and returns a proof π.

Completeness: An argument is complete if given true statement ϕ, a prover with a witness can convince the verifier. For all $(\phi, w) \in R$, the probability of completeness is:

$$Pr\left[\mathsf{Verify}(crs, \phi, \pi) = 1 \middle| \begin{matrix} (crs, td) \leftarrow \mathsf{Setup}(R), \\ \pi \leftarrow \mathsf{Prove}(crs, \phi, w) \end{matrix} \right] = 1$$

Computational knowledge soundness: An argument is computational knowledge sound if the prover must know a witness and such knowledge can be efficiently extracted from the prover by using a knowledge extractor. Proof of knowledge requires that for a PPT adversary $\mathcal{A}$ generating an accepting proof, there must be an extractor $\chi_\mathcal{A}$ that, given the same input of $\mathcal{A}$, outputs a valid witness such that

$$Pr\left[\begin{matrix} \mathsf{Verify}(crs, \phi, \pi) = 1 \\ \wedge (\phi, w) \notin R \end{matrix} \middle| \begin{matrix} (crs, td) \leftarrow \mathsf{Setup}(R), \\ (\phi, \pi, w) \leftarrow (\mathcal{A} | \chi_\mathcal{A})(R, crs, z) \end{matrix} \right] \approx 0$$

where z is auxiliary input.

Succinctness: The length of a proof is

$$|\pi| \leq \mathsf{poly}(k)\mathsf{poly}log(|x| + |w|)$$

.

Perfect zero-knowledge: An argument is zero-knowledge if it does not leak any information other than the truth of the statement. Notably, zk-SNARK is perfect zero-knowledge if for all $(R, z) \leftarrow \mathcal{R}$, $(\phi, w) \leftarrow R$ and all adversaries $\mathcal{A}$, one has the following:

$$Pr\left[\mathcal{A}(R, z, crs, td, \pi) = 1 \middle| \begin{matrix} (crs, td) \leftarrow \mathsf{Setup}(R), \\ \pi \leftarrow \mathsf{Prove}(crs, \phi, w) \end{matrix} \right]$$
$$= Pr\left[\mathcal{A}(R, z, crs, td, \pi) = 1 \middle| \begin{matrix} (crs, td) \leftarrow \mathsf{Setup}(R), \\ \pi \leftarrow \mathsf{Sim}(crs, td, \phi) \end{matrix} \right]$$

3.5 Commit and Prove SNARKs

The commit and prove SNARKs (CP-SNARKs) scheme [8] is a zk-SNARKs scheme to prove the knowledge of (ϕ, w) such that u is a message of commitment cm and a relation $R(\phi, w) = 1$ where the witness $u \in w$.

Definition 3.4. A CP-SNARKs scheme includes the quadruple PPT algorithms (KeyGen, Prove, Verify, Sim) defined as follows.

- $(crs, td) \leftarrow \mathsf{Setup}(ck, R)$: The setup algorithm takes a relation $R \in \mathcal{R}_\lambda$ and commitment key ck as input, and returns a common reference string crs and a trapdoor td.
- $\pi \leftarrow \mathsf{Prove}(crs, \phi, \{c_j, u_j, o_j\}_{j=1}^{l}, w)$: The prover algorithm takes as input a crs for a relation R, $(\phi, w) \in R$, commitments c_j, inputs u_j and opening o_j, and returns a proof π.
- $0/1 \leftarrow \mathsf{Verify}(crs, \phi, \{c_j\}_{j=1}^{l}, \pi)$: The verifier algorithm takes as input a crs, a statement ϕ, commitments c_j and a proof π, and returns 0 (reject) or 1 (accept).
- $\pi \leftarrow \mathsf{Sim}(crs, td, \phi, \{c_j\}_{j=1}^{l})$: The simulator algorithm takes a crs, a trapdoor td, a statement ϕ, and commitments c_j as input, and returns a proof π.

3.6 Commit and Carry SNARKs

Similar to the case of CP-SNARKs, the commit and carry SNARKs (cc-SNARKs) scheme [8] proves a relation with commitment, but it generates a commitment while proving the relation.

Definition 3.5. The cc-SNARKs scheme has the quintuple of PPT algorithms (KeyGen, Prove, Verify, VerifyCom, Sim) defined as follows.

- $(ck, crs, td) \leftarrow \mathsf{Setup}(R)$: The setup algorithm takes as input a relation $R \in \mathcal{R}_\lambda$, and returns a commitment key ck, a crs, and a simulation trapdoor td.
- $(cm, \pi, r) \leftarrow \mathsf{Prove}(crs, \phi, w)$: The prover algorithm takes as a crs for a relation R and $(\phi, w) \in R$, and returns a commitment cm, a proof π, and an opening r.
- $0/1 \leftarrow \mathsf{Verify}(crs, \phi, cm, \pi)$: The verifier algorithm takes as input a crs, a statement ϕ, commitments cm and a proof π, and returns 0(reject) or 1(accept).
- $0/1 \leftarrow \mathsf{VerifyCom}(ck, cm, u, r)$: The verifier algorithm takes as input a commitment key ck, a commitments cm, a message u, and an opening r, and returns 0(reject) or 1(accept).
- $(cm, \pi) \leftarrow \mathsf{Sim}(crs, td, \phi)$: The simulator algorithm takes as a crs, a simulation trapdoor td, and a statement ϕ, and returns a commitment cm and a proof π.

4 VERIFIABLE CONVOLUTIONAL NEURAL NETWORK

This section constructs Verifiable Convolutional Neural Network (vCNN) scheme to prove CNNs efficiently, where it is significantly expensive to prove CNN evaluations in traditional QAP-based zk-SNARKs. Convolution computations deteriorate the proving performance severely, since it requires more than 90% of total proof generation time in CNNs. First, we optimize the convolution relation utilizing QPP [21] and construct an efficient QPP-based zk-SNARKs scheme for convolutions. Although the QPP approach improves

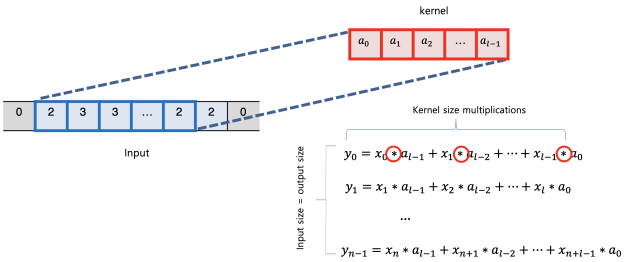


Figure 2: Illustration of convolution

convolution performance, QPP representation of a whole CNN degrades the performance due to the other CNN components, such as ReLU and Pooling. Hence, we propose a new efficient zk-SNARK framework for CNNs by applying QPP to convolutions and QAP to the other components, and we build a connecting proof between QPP and QAP by using CP-SNARKs technique [8].

4.1 Optimizing Convolution Relation

The convolution filters inputs using kernels by computing the inner product for inputs and kernels, as depicted in Figure 2. Thus, convolution can be expressed as

$$y_i = \sum_{j\in[l]} a_j \cdot x_{i-j+l-1} \tag{2}$$

for $i \in [n]$ where $\{a_j\}_{j\in[l]}$ are convolution kernels, $\{x_i\}_{i\in[n]}$ are convolution inputs, and $\{y_i\}_{i\in[n-l]}$ are convolution outputs. When the convolution is represented as QAP, $n\times l$ multiplication gates are required, since there are n outputs and l multiplications per output. Figure 3 shows a small convolution example, where input size is 5, kernel size is 3, and output size is 3, hence the QAP requires 9 multiplication gates.

$$\sum_{i\in[n+l-1]} y_i' = (\sum_{i\in[n]} x_i) \cdot (\sum_{i\in[l]} a_i) \tag{3}$$

Since Equation (2) is the sum of products, which requires many multiplication gates, we transform it into the product of sums as shown in Equation (3) which includes a single multiplication gate to reduce the number of multiplications. However, the naive transformation is not sound, as it is easy to find the incorrect output $\vec{y}'$ which is different from the correct output $\vec{y}$ such that sums of two outputs are equivalent. Therefore, to distinguish each output y_i, we introduce an indeterminate variable Z for each equation as shown in Equation (4) which has $O(|\vec{x}| + |\vec{a}|)(= O(n + l))$ multiplications.

$$\sum_{i\in[n+l-1]} y_i \cdot Z^i = (\sum_{i\in[n]} x_i \cdot Z^i) \cdot (\sum_{i\in[l]} a_i \cdot Z^i) \tag{4}$$

Figure 4 unrolls the Equation (4). Notably, the transformation slightly increases the number of outputs by $2l - 2$ from that in the original Equation (2) with n outputs.

To formulate Equation (4), we can devise two approaches: a point evaluation approach and a polynomial circuit with an indeterminate variable. In the point evaluation approach, for a polynomial of degree d, $d+1$ different points should be evaluated, requiring $O(d^2)$ (multiplicative) operations since there are d multiplications per

	x_0	x_1	x_2	x_3	x_4	Equations
a_2	y_{00}	y_{10}	y_{20}			$y_0 = a_2x_0 + a_1x_1 + a_0x_2$
a_1		y_{01}	y_{11}	y_{21}		$y_1 = a_2x_1 + a_1x_2 + a_0x_3$
a_0			y_{02}	y_{12}	y_{22}	$y_2 = a_2x_2 + a_1x_4 + a_0x_4$

9 multiplications

Figure 3: Example of convolution

$$(a_0 + a_1 + a_2 \cdot Z^2) \cdot (x_0 + x_1 \cdot Z + x_2 \cdot Z^2 + x_3 \cdot Z^3 + x_4 \cdot Z^4) = (y_0 + y_1 \cdot Z + y_2 \cdot Z^2 + dummies)$$

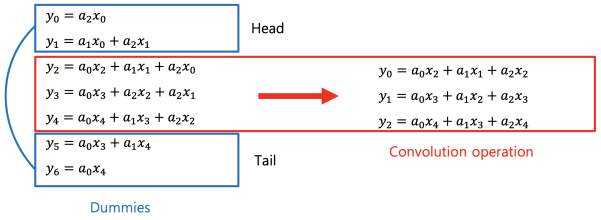


Figure 4: Example of Equation (4)

point evaluation and there are $d + 1$ points. Point evaluation can be performed using number theoretic transform (NTT) in $O(d \log d)$. However, due to the NTT complexity, the computation overhead in NTT is severer than the naive point evaluation, unless d is large enough.

In a polynomial circuit (called Quadratic Polynomial Program (QPP) [21]) a wire can have a polynomial as value. Thus, we can directly express the revised equation as a single multiplication gate with two input polynomials and one output polynomial. While the point evaluation approach requires quadratic $O(d^2)$ or quasi-linear $O(d \log d)$ multiplication operations, the QPP approach requests $O(d)$ operations. Therefore, this paper adopts QPP representation for convolution.

Construction of QPP-based zk-SNARK: We now construct a QPP-based zk-SNARK scheme to prove Equation (4), similar to [21], except we utilize Groth16 [19] rather than the Pinocchio scheme [25]. While each wire can have only a value in QAP, QPP allows each wire to have a polynomial. The proposed concrete QPP-based zk-SNARK scheme is as follows.

$(crs, td) \leftarrow \text{Setup}(R)$: Pick α, β, γ, δ, x, $z \xleftarrow{\$} \mathbb{Z}_p^*$. Define td=(α, β, γ, δ, x, z) and set

$$crs = \begin{pmatrix} G_1^{\alpha}, G_1^{\beta}, G_1^{\delta}, \{G_1^{x^i \cdot z^j}\}_{i=0,j=0}^{d_x-1,d_z}, \\ G_2^{\beta}, G_2^{\gamma}, G_2^{\delta}, \{G_2^{x^i \cdot z^j}\}_{i=0,j=0}^{d_x-1,d_z}, \\ \{G_1^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma} z^j}\}_{i=0,j=0}^{l,d_z}, \\ \{G_1^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta} z^j}\}_{i=l+1,j=0}^{m,d_z}, \\ \{G_1^{\frac{x^i \cdot z^j \cdot t(x)}{\delta}}\}_{i=0,j=0}^{d_x-2,d_z} \end{pmatrix}$$

$\pi \leftarrow \text{Prove}(crs, \phi, w)$: Parse ϕ as $(a_0(Z), a_1(Z), \ldots, a_l(Z))$ and w as $(a_{l+1}(Z), \ldots, a_m(Z))$. Use the witness to compute $h(X, Z)$ from

the QPP. Choose $r, s \xleftarrow{\$} \mathbb{Z}_p^*$ and output a proof $\pi = (G_1^a, G_2^b, G_1^c)$ such that

$$a = \alpha + \sum_{i=0}^{m} a_i(z)u_i(x) + r\delta \qquad b = \beta + \sum_{i=0}^{m} a_i(z)v_i(x) + s\delta$$
$$c = \frac{\Sigma_{i=l+1}^{m} a_i(z) \cdot (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x,z)t(x)}{\delta} + as + rb - rs\delta$$

$0/1 \leftarrow \mathsf{Verify}(crs, \phi, \pi)$: Parse the statement ϕ as $(a_0(Z), a_1(Z), \ldots, a_l(Z))$ and the proof π as (A, B, C). Accept the proof if and only if the following equation is satisfied:

$$e(A,B) = e(G_1^{\alpha}, G_2^{\beta}) \cdot e(\prod_{i=0}^{l} G_1^{a_i(z) \cdot \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}}, G_2^{\gamma}) \cdot e(C, G_2^{\delta})$$

$\pi \leftarrow \mathsf{Sim}(\tau, \phi)$: Pick $a, b \xleftarrow{\$} \mathbb{Z}_p^*$ and compute a simulated proof $\pi = (G_1^a, G_2^b, G_1^c)$ with

$$c = \frac{ab - \alpha\beta - \Sigma_{i=0}^{l} a_i(z)(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}$$

Theorem 4.1. *The above protocol is a non-interactive zero-knowledge arguments of knowledge with completeness and perfect zero-knowledge. It has computational knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations.*

The proposed QPP-based zk-SNARK has the same construction as that of the original QAP-based zk-SNARK except that in the former the terms in CRS include unknown value z to generate a polynomial $f(Z)$. We prove the knowledge soundness for the proposed scheme in Appendix A.1.

Implementation challenge: To prove convolution using Equation (4), a prover computes $h(X, Z)$ by performing polynomial division $(p(X, Z)/t(X))$ for Equation (1). Although the polynomial division can be efficiently performed using NTT for univariate polynomials, NTT is not directly applicable for the bivariate polynomials in QPP. Therefore, we transform bivariate polynomials to univariate polynomials. In QPP, the degree of X in $p(X, Z)$ is $2d_x - 2$, where d_x is the number of multiplication gates. Therefore, by setting $Z = X^{2d_x - 1}$, all terms can be distinct and the degree of $p(X, X^{2d_x-1})$ is $(2d_x - 1)d_z$ where d_z is the maximum degree of Z. Since there is one multiplication in Equation (4), and maximum degree of Z is $n + l - 1$, the degree of $p(X, Z)$ becomes $n + l - 1$. Although converting bivariate polynomials to univariate polynomials increases the equation degree, it is significantly more efficient than QAP based approaches.

Although the total performance is expected to increase significantly since QPP improves convolution proving time dramatically, the actual performance for CNNs is not improved. Even if no Z variable is required in ReLU and Pooling, the transformation of bivariate polynomials to univariate polynomials increases the degree of X, which populates unnecessary terms. The following subsection tackles this problem.

4.2 Connection with ReLU and Pooling

To solve the above problem, QPP is applied only to convolution while QAP is utilized for the other CNN modules, i.e., ReLU and Pooling. To guarantee consistency between the QAP-based ReLU and Pooling circuits and QPP-based convolution circuits, we adopt CP-SNARKs [8].

Construction of commit and prove SNARKs: A commit and prove SNARKs (CP-SNARKs) scheme is a proof system to prove that multiple Pedersen-like commitments are constructed on the same input. We refer to the scheme in LegoSNARK's Appendix. D [8]. Setup takes two commitment keys, ck and ck' as inputs and combines them to generate CRS. Prove creates a new proof π in which the commitments are combined. If commitments c and c' were made using the same input, proof π passes verification.

$$R_{cp} = \left\{ \phi = (c, c'), w = (r, r', \vec{u}) \;\middle|\; \begin{matrix} c = commit(r, \vec{u}) \\ \wedge c' = commit(r', \vec{u}) \end{matrix} \right\}$$

$(crs, td) \leftarrow \mathsf{Setup}(R_{cp}, ck, ck')$: parse $ck = \{G_1^{h_i}\}_{i=0}^{l}, ck' = \{G_1^{f_i}\}_{i=0}^{l}$. Pick $k_1, k_2, a \xleftarrow{\$} \mathbb{Z}_p$ and set $crs = (G_1^{k_1 \cdot h_0}, G_1^{k_2 \cdot f_0}, \{G_1^{k_1 \cdot h_i + k_2 \cdot f_i}\}_{i=1}^{l}, G_2^{ak_1}, G_2^{ak_2}, G_2^{a})$ and trapdoor $td = (k_1, k_2)$.

$\pi \leftarrow \mathsf{Prove}(crs, \phi, w)$: parse $r, r', \{u_i\}_{i=1}^{l} \in w$ and $(A, B, \{C_i\}_{i=1}^{l}, vk_1, vk_2, vk_3) \in crs$. Compute π as

$$\pi = A^r \cdot B^{r'} \cdot \prod_{i=1}^{l} C_i^{u_i} \tag{5}$$

$1/0 \leftarrow \mathsf{Verify}(crs, \phi, \pi)$: parse $c, c' \in \phi$ and $(A, B, \{C_i\}_{i=1}^{l}, vk_1, vk_2, vk_3) \in crs$. Accept the proof iff the following equation is satisfied:

$$e(c, vk_1) \cdot e(c', vk_2) = e(\pi, vk_3)$$

$\pi \leftarrow \mathsf{Sim}(crs, td, \phi)$: parse $k_1, k_2 \in td$ and $c, c' \in \phi$. Compute a proof π as

$$\pi = c^{k_1} \cdot c'^{k_2}$$

Construction of cc-SNARKs from zk-SNARKs: To connect the zk-SNARKs proofs with CP-SNARKs, we need commitments for inputs as well as the proofs. Therefore, we modify the zk-SNARKs scheme in subsection 4.1 to produce a cc-SNARKs scheme that generates a commitment of the wires with a proof, similar to LegoSNARKs [8]. Since the verification in zk-SNARKs includes a form of Pedersen-like commitment as

$$\prod_{i=0}^{l} G_1^{a_i(z) \cdot \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}} = \prod_{i \in [l], j \in [d_z+1]} \left(G_1^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} \cdot z^j} \right)^{a_{i,j}}$$

it can be delegated to the prover, and hence we can create a proof system that carries the commitment. Setup adds a commitment key $G_1^{\frac{\eta}{\gamma}}$ and additional random $G_1^{\frac{\eta}{\delta}}$ to the CRS. Prove additionally generates a commitment G_1^d, and we add the $-\nu\frac{\eta}{\delta}$ term to c to cancel out the random part of the commitment during verification. Verify takes cm as input and verifies proof π. Finally, there is a new algorithm VerifyCom, which verifies the commitment cm. The modified algorithms are as follows.

$(cm, \pi, v) \leftarrow \mathsf{Prove}(crs, \phi, w)$: Parse ϕ as $(a_0(Z), a_1(Z), \ldots, a_l(Z))$ and w as $(a_{l+1}(Z), \ldots, a_m(Z))$. Use the witness to compute $h(X, Z)$ from the QPP. Choose $r, s, v \xleftarrow{\$} \mathbb{Z}_p^*$ and output a random v, a commitment $cm = G_1^d$, and a proof $\pi = (G_1^a, G_2^b, G_1^c)$ such that

$$a = \alpha + \sum_{i=0}^{m} a_i(z)u_i(x) + r\delta \qquad b = \beta + \sum_{i=0}^{m} a_i(z)v_i(x) + s\delta$$

$$c = \frac{\sum_{i=l+1}^{m} a_i(z) \cdot (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x,z)t(x)}{\delta} + As + rB - rs\delta - v\frac{\eta}{\delta}$$

$$d = \frac{\sum_{i=0}^{l} a_i(z) \cdot (\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma} + v\frac{\eta}{\gamma}$$

$0/1 \leftarrow \mathsf{Verify}(crs, \phi, cm, \pi)$: Parse the proof ϕ as $(a_0(Z), a_1(Z), \ldots, a_l(Z))$ and π as (A, B, C). Accept the proof iff the following equation is satisfied:

$$e(A, B) = e(G_1^{\alpha}, G_2^{\beta}) \cdot e(cm, G_2^{\gamma}) \cdot e(C, G_2^{\delta})$$

$0/1 \leftarrow \mathsf{VerifyCom}(ck, w, r, cm)$: Parse the message u in w. Accept the proof iff the following equation is satisfied:

$$cm = (r, \vec{u}) \cdot ck$$

$(v, cm, \pi) \leftarrow \mathsf{Sim}(\tau, \phi)$: Pick $a, b, v \xleftarrow{\$} \mathbb{Z}_p^*$ and compute a simulated commitment $cm = G_1^d$ and simulated proof $\pi = (G_1^a, G_2^b, G_1^c)$ with

$$c = \frac{ab - \alpha\beta - \sum_{i=0}^{l} a_i(z)(\beta u_i(x) + \alpha v_i(x) + w_i(x)) - v\eta}{\delta}$$

$$d = \frac{\Sigma_{i=0}^{l} a_i(z)(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + v\eta}{\gamma}$$

Theorem 4.2. *The protocol given above is a non-interactive zero-knowledge arguments of knowledge with completeness and perfect zero-knowledge. It has computational knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations.*

The proof for Theorem 4.2 is available in Appendix A.1. We omit the concrete construction and security proof for the QAP-based cc-SNARKs here since it is a special case of the QPP-based cc-SNARKs; the degree of Z is zero.

4.3 Construction of Verifiable Convolutional Neural Network

The proposed vCNN proves CNNs using cc-SNARKs and CP-SNARKs. The relation of CNNs, R_{CNN}, comprises R_{convol}, $R_{ReLU+Pool}$, and R_{cp}, where R_{convol} is encoded in QPP containing Z and $R_{ReLU+Pool}$ is in QAP. Let Π_{qap} = (Setup, Prove, Verify, VerifyCom, Sim) be a QAP-based cc-SNARKs scheme, Π_{qpp} = (Setup, Prove, Verify, VerifyCom, Sim) be a QPP-based cc-SNARKs scheme, and Π_{cp} = (Setup, Prove, Verify, Sim) be a CP-SNARKs scheme.

$(crs, td) \leftarrow \mathsf{Setup}(R_{CNN})$: Parse R_{CNN} as relation of convolution R_{convol}, and ReLU and Pooling $R_{ReLU+Pool}$. Compute common reference string crs and trapdoor td as follows:

$$ck_{qap}, crs_{qap}, td_{qap} \leftarrow \Pi_{qap}.\mathsf{Setup}(R_{ReLU+Pool})$$
$$ck_{qpp}, crs_{qpp}, td_{qpp} \leftarrow \Pi_{qpp}.\mathsf{Setup}(R_{conv})$$
$$crs_{cp}, td_{cp} \leftarrow \Pi_{cp}.\mathsf{Setup}(ck_{qap}, ck_{qpp})$$

Set $crs = (crs_{qap}, crs_{qpp}, crs_{cp})$ and $td = (td_{qap}, td_{qpp}, td_{cp})$.

$\pi \leftarrow \mathsf{Prove}(crs, \phi, w)$: Parse (ϕ, w) as (ϕ_{qap}, w_{qap}) and (ϕ_{qpp}, w_{qpp}). Parse crs as $(crs_{qap}, crs_{qpp}, crs_{cp})$. Compute a proof as follows:

$$\pi_{qap}, r_{qap}, cm_{qap} \leftarrow \Pi_{qap}.\mathsf{Prove}(crs_{qap}, \phi_{qap}, w_{qap})$$
$$\pi_{qpp}, r_{qpp}, cm_{qpp} \leftarrow \Pi_{qpp}.\mathsf{Prove}(crs_{qpp}, \phi_{qpp}, w_{qpp})$$
$$parse \quad \pi_{qap} = (A_{qap}, B_{qap}, C_{qap})$$
$$parse \quad \pi_{qpp} = (A_{qpp}, B_{qpp}, C_{qpp})$$
$$\phi_{cp} = (cm_{qap}, cm_{qpp})$$
$$w_{cp} = (r_{qap}, \vec{y}, r_{qpp}, \vec{y'})$$
$$\pi_{cp} \leftarrow \Pi_{cp}.\mathsf{Prove}(crs_{cp}, \phi_{cp}, w_{cp})$$

Set $\pi = (\pi_{qap}, \pi_{qpp}, \pi_{cp}, cm_{qap}, cm_{qpp})$.

$0/1 \leftarrow \mathsf{Verify}(R_{CNN}, crs, \phi, \pi)$: Parse $\phi = (\phi_{qap}, \phi_{qpp})$. Parse crs as $(crs_{qap}, crs_{qpp}, crs_{cp})$ and π as $(\pi_{qap}, \pi_{qpp}, \pi_{cp}, cm_{qap}, cm_{qpp})$. And parse $\pi_{qap} = (A_{qap}, B_{qap}, C_{qap})$ and $\pi_{qpp} = (A_{qpp}, B_{qpp}, C_{qpp})$. Accept the proof iff the following equation is satisfied:

$$assert \quad \Pi_{qap}.\mathsf{Verify}(crs_{qap}, \phi_{qap}, cm_{qap}, \pi_{qap}) = 1$$
$$assert \quad \Pi_{qpp}.\mathsf{Verify}(crs_{qpp}, \phi_{qpp}, cm_{qpp}, \pi_{qpp}) = 1$$
$$assert \quad \Pi_{cp}.\mathsf{Verify}(crs_{cp}, (cm_{qap}, cm_{qpp}), \pi_{cp}) = 1$$

$\pi \leftarrow \mathsf{Sim}(crs, \tau, \phi)$:Parse ϕ=(ϕ_{qap},ϕ_{qpp}) and $td = (td_{qap}, td_{qpp}, td_{cp})$. Compute a proof π as follows:

$$cm_{qap}, \pi_{qap} \leftarrow \Pi_{qap}.\mathsf{Sim}(crs_{qap}, td_{qap}, \phi_{qap})$$
$$cm_{qpp}, \pi_{qpp} \leftarrow \Pi_{qpp}.\mathsf{Sim}(crs_{qpp}, td_{qpp}, \phi_{qpp})$$
$$\phi_{cp} = (cm_{qap}, cm_{qpp})$$
$$\pi_{cp} \leftarrow \Pi_{cp}.\mathsf{Sim}(crs_{cp}, td_{cp}, \phi_{cp})$$

Set $\pi = (\pi_{qap}, \pi_{qpp}, \pi_{cp}, cm_{qap}, cm_{qpp})$.

Theorem 4.3. *If Π_{qap}, Π_{qpp}, and Π_{cp} are computationally knowledge sound and perfect zero-knowledge, then the protocol given above is a non-interactive zero-knowledge arguments of knowledge with completeness and perfect zero-knowledge. It has computational knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations.*

The proposed vCNN scheme generates a constant size proof regardless of the number of layers in the neural network models. Note that since the constraint relations are checked in proof systems, the computation order can be ignored. Therefore, we can build proofs for QPP and QAP at once using given values without iterating layers. Consequently, the proposed vCNN generates 9 group elements as proof; three for QAP, three for QPP, two for commitment, and one for CP-SNARKs.

5 EXPERIMENT

This section describes the implementation of vCNN, and compares the prove time and the CRS size in vCNN with existing QAP-based zk-SNARKs scheme [19]. As real applications, we utilize LeNet-5 [24], AlexNet [23], and VGG16 [27] models. We execute them on a Quad-core Intel CPU i5 3.4 GHz and Ubuntu 16.04.

We implement the proposed QPP-based SNARKs scheme by utilizing libsnark and jsnark [1, 4, 5]. First, we build a generic convolution circuit operation in jsnark, as follows.

> "convol in #input < wire numbers > out #output < wire numbers > state #state < input size, kernel size>"

The circuit operation contains input and output wires. Since a convolution takes kernels as input, keyword "state" is appended to specify the size of input and kernel. And then, we add code for reading "convol" operations and constructing QPP polynomials for convolutions in the library.

5.1 Convolutions

We compare the prove performance in the proposed QPP-based zk-SNARKs scheme with the QAP-based zk-SNARKs scheme [19] for convolution. Figure 5 shows the setup and proof generation time, and Figure 6 shows CRS size by varying the convolution input size for given kernel size. Figures 5 and 6 show that the proposed QPP-based scheme provides higher proving performance and smaller CRS size where the improvement increases as the kernel size increases.

5.2 Convolutional Neural Networks

We compare the proposed vCNN scheme with the QAP-based zk-SNARKs scheme on various deep neural models from small CNNs to real large models, to demonstrate its practicality.

Small size CNNs: Figures 7 and 8 illustrate the experimental results for a small CNN with one convolution layer and one pooling layer. Figures 7 (a), (b), and (c) show setup time, proof generation time, and CRS size, respectively, by varying the convolution input size where the kernel size is 10, depth is 3, and quantization bit depth is 10. Figure 8 increases the kernel size to 50 while the other parameters remain. Figures 7 and 8 show that vCNN produces better results in terms of performance and the CRS size always. In the figures, the CP-SNARKs time in vCNN is ignorable. Performance improves as the kernel size increases. In vCNN Setup is 2.6x faster than Gro16, proving time is 3.3x faster, and CRS size is 3.3x smaller when kernel size is 10; whereas setup is up to 9x faster, proving time is 7.5x faster, and CRS size is 12.3x smaller when kernel size is 50. Note that the prove time of convolutions in Gro16 can be easily estimated by subtracting the time of "ReLU+Pool" in vCNN from the prove time in Gro16.

Figure 9 shows the result for a MNIST CNN model which consists of a single convolution and pooling layer with kernel size is 9 (=3×3) and kernel depth is 64 by varying quantization bit depth from 16 to 32. Since non-linear functions, such as ReLU, are required to be encoded into bitwise operations "split" and "pack," both prove time and CRS size are proportional to the quantization bit depth. Setup and proof generation are up to 20x faster in vCNN than Gro16 and CRS size is up to 30x smaller when quantization bit depth is 32.

Figure 10 illustrates multi-layer CNNs on the MNIST dataset when the kernel size is 9 (=3 × 3) and quantization bit depth is 10. In this model convolution and pooling (including ReLU) layers alternate. The x axis represents the number of layers, e.g., the model with 2 layers consists of a convolution and a pooling layers, whereas in the model with 6 layers there are three convolution layers and three pooling layers, respectively. Each convolution layer has a different kernel depth. Kernel depths are given as 32, 64, and 128 for the first, the second, and the third convolution layer, respectively. Note that the model with 6 layers achieves 98% accuracy. Figures 10 (a)-(c) show that for the two layer model, setup is 10.6x faster in vCNN than Gro16, proof generation is 12x faster, and CRS size is 14.5x smaller. vCNN generates a proof in less than 11 seconds with 55MB size CRS while Gro16 scheme fails to generate proofs when the number of layers is more than two due to the large run-time memory requirement.

Real CNNs: We evaluate vCNN on several canonical CNNs models: LeNet-5 [24], AlexNet [23], and VGG16 [27]. We utilize the average pool rather than the max pool since the average pool requires a smaller circuit than the max pool. In addition, we exclude the fully connected layer in the models.

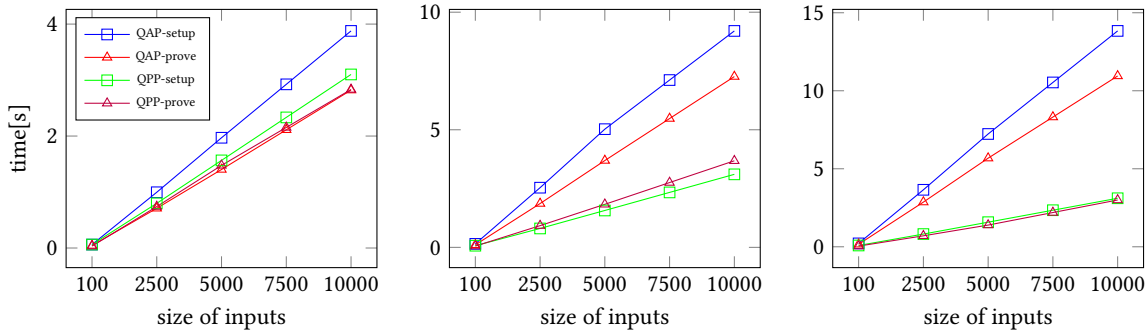
Figures 11 and 12 show the prove time and the CRS size by varying the scale factors for AlexNet and VGG16 models in vCNN. The scale factor includes two subfactors for the kernel depth and the input size. For example, $(\frac{1}{32}, \frac{1}{7})$ denotes that the kernel depth decreases by $\frac{1}{32}$ and the input size by $\frac{1}{7}$ in every layer. Note that $(1, 1)$ represents the real model.

Table 2 summarizes the performance and the size in vCNN and Gro16 [19]. In the table, we estimate the results in Gro16 due to insufficient memory. In vCNN, the setup time, proving time, and CRS size in vCNN are **291x** faster and smaller than Gro16 for LeNet-5. Similarly, they are **1200x** faster and smaller than Gro16 for AlexNet; and **18000x** for VGG16. Note that Gro16 would require more than 10 years to generate a proof for VGG16. Verification time remains for all applications in both vCNN and Gro16.

6 CONCLUSION

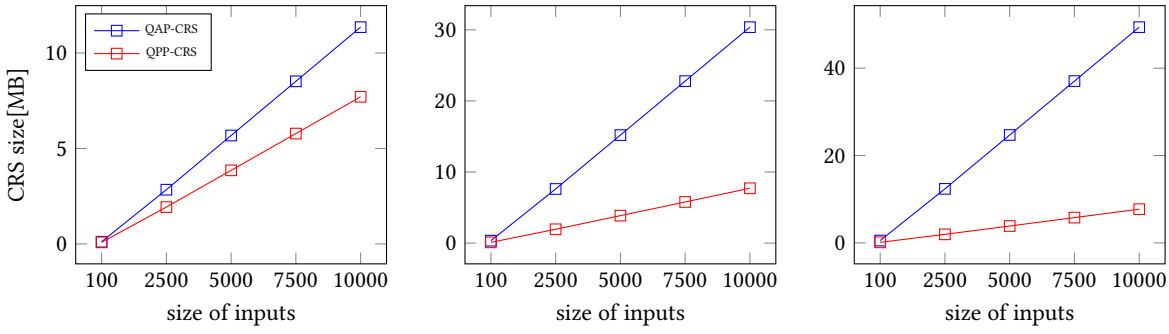
In this paper, we propose the first practical verifiable zk-SNARKs scheme for convolutional neural network models. We devise a new relation to optimally represent convolution operations based on quadratic polynomial program(QPP), which reduces the computational complexity to $O(l + n)$ from $O(l \cdot n)$ in the existing QAP approach where l and n denote the kernel and the input size. However, since the QPP only approach enlarges the circuit for components except convolution we adopt a commit-and-prove approach to combine proofs after applying QPP and QAP to convolution and the other functions. The proposed scheme is proven to be perfectly zero-knowledge and computationally knowledge sound.

The experimental results validate that the proposed vCNN scheme reduce prove time and CRS size approximately **18,000x** for the canonical CNN models on VGG16. In practice, prove time decreases to 8 hours from 10 years, and CRS size reduces to 80GB from 1400 TB compared with [19].



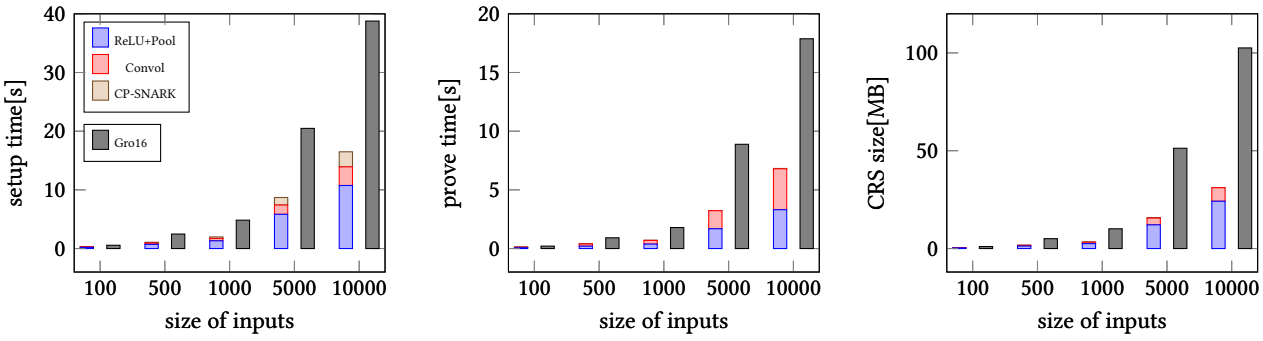
(a) kernel size = 10 (b) kernel size = 30 (c) kernel size = 50

Figure 5: Prove time in QAP and QPP based zk-SNARKs for convolutions



(a) kernel size = 10 (b) kernel size = 30 (c) kernel size = 50

Figure 6: CRS size in QAP and QPP based zk-SNARKs for convolutions



(a) (b) (c)

Figure 7: Comparison between vCNN and Gro16 [19] when the kernel size = 10, depth size = 3, and quantization bit depth = 10 bits

REFERENCES

[1] [n. d.]. Jsnark. https://github.com/akosba/jsnark.

[2] [n. d.]. ZSL on Quorum. https://github.com/jpmorganchase/zsl-q.

[3] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. IEEE Computer Society, 459–474. https://doi.org/10.1109/SP.2014.36

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge, See [7], 90–108. https://doi.org/10.1007/978-3-642-40084-1_6

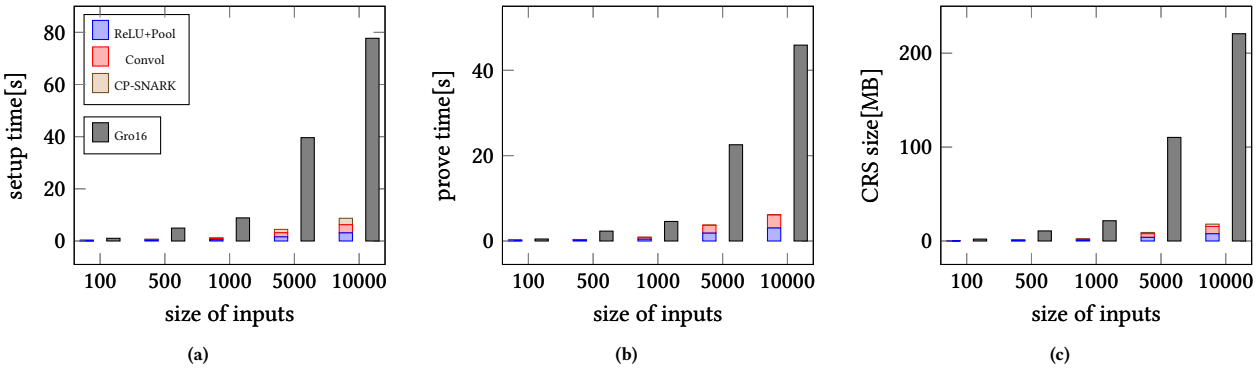


Figure 8: Comparison between vCNN and Gro16 [19] when kernel size = 50, depth size = 3, and quantization bit depth = 10 bits

Table 2: Comparison between vCNN and Gro16 for real CNN models

	vCNN					Gro16				
	setup	prove	verify	|CRS|	|proof|	setup	prove	verify	|CRS|	|proof|
LeNet-5	19.47 s	9.34 s	75ms	40.07MB		1.5 hours	0.75 hours	75ms	11 GB	
AlexNet	20 min	18 min	130ms	2.1 GB	2803 bits	16 days	14 days	130 ms	2.5 TB	1019 bits
VGG16	10 hours	8 hours	19.4s	83 GB		13 years	10 years	19.4s	1400 TB	

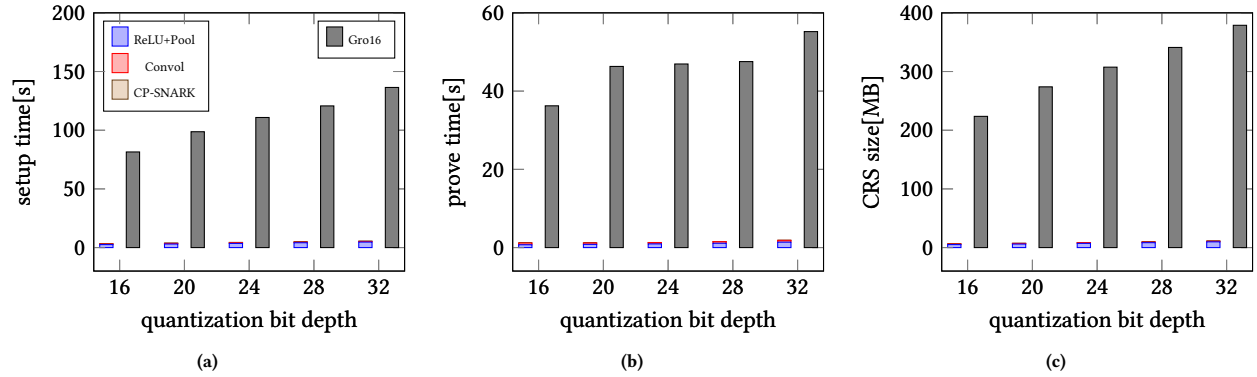


Figure 9: Results when kernel size = 3×3 and kernel depth size = 64

[5] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. 781–796. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[6] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. 2013. Succinct Non-interactive Arguments via Linear Interactive Proofs. In Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings. 315–333. https://doi.org/10.1007/978-3-642-36594-2_18

[7] Ran Canetti and Juan A. Garay (Eds.). 2013. Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, Vol. 8043. Springer. https://doi.org/10.1007/978-3-642-40084-1

[8] Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). 2019. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. ACM. https://doi.org/10.1145/3319535

[9] Hervé Chabanne, Julien Keuffer, and Refik Molva. 2017. Embedded Proofs for Verifiable Neural Networks. IACR Cryptology ePrint Archive 2017 (2017), 1038. http://eprint.iacr.org/2017/1038

[10] Marcus Comiter. 2019. Attacking artificial intelligence: AI's security vulnerability and what policymakers can do about it. Technical Report. Belfer Center for Science and International Affairs, Havard Kennedy School.

[11] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. 90–112. https://doi.org/10.1145/2090236.2090245

[12] Jacob Eberhardt and Stefan Tai. 2018. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018. IEEE, 1084–1091. https://doi.org/10.1109/Cybermatics_2018.2018.00199

[13] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time. In Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto,

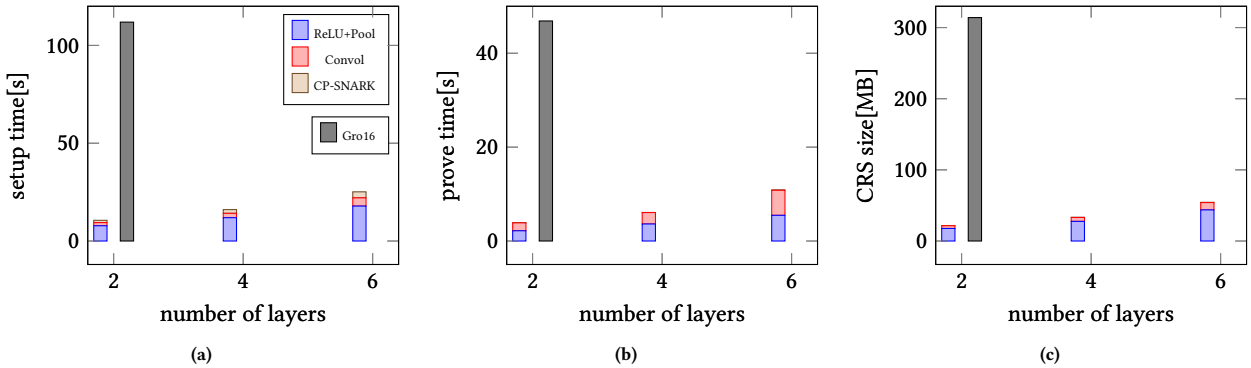


Figure 10: MNIST CNN when kernel size is 3×3 and kernel depths are 32, 64, and 128 for each convolution layer

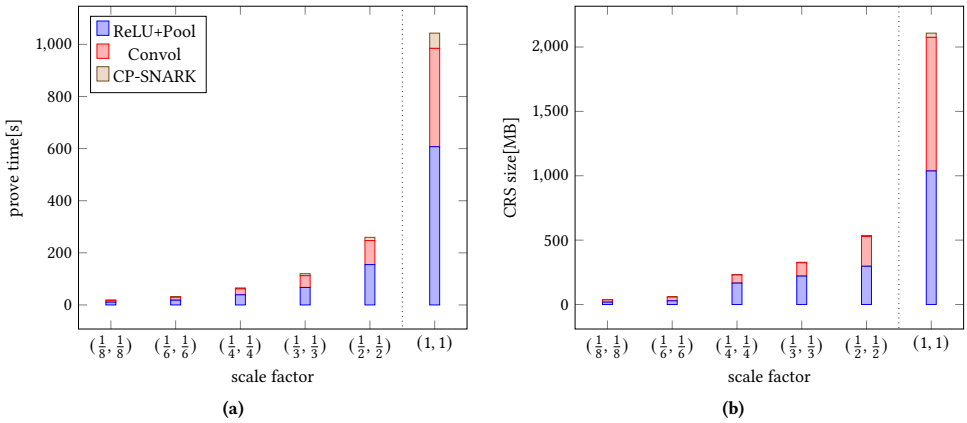


Figure 11: AlexNet in vCNN by varying the scale factor to the kernel depth and the input size

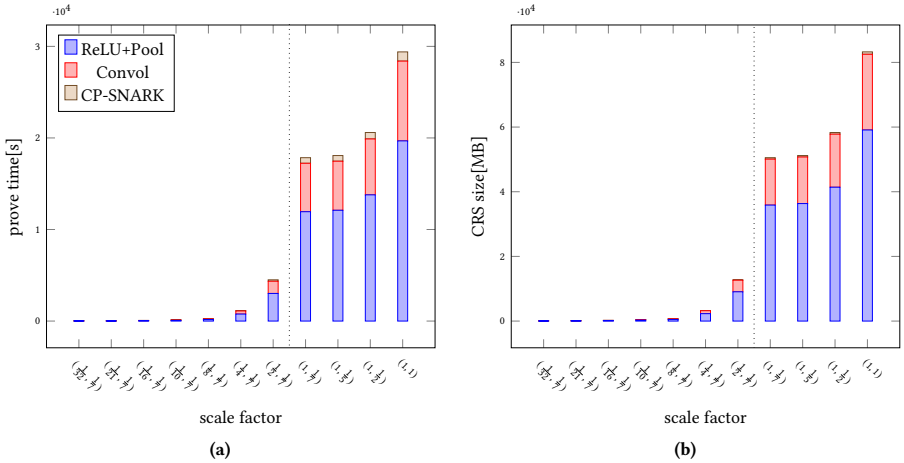


Figure 12: VGG16 in vCNN by varying the scale factorvCNN to the kernel depth and the input size

Canada, August 8-12, 1977, Bruce Gilchrist (Ed.). North-Holland, 839–842.

[14] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In Advances in

Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. 626–645. https://doi.org/10.1007/978-3-642-38348-9_37

[15] 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, 4-9 December 2017, Long Beach, CA, USA. 4675–4684. http://papers.nips.cc/paper/7053-safetynets-verifiable-execution-of-deep-neural-networks-on-an-untrusted-cloud

[16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. 201–210. http://proceedings.mlr.press/v48/gilad-bachrach16.html

[17] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comput. 18, 1 (1989), 186–208. https://doi.org/10.1137/0218012

[18] Shafi Goldwasser, Guy N. Rothblum, and Yael Tauman Kalai. 2017. Delegating Computation: Interactive Proofs for Muggles. Electronic Colloquium on Computational Complexity (ECCC) 24 (2017), 108. https://eccc.weizmann.ac.il/report/2017/108

[19] 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. https://doi.org/10.1007/978-3-662-49896-5_11

[20] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In 27th {USENIX} Security Symposium ({USENIX} Security 18). 1651–1669.

[21] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. 2014. TRUESET: Faster Verifiable Set Computations. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. 765–780. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kosba

[22] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. 2018. xJsnark: A Framework for Efficient Verifiable Computation. In 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. 944–961. https://doi.org/10.1109/SP.2018.00018

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (2017), 84–90. https://doi.org/10.1145/3065386

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (1998), 2278–2324.

[25] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: nearly practical verifiable computation. Commun. ACM 59, 2 (2016), 103–112. https://doi.org/10.1145/2856449

[26] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In Annual international cryptology conference. Springer, 129–140.

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

[28] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation, See [7], 71–89. https://doi.org/10.1007/978-3-642-40084-1_5

[29] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. https://openreview.net/forum?id=rJVorjCcKQ

[30] 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 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. 926–943. https://doi.org/10.1109/SP.2018.00060

[31] 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. CoRR abs/1909.06961 (2019). arXiv:1909.06961 http://arxiv.org/abs/1909.06961

A SECURITY PROOFS

A.1 Proof of Theorem 4.1 and 4.2

Proof. We demonstrate the NILP scheme soundness for the proposed protocol as demonstrated in [19]. If the NILP scheme is proved, then soundness for proposed scheme is guaranteed in the Generic Group Model [19]. zk-SNARK and cc-SNARK are similar aside from the random parameter for the commitment. In the proof, zk-SNARK soundness(4.1) is the special case of cc-SNARK soundness(4.2) when $\nu = 0$. Therefore we only prove Theorem 4.2 here.

We first consider an affine adversary $\mathcal{A}$ strategy with non-negligible success probability of extracting a witness. First, we set $Z = X^{2d_x-1}$ to reducing the variables. Then $\mathcal{A}$ can generate a proof

$$\begin{aligned}A = {} & A_\alpha\alpha + A_\beta\beta + A_\gamma\gamma + A_\delta\delta + A(x, x^{2d_x-1}) \\ & + \sum_{i=0}^{l}\sum_{j=0}^{d_z} A_{i,j}\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}x^{(2d_x-1)\cdot j} \\ & + \sum_{i=l+1}^{m}\sum_{j=0}^{d_z} A_{i,j}\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}x^{(2d_x-1)\cdot j} \\ & + A_h(x, x^{2d_x-1})\frac{t(x)}{\delta} + A_{\eta\gamma}\frac{\eta}{\gamma} + A_{\eta\delta}\frac{\eta}{\delta}\end{aligned}$$

for known filed elements A_α, A_β, A_γ, A_δ, A_i and polynomials $A(x,z)$, $A_h(x,z)$. we construct B and C similarly for the proof. In verification, the equation shows polynomials equality. From the Schwartz-Zippel lemma, verification holds the proof(A, B, and C) for indeterminates α, β, γ, δ, and x if verification succeed.

Terms with indeterminates α^2 are $A_\alpha B_\alpha \alpha^2 = 0$, i.e., $A_\alpha = 0$ or $B_\alpha = 0$. Since field operation is commutative, we can assume $B_\alpha = 0$. Terms with indeterminate $\alpha\beta$ imply $A_\alpha B_\beta + A_\beta B_\alpha = A_\alpha B_\beta = 1$. Thus, $AB = (AB_\beta)(A_\alpha B)$, and we can assume $A_\alpha = B_\beta = 1$. Hence with indeterminate β^2 now imply $A_\beta B_\beta = A_\beta = 0$. This simplifies A and B constructed by the adversary to have the form

$$\begin{aligned}A &= \alpha + A_\gamma\gamma + A_\delta\delta + A(x, x^{2d_x-1}) + \cdots \\ B &= \beta + B_\gamma\gamma + B_\delta\delta + B(x, x^{2d_x-1}) + \cdots\end{aligned}$$

Let us consider terms involving $\frac{1}{\delta^2}$.

$$\left(\sum_{i=l+1}^{m} A_{i,j}(\beta u_i(x) + \alpha v_i(x) + w_i(x))\cdot x^{(2d_x-1)\cdot j} + A_h(x, x^{2d_x-1})t(x)\right) \cdot \left(\sum_{i=l+1}^{m} B_{i,j}(\beta u_i(x) + \alpha v_i(x) + w_i(x))\cdot x^{(2d_x-1)\cdot j} + B_h(x, x^{2d_x-1})t(x)\right) = 0$$

Hence either left factor is 0. From symmetry, let us assume

$$(\Sigma_{i=l+1}^{m} A_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + A_h(x, x^{2d_x-1})t(x)) = 0$$

. Therefore, terms in

$$\alpha\frac{\Sigma_{i=l+1}^{m} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + B_h(x, x^{2d_x-1})t(x)}{\delta} = 0$$

imply that $\Sigma_{i=l+1}^{m} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + B_h(x, x^{2d_x-1})t(x) = 0$.

Therefore, considering terms involving $\frac{1}{\gamma}$,

$$\left(\sum_{i=0}^{l} A_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))\right) \cdot \left(\sum_{i=0}^{l} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))\right)$$

hence either left or right factor is 0. From symmetry, let us assume $(\Sigma_{i=0}^{l} A_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))) = 0$. Thus, terms in

$$\beta \frac{\Sigma_{i=0}^{l} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma} = 0$$

also imply $\Sigma_{i=0}^{l} B_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) = 0$.

Thus, $A_\gamma \beta\gamma = 0$, $B_\gamma \alpha\gamma = 0$, and added terms involving η also $(A_{\eta\gamma}\frac{\eta}{\gamma} + A_{\eta\delta}\frac{\eta}{\delta}) \cdot \beta = 0$, hence $A_\gamma = 0$, $B_\gamma = 0$, $A_{\eta\gamma} = 0$, and $A_{\eta\delta} = 0$.

Collecting these results,

$$A = \alpha + A(x, x^{2d_x-1}) + A_\delta \delta \quad B = \beta + B(x, x^{2d_x-1}) + B_\delta \delta$$

Remaining terms in the verification equation that involve α imply $\alpha B(x, x^{2d_x-1}) = \alpha \sum_{i=0}^{l} a_i(x^{2d_x-1} v_i(x) + \alpha \sum_{i=l+1}^{m} \sum_{j=0}^{d_z} C_{i,j} v_i(x) \cdot x^{(2d_x-1)\cdot j}$. Defining $a_i(x^{2d_x-1}) = C_i(x^{2d_x-1}) = \sum_{j=0}^{d_z} C_{i,j} \cdot x^{(2d_x-1)\cdot j}$ for $i = l+1, \ldots, m$,

$$A(x, x^{2d_x-1}) = \sum_{i=0}^{m} a_i(x^{2d_x-1}) u_i(x) \quad B(x, x^{2d_x-1}) = \sum_{i=0}^{m} a_i(x^{2d_x-1}) v_i(x)$$

Finally,collecting terms involving powers of x,

$$\sum_{i=0}^{m} a_i(x^{2d_x-1}) u_i(x) \cdot \sum_{i=0}^{m} a_i(x^{2d_x-1}) v_i(x) = \sum_{i=0}^{m} a_i(x^{2d_x-1}) w_i(x) + C_h(x, x^{2d_x-1}) t(x)$$

Since $Z = X^{2d_x-1}$, Z degree $\geq X$ degree, and all terms are independent. Thus, $a_i(X^{2d_x-1})$ is irrelevant to $u_i(X), v_i(X), w_i(X)$ and $t(X)$, and hence

$$a_{l+1}(x^{2d_x-1}), \ldots, a_m(x^{2d_x-1}) = C_{l+1}(x^{2d_x-1}), \ldots, C_m(x^{2d_x-1})$$

is a witness for the statement $(a_1(x^{2d_x-1}), \ldots, a_l(x^{2d_x-1}))$. □

A.2 Proof of Theorem 4.3

Proof. We first prove the perfect zero-knowledge. There are simulators for each scheme, and the commitment is the Pedersen [26] vector commitment which provides perfect hiding. Thus, proof has no information regarding witnesses, and hence the scheme supports perfect zero-knowledge.

Next, we prove that the computational knowledge soundness error is negligible. We define the computational knowledge soundness errors for each scheme Π_{qap}, Π_{qpp}, and Π_{cp} as ϵ_{qap}, ϵ_{qpp}, and ϵ_{cp}, respectively, which are negligible; and the extractors for each scheme are χ_{qap}, χ_{qpp}, and χ_{cp}, respectively, which must exist due to the knowledge soundness for each scheme. The extractor χ for the proposed scheme can be composed of three extractors because each extractor can generate a witness and the collection of all the witnesses is the witness for the proposed scheme.

Now, we compute the computation knowledge soundness error for the proposed scheme as follows:

$$\begin{aligned}
&Pr\left[\begin{array}{c|c} \mathsf{Verify}(crs, \phi, \pi) = 1 \\ \wedge (\phi, w) \notin R \end{array}\middle| \begin{array}{c} (crs, td) \leftarrow \mathsf{Setup}(R), \\ (\phi, \pi, w) \leftarrow (\mathcal{A}|\chi_{\mathcal{A}})(R, crs, z) \end{array}\right] \\
&= Pr\left[\begin{array}{c} \Pi_{qap}.\mathsf{Verify}(crs_{qap}, \phi_{qap}, \pi_{qap}) = 1 \\ \wedge \Pi_{qpp}.\mathsf{Verify}(crs_{qpp}, \phi_{qpp}, \pi_{qpp}) = 1 \\ \wedge \Pi_{cp}.\mathsf{Verify}(crs_{cp}, \phi_{cp}, \pi_{cp}) = 1 \\ \wedge ((\phi_{qap}, w_{qap}) \notin R_{ReLU+Pooling} \\ \vee (\phi_{qpp}, w_{qpp}) \notin R_{convol} \vee (\phi_{cp}, w_{cp}) \notin R_{cp}) \end{array}\right] \\
&\leq Pr\left[\begin{array}{c} \Pi.\mathsf{Verify}(crs_{qap}, \phi_{qap}, \pi_{qap}) = 1 \\ \wedge \Pi_{qpp}.\mathsf{Verify}(crs_{qpp}, \phi_{qpp}, \pi_{qpp}) = 1 \\ \wedge \Pi_{cp}.\mathsf{Verify}(crs_{cp}, \phi_{cp}, \pi_{cp}) = 1 \\ \wedge (\phi_{qap}, w_{qap}) \notin R_{ReLU+Pool} \end{array}\right] \\
&+ Pr\left[\begin{array}{c} \Pi_{qap}.\mathsf{Verify}(crs_{qap}, \phi_{qap}, \pi_{qap}) = 1 \\ \wedge \Pi_{qpp}.\mathsf{Verify}(crs_{qpp}, \phi_{qpp}, \pi_{qpp}) = 1 \\ \wedge \Pi_{cp}.\mathsf{Verify}(crs_{cp}, \phi_{cp}, \pi_{cp}) = 1 \\ \wedge (\phi_{qpp}, w_{qpp}) \notin R_{convol} \end{array}\right] \\
&+ Pr\left[\begin{array}{c} \Pi_{qap}.\mathsf{Verify}(crs_{qap}, \phi_{qap}, \pi_{qap}) = 1 \\ \wedge \Pi_{qpp}.\mathsf{Verify}(crs_{qpp}, \phi_{qpp}, \pi_{qpp}) = 1 \\ \wedge \Pi_{cp}.\mathsf{Verify}(crs_{cp}, \phi_{cp}, \pi_{cp}) = 1 \\ \wedge (\phi_{cp}, w_{cp}) \notin R_{cp} \end{array}\right] \\
&\leq \epsilon_{qap} + \epsilon_{qpp} + \epsilon_{cp}
\end{aligned}$$

where we used that ϵ_{qap}, ϵ_{qpp}, ϵ_{cp} are negligible in the last two inequalities. Therefore the computational soundness error is negligible.

□