Efficient Deep Neural Network Verification with QAP-Based ZkSNARK

Subhasis Thakur(✉) and John Breslin

University of Galway, Galway, Ireland
{subhasis.thakur,john.breslin}@universityofgalway.ie

Abstract. In MLaaS, DNN models are kept in a server operated by the service provider and inputs to the DNN models are provided by the clients. Such inputs are used to execute the DNN models and classification results are sent back to the client. In MLaaS, the DNN model owner does not reveal the DNN model parameters to the client. MLaaS there are a few trust problems: (a) The server may not be secure and an attacker may send manipulated classification results to the client. In the case of safety-critical systems using such classification in the decision-making process, an attacker may specifically manipulate the classification result to disrupt the operations of the safety-critical system, (b) The server may intentionally send wrong or random classification results without executing the DNN model to respond to a massive number of classification requests from the clients. In this paper, we investigate the problem of verifying DNN model execution by the service provider in an MLaaS paradigm. A proof of DNN model execution will prove that given an input, the DNN model is executed to generate the classification result by providing sequences of outputs of all functions used in the DNN model. As the service provider in MLaaS does not share the DNN model with the client, we need to verify DNN function outcomes without the knowledge of DNN function parameters. Hence zero-knowledge proof can be used for verifying DNN model execution. In this paper, we use Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) which reduces the size of proof and complexity of proof verification considerably. In particular, we use a quadratic arithmetic program-based zkSNARK for DNN model verification. Our main results in this paper are as follows: (a) We have developed a DNN model execution verification method using a QAP-based zkSNARK. (b) We prove that the verification protocol is correct and privacy-preserving. (c) We analyzed the cost of using such a verification protocol.

Keywords: Deep neural networks · zkSNARK · Zero knowledge proof

1 Introduction

Machine learning as a service (MLaaS) facilitates the outsourcing of Deep Neural Network (DNN) classification tasks. In MLaaS, DNN models are kept in a server operated by the service provider and inputs to the DNN models are provided by the clients. Such inputs are used to execute the DNN models and classification results are sent back to the client. In MLaaS, the DNN model owner does not reveal the DNN model parameters to

A. Fred et al. (Eds.): DeLTA 2024, CCIS 2172, pp. 178–191, 2024.
https://doi.org/10.1007/978-3-031-66705-3_12

the client. The advantages of MLaaS are (a) it can lower the cost to use DNN models in businesses as it can hire a DNN model for low cost, (b) the client can easily scale up and down its classification infrastructure depending on its business requirement by leveraging elasticity in the cloud computing infrastructure of MLaaS. However, in this MLaaS there are a few trust problems:

1. The server may not be secure and an attacker may send manipulated classification results to the client. In the case of safety-critical systems using such classification in the decision-making process, an attacker may specifically manipulate the classification result to disrupt the operations of the safety-critical system.
2. The server may intentionally send wrong or random classification results without executing the DNN model to respond to a massive number of classification requests from the clients.

In this paper, we investigate the problem of verifying DNN model execution by the service provider in an MLaaS paradigm. A proof of DNN model execution will prove that given an input, the DNN model is executed to generate the classification result by providing sequences of outputs of all functions used in the DNN model. As the service provider in MLaaS does not share the DNN model with the client, we need to verify DNN function outcomes without the knowledge of DNN function parameters. Hence zero-knowledge proof can be used for verifying DNN model execution. In this paper, we use Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) which reduces the size of proof and complexity of proof verification considerably. In particular, we use a quadratic arithmetic program-based zkSNARK for DNN model verification. Our main results in this paper are as follows:

1. We have developed a DNN model execution verification method using a QAP-based zkSNARK.
2. We prove that the verification protocol is correct and privacy-preserving.
3. We analyzed the cost of using such a verification protocol.

The paper is organized as follows: in Sect. 2 we discuss related literature, in Sect. 3 we describe the DNN model verification problem, in Sect. 4 we present the DNN model verification procedure, in Sect. 5 we present an analysis of the proposed DNN model execution, and we conclude the paper in Sect. 6.

2 Related Literature

MLaas was previously investigated and many architectures for MLaaS were developed [15]. Privacy and security are important problems for MLaaS. In [14] privacy issues for MLaaS were investigated. The service provider in an MLaaS wants to keep its DNN model secret and model stealing attacks is an important problem in MLaaS. In [11], model stealing attacks are investigated for MLaaS. In the context of MLaaS, the service provider does not share the DNN model with to client and hence we need to use zero-knowledge proof for such a verification. Briefly, previous research in DNN model verification is as follows: In [13] have developed a scalable method to verify deep neural networks during the training phase. In [4], the authors have developed a protocol to

verify the execution of neural networks using an interactive sum check protocol. In [7] the authors have developed a method to verify the machine learning model to be used in safety-critical systems. In [1] uses zkSNARK to verify deep learning models kept in insecure data servers. In [17] the authors have explored verifiable computation for federated machine learning. In [3] developed a method for shared certificates to reduce the cost of verifying neural networks. [9,16] have presented surveys on trustworthy AI.

In this paper, we used zero-knowledge proof for DNN verification. In [2] the authors have developed a zero-knowledge proof protocol that allows the prover to prove that it still knows a number. We use Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) which reduces the size of proof and complexity of proof verification considerably. The KGZ polynomial commitment scheme developed in [8]. Other constructions of zkSNARKs developed in [6,10,12] use quadratic arithmetic program-based construction of zkSNARK. In this paper we used QAP-based zkSNARK developed in [5,12]

3 Problem Statement

We can represent a DNN model as follows:

1. there are k layers,
2. each layer has n nodes,
3. the DNN is fully connected,

The MLaaS service provider builds the DNN models and the client sends input data to the service provider for DNN model execution and gets the classification results. In this settings, the service provider and the client have the following knowledge:

Data	Service Provider	Client
DNN Model parameters	✓	×
Number of nodes in each DNN layer	✓	✓
Number of DNN layers	✓	✓
Input data	×	✓

In the above setting, the client wants to verify that a classification result for its input data is a result of the correct execution of the DNN functions. A malicious service provider can send random classification results to the client without actual execution of the DNN model. We address the following problems to prevent such a malicious service provider:

- We need to generate a zero-knowledge proof protocol for verifying DNN model execution where the service provider acts as the prover and sends proofs of DNN function evaluations to the client (the verifier).
- A DNN model may contain numerous DNN functions with a large number of variables. It is costly to verify all such functions using zero-knowledge proofs. Hence we need to reduce the number of verifications to make the DNN model verification protocol efficient.

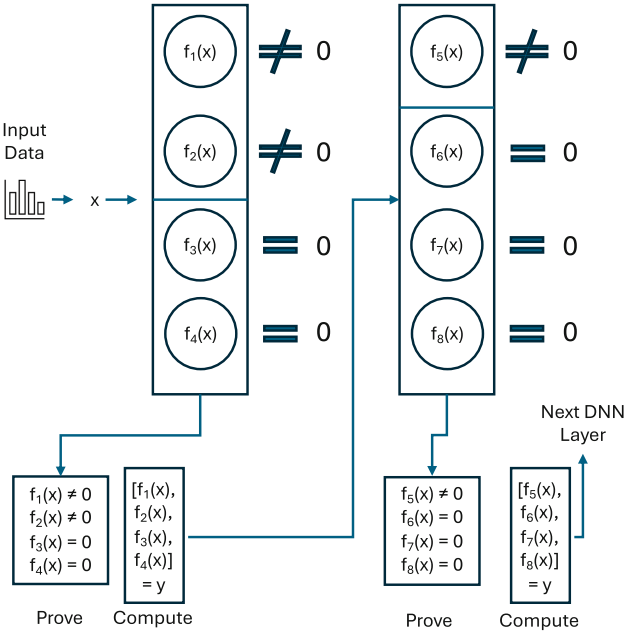

Fig. 1. Overview of DNN verification.

4 DNN Verification Procedure

4.1 Overview of DNN Verification

A DNN model with k layers and n nodes in each layer is as follows:

In the above DNN model, the j'th node at $layer_i$ computes the following function

$$y_j^i = b_j^i + \cup_{x=1}^{n} y_j^{i-1} w_{x,j}^i$$

where all parameters are real numbers. All functions y_j^i will be referred to as DNN functions. An activation function modifies the outcome of each DNN function to be used as input to the next layer. We assume that the activation function is Relu, i.e., if the outcome of a DNN function is positive then its value remains the same otherwise it becomes zero. Given an input $(a_0, \ldots, a_n)$, the activation function will modify the outcome of a DNN function as either 0 or > 0. We will build a binary representation from such an activation function. In this DNN model, the proposed DNN verification protocol (shown in Fig. 1) is as follows:

1. We assume that the activation function of DNN functions is ReLU, i.e., it converts the outcome of each DNN function into either 0 or a positive number $x > 0$.
2. Given the input to the functions of a DNN layer, the DNN functions can be partitioned into two groups. In one group, all DNN function values are more than 1 and in the other group, all DNN function values are 0.

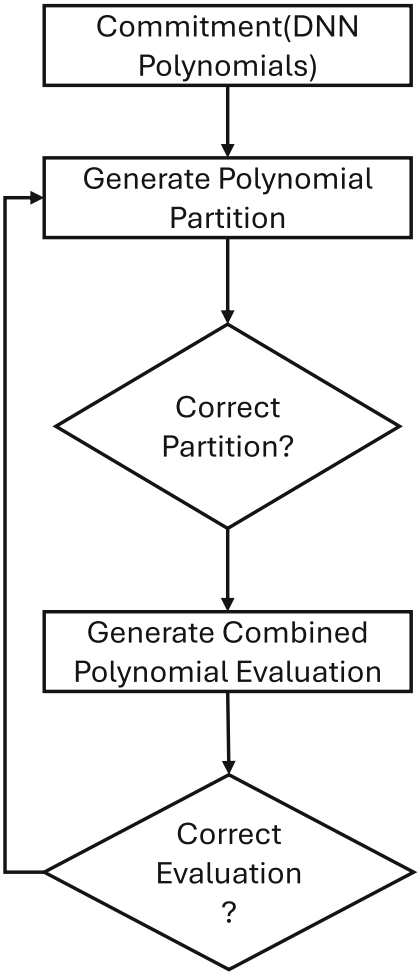

Fig. 2. Procedure for DNN model verification.

3. A valid execution of the DNN functions implies that all such DNN function partitions are correct. Given a partition of DNN functions in layer i into two groups S_i ($value > 0$), T_i ($value = 0$), we can prove the partition in layer $i + 1$ (S_{i+1}, T_{i+1}) is correct if (a) by checking the input from the DNN functions in layer i is correct and (b) by checking evaluation of DNN functions in layer $i + 1$ are correct.
4. We can perform such verification for each layer of the DNN model. In such verification, the prover is the DNN model owner and the verifier is using the DNN model for a classification task on its input data. We will use zero-knowledge proofs to hide the DNN model parameters from the verifier.

Table 1. DNN model.

Input	a_0
	$\dots$
	a_n
$Layer_1$	$y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1$
	$\dots$
	$y_n^1 = b_n^1 + \cup_{i=1}^{n} a_i w_{i,n}^1$
$Layer_2$	$y_1^2 = b_1^2 + \cup_{i=1}^{n} a_i w_{i,1}^2$
	$\dots$
	$y_n^2 = b_n^2 + \cup_{i=1}^{n} a_i w_{i,n}^2$
$\dots$	$\dots$
$Layer_k$	$y_1^k = b_1^k + \cup_{i=1}^{n} a_i w_{i,1}^k$
	$\dots$
	$y_n^k = b_n^k + \cup_{i=1}^{n} a_i w_{i,n}^k$

5. We can use the following procedure for the above verification (shown in Fig. 2)
   (a) The DNN model owner (Prover) sends a commitment of the DNN model to the verifier without revealing the DNN model parameters.
   (b) Given an input to the DNN model, the prover generates the partition of DNN functions for the first layer and generates the DNN function values for the first layer. Then it sends a zero-knowledge proof for proving such a partition is correct along with the DNN function values to the verifier.
   (c) The verifier checks the proofs and calculates the input to the next layer from the DNN function values of the first layer.
   (d) This process continues for all DNN layers.

4.2 Polynomial Representation of DNN Models

In the DNN shown in Table 1, there are $n * k$ functions where each function has degree n if we represent each function as follows:

$$f(x) = y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1 \tag{1}$$

$$= b_1^1 + a_1 x^1 + a_2 x^2 + \cdots + a_n x^n \tag{2}$$

where x represent the set of inputs $a_1, \dots, a_n$. Note that a DNN function $f(x) = y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1$ is evaluated as $b_1^1 + a_1 x^1 + a_2 x^2 + \cdots + a_n x^n$. Hence we need to find a value for x that can represent the set of numbers $b_1^1, a_1, \dots, a_n$. The prover and the verifier generate a number x as follows:

$$
\begin{aligned}
x &= a_1 \\
x^2 &= a_2 \\
x^3 &= a_3 \\
&\cdots \\
x^n &= a_n \\
x + x^2 + x^3 + \cdots + x^n &= a_1 + a_2 + \cdots + a_n \\
x + x^2 + x^3 + \cdots + x^n - (a_1 + a_2 + \cdots + a_n) &= 0 \\
\frac{1}{1-x} &= (a_1 + a_2 + \cdots + a_n) \\
\frac{1}{(a_1 + a_2 + \cdots + a_n)} &= 1 - x \\
1 - \frac{1}{(a_1 + a_2 + \cdots + a_n)} &= x \qquad (3)
\end{aligned}
$$

The above equation holds for large n. We assume that the DNN model is large and hence n is large. Using this value of x, the prover calculates DNN functions for $layer_1$. We denote such a set of evaluations as the set $\{y_i^1\}$. Note that, the set $\{y_i^1\}$ is the input to $layer_2$, and for $layer_2$ we calculate the value of x for DNN functions of $layer_2$ using the same procedure.

4.3 Polynomial Evaluation Problem for DNN Verification

Given a set of polynomials for i'th layer as $\{F_i(x)\}$ and the combined polynomial as $F(x)$, we generate a partition over the polynomials as follows:

1. Let the set of polynomials $S \subset \{F_i(x)\}$ and $T \subset \{F_i(x)\}$ corresponds to the DNN functions with evaluation > 0 and 0 respectively.
2. We create two polynomials as follows

$$\mathbb{S} = \bigcap_{F_i(x) \in S} F_i(x) \qquad (4)$$

$$\mathbb{T} = \bigcup_{F_i(x) \in T} F_i(x) \qquad (5)$$

We need to prove the following:

1. We need to prove evaluation of $\mathbb{S}$ at x (given by Eq. 3) is not zero. Let $F_i(y) = z_i$ for all $F_i \in S$ and given value of x as y. Hence, in order to prove that $\mathbb{S}$ at $x = y$ is not zero if $(x - y)$ is a root of the polynomial $\mathbb{S} - \prod_i z_i$. The prover can present a (zero) knowledge proof that it knows a value of x as y such that $\mathbb{S} - \prod_i z_i = 0$.
2. We need to prove evaluation of $\mathbb{T}$ at x (given by the Eq. 3) is zero. The prover can present a (zero) knowledge proof that it knows a value of x as y such that $\mathbb{T} = 0$.

4.4 QAP-Based ZkSANRK for Polynomial Evaluation Verification

Common Reference String. During this step, the prover and the verifier compute a set of random numbers. Let G_1 be an elliptic curve for the prime number $\mathbb{P}$ and g_1 be the generator of G_1. Let τ be a random positive integer less than $\mathbb{P}$. A set of $l > n$ random numbers are generated from τ as follows:

$$\begin{aligned} p_1 = g_1, p_2 = M(\tau, g_1), & p_3 = M(\tau^2(Mod(\mathbb{P})), g_1), \\ & \ldots, \ldots \\ p_l = M(\tau^{l-1}(Mod(\mathbb{P})), g_1) & \end{aligned} \tag{6}$$

where $M()$ is a function to multiply a positive integer with an elliptic curve point resulting in a new elliptic curve point. We can use a ceremony to form a random number. The prover and verifier know the above random points for both bilinear elliptic curves. We represent these random points for $\mathbb{G}_1, \mathbb{G}_2$ as follows:

$$\begin{aligned} \mathbb{G}_1 : & p_1^1 = g_1, p_2 = M(\tau, g_1), \ldots, \\ & p_l^1 = M(\tau^{l-1}(Mod(\mathbb{P}_1)), g_1) \\ \mathbb{G}_2 : & p_1^1 = g_2, p_2 = M(\tau, g_2), \ldots, \\ & p_l^2 = M(\tau^{l-1}(Mod(\mathbb{P}_2)), g_2) \end{aligned}$$

QAP-Based ZkSNARK. We will use a QAP-based zkSNARK to prove the evaluation of the above polynomials. Let $f(x) = r$ be a polynomial where r is a real number. The prover wants to prove that it knows a value of $x = y$ such that $f(y) = r$ without revealing y to the verifier.

It is as follows:

1. First, we will convert a polynomial $f(x)$ int a set of constraints with three terms $A = B(Op)C$ where A, B, C are either variables or constants (real numbers) and Op is operation representing $*$ or $+$. Algorithm 2 describes the process of converting a polynomial to such constraints. It uses a procedure (shown in Algorithm 1) to convert each term $a_i x^i$ into a set of constraints.
2. Next, we will convert the set of constraints into a ranked 1 constraint system. We set a vector A(line 2 Algorithm 3) representing all variables of the constraint system and one additional field One(will be used to include scalars into the ranked 1 constraint system).
3. Next, we will convert the R1CS into Quadratic-Arithmetic-Programs (QAP). We will convert the R1CS constraints into polynomials using Lagrange Interpolation. The QAP representation will contain coefficients of such polynomials. This conversion process is shown in Algorithm 4.
4. Finally, a zero-knowledge proof multiplies the QAPs with a witness vector, i.e., valuation of vector A (line 2 Algorithm 3). We denote this polynomial as $t(x)$.
5. The verifier generates a polynomial $z(x) = (x-1)(x-2)\ldots(x-p)$ where p is the number of constraints in the R1CS or number of weights in each DNN node.
6. The verifier will accept $z(x)$ as a valid proof for the polynomial $f(x)$ if there exists a polynomial $h(x)$ such as $t(x) = h(x) \times z(x)$, i.e., $t(x)$ is divisible by $z(x)$.

7. The verifier chooses a random positive numbers s, α and computes the following:

$$P_1 = M(s, g_1), P_2 = M(s^2, g_1), \ldots, P_d = M(s^d, g_1)$$
$$Q_1 = M(\alpha s, g_1), Q_2 = M(\alpha s^2, g_1), \ldots, Q_d = M(\alpha s^d, g_1)$$

The verifier sends these elliptic curve points to the prover. For any polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$, the prover can create two elliptic curve points for $f(x)$ evaluated at s as follows:

$$P = Add(M(a_0, g_1), M(a_1, P_1), \ldots, M(a_d, P_d))$$
$$Q = Add(M(a_0, g_1), M(a_1, Q_1), \ldots, M(a_d, Q_d))$$

The prover will generate points P and Q for the polynomials $z(x), t(x), h(x)$ and send it to the verifier.

8. The verifier checks if for all P and Q value of $z(x), t(x), h(x)$, $P = M(\alpha, Q)$. This verification is known as $d - KCA$ (d-power knowledge of coefficient assumption [5]) test and it ensures that the prover has indeed used a witness (A in line 2 Algorithm 3) to generate the polynomials $z(x), t(x), h(x)$ without revealing the polynomials $z(x), t(x), h(x)$.
9. Finally, the verifier checks if $h(x) = t(x) \times z(x)$ using bilinear elliptic curves as it needs to check the multiplication relation over elliptic curve points.

Table 2. Sumamry of DNN verification.

Prover (MLaaS service Provider)	Verifier (Client of MLaaS)
	Sends the input to the DNN model to the verifier
Computes DNN function values of all DNN functions of the first layer	
Generate two functions $\mathbb{S}$ and $\mathbb{T}$ as shown in Eqs. 4 and 5	
	Sends the sets of elliptic curve points $\{P_i\}, \{Q_i\}$ as shown in step 7 in previous section.
Generate zero-knowledge proofs $z(x), t(x), h(x)$ for both polynomials $\mathbb{S} - \prod_i z_i$ and $\mathbb{T}$	
Generate elliptic curve points P, Q for all $z(x), t(x), h(x)$ as mentioned in step 7 in the previous section	
	Check if $P = M(\alpha, Q)$.
	Check if $h(x) = t(x) \times z(x)$ using bilinear elliptic curves.
Repeat the above steps for all DNN layers	

Algorithm 1. Decomposition of term into constraint systems.

input : i'th term $a_i x^i$ a polynomial, Starting x
output: Constraint systems for the term
$Constraints \leftarrow \emptyset$
if $i \leq 1$ **then** $c_1 : s_x \leftarrow a_i * x$
Add c_1 to $Constraints$

else for $j \in [1 : i-1]$ **do**
  if $j == 1$ **then**
    $c_j : s_x \leftarrow x * x$
    x++
  else
    $c_j : s_x \leftarrow c_{j-1} * x$
    x++;
  Add c_j to $Constraints$
$c_i : s_x \leftarrow c_{i-1} * a_i$

Add $\{c_y\}$ to $Constraints$
Return $Constraints, x$

Algorithm 2. Decomposition of all terms into constraint systems.

input : A polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$
output: Constraint systems for $f(x)$
$Constraints \leftarrow \emptyset$
$Index \leftarrow 1$
$Last\_Constraint = \emptyset$
for $i \in [n : 1]$ **do**
  $Term\_Constraints, New\_Index \leftarrow Decompose(a_i x^i, Index)$
  Add $Term\_Constraints$ to $Constraints$
  $Index \leftarrow New\_Index$
  if $Last\_Constraint = \emptyset$ **then**
    $Last\_Constraint = s_x$ where s_x is the output of last equation of $Term\_Constraints$
  else
    s_y is the output of last equation of $Term\_Constraints$
    $c : s_{Index} \leftarrow Last\_Constraint + s_y$
    $Last\_Constraint = s_x$ where s_x is the output of last equation of $Term\_Constraints$
    Add c to $Constraints$
    Index ++
  $c : s_{Index} \leftarrow Last\_Constraint + a_0$
  Add c to $Constraints$
Return $Constraints$

Algorithm 3. R1CS Generation.

input : A set of Constraints
output: R1CS formulation of constraints
$R1CS \leftarrow \emptyset$
$A = [One, x, Out, s_1, \ldots, s_k]$ be a vector representing the set of all symbols from the constraints
for $i \in [1 : k]$ **do**
  $c_i \leftarrow$ i'th constraint
  if $'*' \in c_i$ **then**
    if $Scalar \notin c_i$ **then**
      // c_i: s_x = s_y * x
      // p be index of s_y in A
      // q be index of s_x in A
      $a = [0, 1, \ldots, 0]$
      $b = [0, 0 \ldots, \underbrace{1}_{p'thposition}, \ldots, 0]$
      $c = [0, 0 \ldots, \underbrace{1}_{q'thposition}, \ldots, 0]$
    else
      // c_i: s_x = s_y * 5
      // p be index of s_y in A
      // q be index of s_x in A
      $a = [5, 0, \ldots, 0]$
      $b = [0, 0 \ldots, \underbrace{1}_{p'thposition}, \ldots, 0]$
      $c = [0, 0 \ldots, \underbrace{1}_{q'thposition}, \ldots, 0]$
  if $'+' \in c_i$ **then**
    if $Scalar \notin c_i$ **then**
      // c_i: s_x = s_y + s_z
      // p be index of s_y in A
      // q be index of s_x in A
      // r be index of s_z in A
      $a = [0, \ldots, \underbrace{1}_{q'thposition}, \underbrace{1}_{r'thposition}, \ldots, 0]$
      $b = [1, 0, \ldots, 0]$
      $c = [0, 0 \ldots, \underbrace{1}_{p'thposition}, \ldots, 0]$
    else
      // c_i: s_x = s_y + 5
      // p be index of s_y in A
      // q be index of s_x in A
      $a = [1, 0, \ldots, 0]$
      $b = [5, 0 \ldots, \underbrace{1}_{p'thposition}, \ldots, 0]$
      $c = [0, 0 \ldots, \underbrace{1}_{q'thposition}, \ldots, 0]$
  $A =$ All a vectors, $B =$ All b vectors, $C =$ All c vectors
Return A,B,C

Algorithm 4. Convert R1CS into QAP.

input : R1CS constraints
output: QAP formulation
A, B, C be the R1CS constraints
`// Let` A `constraints be a` $[M \times N]$ `matrix`
Return QAP_A, QAP_B, QAP_C **for** $i \in [1 : N]$ **do**
  $A_i \leftarrow$ be the values of i'th columns of all A constraints
  $F_i \leftarrow$ be the polynomial that goes through $(1, A_1), (2, A_2), \ldots, (N, A_N)$ points
  Add Coefficients of F_i to QAP_A
Repeat above steps to create QAP_B and QAP_C
Return QAP_A, QAP_B, QAP_C

Algorithm 5. Generate zero knowledge proof polynomial.

input : QAP_A, QAP_B, QAP_C, solution vector s
output: zero knowledge proof as function t(x)
for $i \in [1 : length(s)]$ **do**
  Multiply Coefficients of QAP_A's i'th polynomial with $s[i]$
  Multiply Coefficients of QAP_B's i'th polynomial with $s[i]$
  Multiply Coefficients of QAP_C's i'th polynomial with $s[i]$
Add all polynomials of QAP_A into one polynomial A
Add all polynomials of QAP_B into one polynomial B
Add all polynomials of QAP_C into one polynomial C
Generate the polynomial $t(x) = AB - C$
Return $t(x)$

Algorithm 6. Check zero knowledge proof.

input : Zero knowledge proof $t(x)$
output: Verify $t(x)$
Generate a polynomial $z = (x-1)(x-2)\ldots(x-n)$ where n is the length QAP polynomials
if $t(x)/z(x)$ *has no reminder* **then**
  Accept the zero knowledge proof
else
  Do not accept the zero knowledge proof

Summary of DNN Verification. A summary of the proposed DNN verification with QAP-based zkSNARK is shown in Table 2. It is as follows:

1. The Verifier sends inputs to the DNN model to the verifier.
2. For each DNN layer, the prover generates the zero-knowledge proof for the DNN functions of that layer and sends such proof to the verifier.
3. The verifier checks if the zero-knowledge proof is correct and $d - KCA$ check is valid.

5 Analysis

Theorem 1. *DNN verification protocol is correct and privacy-preserving.*

Proof. 1. Note that, we need to prove that each DNN function value in the set $\mathbb{S}$ is not zero for a given value of x(calculated using Eq. 3). Note that in Eq. 4, we construct $\mathbb{S}$ as the multiplication of all such functions and check if $\mathbb{S}$ is not zero at the given value of x. Multiplication of these functions allows us to check all such polynomials (one polynomial for DNN function) at once as if one such polynomial becomes 0 then the entire $\mathbb{S}$ will become zero. Further, we need to prove that DNN function values for $\mathbb{T}$ are zero. We created $\mathbb{T}$ as an addition of polynomials (one polynomial for each DNN function). Hence if the valuation of one such polynomial is not zero then $\mathbb{T}$ will not be zero. Additionally, we assumed ReLU as the activation function of the DNN and hence DNN function values can not be negative. Thus it is not possible that one DNN function value is positive and another is negative to create the addition of all DNN polynomials as zero.

2. Note that the prover does not reveal DNN function parameters or polynomials representing DNN functions. Instead, the prover only sends mappings to elliptic curve points for all such polynomials as mentioned in step 7 in Sect. 4.4. The verifier ensures that the prover knows such polynomials using the d-KCA test as mentioned in step 8 in Sect. 4.4. Hence the DNN model owner preserves the privacy of its DNN model parameter data.

Theorem 2. *The proposed DNN verification method requires zero-knowledge proof checking of $2k$ polynomials of degree at most n where k is the number of DNN layers and n is the number of nodes in each DNN layer.*

Proof. We will quantify the cost of DNN verification in terms of zkSNARK-based verification of a polynomial. We need to verify two polynomials of degree n at each layer. Hence we need to check $2k$ polynomials where k is the number of layers of the DNN model.

6 Conclusion

In this paper, we used a QAP-based zkSNARK to verify the correct execution of DNN functions. We proved that the proposed verification method is correct and efficient as it takes only $2k$ verification where k is the number of DNN layers. In the future, we will extend this verification result to develop a secure MLaaS service platform.

Acknowledgements. This publication has emanated from research supported by grants from Science Foundation Ireland (SFI) under Grant Numbers 21/FFP-A/9174 (Sustain), 16/RC/3835 (VistaMilk) and 12/RC/2289_P2 (Insight). For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

1. Chabanne, H., Keuffer, J., Molva, R.: Embedded proofs for verifiable neural networks. Cryptology ePrint Archive, Paper 2017/1038 (2017). https://eprint.iacr.org/2017/1038
2. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
3. Fischer, M., Sprecher, C., Dimitrov, D.I., Singh, G., Vechev, M.: Shared Certificates for Neural Network Verification. In: Shoham, S., Vizel, Y. (eds.) CAV 2022. LNCS, vol. 13371, pp. 127–148. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13185-1_7
4. Ghodsi, Z., Gu, T., Garg, S.: Safetynets: verifiable execution of deep neural networks on an untrusted cloud. CoRR http://arxiv.org/abs/1706.10268 (2017)
5. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_19
6. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology - EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
7. Isac, O., Barrett, C.W., Zhang, M., Katz, G.: Neural network verification with proof production. In: 2022 Formal Methods in Computer-Aided Design (FMCAD), pp. 38–48 (2022). https://api.semanticscholar.org/CorpusID:249240518
8. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
9. Li, B., et al.: Trustworthy AI: from principles to practices. ACM Comput. Surv. **55**(9), 1–46 (2023). https://doi.org/10.1145/3555803
10. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology - ASIACRYPT 2013. LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_3
11. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: a survey on stealing machine learning models and defences. ACM Comput. Surv. **55**(14s), 1–41 (2023). https://doi.org/10.1145/3595292
12. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252 (2013). https://doi.org/10.1109/SP.2013.47
13. Sun, H., Bai, T., Li, J., Zhang, H.: zkdl: efficient zero-knowledge proofs of deep learning training. Cryptology ePrint Archive, Paper 2023/1174 (2023). https://eprint.iacr.org/2023/1174
14. Tanuwidjaja, H.C., Choi, R., Baek, S., Kim, K.: Privacy-preserving deep learning on machine learning as a service-a comprehensive survey. IEEE Access **8**, 167425–167447 (2020). https://doi.org/10.1109/ACCESS.2020.3023084
15. Toumi, N., Bagaa, M., Ksentini, A.: Machine learning for service migration: a survey. IEEE Commun. Surv. Tutor. **25**(3), 1991–2020 (2023). https://doi.org/10.1109/COMST.2023.3273121
16. Wu, C., Lib, Y.F., Bouvry, P.: Survey of trustworthy AI: a meta decision of AI (2023)
17. Zhang, B., Lu, G., Qiu, P., Gui, X., Shi, Y.: Advancing federated learning through verifiable computations and homomorphic encryption. Entropy **25**(11) (2023). https://doi.org/10.3390/e25111550. https://www.mdpi.com/1099-4300/25/11/1550