zkVML: Zero-Knowledge Verifiable Machine Learning

Mohammad Bilal Aziz, Maryam Siddiqui, Ali Shah Naushad, and Jawwad Ahmed Shamsi
Dept. of Computer Science, *National University of Computer and Emerging Sciences*, Karachi, Pakistan

***Abstract*—Zero Knowledge Proofs(ZKPs) are increasingly used in the domain of privacy-preserving machine learning. However, recent works have not been widely incorporated due to the complexities of ZKPs, and they lack generalization. To address these issues, we present develop zkVML: Zero Knowledge Verifiable Machine Learning, a novel Python library that seamlessly integrates zero-knowledge proofs (ZKPs), allowing the verification of public inference over a private model. zkVML ensures privacy in both on-chain verification and off-chain computations, particularly within decentralized environments like blockchains. We design and develop several APIs, for handling complex computations present in Machine Learning models. We successfully validated our library by developing K-Nearest Neighbors(KNN), Decision Trees(DT) and Convolutional Neural Networks(CNN) models. While zkVML may incur increased execution times for complex models and large datasets, it offers significant privacy improvements alongside enhanced scalability, generalization, and control over model creation. Through high-level Python APIs abstracting complex ZKP functionalities, zkVML empowers developers to maintain flexibility and privacy control while guaranteeing data integrity. Our key contributions include generalized proof generation, user-friendly Python abstractions for easy integration, and on-chain verification via smart contracts. Future work may focus on optimizing proof generation times and extending zkVML's functionality to a wider range of algorithms.**

***Index Terms*—Zero-Knowledge Proof, Machine Learning, Verification, Blockchain**

I. Introduction

A. Applications and Challenges in Machine Learning

Artificial Intelligence (AI) applications have undergone significant advancements, with algorithms such as Decision Trees (DT), K-Nearest Neighbors (KNN), and Convolutional Neural Networks (CNN) leading the forefront of innovation. These algorithms have found widespread applications across various domains, from image recognition to recommendation systems, revolutionizing industries worldwide.

In the domain of text categorization, Soucy and Mineau (2001) presented a seminal work on a simple KNN algorithm tailored specifically for this task during the 2001 IEEE International Conference on Data Mining [7]. This contribution underscored the algorithm's efficacy in natural language processing, a crucial area where precise categorization is paramount.

Similarly, Kotsiantis (2013) provided a comprehensive overview of Decision Trees in his work published in the Artificial Intelligence Review [8]. Decision Trees offer a structured approach to decision-making and have been widely employed for their ability to handle both categorical and numerical data effectively. In the realm of image processing, Albawi, Mohammed, and Al-Zawi (2017) shed light on the intricacies of Convolutional Neural Networks (CNNs) during the 2017 International Conference on Engineering and Technology [9]. Their elucidation of CNNs' hierarchical architecture and feature extraction capabilities highlighted the pivotal role these networks play in tasks such as image classification and object detection.

In outsourced Machine Learning (ML), vulnerabilities like malicious participation, data poisoning, security backdoors, and incorrect inference results pose significant challenges to system integrity and security. Robust countermeasures are essential to ensure reliability and trustworthiness.

Malicious Participation: Threatens ML integrity by compromising computations for personal gain [28].

- Data Poisoning: Injecting poisoned data compromises model integrity, leading to erroneous results [23].
- Security Backdoors: Adversaries introduce backdoors during training, risking unauthorized access [32].
- Incorrect Inference Results: Deliberate manipulation leads to flawed decision-making and erodes trust [28].

Efforts like those of Chen, Pastro, and Raykova (2019) focus on secure computation techniques tailored for ML, while Wang and Hoang (2023) develop privacy-preserving inference pipelines. Xing et al. (2023) explore integrating zero-knowledge proofs with ML to enhance verifiability [28] [23] [32].

B. Zero Knowledge Proofs

Zero-Knowledge Proof (ZKP) is a cryptographic technique that allows one party, the prover, to demonstrate the validity of a statement or computation to another party, the verifier, without revealing any information beyond the correctness of the statement [1].

Originally introduced as interactive proof systems, ZKPs required interaction between the prover and verifier [1]. However, the Fiat-Shamir Transform provided a method to convert interactive proofs into non-interactive ones, enabling ZKPs without direct communication between the parties [2].

zk-SNARKs are a type of ZKP that allows for efficient verification of computations without the need for interaction [3]. They offer succinct proofs that can be efficiently verified, making them suitable for use in resource-constrained environments like blockchains [5]. zk-SNARKs have been utilized in various blockchain applications, including scalability solutions like zk-rollups [6].

C. Zero Knowledge Machine Learning

The integration of ZKP with Machine Learning (ML), known as zkML (Zero Knowledge Machine Learning), offers promising avenues for enhancing privacy and security in ML applications, especially in blockchain environments. zkML enables private inference on blockchain networks, allowing for the execution of ML models while preserving data privacy [11]. Projects like opML and opp/ai explore optimistic and privacy-preserving ML on blockchain platforms, leveraging the capabilities of ZKPs [12] [14].

D. Privacy issues in Blockchain

Blockchain networks inherently store data publicly, raising concerns about data privacy and confidentiality. Integrating ZKPs with blockchain can address these privacy issues by enabling private transactions and computations while maintaining the integrity and transparency of the blockchain [5] [6].

ZKPs offer a powerful tool for enhancing privacy, security, and scalability in blockchain and machine learning applications. By enabling non-interactive proofs and leveraging zk-SNARKs, ZKPs facilitate private inference and transactions on blockchain networks, paving the way for secure and privacy-preserving ML deployments in decentralized environments.

*E. **Our Contributions***

We proposed a library named Zero-Knowledge Verifiable Machine Learning, aimed at enhancing the privacy and security of machine learning models. This library incorporates zero-knowledge proofs (ZKPs) functions to ensure the verifiability of ML model operations, specifically focusing on K-Nearest Neighbors (KNN), Decision Trees, and Convolutional Neural Networks (CNN), with the aim of achieving model generalization and applicability across various ML algorithms. We provide the functions in a high-level language like Python, to abstract the intricate details of zero knowledge proofs, to ensure that model developers can seamlessly integrate it within their existing applications. We further utilize Zokrates to deploy our smart contracts to support on-chain verification. Mainly, we put forward 3 main contributions: ZKPs for different computations, Abstraction in Python language for seamless integrations, and on-chain verification using smart contracts for decentralized blockchain platforms. We incorporate all the APIs, and utilities into a fully functional, privacy preserving Python library, zkVML (Zero Knowledge Verifiable Machine Learning)

Zero Knowledge Proofs for Computations. Our main contribution is to design and develop different functions commonly found in machine learning and deep learning models. Using Zokrates, a toolbox for constructing zkSNARKs, we have designed and evaluated 10 such functions (Distance, Sort, TreeTraversal, Convolution, MaxPooling, Softmax, Relu, Argmax, MaxLabel). Each function asserts whether the computation was correctly applied. We developed and evaluated 3 different models, KNN, Decision Tree and CNN, using our functionalities.

Python Library. Since ZKP is a complex topic, we abstract the intricate details of all the functionalities of compiling, and generating zero knowledge proofs into a Python Library. Our goal is to provide developers with an interface to seamlessly integrate our library, into the inference phase of their models, to allow for verification on the user provided input. Since ZKP works on finite fields, we further provide various utilities to handle the conversions and other transformations on the inputs.

On-chain Verification. Since there is a growing use of AI services on blockchain platforms, we utilize Zokrates to provide on-chain verification of our generated proof. The verifier will provide proof to verify the transaction on the chain. The smart contract can be deployed on any Ethereum supported chain. This deployment paradigm ensures the integrity and transparency of machine learning processes while preserving data privacy and confidentiality.

We've encapsulated our APIs and their integration with blockchain in a Python library named zkVML. This simplifies the workflow for machine learning developers, eliminating the need to retrain models from scratch. With zkVML, developers gain abstracted control over model creation and function development, enhancing efficiency and flexibility in their projects. This streamlined approach enables developers to focus on project-specific requirements while seamlessly integrating machine learning models with blockchain technology.

II. Literature Review

Machine learning has rapidly evolved, drawing attention from sectors prone to data and model attacks [35]. To address these concerns, researchers have explored the use of zero-knowledge proofs (ZKPs), allowing a prover to convince a verifier of a statement's truth without revealing any information about the underlying data [19] [20]. This approach has been particularly effective in protecting sensitive model parameters during inference, as demonstrated by recent work integrating ZKPs into machine learning models for privacy-preserving and verifiable outcomes [18]. Additionally, the concept of zero-knowledge proofs has been extended to protect the privacy of both data and models during the inference process, as highlighted by studies that propose frameworks for privacy-preserving verifiable CNN schemes [17] and the application of ZKPs to decision trees and related models for proof of predictions and accuracy [25].

Several notable works, such as zkCNN [24] and Mystique [22], have applied ZKPs to enhance the security and privacy of machine learning models. zkCNN utilizes an efficient sum-check protocol and Fast Fourier Transform on convolutions to generate interactive proofs ensuring model integrity during public inference. Mystique focuses on efficient conversions using ZKPs for various layers, employing sVOLE-based zero-knowledge circuits. Frameworks like ezDPS [23], integrate differential privacy and ZKP for enhanced privacy and security, to develop more secure and privacy-preserving machine learning models. The integration of ZKPs in machine learning models, as discussed in [21], [36], and [37], further underscores the importance of interactive proofs within the domain of ZKML.

A. Zero Knowledge Proofs

Zero knowledge proofs, as introduced by Goldwasser et al., are a cornerstone in modern cryptography, enabling an interactive system where a prover and verifier, with varying computational powers, exchange information to construct a secure interactive proof system [1]. Fiat and Shamir's work [2] further advanced the field by introducing non-interactive proof systems, eliminating the need for direct interaction between the prover and verifier. This innovation not only enhances efficiency but also maintains the robustness and integrity of the proof system, significantly contributing to the development of cryptographic protocols.

zk-SNARKs (Zero Knowledge Succinct Non-interactive Argument of Knowledge) are a class of non-interactive proofs celebrated for their compact proof sizes and rapid verification capabilities, regardless of the witness size [3]. Their utility is particularly evident in scenarios with high communication overhead, where concise and efficient proofs are crucial. Blockchain technology, known for its transparency, is one such domain where zkSNARKs are highly relevant due to the need for privacy-enhancing mechanisms [5]. The scalability issue in blockchain is another critical area where zkSNARKs offer a solution, as highlighted by [6], which discusses the potential of zkSNARKs to address scalability challenges through roll-up solutions. This application underscores the role of zkSNARKs in enhancing the scalability and maturity of distributed ledger technologies.

B. ZKML using SNARKs

Recent advancements in zero-knowledge proof-based machine learning have focused on zk-SNARKs, with applications like zkDNN [31] integrating zk-SNARKs with the Halo2 proving system for deep neural network (DNN) inference. This protocol addresses dishonest behavior, ensuring robustness and viability. Similarly, zkDL [19] enhances privacy in deep learning by allowing a prover to demonstrate knowledge of model parameters and inputs without compromising sensitive information. It introduces zkReLU for ReLU activation functions, featuring copyright verification, deep learning architecture compatibility, efficient non-arithmetic ReLU operations verification, and scalability for large networks. zkDL also outperforms bit-decomposition-based proofs in proving time and proof size. Another work [16] verifies standard neural network inference, highlighting challenges and solutions in designing real-world privacy-preserving models. [13]

Ganescu et al. [15] used SNARKs to validate Transformer-based models, launching snarkGPT, a practical zkML implementation for transformers using Halo2. To optimize the heavy computations required by zero-knowledge proofs, researchers proposed optimizations. Zhang et al. [26] introduced a methodology for verifying floating-point computations, enhancing efficiency and practicality for machine learning and scientific computing applications. This approach uses commit-and-prove systems for succinct zero-knowledge proofs tailored to floating-point computations. A. Thomas [27] proposed model pruning to optimize zk-SNARKs, addressing the issue of increasing proof generation time with circuit size by pruning nodes in the hidden layer randomly. [30] Modulus Labs' benchmarking analysis [38] compared six zero-knowledge methods for machine learning, focusing on scaling challenges with deep MLPs. The study evaluated proof time and memory usage across interactive and non-interactive approaches, providing insights into their suitability for machine learning applications. Secure matrix multiplication using SPDZ [28] and [10] are highlighted as effective optimizations for privacy-preserving model training.

C. AI in decentralized environments

In recent years, there's been a big increase in using machine learning in decentralized setups. A key study, mentioned in [11], introduces a system for federated learning based on blockchain. They use something called Zero Knowledge Proofs (ZKPs) to make sure data uploaded to the network is accurate and private, which is really important for keeping data safe. Another interesting project, opML [12], combines AI with blockchain. They aim to run big models on regular computers without fancy GPUs. Also, [14] introduces a new approach called zkML to balance privacy and efficiency in AI services on blockchains. They do this by tweaking privacy settings and breaking down the machine learning model into smaller parts. [29]

With all this growth in AI on blockchains, it's clear we need tools to keep data private. Our main goal is to make a library that can create efficient zero knowledge proofs for different types of machine learning. Recent research, like [5] [6] [8], shows that zkSNARKs are great for blockchains. So, we're using zkSNARKs for our proofs to stay in line with what's working best in the field.

III. Methodology

In conducting inference over trained machine learning and deep learning models, a significant computational burden arises. To securely verify a public inference over a private model, one prevalent approach involves committing the entire model. However, this method is not without its limitations. While it may promote the generalization of Zero-Knowledge Proofs (ZKP) over machine learning models, complexities emerge, particularly when dealing with large and intricate models.

In our work, we introduce a new library that contains many common computations used in machine learning and deep learning. For instance, our library includes a secure version of the Relu function, which is commonly used in deep learning models. We use this approach to create various secure computations using Zero-Knowledge Proofs (ZKP). By putting all these computations into a Python library, we've built a system that can be used to make different types of machine learning models. Breaking down the model into smaller computations helps it work better at scale. By making the APIs available through our library, developers have complete control over how they use our secure machine learning computations. This setup offers both enhanced security and flexibility to

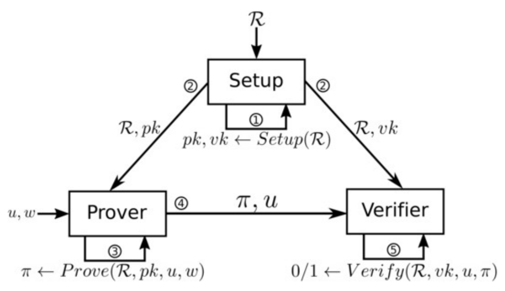
$\mathcal{R}$
Setup
②
①
$pk, vk \leftarrow Setup(\mathcal{R})$
$\mathcal{R}, pk$
$\mathcal{R}, vk$
u, w
Prover
④
π, u
Verifier
③
⑤
$\pi \leftarrow Prove(\mathcal{R}, pk, u, w)$
$0/1 \leftarrow Verify(\mathcal{R}, vk, u, \pi)$

Fig. 1. Snark Process

developers in building reliable and scalable machine learning models.

A. SNARK Construction

The approach utilizes non-interactive zkSNARKs, employing Zokrates for privacy-preserving off-chain computations of proofs. zkSNARK ensures privacy by passing input through stages to prove correctness without interaction between prover and verifier [3] [34].

Arithmetization: Computations are transformed into arithmetic circuits, then into Rank-1 Constraint Systems (R1CS).

Setup: Using gm17 as the backend, a setup phase generates proving key (pk) and verification key (vk) [4].

$$\text{Setup(C, } \lambda) \rightarrow \text{(pk, vk)},$$

Proof Generation: With computed witness 'w' and public input 'x', a proof 'Prf' is generated using pk.

$$\text{Prove(w,x,pk)} \rightarrow \text{Prf}$$

Verification: Verifiers, possessing vk and Prf, can discern honesty using Zokrates' SNARK backend. The Verify function returns a boolean indicating verification status (True/False).

$$\text{Verify(vk,prf,x)} \rightarrow \text{Boolean (True/ False)}$$

Fig. 1. by [40] encapsulates the entire process of zk-SNARKs. This satisfies all the three fundamental properties of a zero-knowledge proof:

- **Completeness**: If the initial statement C(x, w) is correct, a valid proof 'Prf' is always generated and is accepted by the verifier.
- **Soundness**: If the initial statement C(x, w) is incorrect, any proof 'Prf' generated by the prover will be deemed invalid by the verifier.
- **Zero-knowledge**: The verifier verifies the proof 'Prf' without learning anything about the private input (witness) 'w'.

B. zkVML

zkVML, the Zero-Knowledge Verifiable Machine Learning library, is designed to elevate the privacy and security of machine learning models. By integrating zero-knowledge proofs (ZKPs), zkVML ensures the verifiability of ML model operations, focusing on K-Nearest Neighbors (KNN), Decision Trees, and Convolutional Neural Networks (CNN). Our goal is to achieve model generalization and applicability across diverse ML algorithms. Our comprehensive library includes all necessary functions and utilities, providing a fully functional, privacy-preserving solution for machine learning tasks.It contains following functions. Upon successful assertion according to the specified requirements, these functions return 1; otherwise, they indicate failure.

*1) **zkApplyWeights**:* The function involves multiplying a matrix by another matrix (often referred to as a weight matrix in the context of neural networks) and adding a bias vector. The goal of this function is to prove that the operation has been performed correctly without revealing the actual values of the matrices or the bias vector.

```
1:  Input: matrix1[M1][N1], matrix2[M2][N2], bias[bc][1], pyresult[M1][N2]
2:  Output: NULL
3:  for $i$ in 0 to $M1$ do
4:      for $j$ in 0 to $N2$ do
5:          $temp \leftarrow 0$
6:          for $k$ in 0 to $N1$ do
7:              $temp \leftarrow temp + (matrix1[i][k] \times matrix2[k][j])$
8:          end for
9:          $result[i][j] \leftarrow temp + bias[i][0]$
10:     end for
11: end for
12: Assert that $result$ is equal to $py\_result$.
```

*2) **zkConv2d**:* The zkConv2D function is designed to generate zero-knowledge proofs for the output of a 2D convolution operation, allowing the verification of the correctness of the operation without revealing the input data or the filters used. Here n is input size, c is Channels, f is number of filters, fs is size of filters

*3) **zkDist**:* The function calculates the distance between a data point and a set of data points read from a CSV file, using the Zokrates library to validate the computation in zero-knowledge.

*4) **zkMaxLabel**:* This function finds the most frequent label in a given set of labels using the Zokrates library to validate the computation in zero-knowledge.

*5) **zkMaxPooling**:* The zkMaxPooling function generates zero-knowledge proofs (ZKPs) for the operation of max pooling in a neural network, which downsamples input by taking the maximum value in a region.

*6) **zkRelu**:* The function generates zero-knowledge proofs for the ReLU (Rectified Linear Unit) operation, a common activation function in neural networks. The ReLU function outputs the input directly if it is positive; otherwise, it outputs zero.

*7) **zkSoftmax**:* The function generates zero-knowledge proofs and ensures that the output values are in the range (0, 1) and sum up to 1, making them suitable for classification problems where the output represents the probability of each class.

Algorithm 1 zkConv2d

```
1:  Input: input[n][n][c], filters[f][fs][fs][c], bias[f]
2:  Output: NULL
3:  Initialize $output$ as a 3D array of size $[output\_size][output\_size][num\_filters]$ filled with zeros.
4:  for $k$ in $0$ to $num\_filters$ do
5:      for $i$ in $0$ to $output\_size$ do
6:          for $j$ in $0$ to $output\_size$ do
7:              Set $sum$ to 0.
8:              for $x$ in $0$ to $filtersize$ do
9:                  for $y$ in $0$ to $filtersize$ do
10:                     for $z$ in $0$ to $channels$ do
11:                         Add to $sum$ the product of $input[x+i][y+j][z]$ and $filters[k][x][y][z]$.
12:                     end for
13:                 end for
14:             end for
15:             Set $output[i][j][k]$ to $sum + bias[k]$.
16:         end for
17:     end for
18: end for
19: Assert that the output size is correct: $output\_size = input\_size - filter\_size + 1$.
20: Compute the expected output using the provided inputs, filters, and bias.
21: Assert that each element of the computed output matches the corresponding element of the expected output.
```

Algorithm 2 zkDist

```
1:  Input: points[rows][cols], point[test], checkDistance[rows][test]
2:  Output: 1
3:  Initialize $arr$ as a 2D array of size $[rows][test]$ filled with zeros.
4:  for $i$ in points do
5:      Calculate the difference $dx1$ between $points[i][0]$ and $point[0]$.
6:      Calculate the difference $dx2$ between $points[i][1]$ and $point[1]$.
7:      Calculate the distance as the sum of the squares of $dx1$ and $dx2$.
8:      Set $arr[i]$ to $[distance, points[i][2]]$.
9:      Assert that $checkDistance[i][0]$ is equal to $arr[i][0]$ and $checkDistance[i][1]$ is equal to $arr[i][1]$.
10: end for
11: return $1$.
```

Algorithm 3 zkMaxLabel

```
1:  Input: input[size]
2:  Output: maxLab
3:  Initialize an iterator itr to 0.
4:  for $x$ in input do
5:      Initialize a boolean flag to false.
6:      for $j$ in maxLabel do
7:          Update flag to true if the current input element matches the label in maxLabel.
8:          Increment the count of the matching maxLabel element if the input matches.
9:      end for
10:     Update maxLabel[itr] to a new MaxLabel struct if flag is false, using the changefunc to initialize.
11:     Increment itr if flag is false.
12: end for
13: Find the label with the highest count (maxCount) and store it in maxLab.
14: Assert that the last element of the input array is equal to maxLab.
15: return maxLab.
```

Algorithm 4 zkMaxPooling

```
1:  Input: val1[size1], val2[size2], filter
2:  Output: 1
3:  Initialize a boolean variable ‘result‘ to true.
4:  Initialize an index ‘j‘ to 0.
5:  Initialize a variable ‘max‘ to 0.
6:  for $i$ in $0$ to size1 do
7:      Update ‘max‘ to the maximum value between ‘res1[i]‘ and ‘max‘.
8:      if $i \mod$ filter = filter-1 then
9:          if the value at index $j$ in ‘res2‘ is not equal to ‘max‘ then
10:             Set ‘result‘ to false.
11:         end if
12:         Reset ‘max‘ to 0.
13:         Update $j$ based on the current index and the filter size.
14:     end if
15: end for
16: return the final value of ‘result‘.
```

*8) **zkTreeTraversal**:* The function demonstrates a technique for generating zero-knowledge proofs (ZKPs) that verify the correct traversal of a decision tree without revealing the tree's structure or the input data. This approach offers privacy-preserving computation for decision tree models in various applications.

C. Machine Learning Models with zkVML

In this section we will discuss the working of the machine learning models and how our library zkVML is used to construct zero knowledge verifiable models.

Algorithm 5 zkRelu

```
1: Input: val[size], threshold
2: Output: 1
3: Define a function ‘zkRelu‘ that takes an array ‘res‘ and a threshold value ‘a‘. It iterates through the elements of the array and checks if each element is greater than or equal to the threshold.
4: Initialize a boolean variable ‘b‘ to true.
5: for $i$ in 0 to size do
6:     if the value at index $i$ in ‘res‘ is less than ‘threshold‘ then
7:         Set ‘b‘ to false.
8:     end if
9: end for
10: return the final value of ‘b‘.
```

Algorithm 6 zkSoftmax

```
1: Input: val[size], threshold, hp
2: Output: 1
3: Call ‘nonNeg‘ function with ‘val‘ and threshold. It checks if all values are non negative
4: Assert that the result of ‘nonNeg‘ is true.
5: summation ← val + threshold
6: Assert that the result of ‘summation‘ matches the provided hp value.
7: return 1.
```

K-Nearest Neighbors (KNN) model computes distances between input data points and others in the dataset, selecting the K nearest neighbors to determine the majority class for predictions. This intuitive approach is effective for classification and regression tasks but poses privacy challenges with proprietary parameters.

To achieve zero knowledge verifiability, zkVML employs specific steps detailed in Algorithm 8. Public inputs are used for computations, while proprietary model parameters remain hidden. The data undergoes computations through zkDistance, zkSort, and zkMaxLabel APIs, each preserving privacy and generating proofs of correct computation.

Algorithm 7 zkTreeTraversal

```
1: Input: tree[treeSize], X[numFeatures], index
2: Output: finalProbs[numClasses]
3: for $i$ in 0 to $treeSize$ do
4:     $tmp$ ← index
5:     node ← tree[index] {Get the current node}
6:     if $X$[node.feature_idx] < node.feature_val then
7:         $index \leftarrow index * 2 + 1$ {Move to the left child}
8:     else
9:         $index \leftarrow index * 2 + 2$ {Move to the right child}
10:    end if
11:    if node.isLeafNode then
12:        $index \leftarrow tmp$ {If it’s a leaf node, reset index}
13:    end if
14: end for
15: return node.prediction_probs
```

Algorithm 8 KNN with zkVML Algorithm

```
1: Input: Training dataset $D$, test instance $x$, number of neighbors $K$
2: Public Parameters: ZK APIs (zkDistance, zkSort, zkMaxLabel)
3: for each instance $d_i$ in $D$ do
4:     Use zkDistance API to compute distance $dist(x, d_i)$ with proof
5: end for
6: Use zkSort API to sort instances in $D$ based on distance $dist(x, d_i)$ with proof
7: Use zkMaxLabel API to select the first $K$ instances as nearest neighbors, count class label frequencies, and return majority class label with proof
8: Verify all generated proofs
9: Send final proof of correct prediction to user for verification
```

zkDistance calculates distances using ZK-SNARKS, zkSort sorts distances while proving knowledge of a permutation, and zkMaxLabel identifies the most occurring label among the K neighbors. Proofs are verified at each step, and a final proof validates all computations, ensuring prediction accuracy while safeguarding model parameter confidentiality.

The Decision Tree model partitions the feature space into regions, creating a hierarchical tree structure for decision-making. To ensure zero knowledge verifiability and confidentiality, zkVML library employs specific ZK APIs.

After training the model, zkTreetraversal navigates the tree structure based on user inputs, preserving confidentiality and generating proof. zkArgmax determines the most probable class without revealing specific decisions. Proofs are verified at each step, and the final proof validates all computations, allowing users to verify the model while maintaining confidentiality and integrity.

Convolutional Neural Networks (CNNs) are widely used in image recognition and classification tasks due to their ability to capture spatial hierarchies in data. To ensure zero knowledge verifiability and confidentiality, zkVML library is employed using specific zero-knowledge (ZK) APIs throughout the computation process.

In a CNN, layers such as convolutional, pooling, and fully connected layers are used to process input images and extract features. zkConv2D and zkRelu maintain privacy during convolution operations and introduce non-linearity, respectively. Additionally, zkMaxPooling is applied for downsampling.

Further layers in the CNN employ similar ZK APIs for feature extraction, ensuring zero knowledge verifiability. Once features are extracted, zkApplyWeights simulates fully connected layer operations, zkRelu introduces non-linearity, and zkSoftmax computes class probabilities. zkArgmax determines the predicted class.

Proofs of correctness are generated at each step, ensuring accuracy while preserving confidentiality. A final proof validates all computations, maintaining the privacy and integrity of the CNN model and user inputs.

D. Blockchain Integration

To showcase our library's effectiveness, we deployed it on a decentralized platform, leveraging ZK-SNARKs for scalability and privacy [5]. Zokrates facilitates off-chain proof computation and on-chain verification, particularly valuable in blockchain environments [33] [39]. With Ethereum support, our system can deploy smart contracts on any EVM-supported chain, demonstrating its versatility in addressing blockchain challenges.

We specifically chose to fork the Mumbai testnet of the Polygon chain for integration, ensuring controlled yet realistic testing. Zokrates enables the export of verifier smart contracts, facilitating direct submission of proofs and public inputs to the blockchain for verification, providing a Boolean verification status.

Forking was conducted locally for thorough testing before public deployment, ensuring seamless integration into existing blockchain platforms. Off-chain proof generation maintains efficiency and scalability, while on-chain verification remains concise. This deployment underscores our library's potential to enhance blockchain applications with secure off-chain proof generation and efficient on-chain verification.

IV. Results

In this section, we benchmarked the zkVML library across various machine learning and deep learning models. We utilized the KNN model with the diabetes dataset, aiming to predict diabetes in female patients aged 21 or older of Pima Indian heritage. The Decision Tree model was applied to the iris dataset, which includes measurements of three iris types. Lastly, a CNN model was used on the MNIST dataset, consisting of 60,000 training and 10,000 test examples of handwritten digits. The images were resized to 20x20 pixels and centered within a 28x28 frame.

All the benchmarks of K-Nearest Neighbors and Decision Tree with and without zkVML were performed on 11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz along with 64 GB installed RAM and CNN with and without zkVML was performed on Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz with 12 GB installed RAM. We tested this machine learning and deep learning model for time comparison analysis between models with and without zkVML library.

A. Performance of K-Nearest Neighbours

Table I shows performance of the KNN machine learning model was analyzed across various parameter values, with a focus on the time comparison between using and not using zkVML. The analysis revealed that as the number of parameters increased, the model's processing time also increased. However, incorporating zkVML, which enhances privacy, significantly prolonged the processing time, especially with larger datasets. For instance, with 10,000 parameters, the model took over 300 seconds longer when zkVML was used. This trend is illustrated in the KNN graph in Fig.2. The study underscores the trade-off between privacy and computational efficiency, suggesting that decisions on using zkVML with the KNN model should consider the balance between privacy needs and desired model speed.

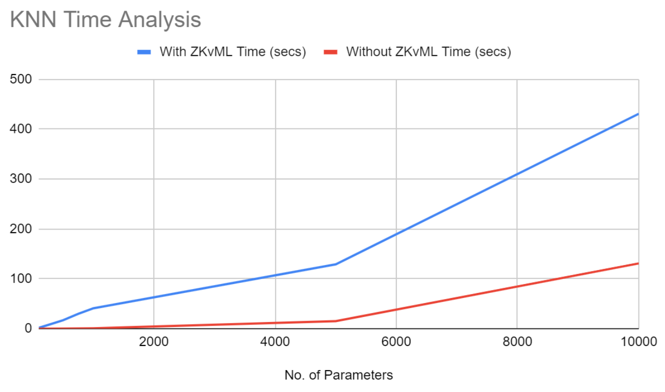

Fig. 2. KNN Time Comparison Analysis.

TABLE I
KNN Time Comparison Analysis

No. of Parameters	Time with zkVML (secs)	Time without zkVML (secs)
100	1.5764	0.02275
500	16.7780	0.1105
750	29.6153	0.35028
1000	40.8302	0.6899
5000	129.310	15.1324
10000	430.9613	130.8092

B. Performance of Decision Tree

Table II analysis of Decision Tree models with and without zkVML integration shows a direct correlation between the number of parameters and execution time. zkVML significantly increases execution times due to the computational

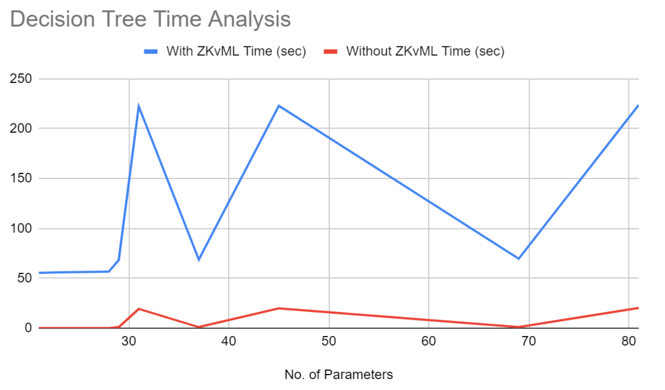

Fig. 3. DT Time Comparison Analysis

TABLE II
DT TIME COMPARISON ANALYSIS

Max Depth	Min Sample leaf	No. of Parameters	With zkVML Time (sec)	Without zkVML Time (sec)
4	1	28	56.61	0.05007
4	2	23	55.891	0.0434
4	3	21	55.429	0.0312
6	1	69	69.58	1.1102
6	2	37	68.789	1.0417
6	3	29	68.385	1.0223
8	1	81	223.881	20.2156
8	2	45	222.978	19.7812
8	3	31	222.309	19.312

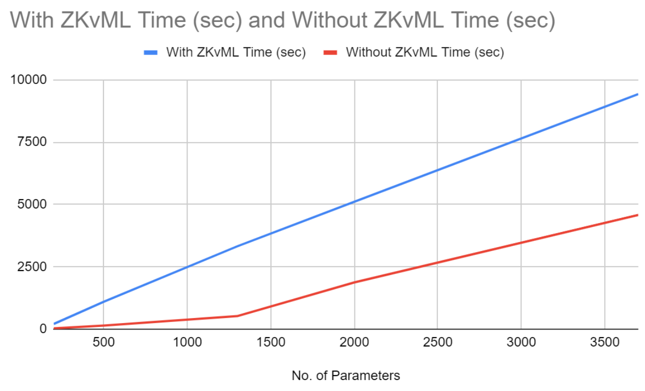

Fig. 4. CNN Time Comparison Analysis

overhead of generating zero-knowledge proofs. For instance, with a Max-Depth of 8 and Min-Sample-leaf of 1, execution time rose from 20.2156 seconds without zkVML to 223.881 seconds with zkVML, as shown in Fig.3. This highlights the need to balance privacy and computational efficiency when using privacy-enhancing tools in machine learning.

C. Performance of Convolution Neural Networks

Table III compares the execution time of a CNN model with and without zkVML integration, showing a significant increase in execution time with zkVML as parameter size grows. For example, with 200 parameters, the model takes 207 seconds with zkVML versus 30 seconds without it. This trend continues, with execution times rising to 1099 seconds for 500 parameters, 3325 seconds for 1300 parameters, and 5107 seconds for 2000 parameters. The disparity is most evident with 3700 parameters, where the model takes 9425 seconds with zkVML, compared to 4578 seconds without it, as seen in Fig.4. This highlights the substantial time overhead of integrating zkVML into CNN models, especially with larger and more complex parameter configurations.

TABLE III
CNN TIME COMPARISON ANALYSIS

No. of Parameters	With zkVML Time (sec)	Without zkVML Time (sec)
200	207	30
500	1099	143
1300	3325	523
2000	5107	1873
3700	9425	4578

zkVML enhances privacy but significantly increases execution times, especially with larger datasets and more complex parameters, underscoring the need for balancing privacy and computational efficiency. Our findings stress the importance of informed decision-making when incorporating privacy-enhancing mechanisms like zkVML in machine learning and deep learning applications. For KNN, DT, and CNN models, the trade-off between privacy and computational efficiency is evident, with zkVML introducing notable time overheads, particularly with larger and more complex parameter configurations. This highlights the critical need to carefully consider the balance between privacy requirements and computational constraints when using zkVML.

V. CONCLUSION

This paper introduces zkVML, a library for verifiable machine learning that ensures privacy by proving computation correctness without revealing model details. However, zkVML's execution times notably increase, particularly for large datasets and complex models. Integrating with blockchains like Ethereum enhances privacy guarantees.

zkVML offers functions for common ML operations, supporting privacy-preserving model building. Benchmarking on various models shows increased execution times, especially for larger and complex datasets, highlighting the privacy-performance trade-off. Optimization techniques like model pruning and fine-tuning can mitigate these limitations, enhancing zkVML's applicability. Leveraging CUDA for proof generation can also significantly reduce computation times.

While zkVML enhances privacy and flexibility, its computational overhead requires careful consideration, especially for real-time or resource-constrained applications. Despite these challenges, zkVML presents a promising approach for privacy-preserving ML, albeit with trade-offs that warrant optimization exploration.

REFERENCES

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," SIAM Journal on Computing, vol. 18, no. 1, pp. 186–208, 1989
[2] Fiat, A., & Shamir, A. (1986, August). How to prove yourself: Practical solutions to identification and signature problems. In Conference on the theory and application of cryptographic techniques (pp. 186-194). Berlin, Heidelberg: Springer Berlin Heidelberg.
[3] Bitansky, N., Canetti, R., Chiesa, A., & Tromer, E. (2012, January). From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Proceedings of the 3rd innovations in theoretical computer science conference (pp. 326-349).
[4] Kim, J., Lee, J., & Oh, H. (2020). Simulation-extractable zk-SNARK with a single verification. IEEE Access, 8, 156569-156581.
[5] Pinto, A. M. (2020). An introduction to the use of zk-SNARKs in blockchains. In Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece (pp. 233-249). Springer International Publishing.

[6] Kuznetsova, K., Yezhov, A., Kuznetsov, O., & Tikhonov, A. (2023, March). Solving Blockchain Scalability Problem Using ZK-SNARK. In The International Conference on Artificial Intelligence and Logistics Engineering (pp. 360-371). Cham: Springer Nature Switzerland.

[7] Soucy, P., & Mineau, G. W. (2001, November). A simple KNN algorithm for text categorization. In Proceedings 2001 IEEE international conference on data mining (pp. 647-648). IEEE.

[8] Kotsiantis, S. B. (2013). Decision trees: a recent overview. Artificial Intelligence Review, 39, 261-283.

[9] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In 2017 international conference on engineering and technology (ICET) (pp. 1-6). Ieee.

[10] Chen, H., Kim, M., Razenshteyn, I., Rotaru, D., Song, Y., & Wagh, S. (2020). Maliciously secure matrix multiplication with applications to private deep learning. In Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26 (pp. 31-59). Springer International Publishing

[11] Mahmood, Z., & Jusas, V. (2021). Implementation framework for a blockchain-based federated learning model for classification problems. Symmetry, 13(7), 1116.

[12] Conway, K. D., So, C., Yu, X., & Wong, K. (2024). opML: Optimistic Machine Learning on Blockchain. arXiv preprint arXiv:2401.17555.

[13] Seriner Gerenli, D. O. (2023). Secure Model Verification and Privacy Preservation with ZK-SNARKs and Neural Networks (Master's thesis, Middle East Technical University).

[14] So, C., Conway, K. D., Yu, X., Yao, S., & Wong, K. (2024). opp/ai: Optimistic Privacy-Preserving AI on Blockchain. arXiv preprint arXiv:2402.15006.

[15] Ganescu, B. M., & Passerat-Palmbach, J. (2024). Trust the Process: Zero-Knowledge Machine Learning to Enhance Trust in Generative AI Interactions. arXiv preprint arXiv:2402.06414.

[16] South, T., Camuto, A., Jain, S., Nguyen, S., Mahari, R., Paquin, C., ... & Pentland, A. S. (2024). Verifiable evaluations of machine learning models using zkSNARKs. arXiv preprint arXiv:2402.02675.

[17] Attrapadung, N., Hanaoaka, G., Hiromasa, R., Koseki, Y., Matsuda, T., Nishida, Y., ... & Yasuda, S. (2024, February). Privacy-Preserving Verifiable CNNs. In International Conference on Applied Cryptography and Network Security (pp. 373-402). Cham: Springer Nature Switzerland.

[18] Zhang, Y. (2020) 'Zero-knowledge proofs for Machine Learning', Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice [Preprint].

[19] H.-C. Sun and H. Zhang, "ZKDL: Efficient Zero-Knowledge Proofs of Deep Learning Training," arXiv (Cornell University), Jul. 2023, doi: 10.48550/arxiv.2307.16273.

[20] S. Garg, A. Jain, and A. Sahai, "Leakage-Resilient zero knowledge," in Lecture Notes in Computer Science, 2011, pp. 297–315. doi: 10.1007/978-3-642-22792-9-17.

[21] T. Gui, W. Tan, and M. Cai, "Privacy-Preserving and trustless verifiable fairness audit of machine learning models," International Journal of Advanced Computer Science and Applications, vol. 14, no. 2, Jan. 2023, doi: 10.14569/ijacsa.2023.0140294.

[22] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning.," IACR Cryptology ePrint Archive, vol. 2021, p. 730, Jan. 2021, [Online]. Available: https://eprint.iacr.org/2021/730.pdf

[23] H. Wang and T. M. Hoang, "EZDPS: An Efficient and Zero-Knowledge Machine Learning Inference Pipeline," Proceedings on Privacy Enhancing Technologies, vol. 2023, no. 2, pp. 430–448, Apr. 2023, doi: 10.56553/popets-2023-0061.

[24] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy.," IACR Cryptology ePrint Archive, vol. 2021, p. 673, Jan. 2021, [Online]. Available: https://eprint.iacr.org/2021/673.pdf

[25] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero Knowledge Proofs for Decision Tree Predictions and Accuracy," ACM SIGSAC Conference on Computer and Communications Security, Oct. 2020, doi: 10.1145/3372297.3417278.

[26] S. Garg, A. Jain, Z. Jin, and Y. Zhang, "Succinct zero knowledge for floating point computations," Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Nov. 2022, doi: 10.1145/3548606.3560653.

[27] A. Thomas, "The Implementation of Model Pruning to Optimize zk-SNARKs," Twelfth Annual Spring Term PRIMES Conference, May 21-22, 2022.

[28] V. Chen, V. Pastro, and M. Raykova, "Secure computation for machine learning with SPDZ.," arXiv (Cornell University), Jan. 2019, [Online]. Available: https://arxiv.org/pdf/1901.00329.pdf

[29] D. A. Luong and J. H. Park, "Privacy-Preserving Identity Management System on Blockchain using ZK-SNARK," IEEE Access, vol. 11, pp. 1840–1853, Jan. 2023, doi: 10.1109/access.2022.3233828.

[30] J. Groth and M. Maller, "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs," in Lecture Notes in Computer Science, 2017, pp. 581–612. doi: 10.1007/978-3-319-63715-0-20.

[31] D. G. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "Scaling up Trustless DNN Inference with Zero-Knowledge Proofs," arXiv (Cornell University), Oct. 2022, doi: 10.48550/arxiv.2210.08674.

[32] Z. Xing et al., "Zero-knowledge proof meets Machine Learning in Verifiability: A survey," arXiv (Cornell University), Oct. 2023, doi: 10.48550/arxiv.2310.14848.

[33] Eberhardt, J., & Tai, S. (2018, July). Zokrates-scalable privacy-preserving off-chain computations. In 2018 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) (pp. 1084-1091). IEEE.

[34] Eberhardt, J. (2021). Scalable and privacy-preserving off-chain computations.

[35] Rigaki, M., & Garcia, S. (2023). A survey of privacy attacks in machine learning. ACM Computing Surveys, 56(4), 1-34

[36] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., & Ward, N. P. (2019). Aurora: Transparent succinct arguments for R1CS. In Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38 (pp. 103-128). Springer International Publishing.

[37] Feng, B., Qin, L., Zhang, Z., Ding, Y., & Chu, S. (2021). Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive.

[38] Modulus Labs, "The Cost of Intelligence: Proving Machine Learning Inference with Zero-Knowledge," Modulus Labs Blog, Jan. 30, 2023. [Online]. Available: https://www.modulus.ai/blog/the-cost-of-intelligence-proving-machine-learning-inference-with-zero-knowledge/. [Accessed: Dec. 8, 2023].

[39] Groth, J. (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 35 (pp. 305-326). Springer Berlin Heidelberg.

[40] Salleras, X., & Daza, V. (2021). ZPiE: Zero-knowledge proofs in embedded systems. Mathematics, 9(20), 2569.