Verification of Deep Neural Networks with KGZ-Based zkSNARK

Subhasis Thakur(✉) and John Breslin

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

Abstract. Verification of a deep neural network is required as large DNN models are used in machine learning as a service procedure where the server providing a classification service may be insecure and provide invalid classifications. A verification of deep neural networks in a machine learning as a service paradigm requires verification of function evaluation for all functions of a DNN model given a specific input where the service provider and the server do not want to reveal the DNN model to the client. In this paper, we investigate the privacy-preserving verification problem of the DNN model with zero-knowledge proofs. We have developed a KGZ polynomial commitment scheme based on zero-knowledge proof for such DNN verification. We present an efficient DNN verification using KGZ zero-knowledge proof. We have developed a batch-processing algorithm that can significantly reduce the number of function evaluation verifications. We also prove that a malicious server may not manipulate the proposed verification protocol.

Keywords: Deep neural networks · Zero knowledge proof · Polynomial commitment schemes

1 Introduction

Deep Neural Networks (DNN) can be used in safety-critical systems where trust in DNN models is important for the system's safety. DNN models are made with significant data and computational resources. Usually, classification tasks for large DNN models are outsourced to reduce the cost of using DNN classifications in systems. Machine learning as a service (MLaaS) facilitates such outsourcing of DNN classification tasks. In such a machine learning as a service procedure, 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 (see Table 1) and classification results are sent back to the client. However, in this MLaS 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.

K. Arai (Ed.): IntelliSys 2024, LNNS 1068, pp. 79–95, 2024.
https://doi.org/10.1007/978-3-031-66336-9_6

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 (see Table 2) 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. The service provider in MLaS can not share the DNN model with the client as proof of DNN model execution due to privacy issues (the service provider wants to keep its DNN model secret) and cost (it may be too costly for the client to execute the DNN model). 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. Our main results are as follows:

1. We have developed a method to use polynomial commitment scheme-based verification of DNN model execution.
2. We have developed a batch processing method for polynomial commitment scheme-based verification to reduce the cost of verifying DNN model execution. The batch processing method combines several verification processes into one verification process to reduce the cost of proving and verifying DNN.
3. We have developed a method for batch processing for DNN verification that prevents a malicious prover who provides a wrong evaluation of DNN functions.

The paper is organized as follows. In Sect. 2 we discuss related literature, in Sect. 3 we describe the verification problem of DNN model execution, in Sect. 4 we present the DNN model verification procedure, in Sect. 5 we present a batch processing protocol for efficient DNN model verification, in Sect. 6 we present an analysis of the proposed DNN model execution, and we conclude the paper in Sect. 7.

2 Related Literature

MLaas was previously investigated and many architectures for MLaaS were developed [14]. Privacy and security are important problems for MLaaS. In [13] 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 study [10], 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 study [12] have developed a scalable method to verify deep neural networks during the training phase. In study [4], the authors have developed a protocol to

verify the execution of neural networks using an interactive sum check protocol. In study [6] the authors have developed a method to verify the machine learning model to be used in safety-critical systems. In study [1] uses zkSNARK to verify deep learning models kept in insecure data servers. In study [16] the authors have explored verifiable computation for federated machine learning. In study [3] developed a method for shared certificates to reduce the cost of verifying neural networks. The research in [8,15] 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. We use the KGZ polynomial commitment scheme developed in [7]. Other constructions of zkSNARKs developed in study [5,9,11] use quadratic arithmetic program-based construction of zkSNARK.

Our contributions in this paper improve the state of the art in DNN verification in the following directions:

1. We proposed a KGZ polynomial commitment scheme-based DNN verification with a constant proof size and efficient verification process.
2. We have proposed a new batch-processing protocol for DNN verification that significantly reduces the number of polynomial verifications needed to verify the DNN.
3. Our proposed DNN verification algorithm prevents a malicious service provider from misusing the batch aggregation process for DNN verification.

3 Problem Statement

A DNN model $< w^x_{i,j}, b^x_{i,j} >$ has the following elements:

- It has m input nodes, i.e., input data is a m-element array.
- It has k layers $\{L_i\}$ and each layer L_i has m nodes.
- The DNN is fully connected, i.e., any node $v_x \in L_i$ has edges to all nodes in the layer L_i.
- It classifies the input data into z values.
- A node in layer L_i is denoted as $v_{i,j}$ where $j \in \{1, \ldots, m\}$. $v_{i,j}$ has m edges (one from each node in the layer L_{i-1}). The weight of the edge from $v_{i-1,x}$ to the node $v_{i,j}$ is denoted as $w^x_{i,j}$ and bias of the node $v_{i,j}$ is denoted as $b_{i,j}$.
- We assume that $f(x)$ is the DNN function for each node where the input to the activation function is the value $\sum_x w^x_{i,j} * a^x_{i,j} + b_{i,j}$ where $a^x_{i,j}$ is the value from node $v_{i-1,x}$ to $v_{i,j}$.

Verification for the above DNN model requires the following elements:

1. The prover computes all DNN functions for $layer_i$ and the verifier needs to check if such evaluations are correct.

2. The prover uses the evaluations of $layer_i$ as input to the next DNN layer and computes the DNN functions for the next layer.

The DNN model is kept and executed by the prover and the verifier provides an input to the DNN model and asks for proof of DNN model execution. The challenges for DNN verification are as follows:

1. A DNN model can contain a large number of nodes, weights, and biases. It is difficult to transform such large arrays into functions and computation for generating for correct execution of such functions can be difficult and verification of such functions can also be difficult. We need a compact representation for example represent a large degree DNN function as an elliptic curve point.
2. It is costly for the verifier to verify the outcomes of all DNN functions. Hence we need a reduce the number of verifications of all functions.
3. The DNN verification must be computed without the knowledge of the DNN model to the verifier.

4 KGZ-Based Verification of a DNN Model

4.1 Preliminaries

We will call the service provider of an MLaaS as the prover and the client who wants to verify a DNN model as the verifier. We assume that both the prover and the verifier agree on the following bilinear pairing-based elliptic curves as follows:

Table 1. Symbols used in verification of DNN models

Symbols	Meaning
$\mathbb{G}_1, \mathbb{G}_2$	Elliptic curves with bilinear pairing
$\mathbb{E}()$	Bilinear pairing function over $\mathbb{G}_1, \mathbb{G}_2$
g_1, g_2	Generators of $\mathbb{G}_1, \mathbb{G}_2$
p_1, p_2	Prime numbers for $\mathbb{G}_1, \mathbb{G}_2$
$Add(), Mul()$	Adds elliptic curve points and multiplies elliptic curve points with integers

4.2 DNN Representation

A DNN model with k layers and n nodes in each layer is as follows:
In the above DNN there are $n * k$ functions where each function has degree n if we represent each function as follows:

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

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

In	$Layer_1$	$Layer_1$		$Layer_k$
a_1	$y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1$	$y_1^2 = b_1^2 + \cup_{i=1}^{n} y_i^1 w_{i,1}^2$	...	$y_1^k = b_1^k + \cup_{i=1}^{n} y_i^{k-1} w_{i,1}^k$
a_2	$y_2^1 = b_2^1 + \cup_{i=1}^{n} a_i w_{i,2}^1$	$y_2^2 = b_2^2 + \cup_{i=1}^{n} y_i^1 w_{i,2}^2$	...	$y_2^k = b_2^k + \cup_{i=1}^{n} y_i^{k-1} w_{i,2}^k$
...	...	...	...	...
...	...	...	...	...
a_n	$y_n^1 = b_n^1 + \cup_{i=1}^{n} a_i w_{i,n}^1$	$y_n^2 = b_n^2 + \cup_{i=1}^{n} y_i^1 w_{i,n}^2$	...	$y_n^k = b_n^k + \cup_{i=1}^{n} y_i^{k-1} w_{i,n}^k$

where, x represent the set of inputs $a_1, \ldots, a_n$. A proof of execution for the DNN will consist of the following:

1. A **commitment to the function** $f(x)$ that can hide the weights and biases to the verifier.
2. **Value of the function evaluation**, i.e., value of $y_1^1 = f(x)$ where x represents the set of inputs. For functions representing nodes from layer 2 to layer k, inputs will be the value of function evaluation such as y_1^1.
3. A **proof of evaluation** that can prove that y_1^1 indeed the result of executing $f(x)$ with the input $a_1, \ldots, a_n$.

A basic protocol for executing a proof of execution is as follows:

1. There are two parties, the prover(who has the DNN model) and the verifier(who has the input to the DNN model). The verifier wants to verify that given the output of the DNN model for input data, the output is generated by executing the DNN functions(i.e., weighted aggregation of inputs and biases at each node).
2. The prover sends the commitment to the DNN functions to the verifier.
3. The verifier sends the input data to the prover.
4. The prover sends the value of function evaluation for the DNN function at $layer_1$ and proof of such function evaluation to the verifier.
5. The verifier checks the correctness of such value of function evaluation using the DNN function commitment for $layer_1$ and proof of function evaluation for $layer_1$.
6. Next, the prover uses the value of function evaluations as inputs to $layer_2$ functions.
7. Next, steps 4 and 5 are repeated until the last layer.
8. The relation between the last layer and classification result is known to the verifier, i.e., each DNN function at the last layer corresponds to a classification result, and such a mapping is known to the verifier.

4.3 Setup for 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:

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

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:

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

4.4 Commitment to the DNN Functions

For the function $f(x) = y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1$, the prover creates a commitment as follows:

1. Let $F(x) = b_1^1 + w_{1,1}^1 x + w_{2,1}^1 x^2 + \cdots + w_{n,1}^1 x^n$, i.e., coefficients of $F(x)$ is the set of weights and bias for a node in the DNN.
2. We will use KGZ polynomial commitment scheme to generate the commitment for $F(x)$.

$$Commit(F_1^1(\tau)) = Add(Mul(b_1^1, \mathbb{P}_1), Mul(w_{1,1}^1, p_2^1), \ldots, Mul(w_{n,1}^1, p_n^1)) \quad (3)$$

where, the *Add* function adds a set of elliptic curve points. Hence for each DNN layer, the prover creates n elliptic curve points for the elliptic curve $\mathbb{G}_1$, and in total, it creates $n * k$ such points. The set of commitments for layer i will be denoted as the set $\cup_{j=1}^{n} Commit(F_j^i(x))$.

4.5 Function Evaluation

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 $b_1^1, a_1, \ldots, a_n$. The prover and the verifier generate a number x as follows:

$$\begin{aligned}
x &= a_1 \\
x^2 &= a_2 \\
x^3 &= a_3 \\
&\cdots\cdots \\
x^n &= a_n \\
x + x^2 + x^3 + \cdots + x^n &= a_1 + a_2 + \cdots + a_n \\
x + x^2 + x^3 + \cdots + x^n - (a_1 + a_2 + \cdots + a_n) &= 0 \\
\frac{1}{1-x} &= (a_1 + a_2 + \cdots + a_n) \\
\frac{1}{(a_1 + a_2 + \cdots + a_n)} &= 1 - x \\
1 - \frac{1}{(a_1 + a_2 + \cdots + a_n)} &= x
\end{aligned} \quad (4)$$

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.

4.6 Proof of Evaluation

Note that, value of evaluation for DNN functions $\{F_i^1(x)\}$ at $layer_1$ is the set $\{y_i^1\}$. For each function F_i^1, the prover creates a proof of evaluation as a point in the elliptic curve in $\mathbb{G}_1$ as follows: Let,

$$\begin{aligned} Q_1^1(x) &= \frac{F_1^1(x) - y_1^1}{x - a} \\ &= r_0 + r_1 x + r_2 x^2 + \cdots + r_n x^n \end{aligned} \tag{5}$$

$$Commit(Q_1^1(\tau)) = Add(Mul(r_0, p_1^1), Mul(r_1, p_2^1), \ldots, M(r_n, p_n^1)) \tag{6}$$

The set of proof of evaluations for $layer_1$ will be denoted as the set

$$\{Commit(Q_i^1(\tau))\}.$$

4.7 Verification of Proof of Evaluation

The verifier has the following values:

1. The set of commitment to the DNN functions at $layer_1$ as $\cup_{j=1}^{n} Commit(F_j^i(x))$.
2. The set of value of DNN function evaluation at $layer_1$ as $\{y_i^1\}$.
3. The set of commitment for proof of function evaluation to the DNN functions at $layer_1$ as $\{Commit(Q_i^1(\tau))\}$.

The verifier checks the validity of the commitment to the function evaluations by checking if the following equivalence holds:

$$\mathbb{E}(Commit(Q_i^1(\tau)), (\tau - x)) \equiv \mathbb{E}(Commit(F_j^i(x)) - y_i^1) \tag{7}$$

Note that the above equation checks if the commitment to the proof of function evaluation is correct, i.e., the polynomial division for Eq. (5) has no reminder. Note that verification requires multiplication of two elliptic curve points and hence we used bilinear pairing-based elliptic curves for this purpose.

4.8 Iteration for Complete DNN Execution Proof

After the completing the procedure for proof of execution for $layer_1$, proof of execution for all remaining layers are also checked as follows:

1. Outcomes from $layer_1$ DNN functions, i.e., $\{y_i^1\}$ act as the input to $layer_2$.
2. x is calculated from $\{y_i^1\}$ using the method shown in Sect. 4.5.
3. The prover again generates the value for function evaluation and computes the proof of evaluation using the method shown in Sect. 4.6.
4. This process continues until the last layer. After verification of the last layer, the verifier can check if the value of the DNN function corresponds to the classification result given by the prover.
5. Algorithms 1 and 2 illustrate the proof generation and verification procedure.

Algorithm 1: Protocol 1 (Prover): Proof Generation for DNN function Evaluation

Data: $\{F_i^j\}$ be a set of n DNN functions for $layer_j$, set of inputs $(a_1, \ldots, a_n)$
Result: $\{Commit(F_i^j(\tau))\}$ as DNN function commitment, $\{y_i^j\}$ as value of DNN functions, $\{Commit(Q_i^j(\tau))\}$ as proof of DNN function evaluation
begin
  for $i \in [1:n]$ **do**
    $Commit(F_i^1(\tau)) \longleftarrow$ using Eq. (3).
  $x \longleftarrow$ computed from $(a_1, \ldots, a_n)$ using Eq. (4). **for** $i \in [1:n]$ **do**
    $y_i^j \longleftarrow f_i^j$.
  for $i \in [1:n]$ **do**
    $Commit(Q_i^j(\tau)) \longleftarrow$ using Eq. (6).
  Return $\{Commit(F_i^1(\tau))\}$, $\{y_i^j\}$, $\{Commit(F_i^1(\tau))\}$

Algorithm 2: Protocol 1 (Verifier): Verification of DNN function Evaluation

Data: $\{Commit(F_i^1(\tau))\}, \{y_i^j\}, \{Commit(F_i^1(\tau))\}$
Result: Yes / No
begin
  if Eq. (7) holds **then**
    Return(Yes)
  else
    Return(No)

5 Batch Processing for DNN Verification

5.1 Batch Processing with Aggregation of DNN Functions

The above procedure for DNN proof of execution requires the verifier to check every DNN functions at every layer, i.e., it needs to check $n * k$ DNN function

execution verification. Using batch processing (shown in ALgorithm 3 and 4) we can reduce the number of such checks. If we need to evaluate one function for a set of inputs then we can follow the this procedure:

1. Let the function is $f(x)$ and x is $a_1, a_2, \ldots, a_n$ such that $f(a_i) = y_i$.
2. We can create a polynomial $I(x)$ such that $I(a_i) = y_i$ using lagrange interpolation.

$$I(x) \leftarrow Langarage(a_1, \ldots, a_n) \tag{8}$$

3. Note that the polynomial $f(x) - I(x) = 0$ at $x = a_1, a_2, \ldots, a_n$.
4. We can create a polynomial

$$Z(x) = (x - a_1)(x - a_2)\ldots(x - a_n). \tag{9}$$

5. To prove that $f(x) - I(x)$ is zero at $a_1, a_2, \ldots, a_n$, we have to check the following function exists:

$$Q(x) = \frac{f(x) - I(x)}{Z(x)}. \tag{10}$$

6. $Q(x)$ exists if $\frac{f(x)-I(x)}{Z(x)}$ has no reminder. The prover sends commitment to $Q(x)$ and the verifier checks if there is no reminder. It can do so by checking the following equivalence in Eq. (7) using bilinear pairing-based elliptic curves.

Algorithm 3: Batch Processing for one DNN function (Prover): Proof Generation for one DNN function Evaluation at multiple point

Data: A DNN function f_i^j, a set of inputs $(a_1, \ldots, a_n$
Result: Generate proof of function evaluation
begin
  Generate $I(x)$ from Eq. (8)
  Generate $Z(x)$ from Eq. (9)
  Generate $Q(x)$ from Eq. (10)
  Return Evaluation of $Q(x)$ at point τ for elliptic curve $\mathbb{G}_1$.

For DNN function execution verification, we need to verify the value of a set of functions for the same input, i.e., for each layer of the DNN there is a set of DNN functions, and each function takes the same input. This gives us the problem of creating the function $I(x)$ is that its coefficients are inf as $I(x)$. We can mitigate this problem as follows:

1. We combine polynomials for all DNN functions of one layer into one polynomial by adding the polynomials together.
2. We randomly choose a set of $n - 1$ input values for the DNN functions.

Algorithm 4: Batch Processing for one DNN function (Verifier): Verification for one DNN function Evaluation at multiple point

Data: Commitment of batch processed DNN function f_i^j as $Commit(Q(x))$, evaluation of F_i^j at $(a_1, \ldots, a_n)$
Result: Yes / No
begin
  if *If the equivalence relation in* Eq. (11) holds **then**
    Return(Yes)
  else
    Return(No)

3. We evaluate the DNN functions for these n_1 inputs.
4. We add the DNN function outcome for each of such $n-1$ points as the outcome of the aggregated function.
5. The outcome of the aggregated function for classification input is calculated as the summation of the individual function outcomes.
6. Figure 1 shows an example of DNN function aggregation.
7. Using such aggregated polynomial for all DNN functions of a layer we calculate the polynomial commitment, commitment to proof of evaluation, and I(x), Z(x) in Eqs. (8) and (9).

Our approach for batch processing is as follows:

1. Let $Poly_1, Poly_2, Poly_3$ are three polynomials such that $Poly_i$ goes through the points $(v_1^i, a_1^i), (v_2^i, a_2^i) \ldots, (v_n^i, a_n^i)$. Additionally, value of these polynomials at a_0 is the set a_1, a_2, a_3.
2. Let $Poly_x = Poly_1 + Poly_2 + Poly_3$.
3. At a_0, value of $Poly_x$ is $a_1 + a_2 + a_3$.

This construction can be used for batch processing of the DNN functions as follows:

1. Let the DNN functions are the set $\{F_i^1\}$ where each function F_i^1 is a n degree polynomial. The value of evaluation for the DNN functions is the set $\{y_i^1\}$ for x_0 (calculated as section). We prove that an aggregation of $\{F_i^1\}$ at x_0 is $\sum y_i^1$.
2. Each function F_i^1 goes through $n-1$ points $\{x_i^j, Y_i^j\}$.
3. We can combine all DNN functions as the polynomial $F = \sum F_i^1$. The valuation of F is as follows:

$$(x_0, \sum y_i^1), \{x_i^j, Y_i^j\}$$

4. We find the polynomial that goes through the points $(x_0, \sum y_i^1), \{x_i^j, Y_i^j\}$ as $I(x)$.
5. Note that the function $F - I(x)$ is zero at points $(x_0), \{x_i^j\}$. Let $Z(x) = (x - \sum y_i^1)\{(x - x_i^j)\}$.

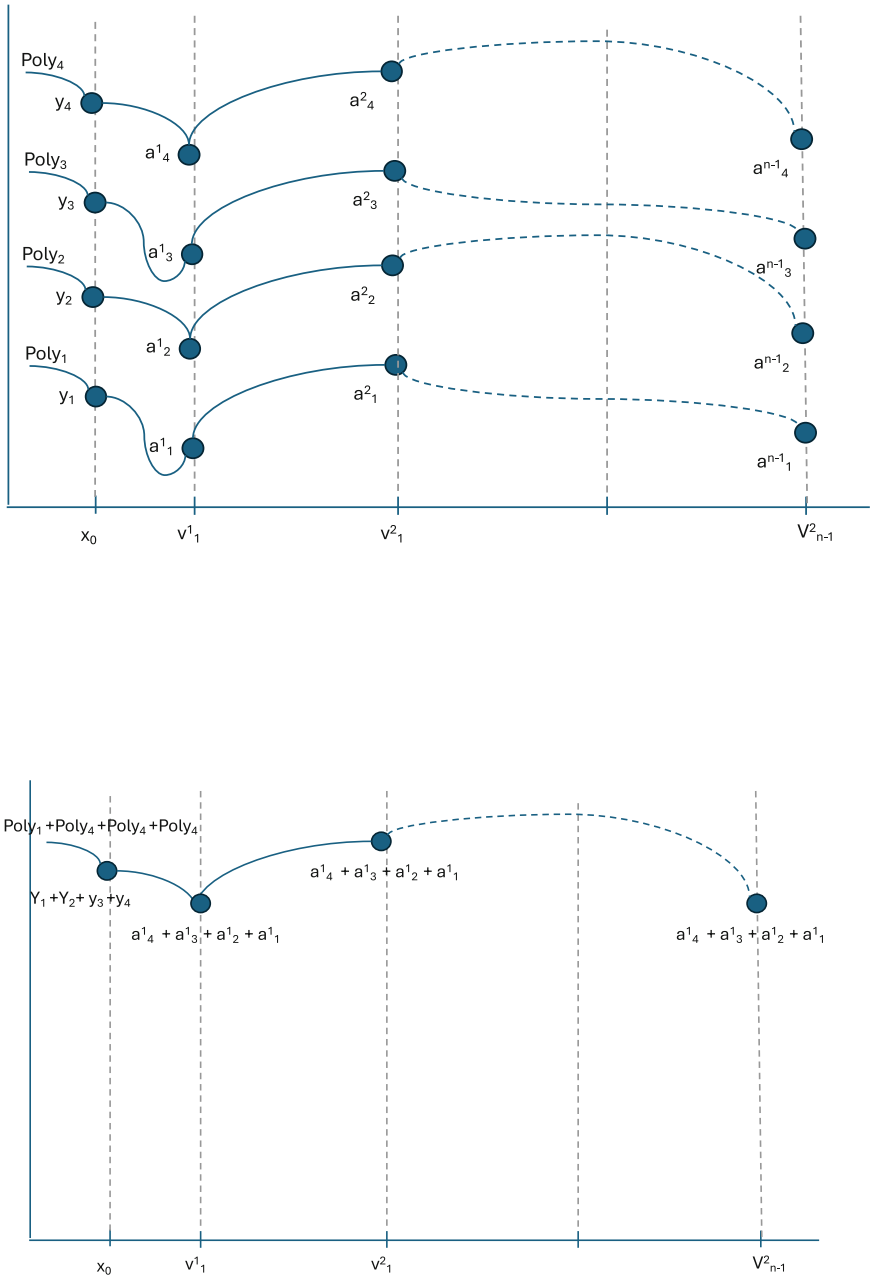


Fig. 1. Batch processing of DNN verification as all DNN functions are aggregated and verifier verifies such an aggregated DNN function

6. Now we can find the function:

$$Q(x) = \frac{F(x) - I(x)}{Z(x)}.$$

7. Now we can prove that $Q(x)$ exists by using KGZ zkSNARK as shown in Algorithm 1 and 2.

5.2 Problem with DNN Function Aggregation for Batch Processing

A malicious prover has a valid set of DNN function evaluation values for layers and using these values it can manipulate the aggregated verification of DNN functions as follows (see Fig. 2):

1. It creates random outcomes for DNN functions for all layers except the last layer such that the summation of such outcomes for each layer matches the sum of a valid outcome.
2. It creates the function shown in Eq. (8) with a set of valid roots one for each DNN function.
3. As the verifier checks the summation of all DNN functions in each layer, it will accept the proof of function execution.
4. It will create the $(n-1)$'th layer's outcome such that (a) its summation is the same as a valid evaluation for this layer and (b) these outcomes may be calculated to make the value of one DNN function in the final layer as the maximum, to make the corresponding classification valid.

5.3 Modified DNN Function Aggregation

The above protocol for batch processing allows the verifier to check if the sum of all DNN function evaluations at each layer is valid. However, a malicious service

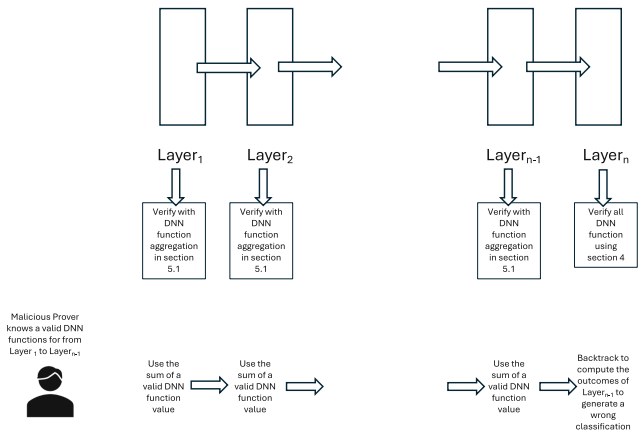


Fig. 2. Procedure for attacking batch-processing-based DNN function verification

provider may change the individual DNN function values while keeping the sum of such evaluation valid. Hence the verifier needs to check if all individual DNN function values are valid. We solve this problem as follows:

1. The verifier randomly chooses a number between 1 to n as h.
2. From the set of DNN function evaluation value $\{y_i^1\}$, we create a set of h pairs as follows:

$$\left(\sum y_1^1, \sum_{i=2}^{n} y_i^1\right), \left(\sum_{i=1}^{2} y_i^1, \sum_{i=3}^{n} y_i^1\right), \ldots, \left(\sum_{i=1}^{n-1} y_i^1, \sum_{i=n} y_i^1\right)$$

3. The verifier will use 1-to-n oblivious transfer to get two set of pairs for a random value d as follows:

$$\left(\sum_{i=1}^{d} y_i^1, \sum_{i=d+1}^{n} y_i^1\right).$$

4. Now the verifier creates three polynomials as follows:

$$Poly_x = F_1^1 + F_2^1 + \cdots + F_d^1 \tag{11}$$

$$Poly_y = F_{d+1}^1 + F_{d+2}^1 + \cdots + F_n^1 \tag{12}$$

$$Poly_z = F_1^1 + \cdots + F_n^1 \tag{13}$$

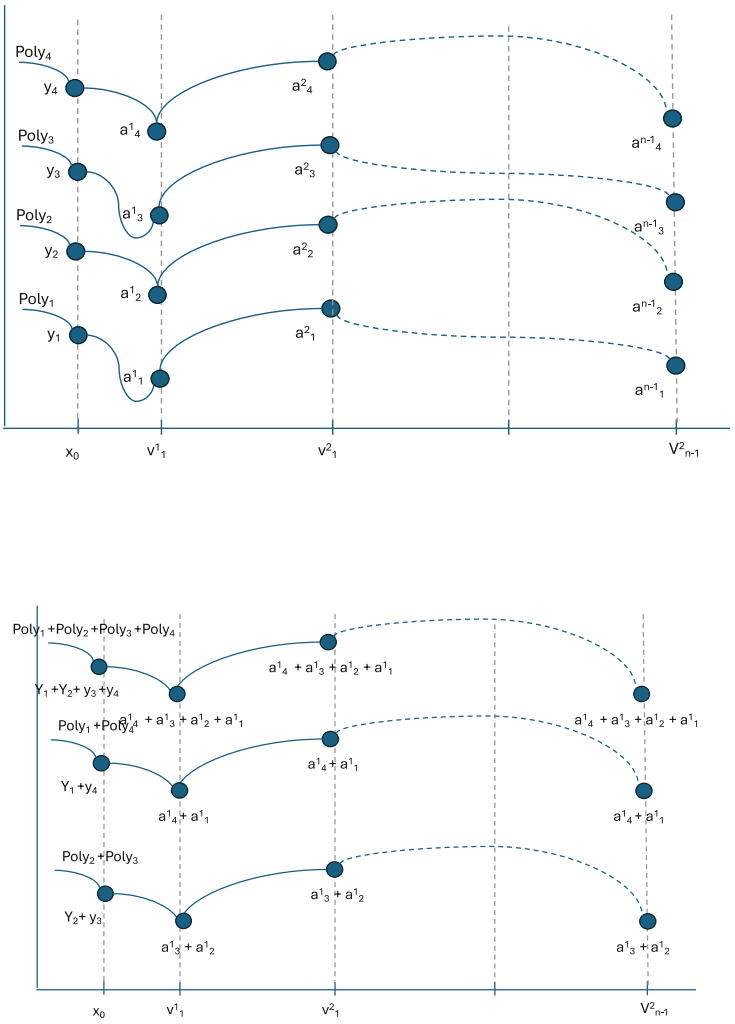
5. Now, the verifier will verify that $Poly_x$ goes through $\sum_{i=1}^{d} y_i^1$ at point x_0, $Poly_y$ goes through $\sum_{i=d+1}^{n} y_i^1$ at point x_0, and $Poly_z$ goes through $\sum_{i=1}^{n} y_i^1$ at point x_0 using KGZ protocol.

A detailed modified batch processing protocol for DNN verification is as follows (see Fig. 3).

6 Analysis

Theorem 1. *The probability that a malicious service provider manipulates the outcome of each layer is* $(\frac{n-1}{n})^z$.

Proof. Note that the modified batch-processing protocol uses oblivious transfer protocol to partition the set of DNN functions into two sets and it is not possible for the prover to know this partition. Also, if the prover changes one DNN function evaluation value then it must change another such value to keep the total of DNN function evaluations constant. Further, such modified DNN function indices must be adjacent to each other to maximize the chances that both functions reside in the same group as the verifier partitions the DNN functions into two groups. If only two DNN function values are modified by the prover then the probability that indices of both DNN functions are in the same group is $(n-1)/n$. If there are z indices of such modifications then the probability that indices of all DNN functions are in the same group is $(\frac{n-1}{n})^z$.



Modified Batch processing of DNN functions

Fig. 3. It partitions the DNN functions into two group and check aggregation of DNN functions for these two groups in the partition and aggregation of all DNN functions

Table 2. Overall DNN verification protocol

	Prover	Verifier
1	Compute $y = y_1^1 \ldots, y_n^1$ as DNN function evaluation for $layer_1$ Compute n pairs of y as $(\sum_{i=1}^{d} y_i^1, \sum_{i=d+1}^{n} y_i^1)$. Compute n pairs of commitment to proof of evaluation $Q^1(x) = \frac{F^1(x)-I^1(x)}{Z^1(x)}, Q^2(x) = \frac{F^2(x)-I^2(x)}{Z^2(x)}$. where $Q^1(x)$ is calculated for first d DNN functions and $Q^2(x)$ is calculated for the remaining functions as shown in Eq. (10).	
2	Generates a random positive integer u and elliptic curve points $U = Mul(u, g_1)$ and $T = Mul(u, U)$. It sends U to the verifier.	
3		Generate a random number c from the set of positive integer $[1 : n]$ Generate random integer b and elliptic curve point $R = Add(Mul(c, U), Mul(b, g_1))$ and sends R to the prover.
4	It generates a set of n elliptic curve points $\{M_j\}$ $M_j = Sub(Mul(u, R), Mul(j, T))$.	
5	It encrypts the DNN function value set $(\sum_{i=1}^{d} y_i^1, \sum_{i=d+1}^{n} y_i^1)$ and $\{Q^1(x), Q^2(x)\}$ with the keys $\{M_i\}$, i.e. i'th pair of function value and proof of evaluation is encrypted with the key M_i. It sends these encrypted values to the verifier.	
6		The encrypted value for $(\sum_{i=1}^{d} y_i^1, \sum_{i=d+1}^{n} y_i^1)$ and $(Q^1(x), Q^2(x))$ is decrypted with the key $Key_d = Mul(b, U)$.
7		Use algorithm 4 and 5 to verify three sets of functions (a) first d DNN functions, (b) remaining DNN functions, (c) all DNN functions (using the batch processing procedure shown in previous section)
8	Repeat for each layer except last layer	
8	Use algorithm 1 to generate proof of execution for the last layer	Use algorithm 2 for verification of last layer

Theorem 2. *The total size of proof verification in batch processing is $n/3$ times less than non-batch processing.*

Proof. Without batch processing, the verifier needs to engage in verification for all DNN functions. There are k layers and each layer has n DNN functions. Hence it needs to verify $k * n$ DNN functions. With the batch processing, the

verifier needs to verify only three functions and hence it will verify $3*k$ functions. Hence it will reduce the number of function verifications by a factor of $n/3$.

7 Conclusion

In this paper, we have developed a method for DNN verification with polynomial commitment scheme-based zkSNARK. Our proposed DNN verification is efficient as it only requires $3 * k$ polynomial verifications to verify a DNN where k is the number of layers in the DNN. In the future, we will explore QAP-based zkSNARKs for DNN verification. We also intend to use this DNN verification procedure for trustworthy federated machine learning.

Acknowledgment. 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: Proceedings on Advances in Cryptology—CRYPTO'86, pp. 186-194. Springer-Verlag, Berlin, Heidelberg (1987)
3. Fischer, M., Sprecher, C., Dimitrov, D.I., Singh, G., Vechev, M.: Shared certificates for neural network verification. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification, pp. 127–148. Springer International Publishing, Cham (2022)
4. Ghodsi, Z., Gu, T., Garg, S.: Safetynets: verifiable execution of deep neural networks on an untrusted cloud (2017). arXiv:1706.10268
5. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) Advances in Cryptology—EUROCRYPT 2016, pp. 305–326. Springer, Berlin, Heidelberg (2016)
6. 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)
7. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Advances in Cryptology—ASIACRYPT 2010, pp. 177–194. Springer, Berlin, Heidelberg (2010)
8. Li, B., Qi, P., Liu, B., Di, S., Liu, J., Pei, J., Yi, J., Zhou, B.: Trustworthy AI: from principles to practices. ACM Comput. Surv. **55**(9) (2023)
9. 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, pp. 41–60. Springer, Berlin, Heidelberg (2013)
10. 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)

11. 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)
12. 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
13. 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)
14. Toumi, N., Bagaa, M., Ksentini, A.: Machine learning for service migration: a survey. IEEE Commun. Surv. Tutor. **25**(3), 1991–2020 (2023)
15. Caesar, W., Lib, Y.-F., Bouvry, P.: A meta decision of AI, survey of trustworthy AI (2023)
16. Zhang, B., Lu, G., Qiu, P., Gui, X., Shi, Y.: Advancing federated learning through verifiable computations and homomorphic encryption. Entropy **25**(11) (2023)