A Verifiable and Privacy-Preserving Federated Learning Training Framework

Haohua Duan, Zedong Peng, Liyao Xiang, *Member, IEEE*, Yuncong Hu, and Bo Li, *Fellow, IEEE*

***Abstract*—Federated learning allows multiple clients to collaboratively train a global model without revealing their private data. Despite its success in many applications, it remains a challenge to prevent malicious clients to corrupt the global model through uploading incorrect model updates. Hence, one critical issue arises in how to validate the training is truly conducted on legitimate neural networks. To address the issue, we propose *VPNNT*, a zero-knowledge proof scheme for neural network backpropagation. *VPNNT* enables each client to prove to others that the model updates (gradients) are indeed calculated on the global model of the previous round, without leaking any information about the client's private training data. Our proof scheme is generally applicable to any type of neural network. Different from conventional verification schemes constructing neural network operations by gate-level circuits, we improve verification efficiency by formulating the training process using custom gates — matrix operations, and apply an optimized linear time zero knowledge protocol for verification. Thanks to the recursive structure of neural network backward propagation, common custom gates are combined in verification thereby reducing prover and verifier costs over conventional zero knowledge proofs. Experimental results show that *VPNNT* is a lightweighted verification scheme for neural network backpropagation with an improved prove time, verification time and proof size.**

***Index Terms*—Zero knowledge proofs, privacy preserving, neural networks.**

I. Introduction

FEDERATED learning has become a recent trend for a server to build a global model by training it over distributed, proprietary datasets hosted by separate clients. Typically, each client locally trains its model and sends the updates to the server for aggregation. However, this computation paradigm faces a privacy and security dilemma: on one hand, the client is concerned about revealing its private data to any other party; on the other, the learning framework is vulnerable to malicious client updates which could compromise the global model's performance. However, the server is restricted from directly inspecting the private data according to the GDPR [1] or for proprietary reasons. Hence we raise the critical question: *how can the server verify the client's update is legitimate without breaching client data privacy?*

Although approaches have been proposed to inspect the valid range of model updates [2], [3] or to ensure global model convergence [4], the methods cannot guarantee the correctness of the local training, and thus do not prevent lazy clients from uploading fake updates without actual training efforts. Existing zero knowledge protocols for verifying neural networks are mainly for the inference phase [5], verifying the prediction outcome is obtained from legitimate models. But the training is much more complicated than the inference and there are yet no efficient protocols for verifying the training procedures.

We propose a verifiable and privacy-preserving neural network training (*VPNNT*) framework on the foundation of the zero-knowledge proof (ZKP) as illustrated in Fig. 1. On the basis of ZKP, we design a protocol to enable each client i (prover) to convince the server (verifier) that the gradient ∇_{W_i} uploaded are correctly computed on its private data by a short proof π_i without divulging any useful information. The probability of any dishonest client convincing the server with incorrect results ∇_{W^*} is negligible. To the best of our knowledge, there is no generalized ZKP for model training; the naive way of expressing the training process as arithmetic circuits is highly complicated incurring exorbitant overhead. Efficient ZKP protocols are designed but are limited to programs only involving addition and multiplication of integers or elements of a finite field [6], or are restricted to layered arithmetic circuits of fan-in two (e.g., [7], [8]). Hence they are unable to deal with non-arithmetic operations such as ReLU activation layer, or complex cross-layer structures in the backpropagation.

The foremost step of zero knowledge proof schemes is to arithmetize the statement into intermediate representations which are directly amenable to the application of interactive proof or argument systems. The core of *VPNNT* lies in a novel intermediate representation, or *custom gates* consisting of matrix operations, which have specific optimizations on interactive proof protocols [9]. On one hand, as defined in [10] that a custom gate is a function $G : \mathbb{F}^l \to \mathbb{F}$ for some l, which can appear many times in the arithmetic circuit. The custom gates greatly reduce the circuit size needed to compute a function, leading

Manuscript received 30 April 2023; revised 10 January 2024; accepted 12 January 2024. Date of publication 26 February 2024; date of current version 4 September 2024. This work was supported in part by the National Key R&D Program of China under Grant 2021ZD0112801, in part by NSF China under Grant 62272306, Grant 62136006, and Grant 62032020, in part by RGC RIF under Grant R6021-20, in part by RGC CRF under Grant C7004-22 G and under Grant C1029-22 G, and in part by RGC GRF under Grant 16209120, Grant 16200221, and Grant 16207922 as well as gifts from Huawei and BIANJIE.AI. *(Corresponding author: Liyao Xiang.)*

Haohua Duan, Zedong Peng, Liyao Xiang, and Yuncong Hu are with Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: duanhaohua@sjtu.edu.cn; zedongpeng@sjtu.edu.cn; xiangliyao08@sjtu.edu.cn; ychu@cs.sjtu.edu.cn).

Bo Li is with The Hong Kong University of Science and Technology, Hong Kong, China (e-mail: bli@cse.ust.hk).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2024.3369658, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2024.3369658

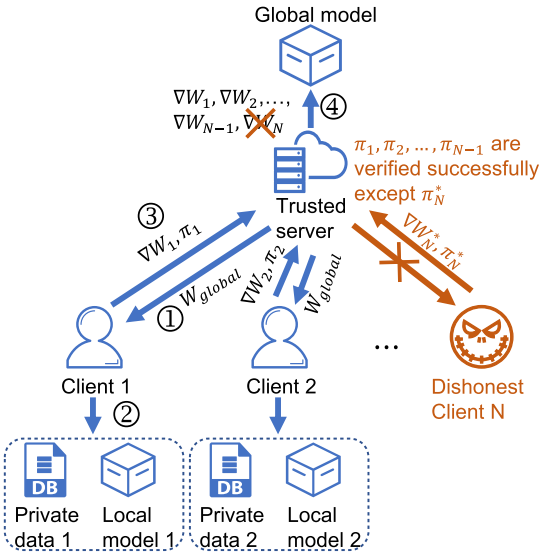


Fig. 1. Framework of *VPNNT* applied to federated learning scene. At the beginning of each training round, ① each client $i \in \{1, \ldots, N-1\}$ downloads the latest parameters of the global model from the server to update its local model; ② each client calculates new gradient ∇_{W_i} on their own local private data and generates the corresponding zero knowledge proof π_i; ③ each client sends ∇_{W_i} and π_i together to the server, and ④ the server only aggregates the gradients $\nabla_{W_1}, \nabla_{W_2}, \ldots$ which are verified successfully to update the global model. The gradients that do not pass verification are discarded.

to a faster prover. On the other hand, we observe that in the backpropagation formula of neural networks, a majority part of the tensor/matrix operations occur repeatedly, and thus can be turned into custom gates if the backpropagation is written as a circuit. Hence we design custom gates as matrix multiplication, Hadamard product, matrix reshaping, matrix re-indexing, etc., to take advantage of the optimized interactive proof protocol.

From the verification point of view, the custom gates are not free but save much costs than the naive approach. For $n \times n$ matrix multiplication, the custom gate using the optimized protocol [9] reduces the prover's runtime from $O(n^3)$ to $O(n^2)$. It should be noted that the prover time is even less than the computation time ($O(n^3)$). For operations that cannot be transformed into matrix forms, i.e., ReLU, Sigmoid, Max Pooling, which are mostly non-arithmetic operations, we collect their claims and express them by another intermediate representation called Rank-1 Constraint System (R1CS) instances. Compared to the more intuitive bit-by-bit expression, we show the R1CS is more efficient in our case. This is attributed to a linear overhead to the number of non-zero entries in the R1CS instances. By expressing the R1CS instances as custom gates, the backpropagation of neural networks is verified by invocations of multi-linear sumcheck protocols.

We further optimize our protocol by combining claims for the same matrix. Due to the recursive structure of neural networks, a matrix can appear multiple times (may or may not be in the same intermediate representation) in the circuit. Since their claims are all converted into multi-linear extensions for sumcheck protocols, we can efficiently combine the multi-linear extensions by interpolating random numbers chosen by the verifier, and thus reduce the overhead.

Highlights of our contributions are as follows. *First*, we are among the first to investigate the problem of neural network training verification in the general setting of federated learning. We propose VPNNT, an efficient, privacy-preserving verification framework on the basis of ZKP. *Second,* we design custom gates and the corresponding zero knowledge argument for verifying the neural network backpropagation with an improved efficiency than the conventional method. *Finally,* our framework is independent of the specific neural network structure and thus is widely applicable. We have implemented VPNNT as an open-source C++ library.

II. Related Work

The most relevant works w.r.t. ZKP for neural networks include SNARK-based ZKP and MPC-based ZKP protocols.

SNARK-Based ZKP Protocols: Our work is mostly related to *vCNN* [11] and *zkCNN* [5]. The two works are on zero-knowledge verification of convolutional neural network (CNN) inference, rather than training. Among them, *vCNN* proposes a quadratic polynomial program (QPP) to verify convolutional operations, and quadratic arithmetic programs (QAPs) to verify ReLU and pooling layers. It combines QPP and QAPs by the commit-and-prove SNARK (CP-SNARK) technique. The work of *zkCNN* generalizes the GKR protocol and uses the sumcheck protocol for fast Fourier transform (FFT) as a building block to verify convolutional operations. Our work also adopts this efficient way to verify convolutional operations. But different from *zkCNN*, *VPNNT* expresses matrix operations as custom gates and non-arithmetic operations as R1CS, the combination of which improves efficiency over *zkCNN*.

Apart from the aforementioned, Ghodsi et al. propose *SafetyNets* [12], a neural network inference verification scheme inspired by the GKR protocol, which is not ZK by requiring public model weights and inputs. Zhang et al. design *zkDT* [13], a zero-knowledge proof scheme for decision tree inference. *VerifyNet* [14] introduces a double-masking protocol exploiting the homomorphic hash function to allow workers to verify the correctness of the server's aggregation result. *ZEN* [15] shows a new, ZKP-friendly quantization approach in supporting neural network inference verification on real numbers. We convert the floating-point numbers into integers in the finite field by scaling but believe our work can be further enhanced by the quantization scheme in *ZEN* to achieve a higher precision.

MPC-Based ZKP Protocols: A recent line of work to build practical ZKP protocols based on subfield vector oblivious linear evaluation (sVOLE) [16], [17], [18], [19], privacy-free garbled circuits [20], [21], [22], or the "MPC-in-the-head" paradigm [23], [24], [25], [26], [27] are efficient in terms of execution time and memory overhead. Zhao et al. propose *VeriML* [28], a verifiable gradient aggregation protocol based on the secret sharing scheme to prevent untrustworthy central servers from affecting the quality of the global model in the federated learning. Weng et al. proposed *Mystique* [29], an efficient ZKP protocol based sVOLE for large-scale neural network inference, but it cannot be converted to a non-interactive protocol. Different

TABLE I
LIST OF NOTATIONS

Symbol	Description
$\mathcal{P}$	Any honest prover
$\mathcal{V}$	Any honest verifier
$\mathcal{P}^*$	Any malicious prover
$\mathcal{A}$	Any PPT adversary
$\mathcal{S}$	The simulator of zero knowledge scheme
$\mathbb{F}$	Finite field
$\beta(\cdot)$	Relation function
$\tilde{\beta}(\cdot)$	The multilinear extension of relation function
$\tilde{A}$	The multilinear extension of matrix A
x	Public input
w	Witness (private input)
$\mathcal{M}$	Any neural network model
$\mathcal{B}$	The gradients calculation formula
∇_{W_i}	The gradients of W_i
N	The number of R1CS constraints
π	The zero knowledge proof

from *VeriML* and *Mystique*, *VPNNT* can be made non-interactive through Fiat-Shamir heuristic [30].

III. PRELIMINARIES

We briefly describe the building blocks of our work. The notations used in this paper are summarized in Table I.

A. Neural Network Backpropagation

Backpropagation, short for 'backward propagation of errors,' represents an algorithm employed in the supervised learning of neural networks through gradient descent. Given a neural network and a loss function, the backpropagation algorithm computes the gradient of the loss function with respect to the neural network's weights. Specifically, assume that $L(X, \theta)$ is a loss function, where X denotes input and θ denotes the neural network's weights. Then, each iteration of gradient descent updates θ according to

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial L(X, \theta^t)}{\partial \theta},$$

where θ^t denotes the neural network's weights at iteration t in gradient descent and α represents the learning rate.

B. Interactive Proofs

In the interactive proof, a prover proves to the verifier the correctness of the information held by the prover. Multiple rounds of statements and inquiries are executed by the two parties. We phrase the process as $\mathcal{P}$ trying to convince $\mathcal{V}$ that $f(x) = y$. The proof system is interesting only when the running time of $\mathcal{V}$ is less than the time of directly computing the function f. We formalize interactive proofs in the following:

Definition 1: Let f be a Boolean function. A pair of interactive machines (P, V) is an interactive proof for f with soundness ϵ if the following holds:

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

Protocol 1: Sumcheck Protocol.

For a l-variate polynomial f, the protocol proceeds in l rounds. The given initial claim is $H \in \mathbb{F}$.

- In the first round, $\mathcal{P}$ sends a univariate polynomial
$$f_1(x_1) \stackrel{def}{=} \sum_{b_2,\ldots,b_l \in \{0,1\}} f(x_1, b_2, \ldots, b_l).$$
- $\mathcal{V}$ checks if $H = f_1(0) + f_1(1)$, and rejects otherwise. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.
- In the ith round ($2 \leq i \leq l-1$), $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial
$$f_i(x_i) \stackrel{def}{=} \sum_{b_{i+1},\ldots,b_l \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_l).$$
$\mathcal{V}$ checks if $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$, and rejects otherwise. Then $\mathcal{V}$ sends a random challenge $r_i \in \mathbb{F}$ to $\mathcal{P}$.
- In the lth round, $\mathcal{P}$ sends a univariate polynomial
$$f_l(x_l) \stackrel{def}{=} f(r_1, r_2, \ldots, r_{l-1}, x_l).$$
$\mathcal{V}$ checks $f_{l-1}(r_{l-1}) = f_l(0) + f_l(1)$, rejecting if not.
- $\mathcal{V}$ generates a random challenge $r_l \in \mathbb{F}$ and evaluates $f(r_1, r_2, \ldots, r_l)$ with a single oracle query to f. $\mathcal{V}$ will accept if and only if $f_l(r_l) = f(r_1, r_2, \ldots, r_l)$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

Sumcheck Protocol: It is one of the most important interactive proofs in the literature. For a multivariate polynomial $f : \mathbb{F}^l \to \mathbb{F}$, the sumcheck problem is to sum f on all binary inputs: $H = \sum_{b_1,b_2,\ldots,b_l \in \{0,1\}} f(b_1, b_2, \ldots, b_l)$. Lund et al. [31] proposed a sumcheck protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ that H is the correct sum. We give the detail of the sumcheck protocol in Protocol 1. The prover time of the sumcheck protocol depends on the degree and sparsity of f, and the verifier time is $O(dl)$, where d is the variable-degree of f. In each round, $\mathcal{P}$ sends a univariate polynomial of one variable in f, which can be uniquely specified by $d+1$ points, so the proof size of the sumcheck protocol is $O(dl)$. The sumcheck protocol is complete and sound with $\epsilon = \frac{dl}{|\mathbb{F}|}$.

Now we introduce the concept of identity function.

Definition 2: (Identity function [5]). Let $I : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}$ be the identity function such that $I(x, y) = 1$ if $x = y$, and $I(x, y) = 0$ otherwise.

With the identity function, now we formally provide the definition of multi-linear extension.

Definition 3: (Multi-linear Extension [32]). Let $V : \{0,1\}^l \to \mathbb{F}$ be a function. The multi-linear extension of $\mathcal{V}$ is the unique polynomial $\tilde{V} : \mathbb{F}^l \to \mathbb{F}$ such that $\tilde{V}(x_1, x_2, \ldots, x_l) = V(x_1, x_2, \ldots, x_l)$ for all $x_1, x_2, \ldots, x_l \in \{0,1\}$. $\tilde{V}$ can be expressed as

$$\tilde{V}(x_1, x_2, \ldots, x_l) = \sum_{b \in \{0,1\}^l} \tilde{I}(x, b) \cdot V(b)$$

$$= \sum_{b\in\{0,1\}^l} \prod_{i=1}^{l}((1-x_i)(1-b_i)+x_i b_i)\cdot V(b), \quad (1)$$

where b_i is the ith bit of b and $\tilde{I}$ is the multi-linear extension of I.

We view the multi-linear extension on an $m \times n$ matrix A as the multi-linear extension on the function $\tilde{A}(\cdot): \{0,1\}^{\log m+\log n} \to \mathbb{F}$ mapping indices to elements of A. Hence, we provide the definition of the relation function describing matrix operations.

Definition 4: (Relation function for binary matrix operations). Let $\beta_{\text{Binary}}(p_1,p_2,p_3,p_4,p_5,p_6): \{0,1\}^{\log s_1+\log s_2+\log s_3+\log s_4+\log s_5+\log s_6} \to \{0,1\}$ be the relation function for a binary matrix operator involving an output matrix of shape $s_1 \times s_2$ and two input matrices of shape $s_3 \times s_4$ and $s_5 \times s_6$, respectively. Let $\beta_{\text{Binary}}(\cdot)=1$ if Bi-Operator$(p_1,p_2,p_3,p_4,p_5,p_6)=True$, and $\beta_{\text{Binary}}(\cdot)=0$ otherwise. Hence, $\tilde{\beta}_{\text{Binary}}(x)=\sum_{b\in\{0,1\}^l}\tilde{I}(x,b)\cdot\beta_{\text{Binary}}(b)$ where $l=\log s_1+\log s_2+\log s_3+\log s_4+\log s_5+\log s_6$.

Definition 5: (Relation function for unary matrix operations). Let $\beta_{\text{Unary}}(p_1,p_2,p_3,p_4): \{0,1\}^{\log s_1+\log s_2+\log s_3+\log s_4} \to \{0,1\}$ be the relation function for a unary matrix operator involving an output matrix of shape $s_1 \times s_2$ and an input matrix of shape $s_3 \times s_4$. Let $\beta_{\text{Unary}}(\cdot)=1$ if Unary-Operator$(p_1,p_2,p_3,p_4)=True$, and $\beta_{\text{Unary}}(\cdot)=0$ otherwise. Hence, $\tilde{\beta}_{\text{Unary}}(x)=\sum_{b\in\{0,1\}^l}\tilde{I}(x,b)\cdot\beta_{\text{Unary}}(b)$ where $l=\log s_1+\log s_2+\log s_3+\log s_4$.

Interactive Proof for MATMULT: Given two $n \times n$ input matrix A, B over finite filed $\mathbb{F}$, the goal of interactive proof for matrix multiplication (MATMULT) is to verify the correctness of the product matrix $C=A\cdot B$. An interactive MATMULT protocol can be obtained by directly applying the sumcheck protocol, which the prover runs in $O(n^3)$ time. Thaler et al. provided an optimized approach in [9] by a summation to establish the connection between the multi-linear extension form of two input matrices and the output matrix

$$\tilde{C}(p_1,p_2)=\sum_{p_3\in\{0,1\}^{\log n}}\tilde{A}(p_1,p_3)\cdot\tilde{B}(p_3,p_2), \quad (2)$$

where $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$ are the multi-linear extensions of $n \times n$ matrix A, B and C, respectively. The prover runtime reduces to $O(n^2)$.

C. Zero-Knowledge Arguments

An argument system for an NP relationship R is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there is a witness w such that $(x;w)\in R$ for some input x. We focus on arguments that if the prover convinces the verifier of the statement validity, the prover must know w. The formal definitions of zero knowledge arguments is provided as follows.

We use $\mathcal{G}$ to represent the generation phase of the public key pp. Formally, consider the definition below, where we assume R is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 6: [7] Let R be an NP relation and w be the witness. A tuple of algorithm $(\mathcal{G},\mathcal{P},\mathcal{V})$ is a zero-knowledge argument for R if the following holds.

- *Completeness:* For every (pk,vk) output by $\mathcal{G}(1^\lambda)$ and $(x,w)\in R$,
$$\langle P(pk,w),V(vk)\rangle(x)=\mathsf{accept}.$$
- *Soundness:* For any PPT prover $\mathcal{P}$, there exists a PPT extractor ε such that for every (pk,vk) output by $\mathcal{G}(1^\lambda)$ and any x, it holds that
$$\Pr[\langle P(pk),V(vk)\rangle(x) = \mathsf{accept}\wedge(x,w)\notin R|w\leftarrow\varepsilon(pk,x)]\le\mathsf{negl}(\lambda),$$
where we use λ to denote the security parameter, and $\mathsf{negl}(\lambda)$ to denote the negligible function in λ.
- *Zero knowledge:* There exists a PPT simulator S such that for any PPT adversary $\mathcal{A}$, auxiliary input $z\in\{0,1\}^{\mathsf{poly}(\lambda)}$, $(x;w)\in R$, it holds that
$$\begin{aligned}&\Pr[\langle P(pk,w),\mathcal{A}\rangle\\&=\mathsf{accept}:(pk,vk)\leftarrow\mathcal{G}(1^\lambda);(x,w)\leftarrow\mathcal{A}(z,pk,vk)]\\&=\Pr[\langle S(\mathsf{trap},z,pk),\mathcal{A}\rangle\\&=\mathsf{accept}\text{: }(pk,vk,\mathsf{trap})\leftarrow S(1^\lambda);(x,w)\leftarrow\mathcal{A}(z,pk,vk)],\end{aligned}$$

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

An efficient way to extend a protocol to a zero-knowledge argument is using zero knowledge verifiable polynomial delegation scheme (zkVPD) to achieve zero knowledge property.

Formally speaking, let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of l-variate polynomial over $\mathbb{F}$, and d be a variable-degree parameter. The zkVPD scheme for $f\in\mathbb{F}$ and $t\in\mathbb{F}^l$ consists of the following algorithms:

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

where r_f denotes the randomness generated by the prover, $y=f(t)$, and π denotes the transcript seen by verifier during the interaction with zkVPD.Open. The zkVPD scheme satisfies completeness, soundness and zero knowledge. The details can be found in [7], [8], [33].

Zero Knowledge Arguments for R1CS Instances: A rank-1 constraint system (R1CS) instance consists of matrices $\mathbf{A},\mathbf{B},\mathbf{C}\in\mathbb{F}^{m\times(n+1)}$ and explicit input $v\in\mathbb{F}^k$. It is satisfiable if there exist $w\in\mathbb{F}^{n-k}$ such that $\mathbf{A}z\circ\mathbf{B}z=\mathbf{C}z$ where $z=(1,v,w)\in\mathbb{F}^{n+1}$ and $\circ$ denotes entry-wise (Hadamard) product. Zero knowledge arguments for R1CS instance using the multi-linear sumcheck for verification are known as *Spartan* [34], and

they do not require a trusted setup. For N constraints, the prove time is $O(N)$, the verification time is $O(\sqrt{N})$ and the proof size is $O(\sqrt{N})$.

IV. Technical Overview

Our main contributions are two-fold: the first is the design of zero knowledge arguments for matrix operations verification using the sumcheck protocol and relation functions. With that serving as the cornerstone, we made a second contribution — a verification framework for neural network training. We provide a technical overview and further explain the detail in Sections V and VI.

Transform Matrix Operations into a General Form: Inspired by Thaler's method for verifying the MALMULT (See Section III-B), we observe that expressing inputs, outputs, and intermediate results in form of matrices, instead of explicitly expressing all operations as gates of the arithmetic circuit, would greatly save costs. But the form of (2) is so primitive that we need to extend it to a general form to include a wide variety of operations. Hence, we construct the relation function for binary matrix operations to rewrite (2) as

$$\tilde{C}(p_1,p_2) = \sum_{(p_3,p_4,p_5,p_6)\in\{0,1\}^{\log s_3+\log s_4+\log s_5+\log s_6}} \tilde{\beta}_{\text{Binary}}(p_1,p_2,p_3,p_4,p_5,p_6)\cdot\tilde{A}(p_3,p_4)\cdot\tilde{B}(p_5,p_6), \quad (3)$$

where $s_3 \times s_4$ and $s_5 \times s_6$ represents the shape of matrices A, B, respectively, the shape of matrix C is $s_1 \times s_2$, and $\tilde{\beta}_{\text{Binary}} : \mathbb{F}^{\log s_1+\log s_2+\log s_3+\log s_4+\log s_5+\log s_6} \to \mathbb{F}$ represents the multi-linear extension of the relation function. The relation function plays the role of selecting specific elements of matrices in different operations. Hence the expression is sufficiently powerful in expressing inner product, outer product, Hadamard product, etc. if different relation functions are applied to different operations (See Section V-A for details of definitions). The relation functions allow us to transform the problems of verifying matrix operations into sumcheck problems, enabling us to invoke the sumcheck protocol to verify the correctness, which is considerably more efficient than verifying the correctness through direct computation. Similarly, we can also verify the operations such as reshaping and re-indexing of a single matrix (See Section V-B for details) by

$$\tilde{C}'(p_1,p_2) = \sum_{(p_3,p_4)\in\{0,1\}^{\log s_3+\log s_4}} \tilde{\beta}_{\text{Unary}}(p_1,p_2,p_3,p_4)\cdot\tilde{C}(p_3,p_4), \quad (4)$$

where $s_3 \times s_4$ represents the shape of matrix C before reshaping or re-indexing, C' denotes the $s_1 \times s_2$ output matrix, and $\tilde{\beta}_{\text{Unary}} : \mathbb{F}^{\log s_1+\log s_2+\log s_3+\log s_4} \to \mathbb{F}$ represents the multi-linear extension of the relation function for unary matrix operations.

The relation function is public as it does not reveal any value of matrix elements, but merely a specific mapping relationship between matrix elements. Therefore, all calculations related to the relation function can be performed offline. The verification process only includes the verification of the input matrices and the output matrix, but not the verification of any intermediate result and thus is far more efficient than the gate-level verification.

A Zero Knowledge Scheme for Verifying Backpropagation: There are many recent works on verifying the correctness of neural network inferences, such as *SafetyNets* [12], *vCNN* [11], *zkCNN* [5], etc. But none of them are suitable for verifying neural network backpropagation. The difficulty mainly lies in the increased computational scale and complex hierarchical structure of backpropagation. In our scheme, we take advantage of the matrix-based, recursive structure of the backpropagation formula. For example, the gradients of the fully-connected layers are expressed by matrix multiplication and Hadamard product. The gradients of the convolutional layers are more complicated, consisting of FFT, IFFT, reshaping, re-indexing, as well as Hadamard product. By decomposing the backward propagation into matrix operations, we build our zero knowledge scheme with the matrix operation verification protocol as the cornerstone. This construction grants our scheme a coarser modularity than the gate-level schemes based on arithmetic circuits. Our scheme enables the construction of backpropagation relations at higher levels, without the need to transform these relations into gate-by-gate hierarchical structures of arithmetic circuits, thereby affording efficiency to various applications. More importantly, our scheme permits the flexible selection of private inputs that require commitment, without the need to commit every gate in the input layer, as mandated by the GKR-based schemes, thereby further improving efficiency over these schemes.

Due to the limitations of the multi-linear extension form in expressing non-arithmetic operations, the sumcheck protocol is challenged to verify the non-arithmetic operations. Noting that the Rank-1 Constraints System (R1CS) can represent any operation including non-arithmetic ones by equations, we model the activation layer and pooling layer operations in backpropagation by R1CS, and adopt its zero knowledge argument for verification (See Section V-D).

V. Zero Knowledge Arguments for Matrix Operations

In this section, we will show our building blocks of zero knowledge arguments for matrix operations, by introducing representative ones including binary operators, unary operators, operators combining multiple basic operations, and non-arithmetic operators. At the high level, we convert operations into the form of matrices operations, and verify them by the sumcheck protocol, meanwhile achieving zero-knowledge with masking polynomial and polynomial commitment scheme.

A. Binary Operators Verification

The fundamental binary operator is matrix multiplication which takes a high proportion in the backpropagation process of neural networks. The naive approach to verify the correctness of matrix multiplication is to implement it using addition and multiplication gates. Given $m \times n$ input matrices A and $n \times t$ B, it is required to compute $C = A \cdot B$. In total, the matrix multiplication circuit contains mnt gates and costs a prove time of $O(mnt)$. The prove time is improved by [9] which we apply in our protocol. Specifically, we use $\tilde{A}$, $\tilde{B}$

and $\tilde{C}$ to denote their corresponding multi-linear extensions. For all $(p_1, p_2) \in \{0,1\}^{\log m} \times \{0,1\}^{\log t}$, the relation of the multi-linear extensions of the three matrices can be written as (2). We could use the sumcheck protocol to verify the correctness of (2), according to the Schwartz-Zippel lemma [35], [36]. The prove time is $O(\max\{mn, nt\})$.

In order to verify matrix operations other than matrix multiplication during backpropagation of neural network in matrix form (see Section VI-B for details), we adopt public relation functions to construct sumcheck formation for more general binary operators founded on matrix multiplications. The relation function essentially acts as a selector for elements in the matrix. It records the mapping from input matrices to the output matrix without carrying any value of the private input. With the relation function, we express binary operators in a general form as (3), where $\tilde{\beta} = \tilde{\beta}(p_1, p_2, p_3, p_4, p_5, p_6) : \mathbb{F}^{2(\log m + \log t + \log n)} \to \{0,1\}$ represents the multi-linear extension of the relation function. The (p_1, p_2) collectively represent the indices of matrix C, while (p_3, p_4) and (p_5, p_6) are indices of A and B, respectively. Since the wiring between the inputs and outputs is fixed a prior, $\tilde{\beta}$ can be computed offline.

We give an example of Hadamard product of matrices to explain the idea. The Hadamard product performs element-wise multiplication, so the same indices are applied to both input matrices. Eq. (3) turns into

$$\tilde{C}(p_1, p_2) = \sum_{(p_3,p_4)\in\{0,1\}^{\log m+\log n}} \tilde{\beta}_1(p_1, p_2, p_3, p_4) \tilde{A}(p_3, p_4) \cdot \tilde{B}(p_3, p_4), \tag{5}$$

where $m \times n$ represents the size of matrix A, B and C, β_1 is the relation function for Hadamard product, i.e., for $p_1, p_3 \in \{0,1\}^{\log m}$ and $p_2, p_4 \in \{0,1\}^{\log n}$,

$$\beta_1(p_1, p_2, p_3, p_4) = \begin{cases} 1, & p_1 = p_3 \,\&\, p_2 = p_4 \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

After $\log m + \log n$ rounds of communication by the sumcheck protocol, we obtain the claims $\tilde{A}(u, v)$ and $\tilde{B}(u, v)$ of matrix A and B, where $u, v \in \mathbb{F}^{\log m + \log n}$ are random challenges. Note that since there is no intermediate-level results in Hadamard product, writing it into matrix forms sees no direct improvement of overhead compared to the gate-level verification. But as the neural network training is mostly composed of matrices multiplication, organizing all operations in terms of matrix operations facilitates reusing claims as we will shortly introduce.

For more complicated binary operations such as convolution, we follow the idea of [5] to transform the convolution operation into multiplication in the spectral domain. The conversion is done by FFT and inverse FFT. In particular, the input matrix and the convolution kernel are flattened into vectors before FFT

$$U = X * W = IFFT(FFT(X) \circ FFT(W)), \tag{7}$$

where X, W, U denotes the input, the convolution kernel, and the convolution output, respectively. Eq. (7) consists of four sumcheck equations, two for FFT, one for IFFT, and one for Hadamard product. FFT and IFFT are essentially inner products and can be accelerated by dynamic programming and bookkeeping tables as in [5].

B. Unary Operators Verification

We also construct the sumcheck formation for the unary operators such as matrix transpose, rotation 180 ° and zero padding, etc. For all $(p_1, p_2) \in \{0,1\}^{\log m} \times \{0,1\}^{\log n}$, the unary operators can be expressed generally as (4).

Letting $D(x)$ denote the decimal representation of x, for $(p_1, p_2), (p_3, p_4) \in \{0,1\}^{\log m} \times \{0,1\}^{\log n}$, we have

$$\beta_2(p_1, p_2, p_3, p_4) = \begin{cases} 1, & p_1 = p_3 \,\&\, p_2 = p_4, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

$$\beta_3(p_1, p_2, p_3, p_4) = \begin{cases} 1, & D(p_1) + D(p_3) = m - 1 \\ & \,\&\, D(p_2) + D(p_4) = n - 1, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

and for padding size p', $(p_1, p_2) \in \{0,1\}^{\log(m+2p')} \times \{0,1\}^{\log(n+2p')}$ and $(p_3, p_4) \in \{0,1\}^{\log m} \times \{0,1\}^{\log n}$, we have

$$\beta_4(p_1, p_2, p_3, p_4) = \begin{cases} 1, & D(p_1) = D(p_3) + p' \\ & \,\&\, D(p_2) = D(p_4) + p' \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

β_2, β_3 and β_4 represent the relation function of matrix transpose, rotation 180 and zero padding, respectively. The interactive protocol for these three matrix operations can be described as evaluating $\tilde{C}'$ in (4) at any given point $(r_1, r_2) \in \mathbb{F}^{\log m} \times \mathbb{F}^{\log n}$ by the sumcheck protocol.

C. Zero-Knowledge for Matrix Operations Verification

All building blocks of matrix operations have to be zero-knowledge to prevent the leakage of the private inputs. Hence we adopt the masking polynomial and commitment scheme [6], [7], [8], [33] to conceal the values in each input matrix. For a $m \times n$ matrix A, we define

$$\dot{A}(x_1, \ldots, x_{\log mn}) \overset{def}{=} \tilde{A}(x_1, \ldots, x_{\log mn}) + Z(x_1, \ldots, x_{\log mn}) \sum_{w\in\{0,1\}} R(x_1, w), \tag{11}$$

where $\tilde{A}$ is the multi-linear extension of A. $Z(x) = \prod_{i=1}^{\log mn} x_i(1 - x_i)$, i.e., $Z(x) = 0$ for all $x \in \{0,1\}^{\log mn}$. R is the masking polynomial and w is the security parameter. Adding the masking polynomial to the multi-linear extension form of the matrix alone is not enough to achieve zero knowledge property. When querying the oracle in the last step of sumcheck, as shown in Protocol 1, there is still a risk of leaking information. So we need to add a new masking polynomial to the right side of the equation verified by the sumcheck protocol.

For zero knowledge sumcheck protocol $H + \rho J = \sum_{x\in\{0,1\}^l} (f(x) + \rho g(x))$, where $H = \sum_{x\in\{0,1\}^l} f(x)$ is the original claim and f denotes a $l-$variate polynomial, the prover generates a random masking polynomial g with the same variables and individual degrees of f and the claim $J = \sum_{x\in\{0,1\}^l} g(x)$ is

TABLE II
COMPARISON BETWEEN R1CS AND ARITHMETIC CIRCUIT FOR RELU

Constraints for ReLU(θ)	Non-zero entries	Gate number
① $\forall i \in [\mathcal{K}] : \theta[i] \cdot (1 - \theta[i]) = 0$	$\mathcal{K}$	$2\mathcal{K}$
② $(1 - sgn\theta) \cdot (1 + sgn\theta) = 0$	1	3
③ $\theta = sgn\theta \cdot \sum_{i=1}^{\mathcal{K}} 2^{i-1} \cdot \theta[i]$	$\mathcal{K}$	$2\mathcal{K}$
④ $\mathcal{I} = (sgn\theta + 1)/2$	2	2
⑤ $ReLU(\theta) = \mathcal{I} \cdot \theta$	1	1

sent to the verifier. ρ is selected by the verifier. At the end of zk-sumcheck protocol, the claim $H + \rho J$ will be reduced to a new claim $f(r) + \rho g(r)$. The prover and the verifier can verify the correctness of the new claim by commitment scheme. We use a transparent polynomial commitment scheme proposed in [33].

Theorem 1: Zero knowledge sumcheck protocol for matrix operations is a zero knowledge argument by Definition 6.

Proof Sketch: The proof of completeness, soundness and zero knowledge is similar to that of the zero-knowledge sumcheck protocol in [7]. We only add public relation functions to construct sumcheck formation for matrix operations, which does not affect the proof. We omit the formal proof here.

D. Non-Arithmetic Operators Verification

To effectively reuse the claims for matrix operations, we express the non-arithmetic operators by $Op(\cdot)$ which works on the elements composing the matrices

$$A'(b) = Op(A(b)), \tag{12}$$

where A is a $m \times n$ matrix and $b \in \{0,1\}^{\log mn}$. Since A typically appears in the multi-linear extension form

$$\dot{A}(u) = \sum_{b \in \{0,1\}^{\log mn}} \tilde{\beta}(u,b)A(b) + Z(u) \sum_{z \in \{0,1\}} R(u_1, z), \tag{13}$$

the elements of $Op(A)$ are merged into a new multilinear extension

$$\dot{A}'(u') = \sum_{b \in \{0,1\}^{\log mn}} \tilde{\beta}(u',b)A'(b) + Z(u') \sum_{z \in \{0,1\}} R'(u'_1, z), \tag{14}$$

where u' denotes the newly introduced randomness from the verifier. By the techniques in [37], we represent (12), (13), and (14) by R1CS instances and verify them by a ZK protocol [34].

We provide an example of constraints for the bitwise decomposition of ReLU operations in Table II, and compare the overhead required to verify it with that of [5]. To ensure a fair comparison, we quantify the number of non-zero entries in R1CS and the number of gates in the bitwise decomposition circuit. This metric is crucial because the prove time of the ZK protocol for R1CS instances is linear with the number of non-zero entries in the instances, and the prove time of the ZK protocol for circuits is linear with the number of gates in the circuit. ReLU(θ) can be written equivalently into five constraints: ① $\theta[i], \forall i \in [\mathcal{K}]$ is the bit representation of θ, which is either 0 or 1; ② A sign variable sgn is either -1 or 1; ③ Bit decomposition of θ; ④ The definition of indicator variable; ⑤ The definition of ReLU.

For ReLU(A), the largest number of non-zero entries are $mn(2\mathcal{K}+4), mn+1, mn+1$, respectively, for (12), (13), and (14), which leads to a prover time of $mn(2\mathcal{K}+6)+2$ in total. Expressing in arithmetic circuits, on the other hand, saves the conversion of (13), and (14), and thus requires a prove time of $mn(4\mathcal{K}+6)$, which is linear to the circuit size. Hence the two representations have the same order of prove time with R1CS lesser by a constant.

Theorem 2: Non-arithmetic operators zero knowledge argument scheme is a zero knowledge argument by Definition 6.

Proof Sketch: The proof of completeness, soundness and zero knowledge follows that of zero-knowledge arguments for R1CS in [34] and Schwartz-Zippel lemma [35], [36]. We omit the formal proof here.

VI. ZERO KNOWLEDGE ARGUMENTS FOR BACKPROPAGATION

The high-level idea of *VPNNT* is to express all the computations of neural networks training (backpropagation) in matrix forms and use the zero knowledge arguments for matrix operations we built in Section V to verify the correctness. We first show how propagation across layers can be expressed by matrix operations, how claims are combined, and then reveal the full protocol at the end of the section. A working example of *VPNNT* is displayed in Fig. 2 where we show how the forward and backward propagations are expressed in matrix forms and how claims are combined. The computation correctness is verified in units of matrix operations. To combine the claims containing the same matrix, we employ a method wherein new randomness are generated by interpolating the randomness in these claims, subsequently generating a new claim. Then the problem of verifying the correctness of multiple original claims is transformed into the problem of verifying the correctness of the new claim, which effectively reduces the verification overhead.

A. Definitions

In our setting, the witness is private input data which is used to train a neural network model. The owner of private inputs, that is the prover, can compute gradients locally using its data and generates a proof together with the gradients to convince others, that is the verifier, that the gradients are computed correctly. Similar to previous work [5], [11], [15], we assume that the structure of the neural network model (e.g., the size of input, the type and number of layers, the dimensions of kernels, and the activation functions) is known to the verifier. In the federated learning scenario, we further consider that the central server acting as the verifier also knows the weights of the model. We recognize that the structure and weights of the model will also leak information in some scenarios but do not consider such leakage in our case. A potential solution is to introduce sub-zero-knowledge protocols to hide the structure and weights of the neural network model, which will be left as our future work. Our zero knowledge arguments for backpropagation in this paper only focus on the privacy of inputs.

Formally speaking, let X denotes the private inputs, $\mathcal{M}$ denotes the neural network model, ∇_W denotes the gradients. Let $\nabla_W = \mathcal{B}(X, \mathcal{M})$ denotes the backpropagation process of

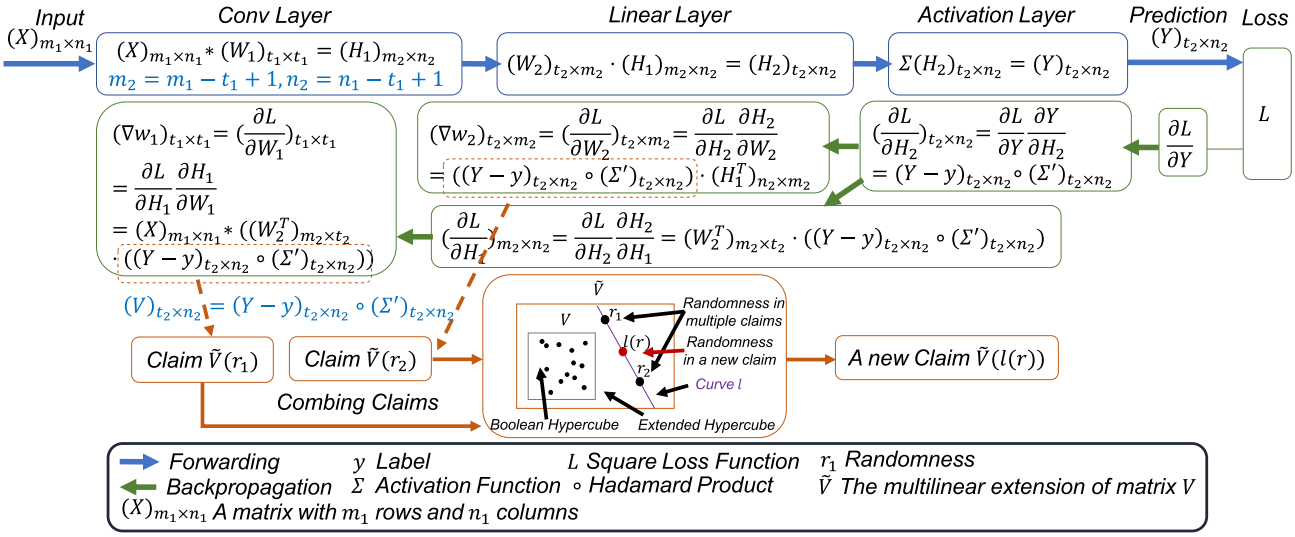


Fig. 2. Example of *VPNNT* applied to a model including a convolutional layer, a linear layer, and an activation layer. The gradients $\nabla_{W_1}, \nabla_{W_2}$ are expressed in matrix forms, and each matrix operation correspond to a claim to be verified. Repeatedly occurring matrix operations can be combined into a single claim.

neural network model where $\mathcal{B}$ represents the gradients calculation formula. A verifiable, and privacy-preserving neural network training (*VPNNT*) framework consists of the following algorithms:

- $(pp, vp) \leftarrow$ *VPNNT*.KeyGen(1^λ): Given the security parameter λ, the algorithm generates the private parameter pp and public parameter vp.
- $com_X \leftarrow$ *VPNNT*.Commit(X, r, pp):The algorithm commits the parameter X which is the private input data using the randomness r and private parameter pp.
- $(\nabla_W, \pi) \leftarrow$ *VPNNT*.Prove$(X, \mathcal{M}, \mathcal{B}, r)$: Given private input data X, a neural network model $\mathcal{M}$ and the gradients calculation formula $\mathcal{B}$, the algorithm runs backpropagation process to get gradients $\nabla_W = \mathcal{B}(X, \mathcal{M})$ and generates the proof π using randomness r.
- $\{0, 1\} \leftarrow$ *VPNNT*.Verify$(com_X, \mathcal{M}, \mathcal{B}, \nabla_W, \pi, vp)$: The algorithm verifies the gradients ∇_W with the commitments com_X, the model $\mathcal{M}$, the gradients calculation formula $\mathcal{B}$, the proof π and the public parameter vp.

VPNNT is correct, where the prover returns correct gradients and can pass the verification; it is sound, where the probability that the prover returns wrong gradients and passes the verification is negligible; it is also zero knowledge, where the proof leaks no information about the private inputs. We give the formal property definitions in the supplementary file due to space limitation. The proof of property is given in Theorem 5 (See Section VI-D for details).

B. Backpropagation in Matrix Forms

We discuss the two most representative layers to show how they can be transformed into matrix operations. Other layers follow the idea in a similar way. In fact, in the plaintext domain, turning neural network operations into tensor, matrix, or vector operations facilitates batch processing, making it ready to be deployed on GPUs for speedup. With a similar spirit, we turn neural network operations into matrix operations to take advantage of the reduced prove time, verification time and proof size compared to the gate-level zero knowledge argument schemes (see Section IV for the comparison).

Linear Layer Backpropagation: The inference process of consecutive linear layers can be written in matrices as

$$\begin{aligned} f(X) &= f(X; W) \\ &= \Sigma_k(W_k \cdot \Sigma_{k-1} \cdots \Sigma_2(W_2 \cdot \Sigma_1(W_1 \cdot X)) \cdots), \end{aligned} \quad (15)$$

where X is the input matrix, W_i are the weight matrices and Σ_i are the activation functions. Backpropagation of the linear layers can be derived by repeated application of the chain rule. For concise expression, we use the following notations: the usual column-by-row product is defined as $A \cdot B$; the Hadamard product is $A \circ B$; and the reversed matrix product is $A \bullet B = B \cdot A$. The backpropagation equation of each layer is listed in the following from the output to the input layer:

$$\begin{cases} \nabla_{W_k} f = \Sigma_k' \cdot \Sigma_{k-1}^T \\ \nabla_{W_{k-1}} f = \Sigma_k' \bullet W_k^T \circ \Sigma_{k-1}' \cdot \Sigma_{k-2}^T \\ \quad \vdots \\ \nabla_{W_2} f = \Sigma_k' \bullet W_k^T \circ \Sigma_{k-1}' \bullet W_{k-1}^T \cdots \circ \Sigma_2' \cdot \Sigma_1^T \\ \nabla_{W_1} f = \Sigma_k' \bullet W_k^T \circ \Sigma_{k-1}' \bullet W_{k-1}^T \cdots \Sigma_2' \bullet W_2^T \circ \Sigma_1' \cdot X^T. \end{cases} \quad (16)$$

Σ_i' denotes the derivative of Σ_i for $i = 1, \ldots, k$, which can be further written in terms of model weights. It is easy to find that these equations are composed by matrix multiplications and Hadamard products, which can be verified by our designed sumcheck protocols.

Convolution Layer Backpropagation: Similarly, we write the inference process of consecutive convolutional layers as

$$f = \Sigma_k(\Sigma_{k-1}(\cdots \Sigma_2(\Sigma_1(X * W_1) * W_2) \cdots) * W_{k-1}) * W_k), \quad (17)$$

where X is the input matrix, W_i are the convolution kernels, Σ_i are the activation functions, and $*$ denotes a convolution operation. Let $\tilde{*}$ denote the reversed convolution: $A\tilde{*}B = B * A$. $Rot_{180}(A)$ means rotating the elements of matrix A by 180 degrees and $\circledast$ represents full convolution in which the input requires zero paddings. By the chain rule, the backpropagation of convolution layers can be written as in (18) shown at the bottom of this page. Zero knowledge arguments for matrix operations introduced in Section V are used to verify the correctness of the gradients.

C. Combining Claims

It is observed that the same structure occurs repeatedly in the formula of (16) and (18), and the equations are recursive by themselves. In verifying gradients of multiple layers, we propose a method to combine multiple claims, i.e., the multi-linear extension forms, of the same matrix, to reduce the prove time, verification time and proof size. Intuitively, the method exploits the layered structure of the neural network, and combines claims of the same component but with different randomness, to remove redundant verification procedures.

In verifying the neural network backpropagation, claims are reduced from the right to the left in (16) and (18). At each sumcheck, different random numbers are generated at the end for the multi-linear extensions of the same matrix. Assume the prover and the verifier collect K claims of the same matrix, and each claim is associated with a set of random numbers (generated during sumcheck). Hence there are K sets of random numbers in total. The verifier generates a degree $K-1$ curve through the K random numbers by interpolation. Then the verifier randomly selects a new point on the curve as the randomness for the combined claim. The details of combining claims are shown in Protocol 2.

Theorem 3 establishes completeness and soundness of Protocol 2, and Theorem 4 gives a brief proof of the zero knowledge property. Benefiting from the recursive structures, the verification of backpropagation is significantly sped up by combining claims for the same matrices.

Theorem 3: Let $\dot{A}$ be a multi-linear polynomial with a masking polynomial over $\mathbb{F}$ in $2\log n$ variables. If $h(x) = \dot{A}(\gamma(x))$, then $h(i) = \dot{A}(\gamma(i)) = \dot{A}(r_i)$ for $i = 1, 2, \ldots, K$, and $h(r) = \dot{A}(\gamma(r)) = \dot{A}(r^*)$ for all $r \in \mathbb{F}$. Meanwhile, if $h(x) \neq \dot{A}(\gamma(x))$, then with probability at least $1 - 2(K-1)\log n/|\mathbb{F}|$ over a randomly chosen $r \in \mathbb{F}$, $\dot{A}(\gamma(r)) \neq h(r) \neq \dot{A}(r^*)$.

Proof: The first half of theorem is immediate from the fact that $\gamma(i) = r_i$ for $i = 1, 2, \ldots, K$. For the second half, we observe that both $h(x)$ and $\dot{A}(\gamma(x))$ are univariate polynomials of degree at most $2(K-1)\log n$. If they are not the same polynomial, given r chosen at random from $\mathbb{F}$, we have $\dot{A}(\gamma(r)) \neq h(r) \neq \dot{A}(r^*)$ with probability at least $1 - 2(K-1)\log n/|\mathbb{F}|$ as suggested by the Schwartz-Zippel lemma [35], [36].

Protocol 2: Combining Claims.

Input: K claims $\dot{A}(r_1), \dot{A}(r_2), \ldots \dot{A}(r_K)$ about a same matrix A with different randomness $r_1, r_2, \ldots, r_K \in \mathbb{F}^{2\log n}$ where n denotes the size of matrix A.

Output: A new claim $\dot{A}(r^*)$ about the matrix A with randomness r^*.

1: $\mathcal{V}$ defines a degree $(K-1)$ curve $\gamma : \mathbb{F} \to \mathbb{F}^{2\log n}$ with $\gamma(i) = r_i$ for $i = 1, 2, \ldots, K$ and sends $\gamma(x)$ to $\mathcal{P}$.
2: $\mathcal{P}$ sends $\mathcal{V}$ a degree $2(K-1)\log n$ univariate polynomial $h(x) = \dot{A}(\gamma(x))$.
3: $\mathcal{V}$ checks that $q(i) = \dot{V}(r_i)$ for $i = 1, \ldots, K$. If failing, the protocol will be aborted.
4: $\mathcal{V}$ randomly selects $r \in \mathbb{F}$ and computes a new claim $\dot{A}(r^*) = h(r) = \dot{A}(\gamma(r))$ on $r^* = h(r) \in \mathbb{F}^{2\log n}$.
5: Return the new claim $\dot{A}(r^*)$.

Theorem 4: Protocol 2 is zero-knowledge.

Proof Sketch: The polynomial $\dot{A}$ is a multi-linear polynomial with a masking polynomial shown as (11). The interaction between $\mathcal{P}$ and $\mathcal{V}$ are the masked values, and thus $\mathcal{V}$ obtains nothing on the real values.

D. The Full Protocol

With the building blocks presented in Sections VI-B and VI-C, we construct a privacy-preserving neural network training verification scheme. The details are shown in Protocol 3. The prover commits to the private input X by the commitment step in zkVPD (the details can be found in [33]). Then the prover computes the gradients of the neural network on the private input X, and sends them to the verifier. After the prover releasing the calculation process and the lookup table which will be used in the sumcheck protocol, the prover and verifier interactively perform the verification process in Line 3 to Line 9 in Protocol 3. During the verification process, if multiple claims about the same matrix are recorded, the prover and the verifier invoke Protocol 2 to combine the claims. Finally, the prover and the verifier verify the correctness of the claims by going through the open-commitment step and the verify-commitment step in zkVPD.

$$
\begin{cases}
\nabla_{W_k} f = \Sigma'_k \tilde{*} \Sigma_{k-1} \\
\nabla_{W_{k-1}} f = \Sigma'_k \circledast Rot_{180}(W_k) \circ \Sigma'_{k-1} \tilde{*} \Sigma_{k-2} \\
\nabla_{W_{k-2}} f = \Sigma'_k \circledast Rot_{180}(W_k) \circ \Sigma'_{k-1} \circledast Rot_{180}(W_{k-1}) \circ \Sigma'_{k-2} \tilde{*} \Sigma_{k-3} \\
\quad \vdots \\
\nabla_{W_2} f = \Sigma'_k \circledast Rot_{180}(W_k) \circ \Sigma'_{k-1} \circledast Rot_{180}(W_{k-1}) \circ \Sigma'_{k-2} \circledast Rot_{180}(W_{k-2}) \cdots \circ \Sigma'_2 \tilde{*} \Sigma_1 \\
\nabla_{W_1} f = \Sigma'_k \circledast Rot_{180}(W_k) \circ \Sigma'_{k-1} \circledast Rot_{180}(W_{k-1}) \circ \Sigma'_{k-2} \circledast Rot_{180}(W_{k-2}) \cdots \circ \Sigma'_2 \circledast Rot_{180}(W_2) \circ \Sigma'_1 \tilde{*} X.
\end{cases}
\tag{18}
$$

Protocol 3: VPNNT: Verifiable and Privacy-Preserving Neural Network Training (Sketch).

For a neural network ψ, the prover $\mathcal{P}$ try to convince the verifier $\mathcal{V}$ that the gradients $\nabla_{W_1}, \nabla_{W_2}, \ldots, \nabla_{W_k}$ are correctly computed on the private input matrix (witness) X. For simplicity, we omit public parameter pp, vp and the details of zkVPD scheme (which can be found in [33]) without any ambiguity.
Let Q_i stores the computation process of ∇_{W_i} in matrix forms for $i = 1, \ldots, k$, and T stores the claims of matrices generated by sumcheck protocol. $\mathcal{P}$ releases $Q_1, Q_2, \ldots, Q_k$ and the lookup tables involved. $\mathcal{P}$ commits the witness X.
for $i = 1, \ldots, k$: **do**
    Verify Q_i successively with matrix operations as the basic unit.
    if the current matrix operation is non-arithmetic **then**
        Model the non-arithmetic operation by R1CS, and invoke sumcheck protocol for R1CS equations to verify the correctness. If failing, $\mathcal{V}$ outputs 0 and abort.
    else
        Invoke sumcheck protocol for matrix operations introduced in Section V to verify the correctness. If failing, $\mathcal{V}$ outputs 0 and abort.
    end if
    At the end of the sumcheck protocol, $\mathcal{V}$ receives claims of the matrices in the current round. $\mathcal{P}$ and $\mathcal{V}$ look up whether there are other claims associated with the same matrix in T. If so, invoke Protocol 2 to get the new claims and store them in T.
end for
$\mathcal{P}$ and $\mathcal{V}$ verify the correctness of all claims in T by opening and verifying the commitment in zkVPD, and $\mathcal{V}$ rejects if either fails. If all checks pass, $\mathcal{V}$ output 1.

Theorem 5: Protocol 3 is a zero knowledge argument by Definition 6.

The correctness of our scheme follows the correctness of the sumcheck protocol for matrix operations proven in Theorem 1, the correctness of Protocol 2, which is established by Theorem 3 and the correctness of the sumcheck protocol for R1CS, as proven in [34]. The soundness is implied by the soundness of the sumcheck protocol introduced in Section III-B, the soundness of the claim combination scheme in Section VI-C proven in Theorem 3 and the zkVPD scheme proven in [33]. We give the proof of zero knowledge here.

Proof: (Zero Knowledge.) For every verifier V^*, there exists a simulator $\mathcal{S}$ such that given oracle access to the output $\nabla_{W_1}, \nabla_{W_2}, \ldots, \nabla_{W_k}$, $\mathcal{S}$ is able to simulate the view of V^* in Protocol 3. The details are as follows.

With oracle access to the output $\nabla_{W_1}, \nabla_{W_2}, \ldots, \nabla_{W_k}$, and the simulator $\mathcal{S}_1$ of the zero-knowledge arguments for matrix operations, the simulator $\mathcal{S}_2$ of the zero-knowledge arguments for R1CS introduced in [34], the simulator $\mathcal{S}_3$ of Protocol 2 and the simulator $\mathcal{S}_4$ of zkVPD introduced in [33] as subroutines, we construct the simulator $\mathcal{S}$ as following:

1) $\mathcal{S}$ initializes the relevant parameters and sends the output $\nabla_{W_1}, \nabla_{W_2}, \ldots, \nabla_{W_k}$ to V^*.
2) $\mathcal{S}$ releases $Q_1^*, Q_2^*, \ldots, Q_k^*$ and the lookup tables involved.
3) $\mathcal{S}$ commits the witness X^* by invoking $\mathcal{S}_4$.
4) For $i = 1, 2, \ldots, k$:
    a) If the current matrix operation is non-arithmetic, $\mathcal{S}$ calls $\mathcal{S}_2$ to simulate the partial view of the zero-knowledge sumcheck protocol for R1CS.
    b) Otherwise, $\mathcal{S}$ calls $\mathcal{S}_1$ to simulate the partial view of the zero-knowledge sumcheck protocol for matrix operations.
    c) At the end of the sumcheck protocol, V^* receives claims of the matrices in the current round. $\mathcal{S}$ and V^* look up whether there are other claims associated with the same matrix in T^*. If so, $\mathcal{S}$ calls $\mathcal{S}_3$ to simulate the partial view of Protocol 2 and gets the new claims to store in T^*.
5) $\mathcal{S}$ and V^* verify all claims in T^* by invoking $\mathcal{S}_4$ to open and verify all the commitment.

To prove zero-knowledge, step 1 and step 2 are obviously indistinguishable in both real world and ideal world as the output $\nabla_{W_1}, \nabla_{W_2}, \ldots, \nabla_{W_k}$, $Q_1^*, Q_2^*, \ldots, Q_k^*$ and the lookup tables are public. Step 3 and step 5 of both worlds are indistinguishable because of the zero knowledge property of the zkVPD proven in [33]. Step 4(a) is indistinguishable as proven in [34] for zero-knowledge sumcheck protocol for R1CS. Step 4(b) is indistinguishable as proven in Theorem 1 for $\mathcal{S}_1$ and Step 4(c) is indistinguishable as proven in Theorem 4 for $\mathcal{S}_2$ in both real world and ideal world, which completes the proof. □

Complexity: The prover time of Protocol 3 is $O(\sum_{i=0}^{m} S_i + N)$ where $S_i = (w_i^2 + n_i^2) ch_{in,i} ch_{out,i}$ and N denotes the R1CS constraints number in order to represent the computation process of non-arithmetic operations. Other denotations include m the total layer numbers, n_i denoting the dimension of input matrix in layer i, w_i the dimension of weight matrix in layer i, $ch_{in,i}$, $ch_{out,i}$ representing the input channel and the output channel in layer i, respectively. The verifier time and the proof size is $O(\sqrt{N} + \sum_{i=0}^{m} \log S_i)$ accordingly.

VII. Implementation and Evaluation

We show the implementation details, and report the experimental results compared with baselines.

A. Implementation Details

VPNNT is implemented by around 7000 lines of `C++` code. Partial implementation of the sumcheck protocol is based on the open-source library `Libra` [38] and `Virgo` [39] whereas the zero knowledge arguments for R1CS is adapted from `Spartan` [40] for non-arithmetic operations. We select the zero knowledge verifiable polynomial delegation scheme in [33], [34] as it has a prove time $O(\mathcal{N})$, a verifier time $O(\sqrt{\mathcal{N}})$ and a proof size $O(\sqrt{\mathcal{N}})$ for a polynomial of size $\mathcal{N}$. The protocol does not need a trusted setup and its security is based on discrete-log assumption. We follow the curve implementation in `zkCNN` [41] to use the FFT-friendly Curve P224 instead of

the Curve-25519 in `Hyrax` [42] to construct the polynomial commitment scheme. *VPNNT* operates on elements in a finite field $\mathbb{F}$, and thus requires floating point numbers to be converted into finite field numbers by scaling. All experiments are done on Intel Xeon E5-2680 with a single CPU core. Potential speedup can be realized through parallel computation. All reported results are averaged over 10 executions.

Baselines include the most recent zero knowledge argument systems *Libra* [7], *Virgo* [8] and *zkCNN* [5]. To run *Virgo* for neural network verification, we turn the forward and backward propagation into arithmetic layered circuits, store network parameters, and export the gates layer by layer to feed into *Virgo*.

To specifically understand the performance on non-arithmetic operations, we compare *VPNNT* to *Libra* on different numbers of ReLU gates within the input range $[-2^9, 2^9)$. *Libra* has a linear prove time faster than *Virgo* and is very efficient for arithmetic circuits.

B. Experimental Setup

Our experiment is divided into two parts, one is to compare the performance of *VPNNT* with baselines on different types of neural network layers, and the other is to evaluate *VPNNT* in the practical setting of federated learning verifications.

To compare the performance on varied neural networks, we select linear layers and convolutional layers of different sizes which are commonly used. The kernel size in convolutional layers is fixed to 3×3. We generate the corresponding arithmetic circuits using our circuit construction program for baselines.

We test *VPNNT* in the setting of verifiable federated learning. In each training round, the trusted server initially distributes the global model parameters to all clients. The clients train the model locally on their private data, generating corresponding proofs in the process. Subsequently, each client uploads their computed gradients to the server, along with the proofs. The central server verifies the proofs by our interactive proof protocol with the clients, aggregates the correctly computed gradients, and then initiates the next round of training. Our experiment specifically focuses on the interaction process between one client and the server, and the verification for other clients is the same.

We choose MNIST [43], a dataset of 70,000 images of hand-written digits with size 28×28, to test LeNet and choose CIFAR-10 [44], a dataset of 60,000 images with size $32 \times 32 \times 3$ in 10 classes to test VGG11 and VGG13. In LeNet, there are 7 layers including 2 convolutional layers, 2 pooling layers and 3 fully-connected layers. In VGG11, there are 8 convolutional layers, 4 pooling layers and 3 fully-connected layers and VGG13 has 2 more convolutional layers than VGG11. Limited by the memory space of the test machine and the capability of *Virgo*, we reduce the channel of the convolution kernel in the VGG structure network to 1/32 of the original and adopt average pooling for the pooling layers. For the cross-layer gates in the arithmetic circuit, we use the relay gates [7] to patch the circuit into a layered one to meet the requirement of *Virgo*. Since the protocol of *zkCNN* cannot verify cross-layer input gates located in other layers except the input layer, we only compare with the modified *Virgo*.

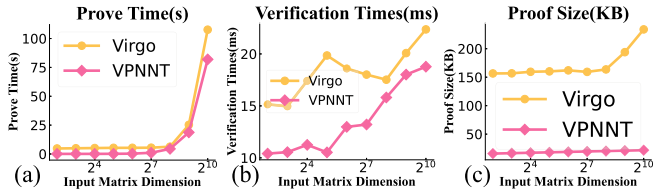


Fig. 3. Comparison of prover time, verification time and proof size between *VPNNT* and *Virgo* on convolutional layers.

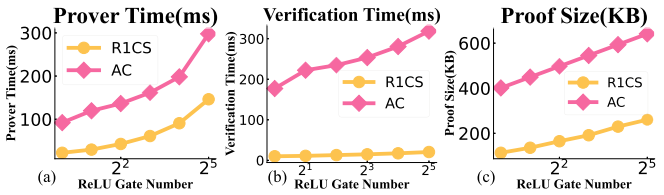


Fig. 4. Comparison of prover time, verification time and proof size between represented-in-R1CS verified by *VPNNT* and represented-in-arithmetic-circuit(AC) verified by *Libra* on ReLU operations.

C. Results and Analysis

Fully-Connected Layers and Convolutional Layers: Table III and Fig. 3 show the prover time, the verification time, and the proof size of *VPNNT*, in comparison with *Virgo* and *zkCNN*. Matrix dimension up to 2^{10} is selected, as we consider any matrix larger than $(2^{10})^2$ is rare in neural networks. The prover time of *VPNNT* is 1.3–1000× faster than *Virgo* depending on different layers. The verification time of *VPNNT* is approximately at the same order (ms) with *Virgo*. It should be noted from Fig. 3(c) that *Virgo* has an almost constant proof size when the input size is small. This is because *Virgo* is designed for large-scale circuits verification and thus any circuit has to be patched to a certain size to be verified.

Similarly, *VPNNT* has advantages in prove time, verification time and proof size compared with *zkCNN*. The speedup is mainly contributed by the improved sumcheck protocol for matrix operations. In addition, due to the layer-by-layer verification of arithmetic circuits by the GKR protocol, in the zero knowledge verifiable polynomial commitment scheme of *Virgo* and *zkCNN*, all gates in the input layer of the circuit need to be committed, including both the private inputs and the public inputs. In contrast, *VPNNT* only selectively commits to the private input matrices since it adopts matrix as a basic unit for verification, which is another speedup factor.

Non-Arithmetic Operations: From Fig. 4, it can be found that the prover time, verification time and proof size of *VPNNT* are competitive. The prover time of *VPNNT* is 2–4× and the verification time is 16–20× faster than *Libra*. Besides, *VPNNT* has a proof size 2–3.5× smaller than that of *Libra*, exhibiting an obvious advantage in communication-intensive situations.

Federated Learning: The results are shown in Table IV. We compare the prover time, the verification time and the communication overhead of each worker running VeriFed using *VPNNT* and *Virgo*. Obviously, *VPNNT* is highly efficient with a much smaller cost. In particular, the prover time is 1.4–2.3× faster than *Virgo* under a similar verifier time. We also show the training time along with other costs. It can be observed

TABLE III
COMPARISON OF PROVE TIME, VERIFICATION TIME AND PROOF SIZE BETWEEN VPNNT AND BASELINES ON FULLY-CONNECTED LAYERS

2^N	Virgo			zkCNN			VPNNT		
	PT (s)	VT (ms)	PS (KB)	PT (ms)	VT (ms)	PS (KB)	PT (ms)	VT (ms)	PS (KB)
2^2	2.51	3.66	153.41	0.36	0.23	2.47	0.31	0.22	0.72
2^3	2.52	3.65	156.35	1.57	0.41	3.56	1.01	0.47	1.16
2^4	2.37	3.69	159.58	3.54	2.58	5.03	1.28	0.88	1.78
2^5	2.23	3.71	162.29	5.80	4.87	7.25	2.38	1.16	2.78
2^6	2.54	3.78	165.87	28.85	7.55	10.97	6.79	2.87	4.53
2^7	3.77	3.82	166.42	94.97	22.51	17.69	23.82	4.92	7.78

2^N denotes the matrix dimension patched to the nearest power of 2, e.g., 2^5 means a 32×32 input matrix is multiplied by a weight matrix of the same size. PT, VT, PS means prove time, verification time and proof size, respectively.

TABLE IV
COMPARISON OF THE COSTS IN VERIFIABLE FEDERATED LEARNING

Cost	LeNet		VGG11		VGG13	
	Virgo	VPNNT	Virgo	VPNNT	Virgo	VPNNT
TT (s)	0.30	0.30	1.63	1.63	2.14	2.14
PT (s)	8.95	6.37	153.51	65.36	154.41	67.88
VT (s)	0.17	0.54	3.08	2.41	3.31	2.56
CO (s)	9.05	0.76	12.96	1.06	13.51	1.07

TT, PT, V T, CO mean the training time, the prover time, the verification time, and the communication overhead, respectively.

that the prover time of *VPNNT* is 30–40× of) the training time, which is reasonable as the training can be accelerated by GPU operations, but the verification procedure faces difficulty turning into batch operations. We believe that it leaves an opportunity for further optimization. The small prove time, which is far less than the baseline, would encourage each FL worker to submit a proof along with the gradients. The verification time of the two schemes is on the same order of magnitude as the training time. We claim that the verification cost is acceptable, as the FL server, acting as the verifier, can leverage multi-core CPUs to parallelize the verification of multiple proofs, which avoids incurring significant latency for the iterative training process. Moreover, *VPNNT* reduces the proof size of *Virgo* by 10×, exhibiting a large saving in the transmission cost and a reduction in the delay between the server and the worker. Overall, the experimental results in verifiable FL have shown the strength of *VPNNT* to applied in real-world settings.

VIII. CONCLUSION

We present *VPNNT*, an efficient, verifiable, and privacy-preserving neural network training framework based on zero-knowledge proofs. Our framework is featured by the custom gates for matrix operations with an optimized interactive proof protocol. By expressing the backpropagation in matrix forms, *VPNNT* is able to efficiently verify the neural network training. The superior efficiency of *VPNNT* is demonstrated by experiments on different types of neural network layers as well as the practical federated learning verification setting. Currently, designing suitable and efficient custom gates for specific tasks is a research hotspot. It is still an open problem to design general-purpose custom gates of high efficiency.

ACKNOWLEDGMENT

The authors would like to appreciate the Student Innovation Center of SJTU for providing GPUs.

REFERENCES

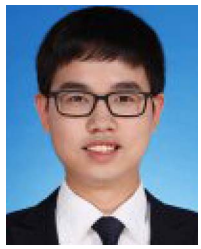
[1] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st Ed., Cham, Switzerland: Springer, 2017.
[2] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 508–519.
[3] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1486–1500, 2019.
[4] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 118–128.
[5] T. Liu, X. Xie, and Y. Zhang, "ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2968–2985.
[6] J. Zhang et al., "Doubly efficient interactive proofs for general arithmetic circuits with linear prover time," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 159–177.
[7] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2019, pp. 733–764.
[8] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 859–876.
[9] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Proc. Annu. Cryptol. Conf.*, 2013, pp. 71–89.
[10] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "HyperPlonk: Plonk with linear-time prover and high-degree custom gates," *Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2023, pp. 499–530.
[11] S. Lee, H. Ko, J. Kim, and H. Oh, "vCNN: Verifiable convolutional neural network based on zk-SNARKS," *IEEE Trans. Dependable Secure Comput.*, pp. 1–17, 2023, doi: 10.1109/TDSC.2023.3348760.
[12] Z. Ghodsi, T. Gu, and S. Garg, "SafetyNets: Verifiable execution of deep neural networks on an untrusted cloud," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4672–4681.
[13] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero knowledge proofs for decision tree predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 2039–2053.
[14] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 911–926, 2019.
[15] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences," *Cryptol. ePrint Arch.*, 2021. [Online]. Available: https://eprint.iacr.org/2021/087
[16] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl, "Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions," in *Proc. 41st Annu. Int. Cryptol. Conf.*, Springer, 2021, pp. 92–122.

[17] S. Dittmer, Y. Ishai, and R. Ostrovsky, "Line-point zero knowledge and its applications," *Cryptol. ePrint Arch.*, 2020. [Online]. Available: https://eprint.iacr.org/2020/1446
[18] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1074–1091.
[19] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2986–3001.
[20] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi, "Privacy-free garbled circuits with applications to efficient zero-knowledge," in *Proc. 34th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Sofia, Bulgaria, Springer, 2015, pp. 191–219.
[21] D. Heath and V. Kolesnikov, "Stacked garbling for disjunctive zero-knowledge proofs," in *Proc. 39th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Zagreb, Croatia, Springer, 2020, pp. 569–598.
[22] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 955–966.
[23] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2087–2104.
[24] C. Baum and A. Nof, "Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography," in *Proc. 23rd IACR Int. Conf. Pract. Theory Public-Key Cryptography*, Edinburgh, U.K., Springer, 2020, pp. 495–526.
[25] M. Chase et al., "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1825–1842.
[26] I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster zero-knowledge for boolean circuits," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 1069–1083.
[27] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 525–537.
[28] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "VeriML: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2524–2540, Oct. 2021.
[29] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *Proc. USENIX Secur. Symp.*, 2021, pp. 501–518.
[30] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Conf. Theory Appl. Cryptographic Techn.*, Springer, 1986, pp. 186–194.
[31] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *J. ACM*, vol. 39, no. 4, pp. 859–868, 1992.
[32] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proc. 3rd Innovations Theor. Comput. Sci. Conf.*, 2012, pp. 90–112.
[33] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 926–943.
[34] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2020, pp. 704–737.
[35] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, vol. 27, no. 4, pp. 701–717, 1980.
[36] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Proc. Int. Symp. Symbolic Algebr. Manipulation*, Springer, 1979, pp. 216–226.
[37] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computsation a few steps closer to practicality," in *Proc. 21st USENIX Secur. Symp.*, 2012, pp. 253–268.
[38] Libra, 2019. [Online]. Available: https://github.com/sunblaze-ucb/Libra
[39] Virgo, 2020. [Online]. Available: https://github.com/sunblaze-ucb/Virgo
[40] Spartan, 2020. [Online]. Available: https://github.com/microsoft/Spartan
[41] zkcnn, 2021. [Online]. Available: https://github.com/TAMUCrypto/zkCNN
[42] Hyrax, 2018. [Online]. Available: https://github.com/hyraxZK/hyraxZK
[43] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[44] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (canadian institute for advanced research)," 2010. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

Haohua Duan received the BS degree in computer science and technology from Jilin University, Changchun, China, in 2020. He is currently working toward the PhD degree in computer science with Shanghai Jiao Tong University. His research interests include security and privacy in machine learning and blockchain.

Zedong Peng is currently working toward the fourth-year undergraduate degree majoring in information engineering with Shanghai Jiao Tong University. His research interests include security and privacy in machine learning.

Liyao Xiang (Member, IEEE) received the BEng degree in electrical and computer engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012, and the PhD degree in computer engineering from the University of Toronto, Toronto, ON, Canada, in 2018. She is currently a tenure-track Associate Professor with Shanghai Jiao Tong University. Her research interests include security and privacy, privacy analysis in data mining, and mobile computing.



Yuncong Hu received the bachelor's degree from Shanghai Jiao Tong University, in 2017, where he was a member of the ACM Honored Class, and the CS PhD degree from UC Berkeley RISE Lab, where he was co-advised by Raluca Ada Popa and Alessandro Chiesa. He is currently an assistant professor with Shanghai Jiao Tong University. His research interests include applied cryptography, decentralized systems, and zero-knowledge proofs.

Bo Li (Fellow, IEEE) received the BEng degree in computer science from Tsinghua University, Beijing, and the PhD degree in electrical and computer engineering from the University of Massachusetts at Amherst. He is a professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He was the chief technical advisor of China Cache Corporation (NASDAQ CCIH), the largest CDN operator in China. He was a Cheung Kong visiting chair professor with Shanghai Jiao Tong University (2010–2013) and an adjunct researcher with Microsoft Research Asia (1999–2007) and with Microsoft Advance Technology Center (2007–2009). His current research interests include multimedia communications, Internet content distribution,datacenter networking, cloud computing, and wireless sensor networks. He made pioneering contributions in Internet video broadcast with the system Coolstreaming, which was credited as the world first large-scale peer-to-peer live video streaming system. The work appeared in IEEE INFOCOM (2005) and received the IEEE INFOCOM 2015 Test-of-Time Award. He has been an editor or a guest editor of more than a dozen IEEE journals and magazines. He was the co-TPC chair of IEEE INFOCOM 2004. He received five Best Paper Awards from the IEEE. He received the Young Investigator Award from Natural Science Foundation of China (NFSC), in 2005, and the State Natural Science Award (2nd Class) from China, in 2011.