scientific reports

Check for updates

OPEN

A polynomial proxy model approach to verifiable decentralized federated learning

Tan Li[1]✉, Samuel Cheng[3], Tak Lam Chan[1] & Haibo Hu[1,2]

Decentralized Federated Learning improves data privacy and eliminates single points of failure by removing reliance on centralized storage and model aggregation in distributed computing systems. Ensuring the integrity of computations during local model training is a significant challenge, especially before sharing gradient updates from each local client. Current methods for ensuring computation integrity often involve patching local models to implement cryptographic techniques, such as Zero-Knowledge Proofs. However, this approach becomes highly complex and sometimes impractical for large-scale models that use techniques such as random dropouts to improve training convergence. These random dropouts create non-deterministic behavior, making it challenging to verify model updates under deterministic protocols. We propose ProxyZKP, a novel framework combining Zero-Knowledge Proofs with polynomial proxy models to provide computation integrity in local training to address this issue. Each local node combines a private model for online deep learning applications and a proxy model that mediates decentralized model training by exchanging gradient updates. The multivariate polynomial nature of proxy models facilitates the application of Zero-Knowledge Proofs. These proofs verify the computation integrity of updates from each node without disclosing private data. Experimental results indicate that ProxyZKP significantly reduces computational load. Specifically, ProxyZKP achieves proof generation times that are 30–50% faster compared to established methods like zk-SNARKs and Bulletproofs. This improvement is largely due to the high parallelization potential of the univariate polynomial decomposition approach. Additionally, integrating Differential Privacy into the ProxyZKP framework reduces the risk of Gradient Inversion attacks by adding calibrated noise to the gradients, while maintaining competitive model accuracy. The results demonstrate that ProxyZKP is a scalable and efficient solution for ensuring training integrity in decentralized federated learning environments, particularly in scenarios with frequent model updates and the need for strong model scalability.

Keywords Zero knowledge proof, Computation integrity, Decentralized federate learning

Federated Learning (FL)[1] has revolutionized machine learning by moving away from centralized data processing to decentralized methods, enhancing data privacy, and reducing dependence on centralized data storage. Google researchers initially developed FL, allowing client devices to perform local computations on model updates, which a central server then aggregates. While FL preserves user privacy by keeping data local, it faces significant challenges in multi-institutional collaborations, particularly in sectors such as healthcare, where stringent security standards are mandatory. These problems mostly come from the fact that it relies on a single authority, which leaves it open to problems like a single point of failure and not enough ways to check that model gradient transfers are being done correctly. To overcome these limitations, there is growing adoption of decentralized learning platforms[2,3] because the role of the central aggregator is not required. Therefore, these vulnerabilities can be diminished. However, decentralized architectures[4,5] introduce new complexities in verifying the integrity of model updates without compromising privacy. Advanced cryptographic techniques, like Zero-Knowledge Proof (ZKP)[6,7] which allows one party to prove the correctness of a statement without revealing any additional information, offer a promising solution to these challenges by enabling secure verification of model updates at each local node[8,9] in a decentralized learning environment.

However, conventional ZKP approaches, which often include intricate structures like Rank-1 Constrained System (R1CS) and Quadratic Arithmetic Programs (QAP)[10,11], are challenging to implement on large-scale

[1]Centre for Advances in Reliability and Safety (CAiRS), Hong Kong, China. [2]Department of Electrical and Electronics Engineering, The Hong Kong Polytechnic University, Hong Kong, China. [3]Department of Electrical and Computer Engineering, University of Oklahoma, Tulsa, OK 74135, USA. ✉email: Tan.li@cairs.hk

models. Moreover, random elements such as dropouts during model training, which contradict the deterministic requirements of ZKP procedures, make it fairly impractical to patch the original model for merging with the ZKP process.

This study introduces ProxyZKP, a novel framework that integrates Deep Mutual Learning[12,13] with ZKP to provide the computation integrity signature in training local models. Each node in decentralized architecture operates with a dual-model setup: a private model for online deployment on local data and a proxy model guided by multivariate polynomial loss function[14], providing gradient updates during federated model training. Within the framework of Deep Mutual Learning, the proxy model is linked to the private models to maintain consistency and alignment. The ZKP is used to verify the integrity of the computation on the proxy model's gradient update before the new gradient is exchanged among decentralized peers. ProxyZKP accomplishes training integrity verification of proxy model by efficiently applying Kate Polynomial Commitment[15,16] with a two-stage multivariate polynomial decomposition method. Our numerical experiments demonstrate the efficacy of this approach, showing that ProxyZKP delivers the best trade-off among alternative Zero-Knowledge Proof methods like ZK-SNARKs[17] and Bulletproofs[18] for training integrity verification in decentralized, federated learning models.

Additionally, privacy breaches caused by Gradient Inversion[19] attacks can be mitigated by implementing an additional defense measure such as Differential Privacy (DP)[20,21] after the ZKP process. This involves adding calibrated noise to the gradients, based on a privacy budget (ϵ, δ), before using them for federated model training. Our benchmarks show that this method effectively reduces the risk of Gradient Inversion attacks, where attackers can reconstruct original data from gradient updates. Specifically, a lower privacy budget $(\epsilon = 0.5)$ requires adding significant noise $(\sigma = 1.5)$, resulting in a model accuracy of 60.0% for MLP on the MNIST dataset and 52.3% for CNN on the CIFAR dataset. In contrast, a higher privacy budget $(\epsilon = 5.0)$ reduces the noise level $(\sigma = 0.5)$, leading to an accuracy of 80.1% for MLP and 62.0% for CNN.

The results show that ProxyZKP effectively strikes a balance between privacy and utility. It offers strong protection while maintaining high performance of private models. Therefore, it has the potential to address both the integrity of model training and local data privacy in decentralized federated learning environments.

The previous works of Joen et al.[22] and Zhao et al.[23] also focus on addressing security and privacy concerns in decentralized federated machine learning. The unique aspect of ProxyZKP is its utilization of ZKPs to verify the entire gradient update of the proxy model. This is further strengthened by the application of local differential privacy (LDP), which enhances privacy by adding noise to the verified updates. In contrast, Zhao et al.[23] concentrates on verifying only the linear layers of the machine learning model through a verifiable matrix multiplication technique. Joen et al.[22] achieve privacy by adopting a communication-based approach, without relying on encryption or noise addition. ProxyZKP provides a more comprehensive solution by combining ZKPs and differential privacy to verify the full gradient update and enhance privacy.

Trusted gradient computation in decentralized learning

The basic components that constitute Decentralized Federated Learning[2,3] and the role of gradient computation and verification by Zero-Knowledge Proof (ZKP) under the ProxyZKP framework are discussed in the section. These components work in cohort to provide secure, privacy-preserving, and reliable model training across multiple participants.

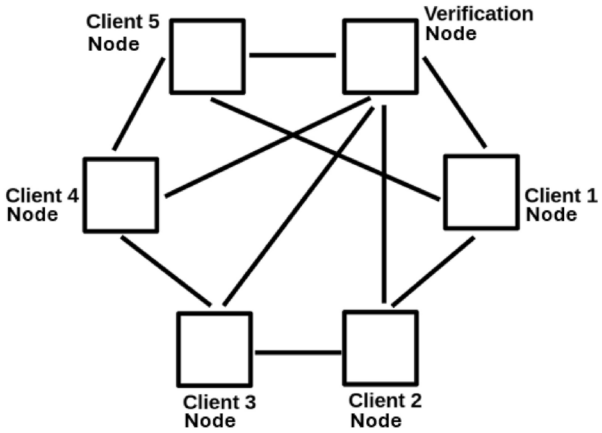
Decentralized federated learning procedure

ProxyZKP, as depicted in Fig. 1a, enables multiple clients to collaboratively train machine learning models without relying on a central aggregation server. It involves several key components:

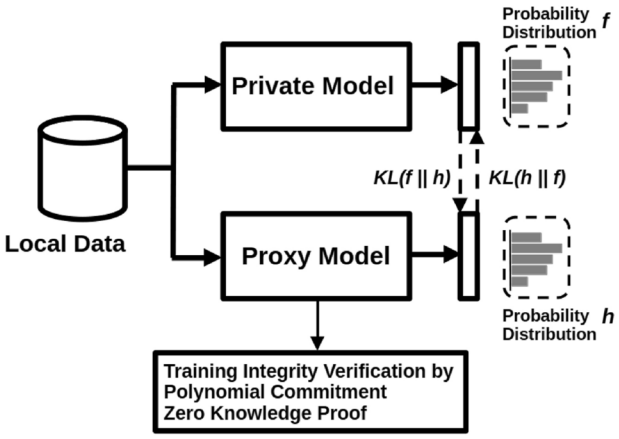
- *Local gradient computation and verification:* As shown in Fig. 1b, each local client independently trains a dual-model setup using its private data under the Deep Mutual Learning framework[12,13]. This setup involves two models: a private model and a proxy model. The private model performs online inference on local data after training completion and remains isolated from the network, while the proxy model provides training updates across the decentralized network. Once the gradients are computed, each client generates a proof of the integrity of the update. This proof, along with the gradient, is transmitted to the network for verification
- *Verification of gradient updates:* Upon receiving proof from each local client, the verifier node checks the integrity of the gradient updates. If the verification succeeds, the gradients are accepted and shared with all other nodes. If the verification fails, the gradient update is discarded to ensure that only valid updates are used.
- *Decentralized gradient averaging at each client:* Each client computes the averaged gradient from all verified updates. This decentralized averaging ensures that every node calculates the same average gradient without needing a central aggregator.
- *Model update and iteration:* After averaging the gradients, each client updates its local model using the averaged gradient. This iterative process continues with further rounds of gradient computations and verifications until the models converge or meet predefined performance criteria.
- *Decentralized communication and coordination:* Efficient communication of model updates is crucial to decentralized machine learning. Network structures such as peer-to-peer, ring, or mesh topologies are commonly used to facilitate the exchange and synchronization of information without the need for a central server.

ProxyZKP in decentralized federated learning

The integration of ProxyZKP within the DFL framework addresses the challenges of securely verifying model updates without revealing sensitive data:



(a) The architecture of the Decentralized Federated Learning Network is depicted, highlighting the network topology and specifically focusing on the verification node, which plays a crucial role in the decentralized learning process.



(b) The dual-model configuration in the local client demonstrates the connection between the private and proxy models using KL divergence. The primary objective is to verify the training integrity of the proxy model by zero-knowledge proof.

Fig. 1. ProxyZKP is a decentralized federated learning technique in which, each client, maintains a private model, a proxy model, and private data. During distributed training, the client exclusively connects with others by sharing their proxy model, which facilitates data and model privacy. The verification node checks the training integrity of each proxy model before it is shared. Following the completion of training, a client can utilize their private model for inference purposes.

- *Proof generation:* Each node generates ZKP on the integrity of its proxy model updates after local training. This proof provides the signature of correct gradient computations without exposing any underlying data. The use of Kate polynomial commitment scheme[15] allows for the compact proof generation, due to the multivariate polynomial nature of the proxy model.
- *Verification and decentralized aggregation:* Verification nodes (or a subset of nodes) validate the submitted ZKP to ensure that updates are both accurate and untampered before aggregation. This process prevents malicious updates from being included in the new global model and safeguard the reliability of the decentralized federated learning environment.

Dual-model training framework

The dual-model training approach, as shown Fig. 1b, is based the work on Deep Mutual Learning[12]. It involves the simultaneous training of a private deep learning model, $f_{\text{NN}}(x;\theta_{\text{NN}})$, and a multivariate polynomial proxy model, $f_{\text{poly}}(x;\theta_{\text{poly}})$. The training objectives are defined as:

$$L_{\theta_k} = (1-\alpha)\cdot L_{\text{Cross-Entropy}}(f_{\text{NN},\theta_k}) + \alpha \cdot L_2(f_{\text{NN},\theta_k}, f_{\text{poly},\phi_k}), \tag{1}$$

$$L_{\phi_k} = (1-\beta)\cdot L_{\text{MSE}}(f_{\text{poly},\phi_k}) + \beta \cdot L_2(f_{\text{poly},\phi_k}, f_{\text{NN},\theta_k}), \tag{2}$$

where α and β balance the contributions of individual model training and cross-model mismatch. To couple the learning of the two models, we replace the Kullback–Leibler (KL) divergence used in the original Deep Mutual Learning[12] formulation with an L_2 divergence[24] as the distance metric between the outputs of the private model and the proxy model. This modification aligns their predictions while ensuring that the gradient updates for the proxy model f_{poly} remain in multivariate polynomial form, which is crucial for meeting the structural requirements of subsequent cryptographic operations. In this configuration, the private model f_{NN} is trained using the Cross-Entropy loss function, while the proxy model f_{poly} uses Mean Square Error (MSE) as its loss function. The MSE loss function is essential for maintaining the polynomial structure in the gradient updates of the proxy model, as required by the Kate polynomial commitment[15] ZKP process. Soft labels from private model pre-training help address limitations in regression-based training and improve consistency with the private model. For more details on soft label generation, refer to Hinton et al.[25] and Vargas et al.[26].

Multivariate polynomial model for multi-layer perceptron and convolutional neural network

The use of a multivariate polynomial form as the proxy model in neural networks presents challenges due to the increasing complexity of terms, which grows exponentially with the number of inputs (n) and the polynomial degree (d). Our work introduces specialized polynomial proxy models named PolyMLP and PolyCNN to address this issue. These models use a Taylor expansion-like structure for deep learning, where the ReLU activation function is expanded into 2nd order polynomials[27]. The outputs of both the PolyMLP and PolyCNN models can be represented as multivariate polynomials in terms of their input variables. Let $x = (x_1, x_2, \ldots, x_n)$

denote the input vector, where $n = 784$ for the MLP model (MNIST dataset) and $n = 1024$ for the CNN model (CIFAR dataset). The unified polynomial expression for the output of both models is given by:

$$f_{\text{poly}}(x) = \sum_{i_1=0}^{2} \sum_{i_2=0}^{2} \cdots \sum_{i_n=0}^{2} a_{i_1 i_2 \cdots i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}, \tag{3}$$

where:

- $a_{i_1 i_2 \cdots i_n}$ are the polynomial coefficients,
- $i_1, i_2, \ldots, i_n$ are the degrees associated with each variable, andThe multivariate polynomial models approximate the behavior of the original MLP and CNN using quadratic expansions up to the 2^{nd} order. It is important to note that the PolyMLP and PolyCNN models, in their multivariate polynomial forms, have the same number of learnable parameters as the original MLP and CNN models before expansion. This means the complexity in terms of the number of parameters remains the same. However, polynomial models require significantly more dense computations due to the lack of optimizations inherent in neural networks, such as parameter sharing and hardware acceleration[28,29]. Consequently, the original MLP and CNN models are more computationally efficient and better suited for online applications. In contrast, the polynomial models are particularly valuable in contexts where polynomial representations are crucial, such as in cryptographic and privacy-preserving computations[30].

Polynomial model for multi-layer perceptron (PolyMLP)
The Multi-Layer Perception (MLP) model processes input data from the MNIST dataset. The dataset consists of 28×28 grayscale images that are flattened into vectors of 784 elements. The MLP architecture considered in this work includes one hidden layer with 10 neurons, respectively, and an output layer corresponding to the 10 possible digit classes.

Each layer of the MLP performs a linear transformation followed by a non-linear activation function. By approximating the ReLU activation function with a quadratic polynomial, the output of each layer can be expressed as a polynomial:

Layer 1 The input layer computes a linear transformation of the input vector *x*:

$$z_i^{(1)} = \sum_{j=1}^{784} W_{ij}^{(1)} x_j + b_i^{(1)}, \tag{4}$$

where $W_{ij}^{(1)}$ are the weights and $b_i^{(1)}$ are the biases for the first layer. The ReLU activation is approximated quadratically[27] as:

$$\text{ReLU}\left(z_i^{(1)}\right) \approx a_0 + a_1 z_i^{(1)} + a_2 \left(z_i^{(1)}\right)^2. \tag{5}$$

Substituting the expression for $z_i^{(1)}$, the output of the first layer becomes:

$$y_i^{(1)} = a_0 + a_1 \left(\sum_{j=1}^{784} W_{ij}^{(1)} x_j + b_i^{(1)}\right) + a_2 \left(\sum_{j=1}^{784} W_{ij}^{(1)} x_j + b_i^{(1)}\right)^2. \tag{6}$$

Expanding the square term yields a polynomial that includes linear and quadratic terms in the input features x_j.

Layer 2 The output of the first layer is processed by the second hidden layer:

$$z_i^{(2)} = \sum_{k=1}^{10} W_{ik}^{(2)} y_k^{(1)} + b_i^{(2)}. \tag{7}$$

Substituting $y_k^{(1)}$ from the first layer, we obtain:

$$z_i^{(2)} = \sum_{k=1}^{10} W_{ik}^{(2)} \left[a_0 + a_1 \left(\sum_{j=1}^{784} W_{kj}^{(1)} x_j + b_k^{(1)}\right) + a_2 \left(\sum_{j=1}^{784} W_{kj}^{(1)} x_j + b_k^{(1)}\right)^2\right] + b_i^{(2)}. \tag{8}$$

Since we apply a linear activation function in the second layer, the output remains a quadratic polynomial in the input variables x_j.

Polynomial model for convolutional neural network (PolyCNN)
The Convolutional Neural Network (CNN) model processes input data from the CIFAR dataset. Each image is a single color channel flattened into a vector of 1024 elements. The CNN architecture includes two convolutional layers, followed by a fully connected layer and an output layer corresponding to the 10 possible classes. The PolyCNN model applies block Toeplitz matrices[31] to input vectors from the CIFAR dataset.

Layer 1 (Convolution): Each filter in the first convolutional layer is represented as a multiplication by a block Toeplitz matrix T. The convolution operation is represented as:

$$\mathrm{conv}\left(\mathrm{T}, \mathrm{x}\right) = \mathrm{T} \cdot \mathrm{x}, \tag{9}$$

where T is the block Toeplitz matrix and x is the input vector. The ReLU activation is approximated quadratically[27] as:

$$y^{(1)} = a_0 + a_1 \left(\mathrm{T} \cdot \mathrm{x} + b_1\right) + a_2 \left(\mathrm{T} \cdot \mathrm{x} + b_1\right)^2. \tag{10}$$

Expanding the square term gives:

$$\left(\mathrm{T} \cdot \mathrm{x} + b_1\right)^2 = \left(\mathrm{T} \cdot \mathrm{x}\right)^2 + 2b_1 \mathrm{T} \cdot \mathrm{x} + b_1^2, \tag{11}$$

which produces a quadratic polynomial in the input variables x_i with interaction terms.

Layer 2 (Convolution with linear activation) The second convolutional layer applies a linear activation function, meaning no quadratic terms are introduced. The convolution is represented as:

$$\mathrm{conv}_f\left(\mathrm{T}_2, \mathrm{y}^{(1)}\right) = \mathrm{T}_2 \cdot \mathrm{y}^{(1)}, \tag{12}$$

where T_2 is the block Toeplitz matrix in the second layer and $\mathrm{y}^{(1)}$ is the output from the first layer. Since this layer uses a linear activation function, the output $z_f^{(2)}$ is:

$$z_f^{(2)} = \mathrm{T}_2 \cdot \mathrm{y}^{(1)} + b_2, \tag{13}$$

where b_2 is the bias term, and no additional non-linearity is applied.

Complexity comparison
The complexity of both PolyMLP and PolyCNN stems from the use of polynomial activation, which increase computational complexity during forward and backward passes. Specifically, the PolyCNN processes images of size 32×32 (1,024 input elements) and introduces approximately $648{,}000$ quadratic terms in the hidden layer. In contrast, the PolyMLP processes images of size 28×28 (784 input elements) and introduces $307{,}720$ quadratic terms in its first hidden layer. While the polynomial activation capture more complex patterns, they increase the computational load but maintain the same number of learnable parameters as their conventional counterparts. For PolyMLP, with one hidden layer of 10 neurons, the total number of parameters remains 7960, the same as in the original unexpanded MLP. Similarly, for PolyCNN, using block Toeplitz matrices to represent convolutional layers does not increase the parameter count, which stays at 2,590 for a minimal CNN structure with two convolutional layers and one fully connected layer.

Gradient update for the original network parameters with polynomial mapping
To train the model, we update the original network parameters (e.g., weights W and biases b) instead of the polynomial coefficients $a_{i_1 i_2 \ldots i_n}$. The relationship between the original network parameters and the polynomial coefficients is managed by a mapping function $g_{i_1 i_2 \ldots i_n}(\theta_k)$, where θ_k denotes the original parameters. This mapping is handled using symbolic mathematics to compute the final gradients in polynomial form.

The polynomial coefficients $a_{i_1 i_2 \ldots i_n}$ are functions of the original network parameters θ_k:

$$a_{i_1 i_2 \ldots i_n} = g_{i_1 i_2 \ldots i_n}(\theta_k) \tag{14}$$

This relationship captures how the original parameters influence the polynomial coefficients and how the input features interact through the polynomial structure.

The gradient of the loss function L_{ϕ_k} with respect to each original network parameter θ_k is given by:

$$\nabla_{\theta_k} L_{\phi_k} = (1-\beta) \sum_{j=1}^{N} (f_{\text{Poly}}(\phi_k, x_j) - y_j) \cdot \nabla_{\theta_k} f_{\text{Poly}}(\phi_k, x_j) + \beta \sum_{j=1}^{N} (f_{\text{Poly}}(\phi_k, x_j) - f_{\text{CNN}}(\theta_k, x_j)) \cdot \nabla_{\theta_k} f_{\text{Poly}}(\phi_k, x_j), \tag{15}$$

where:

- $f_{\text{Poly}}(\phi_k, x_j)$ is the output of the polynomial model,
- $f_{\text{CNN}}(\theta_k, x_j)$ is the output of the CNN model,
- $\nabla_{\theta_k} f_{\text{Poly}}(\phi_k, x_j)$ is the gradient of the polynomial model with respect to the original network parameters.Using symbolic mathematics, we can compute the gradient of the polynomial model output f_{Poly} with respect to θ_k as follows:

$$\nabla_{\theta_k} f_{\text{Poly}}(\phi_k, x_j) = \sum_{(i_1, i_2, \ldots, i_n)} x_{1,j}^{i_1} x_{2,j}^{i_2} \cdots x_{n,j}^{i_n} \cdot \nabla_{\theta_k} a_{i_1 i_2 \ldots i_n}. \tag{16}$$

Since $a_{i_1 i_2 \ldots i_n}$ is a function of θ_k, a symbolic differentiation tool like SymPy (Meurer et al., 2017)[39] enables symbolic differentiation of polynomial coefficients and original network parameters. Finally, the update rule for each original network parameter θ_k is given by:

$$\theta_k^{(t+1)} = \theta_k^{(t)} - \eta \cdot \nabla_{\theta_k} L_{\phi_k}, \tag{17}$$

where η is the learning rate.

Zero-knowledge proof construction for multivariate polynomial computations

We propose a two-stage proof scheme to verify the accuracy of multivariate polynomial computations using the Kate Commitment ZKP[15]. The scheme begins by breaking down the multivariate polynomial into a series of checks for univariate polynomials. A consistency check is then performed using Sigma protocol to ensure the overall evaluation's validity. Unlike existing methods[32,33] that rely on computationally intensive algebraic structures, our approach transforms the problem into a series of univariate checks, thereby reducing verification computational workload. By simplifying the complexity of Zero-Knowledge Proofs (ZKP), our scheme is well-suited for large-scale applications, such as decentralized federated learning. A detailed schematic of this scheme is shown in Fig. 2.

Decomposition and commitment to univariate polynomials

In order to efficiently and securely verify a multivariate polynomial $P(x_1, x_2, \ldots, x_m)$, the prover breaks it down into a series of univariate polynomials by fixing all variables except one. Upon receiving the challenge point $x_o = (x_{o1}, x_{o2}, \ldots, x_{om})$ from the verifier, the multivariate polynomial $P(x_1, x_2, \ldots, x_m)$ is broken down into univariate polynomials $F_i(x_i)$, as follows:

$$F_i(x_i) = P(x_{o1}, \ldots, x_i, \ldots, x_{om}), \tag{18}$$

where x_o is a fixed value for all variables except x_i. When evaluated at x_o, each individual polynomial $F_i(x_{oi})$ equals the evaluation of the original multivariate polynomial at the point where all variables are set to x_0:

$$P(x_o) = F_i(x_{oi}) \quad \text{for all } i. \tag{19}$$

Next, the prover commits to each univariate polynomial $F_i(x_i)$ using the Kate polynomial commitment scheme, which relies on cryptographic commitments in group G_1 with a trusted setup parameter τ. The commitment to a univariate polynomial $F_i(x_i)$ is calculated as:

$$C(F_i) = g_1^{F_i(\tau)} = g_1^{a_0 + a_1\tau + \cdots + a_d\tau^d}, \tag{20}$$

where g_1 is the generator of the elliptic curve group G_1[34], and a_j are the coefficients of the polynomial. These commitments ensure that the polynomial coefficients are securely encapsulated, allowing the prover to respond to verification challenges while maintaining data privacy.

Commitment to the quotient polynomial

The prover evaluates the univariate polynomial $F_i(x_i)$ at the challenge point x_o, obtaining $F_i(x_{oi})$. Instead of sending this value directly, the prover constructs a quotient polynomial $q_i(x)$ to capture the difference between $F_i(x)$ and its evaluation at x_0:

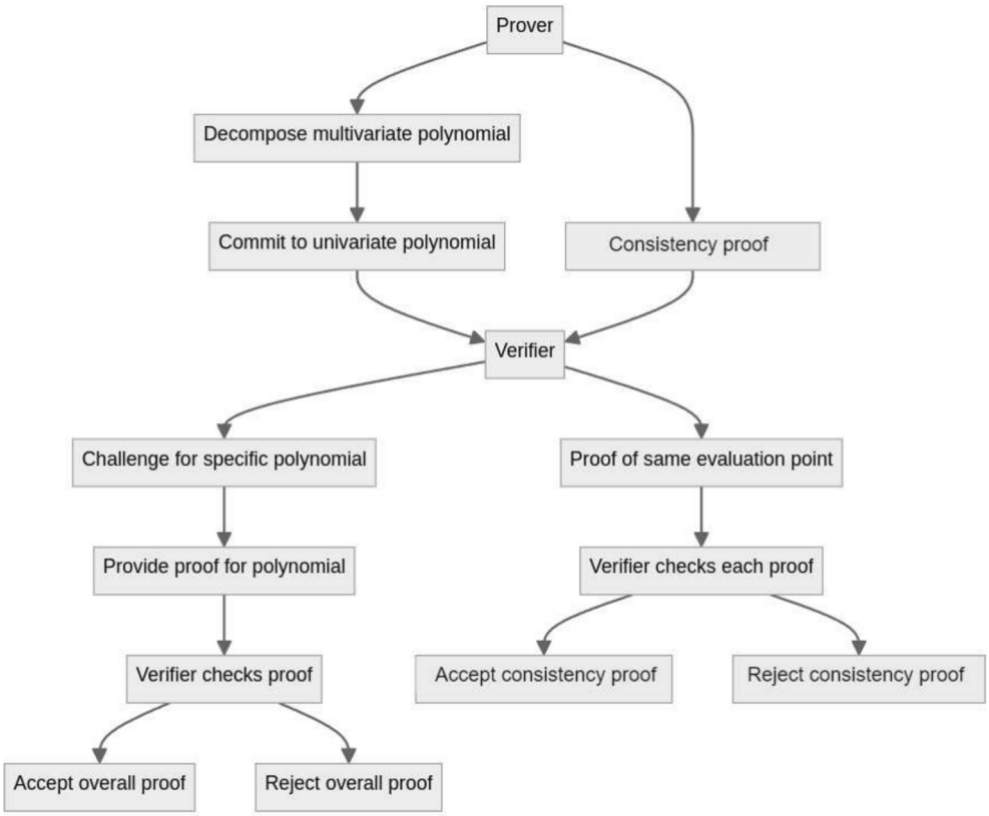


Fig. 2. Two-stage proof of multivariate polynomial computation integrity scheme.

$$q_i(x) = \frac{F_i(x) - F_i(x_{oi})}{x - x_{oi}} \tag{21}$$

This quotient polynomial enables the verifier to confirm the correctness of $F_i(x_0)$.

The prover also computes a commitment to the quotient polynomial $q_i(x)$, denoted as $C(q_i)$:

$$C(q_i) = g_1^{q_i(\tau)} = g_1^{b_0 + b_1\tau + \cdots + b_{d-1}\tau^{d-1}} \tag{22}$$

where b_j are the coefficients of $q_i(x)$.

Proof generation

The prover sends the following to the verifier:

- The commitment $C(F_i)$ to the univariate polynomial $F_i(x_i)$,
- The commitment $C(q_i)$ to the quotient polynomial $q_i(x)$,
- The evaluation $F_i(x_0)$ (not sent directly, but part of the proof).

Verification using bilinear pairing

To verify the correctness of $F_i(x_0)$, the verifier checks the following bilinear pairing equation:

$$e\left(C(F_i) - g_1^{F_i(x_0)},\ g_2\right) \stackrel{?}{=} e\left(C(q_i),\ g_2^{x_0} \cdot h\right) \tag{23}$$

where $e(\cdot,\cdot)$ denotes the bilinear pairing operation[35], g_2 is the generator of a different elliptic curve group G_2[34], and $h = g_2^{-\tau}$ is a public parameter from the trusted setup. If the equation holds, the verifier is convinced of the correctness of $F_i(x_0)$, without learning the actual value.

Sigma protocol for consistency verification

After the zero-knowledge proof for each univariate polynomial $F_i(x_0)$, the prover must demonstrate that these evaluations are consistent with the evaluation of the original multivariate polynomial $P(x_0)$. This consistency is

verified using the Sigma protocol[36], which allows the prover to prove that the committed evaluations $F_i(x_0)$ are consistent with $P(x_0)$, without revealing any additional information.

Commitments to $P(x_0)$ and $F_i(x_0)$
The prover begins by committing to both the evaluation of the multivariate polynomial $P(x_0)$ and each univariate polynomial $F_i(x_{0i})$. The prover commits to the value $P(x_0)$ as:

$$C_P = g^{P(x_0)} h^{r_P} \tag{24}$$

where g and h are generators of a group G, and r_P is a random blinding factor. For each univariate polynomial $F_i(x_0)$, the verifier knows:

$$C_{F_i} = g^{F_i(x_0)} \tag{25}$$

Sigma protocol execution
To prove the consistency between $P(x_0)$ and the set of univariate polynomials $F_i(x_0)$, the prover engages in a Sigma protocol with the verifier. The protocol proceeds as follows: For each $i = 1, \ldots, n$, the prover and verifier follow these steps:

1. *Verifier computes the difference commitment* The verifier calculates the difference between the commitments:

$$C_{d_i} = C_P \cdot C_{F_i}^{-1} = g^{P(x_0) - F_i(x_0)} h^{r_P} \tag{26}$$

Since $P(x_0) = F_i(x_0)$, this simplifies to:

$$C_{d_i} = h^{r_P} \tag{27}$$

2. *Prover's commitment (first message)* The prover picks a random value $s \in \mathbb{Z}_q$ and computes:

$$t = h^s \tag{28}$$

Prover Sends: t to the verifier.

3. *Verifier's challenge (Second Message)* The verifier selects a random challenge $e \in \mathbb{Z}_q$. *Verifier Sends:* e to the prover.
4. *Prover's response (Third Message)* The prover computes:

$$z = s + e \cdot r_P \tag{29}$$

Prover Sends: z to the verifier.

5. *Verifier's verification* The verifier checks if:

$$h^z \stackrel{?}{=} t \cdot (C_{d_i})^e \tag{30}$$

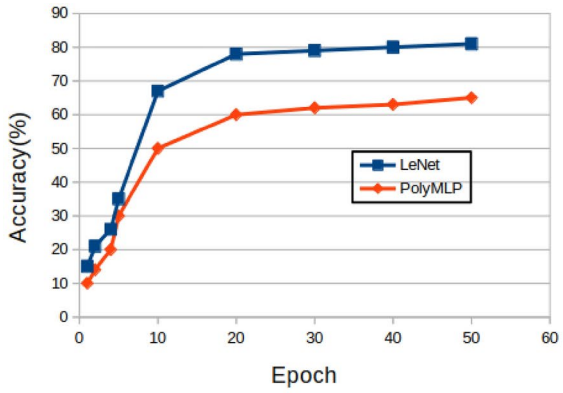
If this holds, the verifier is convinced that $P(x_0) = F_i(x_0)$. This sigma protocol efficiently proves that $P(x_0) = F_i(x_0)$ for all $i = 1, \ldots, n$, while maintaining zero-knowledge, ensuring privacy and consistency without additional information leakage.

Final decision and summary of the proof system

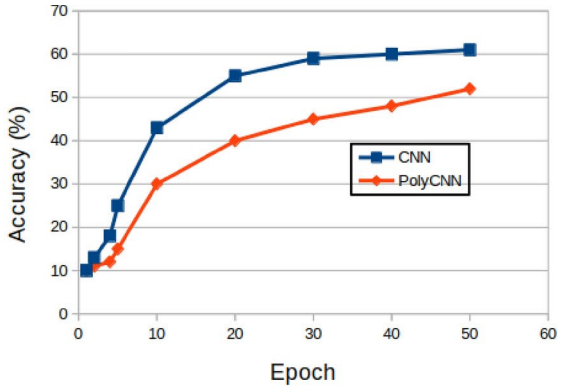
The proof system combines the Kate polynomial commitment scheme, which verifies the correct evaluation of univariate polynomials $F_i(x_i)$ at specific points, with the Sigma protocol, which ensures consistency in evaluations of univariate $F_i(x_i)$ align with the multivariate polynomial $P(x_0)$. The verifier's final decision is based on the bilinear pairing and consistency checks. If both are validated, the proof is accepted, confirming correct computations. If either check fails, the proof is rejected, indicating a discrepancy. As outlined in Table 1,

Property	Definition	Protocol's assurance
Soundness	If the statement is false, no dishonest prover can convince the honest verifier of its truth, except with negligible probability.	The Kate polynomial commitment ensures integrity by guaranteeing the correctness of $F_i(x_i)$. If $F_i(x_i)$ is incorrect, the prover cannot generate a valid commitment that matches the verifier's challenge.The Sigma protocol ensures consistency and soundness by verifying the equivalence between $F_i(x_0)$ and $P(x_0)$, catching any inconsistency with negligible probability.
Completeness	If the statement is true, an honest prover can convince an honest verifier of its truth with high probability.	The Kate polynomial commitment ensures that if the prover has correctly computed $F_i(x_i)$, the commitment will pass the verifier's checks. The Sigma protocol ensures that if the evaluations of $F_i(x_0)$ are consistent with $P(x_0)$, the prover can successfully pass the consistency proof, thereby ensuring completeness.
Zero-Knowledge	The verifier learns nothing beyond the validity of the statement being proven.	The Sigma protocol ensures that no information about $P(x_0)$ and $F_i(x_0)$ is revealed beyond their consistency. To ensure the integrity of the computation for $F_i(x_i)$, the Kate polynomial commitment binds the prover to the polynomial, and its structure enables efficient verification of evaluations at specific points without disclosing the polynomial itself.

Table 1. Soundness, completeness, and zero-knowledge properties of ProxyZKP.



(a) Training performance of LeNet5 and PolyMLP coupled by L_2 divergence on MNIST.



(b) Training performance of CNN2 and PolyCNN coupled by L_2 divergence on CIFAR10.

Fig. 3. Training performance of different private models and polynomial proxy models.

the soundness of the system is guaranteed by the binding property of Kate polynomial commitment and the robustness of the Sigma protocol.

Zero-knowledge proof of training integrity with polynomial proxy model

To demonstrate the effectiveness of the ProxZKP platform on decentralized models, benchmarks on scalability, proof generation and verification are obtained on a high-performance computing system consisting of 8 nodes. Each node is equipped with AMD EPYC 7742 64-core processors and 1 TB of system memory, providing sufficient resources to manage the intensive proof generation tasks. Efficient networking protocols like openMPI are utilized as the communication protocol, similar to the work by Kalra et al.[13]. For the evaluation, we employed two image classification datasets: MNIST and CIFAR. The MNIST dataset consists of 60,000 grayscale images of size 28×28, while the CIFAR dataset comprises 50,000 RGB images of size 32×32, both with 10,000 test images used for model evaluation.

Local dual model training performance

Each local client holds a subset of the training data with non-overlapping private datasets. In this configuration, every client is assigned 1,000 images from the MNIST dataset or 3,000 images from the CIFAR dataset. The experiment involved an 8-client decentralized network. We used the local dual models LeNet5[37]-PolyMLP and CNN2[38]-PolyCNN to train on the MNIST and CIFAR datasets, respectively. A decentralized learning environment was established using a peer-to-peer communication protocol based on OpenMPI. Experimental results from training local dual model according to Eqs. 1 and 2 are shown in Fig. 3a and b. The training process is carried out with the Adam optimizer at a learning rate of 0.001, and the deep mutual learning parameters, α and β, are set to 0.5.

Figure 3a and b depict the training accuracy of both the private and proxy models across several epochs for both datasets. Because of their optimised architecture, the private models, LeNet5[37] and CNN2[38], demonstrate high accuracy in capturing intricate patterns within the datasets. In contrast, the observed lower accuracy of the proxy models, PolyMLP and PolyCNN, can be attributed to two factors. The primary factor arises from the reduced expressiveness of 2nd polynomial models, which restricts their ability to model more complex, non-linear relationships in the data. The second contributing factor is the choice of a regression-based loss function, Mean Squared Error (MSE), that further misaligns with the classification task objectives and leads to slower convergence.

Parameter/metric	MNIST dataset	CIFAR dataset
Converged private model accuracy	80%	63%
Converged proxy model accuracy	73%	53%
Global communication rounds	200	200
Local training epochs per round	5	5
Batch size	250	250
Optimizer	Adam (LR: 0.001, Weight Decay: 1×10^{-4})	Adam (LR: 0.001, Weight Decay: 1×10^{-4})

Table 2. Training parameters and model accuracy.

Metric	MNIST dataset	CIFAR dataset
Average verification Time (ms)	80	95
Computational overhead (%)	400%	500%
Proof size (KB)	50	68
Proof generation time (ms)	600	780
Kate commitment overhead (%)	90%	90%
Sigma protocol overhead (%)	10%	10%

Table 3. ProxyZKP benchmarks across MNIST and CIFAR datasets.

Step	Training (PolyMLP/PolyCNN)	Cryptographic proof generation
Multivariate polynomial evaluation	Shared between training and proof generation	Shared between training and proof generation
Univariate polynomial Evaluation	Not required in training	Required for proof generation (extra workload)
Cryptographic operations	Not required in training	Required for integrity/consistency proof generation (elliptic curve operations, bilinear pairing)
Overall computational cost	Standard arithmetic operations	Cryptographic overhead

Table 4. Summary of analysis: training vs cryptographic proof generation.

Although proxy models show lower classification accuracy than their private model counterparts, this trade-off in machine learning performance is justified by the computation integrity and privacy gains they offer. Polynomial proxy models fulfill the system's security objectives by providing a verifiable mechanism for model updates and ensuring that the private models are isolated from external tempering. The quadratic mathematical structure as shown in Eq. 3 is crucial for maintaining compatibility with cryptographic proofs, such as Kate polynomial Commitment, even though this limits proxy models' ability to capture complex data relationships.

Table 2 shows the performance and convergence results for both datasets. The private model achieved a converged accuracy of about 80% on the MNIST dataset, while the proxy model achieved around 65%. The CIFAR dataset, which is more complex, led to converged accuracy of approximately 63% for the private model and 53% for the proxy model. These results highlight the ability of federated learning models to adjust to different data complexities while still achieving effective learning outcomes.

Proof generation performance studies

Experimental results from implementing the two-stage polynomial commitment scheme, as shown in Fig. 2, to verify the training integrity of the local node through the proxy model are presented in Table 3. We evaluated the effectiveness of this approach using the MNIST and CIFAR datasets. The key to this process is the decomposition of complex multivariate polynomials into univariate components, facilitated by SymPy[39], a Python library for symbolic mathematics. By isolating each variable and treating the others as constants, SymPy converts a single multivariate polynomial terms across 784 variables (PolyMLP) or 1024 variables (PolyCNN) into 784 or 1024 univariate polynomials, respectively. The decomposition simplifies the task of verifying the polynomial in Eq. 15 by its reducing complexity. This reduced complexity facilitates faster proof generation and verification, primarily through the use of the Kate polynomial commitment scheme. This scheme is implemented using open-source libraries such as Galois[40] and py_ecc[41], allowing for succinct proofs and computationally efficient verification on the resulting univariate polynomials.

The results in Table 3 indicate that the proof generation processes introduce significant overhead, but it remains manageable for both the MNIST and CIFAR datasets. The verification times are relatively low, with an average of 80 ms for PolyMLP and 95 ms for PolyCNN. The comparison between training and proof generation, as shown in Table 4, highlights the significant computational overhead introduced during cryptographic proof generation. While both processes share the step of multivariate polynomial evaluation, proof generation adds the burden of univariate polynomial evaluation, which is not required in training. Additionally, cryptographic operations, such as elliptic curve operations and bilinear pairings, further increase the workload, introducing

complexities absent from the training phase. The need for a cryptographic consistency proof to verify the integrity of the univariate evaluations adds to the computational cost. The Sigma protocol, implemented using open-source Python libraries such as PyCryptodome[42], ensures the consistency of polynomial evaluations. The computational overhead is divided between the Sigma protocol and the Kate polynomial commitment, as shown in Table 3. This overhead is primarily determined by the overall complexity of the proxy model and the dataset used.

Overall, the proof generation process is approximately 400-500% more computationally expensive than training due to these added tasks. However, the compact proof sizes-50 KB for PolyMLP on MNIST and 68 KB for PolyCNN on CIFAR-demonstrate a balance between the increased complexity and scalability.

Privacy-preserving mechanisms in decentralized model training: zero-knowledge proofs and differential privacy

Although ZKP can provide integrity verification for the gradient data without revealing any data, sharing gradient updates across the network to train a decentralized federated model is not without privacy challenges. Adversaries can exploit these updates through attacks like Gradient Inversion[19], where gradient information is used to reconstruct the private training data. To mitigate this risk, Differential Privacy (DP) techniques[20,21] have been proposed to reduce the ability of attackers to extract individual data points by adding calibrated noise to the transmitting data.

On the other hand, DP offers privacy protection by obscuring sensitive information, but it does not ensure the integrity of the underlying data generation process, such as gradient updates in the present case. Malicious clients can still submit manipulated gradients to undermine the integrity of the model training. By combining ZKP with DP in the ProxyZKP framework, an extra layer of data privacy protection can be deployed to minimize the risk of Gradient Inversion attacks.

Impact of noise addition on gradient updates

In the ProxyZKP framework, gradient updates are represented by multivariate polynomials, as shown in Eq. 15, where each gradient $\nabla_{\theta_k} L_{\phi_k}$ is computed locally based on private data and model parameters. To ensure computation integrity and data privacy, each local node first computes its local gradient and generates a ZKP to prove the correctness of this computation. Once the ZKP is verified, a calibrated Gaussian noise $\mathcal{N}(0, \sigma^2)$ vector, is added to the gradient vector to satisfy local DP requirements by extending Eq. 15 to:

$$\nabla_{\theta_k} \tilde{L}_{\phi_k} = \nabla_{\theta_k} L_{\phi_k} + \mathcal{N}(0, \sigma^2). \tag{31}$$

The Diffprivlib library, a Python based open-source differential privacy tool developed by Holohan et al.[43], is used to provide this calibrated noise. The amount of noise added to each element of the gradient vector is determined by the privacy parameters ϵ (privacy budget) and δ (failure probability), along with the sensitivity. In the present case, the sensitivity is controlled by the clipping factor set to unity[13]. After clipping the gradient's norm at 1.0, noise is added to each element of the gradient vector based on the given privacy parameters, to make sure that each gradient vector element is protected according to the local DP practice. The noisy gradient $\nabla_{\theta_k} \tilde{L}_{\phi_k}$ is then shared across the network to ensure that the gradients are both valid and private.

Table 5 presents the combined results of proxy models. It demonstrates how the accuracy of the private model and computational overhead vary under different privacy budgets and noise levels.

For the MLP model on MNIST dataset, accuracy decreases from 80% to 60% as the privacy budget becomes stricter, with ϵ reduced from 5.0 to 0.5 and the noise level σ increasing from 0.5 to 1.5. The stricter privacy budget introduces more noise during training, which hinders the model's ability to learn effectively from the data, thereby reducing its accuracy.

Similarly, for CNN model on CIFAR dataset, accuracy decreases from 62% to 52% under the same privacy budget reduction. However, the CNN model experiences a more pronounced drop in accuracy due to the complexity of the CIFAR dataset. The proxy model for CIFAR, PolyCNN, which needs to capture more spatial

Model/dataset	Privacy budget (ϵ, δ)	Noise Level (σ)	Accuracy (%)	Local DP overhead (ms)
MLP/MNIST	$(0.5, 10^{-5})$	1.5	60.0	15–20
MLP/MNIST	$(1.0, 10^{-5})$	1.0	70.2	15–20
MLP/MNIST	$(2.0, 10^{-5})$	0.8	75.3	15–20
MLP/MNIST	$(5.0, 10^{-5})$	0.5	80.1	15–20
CNN/CIFAR	$(0.5, 10^{-5})$	1.5	52.3	30–40
CNN/CIFAR	$(1.0, 10^{-5})$	1.0	54.0	30–40
CNN/CIFAR	$(2.0, 10^{-5})$	0.8	58.4	30–40
CNN/CIFAR	$(5.0, 10^{-5})$	0.5	62.0	30–40

Table 5. Local DP protection level, private model accuracy, and computational overhead.

features, is more sensitive to noisy gradients. The L_2 divergence coupling transfers incomplete information to private model which converges sub-optimally, resulting in a sharper accuracy drop.

Diffprivlib employs well-established algorithms such as the Box-Muller transform[44] and the Ziggurat algorithm[45] to efficiently generate Gaussian-distributed samples in constant time, independent of the variance of the distribution. As a result, the computational overhead incurred by Diffprivlib is determined primarily by the size of the gradient vector (i.e., the number of elements to which noise is added), rather than by the variance or magnitude of the noise itself. The local differential privacy (DP) overhead is minimal compared to the proof generation overhead, which remains constant at 600 ms for PolyMLP and 780 ms for PolyCNN. The local DP process accounts for about 10% of the computational workload of proof generation. Therefore, adding DP as an additional layer of privacy protection only slightly increases the overall workload, ensuring that the extra privacy measures have insignificant impact on the performance of local clients.

Scalability and performance

The scalability of decentralized learning systems is critical, particularly in scenarios where the number of participating nodes varies. In this study, we benchmarked the scaling performance of ProxyZKP by focusing on the relationship between the verification node's computational load and overall communication delay, as both factors significantly impact the system's efficiency. These key performance metrics are needed to be examined to help optimizing the system's performance.

As summarized in Table 6, the impact of network size on communication time and computational overhead for MLP on the MNIST dataset and CNN on the CIFAR dataset reveals important trends. The delay time increases predictably with the number of nodes, ranging from 6 to 13 ms as the number of nodes increases from 2 to 8 for both MLP and CNN models. This linear growth in communication time remains manageable and does not pose a major issue for system scalability. On the other hand, the local node overhead grows more gradually, by as much as 15% at most up to 8 nodes. The small increase in local node overhead indicates that local tasks, such as model training and cryptographic processes, are only minimally affected as the number of nodes increases

However, when scaling up to 8 nodes from a baseline of 2 nodes, the overhead of the verifier node increases to +100–120%. This significant growth suggests that the verifier node becomes a scalability bottleneck. Introducing a multi-verifier ZKP approach[46] could help reduce the load on a single verifier by distributing the verification tasks among multiple nodes and enables better scalability. This approach would enable ProxyZKP to maintain the advantages of decentralized learning without facing performance bottlenecks as the network grows.

Comparative analysis: proxy model zero-knowledge verification system

In the ProxyZKP, where each local client node has a private model coupled to a proxy multivariate polynomial model, selecting the right cryptographic proof approach is crucial for efficient communication of gradient updates in addition to integrity verification. In the two-stage proof scheme, multivariate polynomials are decomposed into multiple univariate polynomials, resulting in simpler cryptographic computations. Due to the simplicity of univariate polynomials, the multi-scalar multiplications[47] required in elliptic curve operations for commitments and proofs are smaller and less computationally intensive. ProxyZKP can independently generate proofs for each univariate polynomial, enabling concurrent proof generation and verification through multi-threaded processing. To provide more context on ProxyZKP's performance, we compare it with establish ZKP methods, including Bulletproofs[18] and zk-SNARKs[17]. Open-source Python libraries such as PyBulletproofs[48] and PySNARK[49] are used for benchmarking these ZKP schemes.

Table 7 presents the main findings regarding the performance of PolyMLP. By examining these aspects, it becomes clear why the two-stage proof scheme (univariate polynomial decomposition) may be the most suitable for this use case.

Parallelization

When PolyMLP is used under ProxyZKP, each node processes a multivariate polynomial with 784 variables. In the two-stage proof scheme, this multivariate polynomial is decomposed into 784 univariate quadratic polynomials, each representing a single variable, while the others are treated as constants. Since each univariate polynomial is independent, the two-stage proof scheme can distribute the computation across multiple threads, making it highly parallelizable. This parallelization greatly reduces the computational workload. The time required for

Number of nodes	Model/dataset	Delay time (ms)	Local node overhead	Verifier node overhead
2	MLP/MNIST	6–7	Baseline (0%)	Baseline (0%)
4	MLP/MNIST	8–9	+5–10%	+30–40%
6	MLP/MNIST	10–11	+10–12%	+60–70%
8	MLP/MNIST	12–13	+15%	+100–120%
2	CNN/CIFAR	6–7	Baseline (0%)	Baseline (0%)
4	CNN/CIFAR	8–9	+5–10%	+30–40%
6	CNN/CIFAR	10–11	+10–12%	+60–70%
8	CNN/CIFAR	12–13	+15%	+100–120%

Table 6. Network size impact on communication time and computational overhead for MLP and CNN on MNIST and CIFAR datasets.

Aspect	Univariate polynomial decomposition (with sigma protocol)	zk-SNARKs	Bulletproofs
Number of variables	784	784	784
Number of terms	784 univariate polynomials (3 terms each)	308,505 terms	308,505 terms
Exponentiations	3920 exponentiation	308,505 exponentiations	617,148 exponentiations
Sigma protocol for consistency check	Proves consistency between univariate and multivariate polynomials, efficient proof of agreement without full computation	Not needed	Not needed
Proof generation time	600 milliseconds (including sigma protocol overhead)	7 s	12 s
Proof size	50KB (including sigma protocol impact on proof size)	300 bytes (trusted setup excluded)	2700 bytes (logarithmic)
Scalability	Good scalability due to parallelization, sigma protocol ensures correctness	Limited scalability	Logarithmic scalability

Table 7. Comparison of univariate polynomial decomposition, zk-SNARKs, and bulletproofs for federated learning (with sigma protocol impact and multivariate polynomial).

proof generation in this approach benefits from this parallelization. On a 64-thread processor, proof generation for all 784 univariate polynomials takes approximately 600 milliseconds. This makes the univariate approach particularly efficient for decentralized federated learning, where real-time communication of updates is critical. In contrast, zk-SNARKs and Bulletproofs handle the entire multivariate polynomial directly, with 308,505 terms. While zk-SNARKs and Bulletproofs allow for some degree of parallelization during certain elliptic curve operations, they lack the modular, parallel structure of the univariate approach. As a result, zk-SNARKs and Bulletproofs take about 7 seconds and 12 seconds, respectively, to generate proofs on the same hardware, which makes them notably slower in situations that require rapid exchanges between nodes.

Cryptographic operations

Exponentiations in the proof generation process involve raising a group element, such as a generator g, to a power, typically a secret value or a polynomial coefficient, within a cryptographic group, often an elliptic curve or a finite field. In zero-knowledge proof systems like zk-SNARKs, Bulletproofs, or the two-stage proof scheme, exponentiations play a central role. They are used to commit to polynomial coefficients or evaluations, such as when computing g^a, where a represents a value or a polynomial evaluation. These commitments allow the prover to convince the verifier of the correctness of a statement without revealing the underlying data. Exponentiations are computationally demanding, but they are vital to the security of the proof, ensuring that commitments are binding and unforgeable, thus maintaining the overall integrity and privacy of the proof system.

In terms of exponentiations, Table 7 reveals distinct differences in the computational demands of each approach. For the two-stage proof scheme(univariate polyomial appaorach), the prover needs to commit to each univariate polynomial's evaluation, which requires 2304 exponentiations(3 for each of the 784 univariate polynomials). Additionally, to ensure consistency between the evaluations of the univariate polynomials and the full multivariate polynomial, Sigma protocol requires one more exponentiation is required for the commitment to the multivariate polynomial at the evaluation point. This brings the total number of exponentiations to 3920. The workload remains manageable and parallelizable due to the independent processing of each univariate polynomial.

In contrast, zk-SNARKs and Bulletproofs require 308,505 and 617,148 exponentiations, respectively, to handle the entire multivariate polynomial at once. This makes them significantly more computationally expensive compared to the univariate decomposition approach.

The combination of highly parallelizable computations and fewer cryptographic operations gives the univariate polynomial approach a significant computational workload advantage over zk-SNARKs and Bulletproofs. This makes the two-stage proof scheme particularly efficient for scenarios that require rapid proof generation and frequent exchanges between nodes, such as in decentralized federated learning environments.

Proof size

Another important factor for ProxyZKP is proof size, as it directly affects network bandwidth and storage requirements. The univariate polynomial decomposition approach generates proofs for each of the 784 univariate polynomials, resulting in a linear growth of proof size. For 784 univariate polynomials, each requiring 2 group elements (one for the commitment and one for the evaluation proof), the total proof size is 50 KB. This is significantly larger than the 300 bytes required by zk-SNARKs, which provide a constant proof size regardless of the complexity of the polynomial. However, zk-SNARKs come with a much higher computational cost, which makes them less attractive for systems prioritizing parallelization and speed.

Bulletproofs offer a middle ground with logarithmic proof size scaling. For a polynomial with 308,505 terms, the Bulletproof proof size is approximately 2700 bytes. This is significantly smaller than the proof size for the univariate polynomial decomposition approach but larger than zk-SNARKs. While Bulletproofs are more scalable in terms of proof size, they still do not match the univariate approach's speed in parallelized environments.

Scalability

Finally, scalability is a key consideration for ProxyZKP. The univariate polynomial decomposition approach scales well in terms of computational workload because each univariate polynomial can be handled independently. This makes it ideal for systems with a growing number of variables, as the workload can be distributed across multiple processor threads. However, the proof size grows linearly, which could become a limitation in systems with very large numbers of variables or polynomials.

zk-SNARKs, on the other hand, are highly scalable in terms of proof size, as they remain constant regardless of the number of variables or terms. However, zk-SNARKs do not scale as well in terms of computational workload, especially when dealing with larger polynomials or more complex models.

Bulletproofs provide strong scalability in both proof size and computational workload. The logarithmic growth in proof size makes them more efficient than the univariate approach for larger polynomials, while their computational workload can still be distributed across multiple threads, although not as efficiently as in the univariate approach.

Conclusion

The ProxyZKP framework combines Zero-Knowledge Proofs with polynomial proxy models to verify the integrity of model training at each local node in a decentralized federated machine-learning network. The two-stage proof scheme allows gradient updates be verified in highly parallelization manner, leading to fast proof generation and verification. While the proof size grows linearly with the number of variables, the computational efficiency of this approach makes it ideal in distributed settings. By comparison, zk-SNARKs offer small, constant proof sizes but are less practical due to their high computational cost and limited parallelization. Bulletproofs, with logarithmic proof size and better parallelization potential, offer a balanced solution for larger networks, but they still fall short of the univariate approach in terms of speed and parallelization. Overall, the two-stage proof scheme, integrated with the ProxyZKP framework, provides a highly efficient and secure solution for decentralized federated learning, maintaining both computational integrity and data privacy.

Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Received: 23 January 2024; Accepted: 12 November 2024
Published online: 20 November 2024

References

1. Pati, S. et al. Federated learning enables big data for rare cancer boundary detection. *Nat. Commun.* **13**, 7346 (2022).
2. Nguyen, T. V. et al. A novel decentralized federated learning approach to train on globally distributed, poor quality, and protected private medical data. *Sci. Rep.* **12**, 8888 (2022).
3. Camajori Tedeschini, B. et al. Decentralized federated learning for healthcare networks: A case study on tumor segmentation. *IEEE Access* **10**, 8693–8708 (2022).
4. Wu, C. et al. Communication-efficient federated learning via knowledge distillation. *Nat. Commun.* **13**, 2032 (2022).
5. Lam, M., Wei, G.-Y., Brooks, D., Reddi, V. J. & Mitzenmacher, M. Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix. In *International Conference on Machine Learning*. PMLR, pp. 5959–5968 (2021).
6. Papamanthou, C., Shi, E. & Tamassia, R. Signatures of correct computation. In *Theory of Cryptography TCC 2013. Lecture Notes in Computer Science* Vol. 7785 (ed. Sahai, A.) (Springer, 2013).
7. Alikhani, P. et al. Experimental relativistic zero-knowledge proofs. *Nature* **599**(7883), 47–50 (2021).
8. Turturica, G. V. & Iancu, V. Homomorphic inference of deep neural networks for zero-knowledge verification of nuclear warheads. *Sci. Rep.* **13**, 7464 (2023).
9. Lee, S., Ko, H., Kim, J. & Oh, H. vCNN: Verifiable Convolutional Neural Network Based on zk-SNARKs. In *IEEE Transactions on Dependable and Secure Computing* (2024).
10. Banerjee, A., Clear, M. & Tewari, H. Demystifying the Role of zk-SNARKs in Zcash, In *2020 IEEE Conference on Application, Information and Network Security (AINS)* (pp. 12-19). IEEE.
11. Parno, B., Howell, J., Gentry, C. & Raykova, M. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* **59**(2), 103–112 (2016).
12. Zhang, Y., Xiang, T., Hospedales, T. M. & Lu, H. Deep mutual learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4320–4328. (2018)
13. Kalra, S. et al. Decentralized federated learning through proxy model sharing. *Nat. Commun.* **14**(1), 2899 (2023).
14. Leng, Z., et al. Polyloss: A polynomial expansion perspective of classification loss functions. arXiv preprint arXiv:2204.12511 (2022).
15. Boneh, D., Drake, J., Fisch, B., and Gabizon, A. (2020). Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive.
16. Kate, A., Gregory M. Z. & Ian, G. Constant-size commitments to polynomials and their applications. Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5–9, 2010. Proceedings 16. Springer Berlin Heidelberg (2010).
17. Campanelli, Matteo, D. F., Anaïs, Q. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019).
18. Bünz, B., et al. Bulletproofs: Short proofs for confidential transactions and more. 2018 IEEE symposium on security and privacy (SP). IEEE (2018).
19. Ovi, P. R. & Gangopadhyay, A. A comprehensive study of gradient inversion attacks in federated learning and baseline defense strategies. In *2023 57th Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, USA, pp. 1–6 (2023).
20. Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., Zhang, L. (2016). Deep Learning with Differential Privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 308-318).
21. Truex, S., Liu, L., Mohan, S., Li, N. & Cao, L. Hybrid Federated Learning: Algorithms and Systems for Privacy-Preserving Machine Learning. In *Proceedings of the 2019 IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)* (pp. 164–172) (2019).

22. Jeon, B., et al. Privacy-preserving decentralized aggregation for federated learning. IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, (2021).
23. Zhao, J. et al. PVD-FL: A privacy-preserving and verifiable decentralized federated learning framework. *IEEE Trans. Inf. Forens. Secur.* **17**, 2059–2073 (2022).
24. Yim, J., Joo, D., Bae, J. & Kim, J. A gift from knowledge distillation: Fast optimization, network minimization, and transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2017).
25. Hinton, G. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015).
26. Vargas, V. M. et al. Soft labelling based on triangular distributions for ordinal classification. *Inf. Fusion* **93**, 258–267 (2023).
27. Hagay M., Tomer M. & Daniel, S., Alias free convnets: Fractional shift invariance via polynomial activations. In *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16333–16342 (2023).
28. Pham, H., et al. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR (2018).
29. Capra, M. et al. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access* **8**, 225134–225180 (2020).
30. Song, W. et al. Publicly verifiable computation of polynomials over outsourced data with multiple sources. *IEEE Trans. Inf. Forens. Secur.* **12**(10), 2334–2347 (2017).
31. Liao, S., Samiee, A., Deng, C., Bai, Y. & Yuan, B. Compressing deep neural networks using Toeplitz matrix: Algorithm design and FPGA implementation. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, pp. 1443–1447 (2019).
32. Bunz, B., Fisch, B. & Szepieniec, A. Transparent SNARKs from DARK compilers. in *IACR Cryptology ePrint Archive*, Report 2019/1229 (2019).
33. de Castro, L. & Chris, P. Functional commitments for all functions, with transparent setup and from SIS. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer Nature Switzerland*, (2023).
34. Enge, A. Elliptic curve cryptographic systems. Handbook of Finite Fields (2013): 784-796.
35. Zhang, J., et al. Polynomial commitment with a One-to-Many prover and applications. *31st USENIX Security Symposium (USENIX Security 22)* (2022).
36. Bootle, J., et al. Efficient zero-knowledge proof systems. Foundations of Security Analysis and Design VIII: FOSAD 2014/2015/2016 Tutorial Lectures **15**. pp 1–31 (2016).
37. Meir, Y., Ben-Noam, I., Tzach, Y., Hodassman, S. & Kanter, I. Learning on tree architectures outperforms a convolutional feedforward network. *Sci. Rep.* **13**(1), 962 (2023).
38. Kwak, N. S., Müller, K. R. & Lee, S. W. A convolutional neural network for steady state visual evoked potential classification under ambulatory environment. *PLoS One* **12**(2), e0172578 (2017).
39. Meurer, A. et al. SymPy: Symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017).
40. A performant NumPy extension for Galois fields. GitHub Repository, https://github.com/mhostetter/galois.
41. py_ecc: Elliptic curve crypto in python. GitHub Repository, https://github.com/ethereum/py_ecc.
42. Legrandin, S. PyCryptodome: Cryptographic library for python. GitHub Repository, https://github.com/Legrandin/pycryptodome.
43. Holohan, N., et al. Diffprivlib: The IBM differential privacy library. arXiv preprint arXiv:1907.02444 (2019).
44. Box, G. E. P. & Muller, M. E. A note on the generation of random normal deviates. *Ann. Math. Stat.* **29**(2), 610–611 (1958).
45. Marsaglia, George & Tsang, Wai Wan. The ziggurat method for generating random variables. *J. Stat. Softw.* **5**, 1–7 (2000).
46. Yang, K., Xiao, W. Non-interactive zero-knowledge proofs to multiple verifiers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Cham: Springer Nature Switzerland (2022).
47. Lu, T., et al. Cuzk: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on gpus. Cryptology ePrint Archive (2022).
48. Python3 implementation of bulletproofs, GitHub Repository, https://github.com/wborgeaud/ (2022).
49. Koninklijke Philips N.V., Xavier, G., Veeningen, M. PySNARK. GitHub Repository, https://github.com/meilof/pysnark

Acknowledgements

The work presented in this article is supported by Center for Advances in Reliability and Safety (CAiRS) admitted under AIR@InnoHK Research Cluster. Access to the computing facilities in the center is gratefully acknowledged.

Author contributions

TL conceived the project and conducted the experiment(s). All authors reviewed the manuscript.

Declaration

Competing interests

The corresponding author is responsible for submitting a competing interests statement on behalf of all authors of the paper. This statement must be included in the submitted article file.

Additional information

Correspondence and requests for materials should be addressed to T.L.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.