zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training

Haochen Sun, Tonghe Bai, Jason Li, Hongyang Zhang

***Abstract*—The recent advancements in deep learning have brought about significant changes in various aspects of people's lives. Meanwhile, these rapid developments have raised concerns about the legitimacy of the training process of deep neural networks. To protect the intellectual properties of AI developers, directly examining the training process by accessing the model parameters and training data is often prohibited for verifiers.**

In response to this challenge, we present zero-knowledge deep learning (zkDL), an efficient zero-knowledge proof for deep learning training. To address the long-standing challenge of verifiable computations of non-linearities in deep learning training, we introduce zkReLU, a specialized proof for the ReLU activation and its backpropagation. zkReLU turns the disadvantage of non-arithmetic relations into an advantage, leading to the creation of FAC4DNN, our specialized arithmetic circuit design for modelling neural networks. This design aggregates the proofs over different layers and training steps, without being constrained by their sequential order in the training process.

With our new CUDA implementation that achieves full compatibility with the tensor structures and the aggregated proof design, zkDL enables the generation of complete and sound proofs in less than a second per batch update for an 8-layer neural network with 10M parameters and a batch size of 64, while provably ensuring the privacy of data and model parameters. To our best knowledge, we are not aware of any existing work on zero-knowledge proof of deep learning training that is scalable to million-size networks.

I. Introduction

The rapid development of deep learning has garnered unprecedented attention over the past decade. However, with these advancements, concerns about the legitimacy of deep learning training have also arisen. In March 2023, Italy became the first Western country to ban ChatGPT amid an investigation into potential violations of the European Union's General Data Protection Regulation (GDPR). Furthermore, in January 2023, Stable Diffusion, a prominent image-generative model, faced accusations from a group of artist representatives over the infringement of copyrights on millions of images in its training data. As governments continue to impose new regulatory requirements on increasingly advanced AI technologies, there is an urgent need to develop a protocol that verifies the legitimacy of the training process for deep learning models. However, due to intellectual property and business secret concerns, model owners are typically hesitant to disclose their proprietary training data or model snapshots for legitimacy investigations.

Despite considerable efforts in verifiable machine learning to address this dilemma, many fundamental questions remain unanswered. Currently, cryptography-based approaches have mainly focused on inference-time verification [21], [8], [20], [30], [19], [29], [28], leaving training-time verification largely unexplored because of the significant computational demands and complex operations involved. Pioneering works in verifiable training [36], [6], [31] have mostly concentrated on basic machine learning algorithms, such as linear regression, logistic regression, and SVMs. The deep learning algorithms explored are limited by the size of the neural networks that the proof system backend can efficiently handle, making them inapplicable to modern deep neural networks.

Additionally, the advancement of verifiable deep learning is hindered by non-arithmetic operations, notably activation functions such as ReLU. These functions, although prevalent in deep learning, are not intrinsically supported by zero-knowledge proof (ZKP) systems. Early studies in verifiable deep learning training have explored square activation and polynomial approximation as arithmetic alternatives to traditional activation functions [36], [6], [1]. Yet, these alternatives diverge from standard deep learning architectures, raising questions about the effectiveness of such models. In light of the present state of verifiable deep learning training, confronting the challenges posed by activation functions such as ReLU is crucial for practical applications.

Furthermore, the representation of neural networks and their training process as arithmetic circuits that can accommodate activation function management is still unclear. Notably, beyond the innate layered structure, the training process encompasses both forward and backward propagations across numerous training steps, amplifying the complexity of the system to be modelled. Thus, a modelling approach for the training process over the neural network that aligns with not only the tensor structures but also the layered architecture and the numerous training steps is pivotal to the formulation of an efficient ZKP scheme for deep learning training.

In response to these challenges, we introduce zkDL, the first zero-knowledge proof for deep learning training. Through zkDL, model developers can provide assurances to regulatory bodies that the model has undergone proper training in accordance with the committed training data, consistent with the specified training logic, thereby addressing concerns about model legitimacy. Our principal contributions include:

- We present zkReLU, an efficient specialized zero-knowledge proof designed specifically for the exact computation of the Rectified Linear Unit (ReLU) and its backpropagation. This is achieved without the need for polynomial approximations to manage non-arithmetic operations. The foundational structure of zkReLU also facilitates our innovative arithmetic circuit design to depict the entire training procedure.

- We introduce FAC4DNN, a modelling scheme that represents the training process over deep neural networks as arithmetic circuits. FAC4DNN is acutely aware of the unique structures inherent in the entire training process, including both tensor-based and layer-based configurations, as well as the repeated execution of similar operations across multiple training steps. Astutely, FAC4DNN leverages the unavoidable alternations made in zkReLU, turning them into an advantage. This enables proofs for different layers and training steps to be aggregated, bypassing the traditional sequential processing as in the training process. As a result, there are both empirical and theoretical reductions in computational and communicational overheads when conducting the proof.
- In addition to pioneering zero-knowledge verifiability for real-world scale neural networks, we have implemented zkDL as the first zero-knowledge deep-learning framework using CUDA. Benefiting from the combined strengths of zkDL's design and implementation, we markedly advance toward practical zero-knowledge verifiable deep learning training for real-world industrial applications. Specifically, on an 8-layer network containing over 10 million parameters and a batch size of 64 using the CIFAR-10 dataset, we have confined the proof generation time to less than 1 second per batch update.

A. Overview of zkDL

In this section, we present an overview of zkDL, as depicted in Figure 1. zkDL, tailored as a Zero-Knowledge Proof (ZKP) for deep learning training, models both forward and backward propagations within neural networks (NNs) as arithmetic circuits (ACs). It adeptly manages non-arithmetic operations, which inherently resist direct proof. Furthermore, leveraging the inherent relatedness of tensor-based and layered structures across all training steps, zkDL effectively batches and compresses the proof. This strategic approach significantly diminishes the computational and communicational burdens for both the prover and verifier.

1) Threat model: We assume the presence of two probabilistic polynomial-time (PPT) parties: a prover $\mathcal{P}$ (e.g., an AI company) and a verifier $\mathcal{V}$ (e.g., a government agency). Both parties concur on a predetermined neural network structure and training logic, including aspects such as the number of training steps, optimization algorithms, and learning rates. The prover's role is to strictly adhere to the training logic when training the model and then to demonstrate this adherence to the verifier. The verifier, while being semi-honest in adhering to the prescribed protocol for validation, remains interested in the training data and model parameters privately held by the prover.

2) Haunt of non-arithmetic and zkReLU: As illustrated in Figure 1a, training NNs primarily comprises two components: tensor operations within each layer and the forward and backward propagation through activation functions, exemplified by ReLU, situated between the layers. In the realm of ZKP, the intra-layer operations such as matrix multiplications, convolutions, and their backward propagations correspond to arithmetic operations. These operations are inherently composed of additions, subtractions, and multiplications, aligning naturally with ZKP schemes. In contrast, ReLU, akin to many other activation functions, is inherently non-arithmetic. This distinction necessitates dedicated ZKP protocols for its forward and backward propagations, ensuring they integrate coherently with the proof mechanisms for the training's arithmetic-centric segments.

As depicted in Figure 1b, due to the nature of ReLU and its backward propagation not establishing input-output relationships based on any arithmetic operation, the arithmetic connections between successive NN layers are absent. This absence necessitates that the dangling inputs and outputs related to ReLU's forward and backward passes—specifically, the preactivation $\mathbf{Z}$, the activation $\mathbf{A}$, and their respective gradients $\mathbf{G_Z}$ and $\mathbf{G_A}$, each of the same dimension D—be bound by commitments. They cannot merely be considered as intermediate computational values over AC. Additionally, it is imperative to reinstate this connection to hinder any potential deceitful actions by the prover during the ReLU's forward and backward computation, all while ensuring that the arithmetic operations within the layers are computed correctly and pass the verification process.

To reestablish this connection, the terms $\mathbf{Z}, \mathbf{G_A}$ and $\mathbf{A}, \mathbf{G_Z}$ can no longer be viewed as the "input" and "output" for forward and backward propagations. We decompose these values into an auxiliary input, represented as $\mathbf{aux} = (\mathbf{Z}', \mathbf{S_Z})$. Here, $\mathbf{Z}'$ denotes the absolute value of $\mathbf{Z}$, and $\mathbf{S_Z}$ represents the sign of $\mathbf{Z}$. The design of zkReLU should ensure that these auxiliary inputs can effectively reconstruct the four tensors, which are separated due to non-linearity. Given a randomness r chosen by the verifier, Schwartz–Zippel lemma [24], [37] guarantees that the verification can be represented by:

$$\mathbf{Z} + \mathbf{Z}' + r\mathbf{A} + r^2\mathbf{G_Z} = \left((2+r)\mathbf{Z}' + r^2\mathbf{G_A}\right) \odot \mathbf{S_Z}, \quad (1)$$

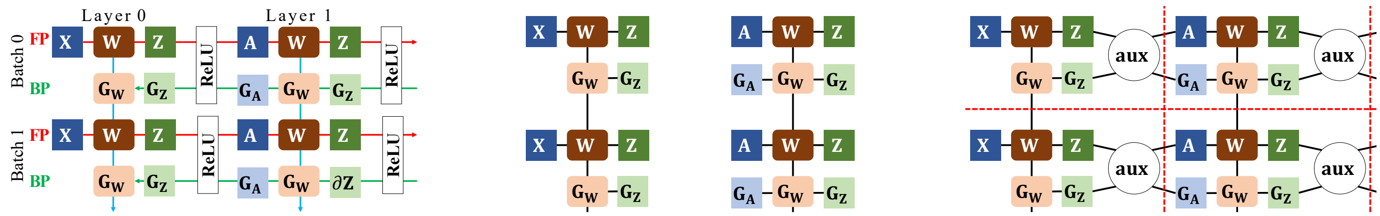
or equivalently,

$$(\mathbf{Z} - \mathbf{Z}' \odot (2\mathbf{S_Z} - 1)) + (\mathbf{A} - \mathbf{Z}' \odot \mathbf{S_Z})\, r + (\mathbf{G_Z} - \mathbf{G_A} \odot \mathbf{S_Z})\, r^2 = 0. \quad (2)$$

This ensures that the coefficient of each term is zero, and therefore the correctness of the ReLU operation.

However, while the incorporation of $\mathbf{aux}$ addresses the gap resulting from the ReLU non-linearity, the original logical orderings between the layers are not retained. Yet, this seemingly disadvantageous situation opens the door for a surprising optimization. Specifically, the tensor operations, including ReLU augmented with $\mathbf{aux}$, from various layers and training steps can be aggregated regardless of their logical orderings in training, which has already been disrupted by zkReLU.

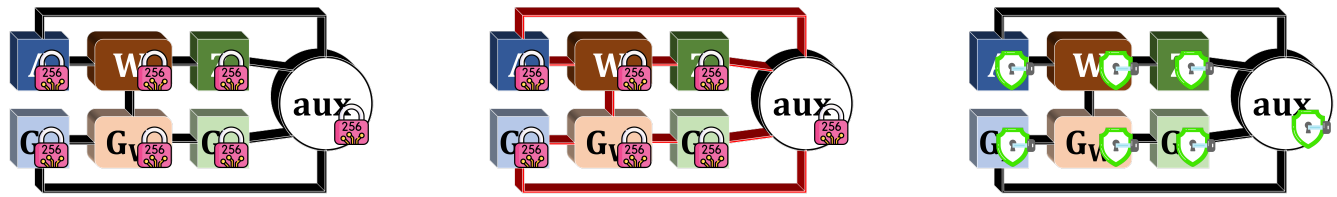
3) FAC4DNN: Aggregating and compressing proofs of deep learning training: In Figure 1c, a consistent pattern of tensors—including auxiliary inputs—and their corresponding arithmetic relations is evident, spanning multiple layers and training steps. During the training phases, the prover must adhere to the designated sequence of layers and training steps to finalize the computations intrinsic to the training process. However, as the disconnection of the arithmetic circuit and the commitment to the auxiliary inputs fundamentally rewire the



(a) Prover executes training: Forward Propagation (FP, red arrows) & Backward Propagation (BP, green arrows), and parameter updates (blue arrows).

(b) Activation functions disrupt arithmetic relations (black lines), compromising proof integrity.

(c) Prover uses auxiliary inputs (absolute values and signs) to reconnect tensors, restoring arithmetic relations. Note that the same pattern is repeated in each cell.



(d) Due to the repeated pattern, the prover stacks tensors across layers and steps, commits to them (represented by the lock), and batches the arithmetic relations accordingly.

(e) Generating batched proof on arithmetic relations (highlighted in red) among stacked tensors.

(f) Concluding with proof of evaluations (represented by unlocking) on stacked tensors.

Fig. 1: Overview of zkDL. One of our key contributions is the **compressed proof** across data batches, DL layers and training steps, by leveraging the repeated structures in deep learning training for acceleration.

arithmetic circuit, tensors in **different NN layers and training steps** reside in the **same AC layer** for the proof generation. Therefore, the prover can collectively group the proofs across these repetitive patterns.

As illustrated in Figure 1d, throughout the proof generation phase, the prover compiles similar tensors from all repetitive units and simultaneously amalgamates their arithmetic relationships. To delve deeper, consider N instances of matrix multiplication, a common tensor operation in deep learning, such as $\mathbf{Z}^{(n)} \leftarrow \mathbf{X}^{(n)}\mathbf{Y}^{(n)}$ $(0 \leq n \leq N-1)$. These instances span several layers and training steps that require verification. Each matrix is defined as $\mathbf{Z}^{(n)} \in \mathbb{F}^{D_1 \times D_3}, \mathbf{X}^{(n)} \in \mathbb{F}^{D_1 \times D_2}$, and $\mathbf{Y}^{(n)} \in \mathbb{F}^{D_2 \times D_3}$. Instead of verifying all N instances individually, the proof directly addresses the stacked matrices $\mathbf{X} \in \mathbb{F}^{N \times D_1 \times D_2}, \mathbf{Y} \in \mathbb{F}^{N \times D_2 \times D_3}, \mathbf{Z} \in \mathbb{F}^{N \times D_1 \times D_3}$, which are in the form of 3-order tensors. Using randomness designated by the verifier, namely $\mathbf{w} \sim \mathbb{F}^{\log_2 N}, \mathbf{u}_1 \sim \mathbb{F}^{\log_2 D_1}, \mathbf{u}_3 \sim \mathbb{F}^{\log_2 D_3}$, the aggregated proof is represented as:

$$\widetilde{\mathbf{Z}}\left(\mathbf{w}, \mathbf{u}_1, \mathbf{u}_3\right) = \sum_{\mathbf{i}=0}^{N} \widetilde{\beta}\left(\mathbf{w}, \mathbf{i}\right) \sum_{\mathbf{j}=0}^{D_2} \widetilde{\mathbf{X}}\left(\mathbf{i}, \mathbf{u}_1, \mathbf{j}\right) \widetilde{\mathbf{Y}}\left(\mathbf{i}, \mathbf{j}, \mathbf{u}_3\right), \quad (3)$$

where $\widetilde{\mathbf{Z}}\left(\cdot\right), \widetilde{\mathbf{X}}\left(\cdot\right), \widetilde{\mathbf{Y}}\left(\cdot\right), \widetilde{\beta}\left(\cdot\right)$ are multilinear extensions [22] of $\mathbf{Z}, \mathbf{X}, \mathbf{Y}$ (viewed as a function mapping from binary representations of indices to values of the corresponding dimensions) and $\beta : \{0,1\}^{\log_2 N} \times \{0,1\}^{\log_2 N} \to \{0,1\}$ where $\beta(\mathbf{b}_1, \mathbf{b}_2) = \mathbb{1}\{\mathbf{b}_1 = \mathbf{b}_2\}$.

Notably, executing the sumcheck (3) over $\mathbf{i}$, which pertains to the extra dimension from stacking, compresses the proof of (3) to:

$$\alpha = \widetilde{\beta}\left(\mathbf{w}, \mathbf{v}\right) \sum_{\mathbf{j}=0}^{D_2} \widetilde{\mathbf{X}}\left(\mathbf{v}, \mathbf{u}_1, \mathbf{j}\right) \widetilde{\mathbf{Y}}\left(\mathbf{v}, \mathbf{j}, \mathbf{u}_3\right). \quad (4)$$

In this context, the prover's claim α and randomness $\mathbf{v}$ arise from the compression's execution. Therefore, validating:

$$\alpha \widetilde{\beta}\left(\mathbf{w}, \mathbf{v}\right)^{-1} = \sum_{\mathbf{j}=0}^{D_2} \widetilde{\mathbf{X}}\left(\mathbf{v}, \mathbf{u}_1, \mathbf{j}\right) \widetilde{\mathbf{Y}}\left(\mathbf{v}, \mathbf{j}, \mathbf{u}_3\right) \quad (5)$$

revives the sumcheck for singular matrix multiplication.

This technique of aggregation and compression extends to all tensor operations that can be verified using the sumcheck protocol individually, regardless of the number of inputs. Specifically, all N instances of each input are condensed into one, contingent upon the randomness of $\mathbf{v}$. This ensures a seamless transition to the sumcheck for singular operations. As a result, the original logical ordering of the N operational instances, whether they belong to consecutive layers or training phases, becomes inconsequential. From an arithmetic circuit (AC) perspective, this method effectively "flattens" the circuit, reducing its depth by a magnitude of $O(N)$. However, this comes at the cost of expanding the width of each AC layer by the same magnitude.

Building on zkReLU, it is inescapable that tensor commitments occur at every layer and training step, exemplified by each repetitive unit in Figure 1c. Hence, there is no asymptotic overhead in the cumulative tensor size bound by the commitments, nor in the requisite commitment time. Conversely, assuming the adoption of sub-linear size commitments, such as Hyrax [27] in this study (whose commitment sizes grow as fast as the square root of the overall committed value size),

committing to a stack of N tensors could culminate in an asymptotic reduction in the overall commitment size by an order of $O\left(\sqrt{N}\right)$.

Similarly, due to the flattened design of the arithmetic circuit facilitated by FAC4DNN, there are significant reductions in both proof sizes and verification times. By aggregating N instances of the same operation, the proof size does not increase at a $\Theta(N)$ rate. Only a single copy of the proof for a singular tensor operation is needed, instead of N copies, and the compression steps introduce only an $O(\log N)$ overhead. Additionally, when summing up the components of zkDL, the proof verification depicted in Figure 1f also witnesses a reduction by an order of $O\left(\sqrt{N}\right)$. Within zkDL, the batched tensors seamlessly integrate into the parallel computing environment of deep learning training, further reducing the total proof time, especially since there is no need to strictly follow the original computation sequence throughout training.

II. Preliminaries

A. Notations

In this study, vectors and tensors are denoted using boldface notation, such as $\mathbf{v}$ and $\mathbf{S}$. To ensure compatibility with the cryptographic tools employed, we adopt a 0-indexing scheme for these vectors and tensors. Our indexing approach for multi-dimensional tensors aligns with the PyTorch convention, exemplified by $\mathbf{S}_{[i,j_0:j_1,:]}$. Additionally, for any positive integer N, we employ the shorthand notation $[N]$ to represent the set $\{0,1,\ldots,N-1\}$.

B. Sumcheck and GKR protocols

The sumcheck protocol [5], [22] serves as a fundamental component in modern proof systems, allowing for the verification of the correctness of the summation $\sum_{\mathbf{b}\in\{0,1\}^d} f(\mathbf{b})$ for a d-variate polynomial f. This protocol offers an efficient proving time of $O(2^d)$ and a compact proof size of $O(d)$.

Building upon the sumcheck protocol, the GKR protocol [14] provides an interactive proof for the correct computation of arithmetic circuits. It leverages the sumcheck protocol between the layers of the arithmetic circuit, as well as the Pedersen commitments to the private inputs.

The sumcheck and GKR protocols have found wide applications in verifying the proper execution of deep learning models, thanks to their compatibility with tensor structures. In particular, the tensor operations can often be expressed in the form of sumchecks, via the multilinear extensions of the tensors: for each tensor $\mathbf{S}\in\mathbb{F}^D$ that is discretized from real numbers (without loss of generality, assume D is a power of 2, or zero-padding may be applied), its multilinear extension $\widetilde{\mathbf{S}}(\cdot):\mathbb{F}^{\log_2 D}\to\mathbb{F}$ is a multivariate polynomial defined as

$$\widetilde{\mathbf{S}}(\mathbf{u})=\sum_{\mathbf{b}\in\{0,1\}^{\log_2 D}}\mathbf{S}(\mathbf{b})\widetilde{\beta}(\mathbf{u},\mathbf{b}), \tag{6}$$

where $\mathbf{b}$ represents the $\mathbf{b}$-th element of $\mathbf{S}$ (identifying the index by the binary string), and $\widetilde{\beta}(\cdot,\cdot):\mathbb{F}^{\log_2 D}\times\mathbb{F}^{\log_2 D}\to\mathbb{F}$ is a polynomial. When restricted to $\{0,1\}^{\log_2 D}\times\{0,1\}^{\log_2 D}$, $\widetilde{\beta}(\mathbf{b}_1,\mathbf{b}_2)=\begin{cases}1, & \text{if } \mathbf{b}_1=\mathbf{b}_2;\\ 0, & \text{if } \mathbf{b}_1\neq\mathbf{b}_2,\end{cases}$ for $\mathbf{b}_1,\mathbf{b}_2\in\{0,1\}^{\log_2 D}$.

In the context of multilinear extensions, we use the notation of indices and their binary representations interchangeably. Moreover, we use $\widetilde{S}(\mathbf{u}_1,\mathbf{u}_2,\ldots,\mathbf{u}_k)$ to denote the evaluation of $\widetilde{S}(\cdot)$ at the concatenation of multiple vectors $\mathbf{u}_1,\mathbf{u}_2,\ldots,\mathbf{u}_k$ whose dimensions sum up to $\log_2 D$.

Specialized adaptations of the sumcheck protocols cater to prevalent deep learning operations like matrix multiplication [26], [13] and convolution [21]. When the sumcheck protocol is applied to these operations, it yields the proclaimed evaluation $\widetilde{\mathbf{S}}(\mathbf{v})$ of the multilinear extension for each involved tensor $\mathbf{S}$, with $\mathbf{v}$ being selected due to the inherent randomness of the protocol. Subsequent verification of these claimed evaluations employs the proof of evaluations relative to the commitment of $\mathbf{S}$, as detailed in II-C. Additionally, zero-knowledge variants of the sumcheck protocol [4], [32], [33] have been developed with asymptotically negligible overhead, which leaks no information on the tensors involved once employed.

Historically, representing NNs as ACs has posed challenges due to the inclusion of non-arithmetic operations, making it ambiguous and inefficient. In this study, we tackle this problem by refining and enhancing the modelling approach. Rather than directly representing NNs as ACs with identical layer structures and invoking the GKR protocol, we adeptly aggregate the sumcheck protocols for tensor operations over different layers and training steps, grounded in our novel AC design of FAC4DNN. This leads to improved running times and more concise proof sizes.

C. Pedersen commitments

The Pedersen commitment is a zero-knowledge commitment scheme that relies on the hardness of the discrete log problem (DLP). Specifically, in a finite field $\mathbb{F}$ with prime order p, committing to d-dimensional vectors requires an order-p cyclic group $\mathbb{G}$ (e.g., an elliptic curve) and uniformly independently sampled values $\mathbf{g}=(g_0,g_1,\ldots,g_{d-1})^\top\sim\mathbb{G}^d$, and $h\sim\mathbb{G}$. This scheme allows any d-dimensional tensor $\mathbf{S}=(S_0,S_1,\ldots,S_{d-1})^\top\in\mathbb{F}^d$ to be committed as:

$$\texttt{com}_{\mathbf{S}}\leftarrow\texttt{Commit}(\mathbf{S};r)=h^r\mathbf{g}^{\mathbf{S}}=h^r\prod_{i=0}^{d-1}g_i^{S_i},$$

where $r\sim\mathbb{F}$ is uniformly sampled, ensuring zero-knowledgeness of the committed value $\mathbf{S}$. A complete and sound proof of evaluation can be conducted for $\widetilde{\mathbf{S}}(\mathbf{v})$ with respect to $\texttt{com}_{\mathbf{S}}$ for any randomness $\mathbf{v}$ chosen by the verifier. This can be utilized as a component of the proofs for operations on private tensors.

Additionally, the Pedersen commitment scheme exhibits homomorphic properties. Specifically, for two commitments, $\texttt{com}_{\mathbf{S}_1}=h^{r_1}\mathbf{g}^{\mathbf{S}_1}$ and $\texttt{com}_{\mathbf{S}_2}=h^{r_2}\mathbf{g}^{\mathbf{S}_2}$, corresponding to tensors $\mathbf{S}_1$ and $\mathbf{S}_2$, their multiplied result, $\texttt{com}_{\mathbf{S}_1}\cdot\texttt{com}_{\mathbf{S}_2}=h^{r_1+r_2}\mathbf{g}^{\mathbf{S}_1+\mathbf{S}_2}$, is a valid commitment to the sum, $\mathbf{S}_1+\mathbf{S}_2$.

Incorporating the Pedersen commitment leads to a $O(d)$ runtime, both for committing to a tensor and for conducting a proof of evaluation from the prover's side. In real-world applications, several variations of the Pedersen commitment are utilized to enhance verifier efficiency and curtail communication demands. For example, Hyrax [27] is a commitment

scheme that does not require a trusted setup, refining the commitment size, proof of evaluation size, and the time it takes for a verifier to evaluate the proof to $O(\sqrt{d}), O(\log d)$, and $O(\sqrt{d})$, respectively. These advancements are strategically integrated into the blueprint of zkDL, particularly FAC4DNN, aiming to improve the prover time, proof sizes, and verifier times in the realm of deep learning.

D. Security assumptions

We assume that the commitment scheme employed in our research offers λ-bit security. In line with this, the finite field $\mathbb{F}$, wherein all computations are discretized, is of size $\Omega(2^{2\lambda})$. Furthermore, we posit that all aspects of the training procedure, encompassing the number of training steps, the number of layers, tensor dimensions, and the complexity of operations between them, are all polynomial in λ.

III. zkReLU: Proof of forward and backward propagations through ReLU activation

The proper and tailored handling of non-linearities, especially ReLU, is essential to achieve efficient zero-knowledge verifiable training on deep neural networks. In this section, we introduce zkReLU, a zero-knowledge protocol designed specifically to verify the training of deep neural networks that incorporate ReLU non-linearity. Our scheme employs auxiliary inputs, allowing for the verification of both the forward and backward propagations involving ReLU. Furthermore, zkReLU integrates the ReLU function into the FAC4DNN framework, which is primarily concerned with the arithmetic operations between tensors. This integration ensures that the efficiencies brought about by FAC4DNN extend to the proof of the entire training process.

When the ReLU activation function is applied to the output of layer ℓ (with $1 \leq \ell \leq L-1$ and L representing the total number of layers), denoted as $\mathbf{Z}^{(\ell)}$, it pertains to a linear layer, either fully connected or convolutional. Given that multiplication operations play a role in computing $\mathbf{Z}^{(\ell)}$, $\mathbf{Z}^{(\ell)}$ undergoes scaling twice by the scaling factor, assumed to be a power of 2, specifically 2^R. Consequently, when dealing with quantized values, the ReLU operation must also reduce the input by a factor of 2^R. This mechanism can be articulated as the activation function $\mathbf{A}^{(\ell)} = \text{ReLU}\left(\left\lfloor \frac{\mathbf{Z}^{(\ell)}}{2^R} \right\rfloor\right) = \mathbb{1}\left\{\left\lfloor \frac{\mathbf{Z}^{(\ell)}}{2^R} \right\rfloor \geq 0\right\} \odot \left\lfloor \frac{\mathbf{Z}^{(\ell)}}{2^R} \right\rfloor$.

To simplify the notation, we introduce the rescaled $\mathbf{Z}^{(\ell)'} := \left\lfloor \frac{\mathbf{Z}^{(\ell)}}{2^R} \right\rfloor$. This representation allows for the expression of $\mathbf{Z}^{(\ell)}$ as $\mathbf{Z}^{(\ell)} = 2^R \mathbf{Z}^{(\ell)'} + \mathbf{R}_{\mathbf{Z}}^{(\ell)}$, where $\mathbf{R}_{\mathbf{Z}}^{(\ell)}$ denotes the remainder resulting from rounding. To adequately define the concept of "non-negative" within the finite field, it becomes necessary to restrict the scale of $\mathbf{Z}^{(\ell)'}$. We assume each element of $\mathbf{Z}^{(\ell)'}$ is an Q-bit signed integer, with $2^Q \ll |\mathbb{F}|$. Solely for analytical reasons, we decompose $\mathbf{Z}^{(\ell)'}$ into its magnitude bits and sign bits, such that $\mathbf{Z}^{(\ell)'} = \sum_{j=0}^{Q-2} 2^j \mathbf{B}_j^{(\ell)} - 2^{Q-1}\mathbf{B}_{Q-1}^{(\ell)}$, with each $\mathbf{B}_j^{(\ell)}$ for $0 \leq j \leq Q-1$ being binary. Furthermore, $\mathbf{B}_{Q-1}^{(\ell)}$ represents the negativity of each dimension in $\mathbf{Z}^{(\ell)}$ (assigning 1 for negative values and 0 otherwise). The arithmetic relations between these intermediate values and the input and outputs of the forward propagation, notably $\mathbf{A}^{(\ell)}$ and $\mathbf{Z}^{(\ell)}$, can be captured as:

$$\mathbf{A}^{(\ell)} = (\mathbf{1} - \mathbf{B}_{Q-1}^{(\ell)}) \odot \mathbf{Z}^{(\ell)'}, \tag{7}$$

$$\mathbf{Z}^{(\ell)} = 2^R \mathbf{Z}^{(\ell)'} + \mathbf{R}_{\mathbf{Z}}^{(\ell)}. \tag{8}$$

During the backpropagation phase, the gradient of $\mathbf{A}^{(\ell)}$, denoted as $\mathbf{G}_{\mathbf{A}}^{(\ell)}$, is typically scaled twice by 2^R owing to the multiplication operations involved. As such, it becomes necessary for the prover to rescale this gradient to $\mathbf{G}_{\mathbf{A}}^{(\ell)'} := \left\lfloor \frac{\mathbf{G}_{\mathbf{A}}^{(\ell)}}{2^R} \right\rfloor$, with the resulting remainder being $\mathbf{R}_{\mathbf{G}_{\mathbf{A}}}^{(\ell)}$. Subsequently, $\mathbf{G}_{\mathbf{A}}^{(\ell)'}$ is employed to compute the gradient of $\mathbf{Z}^{(\ell)}$, represented as $\mathbf{G}_{\mathbf{Z}}^{(\ell)}$, through the Hadamard product $\odot$ with $\mathbf{1} - \mathbf{B}_{Q-1}^{(\ell)}$. Analogous to the forward propagation, the correctness of the backward propagation can be outlined through the following arithmetic relations:

$$\mathbf{G}_{\mathbf{Z}}^{(\ell)} = (\mathbf{1} - \mathbf{B}_{Q-1}^{(\ell)}) \odot \mathbf{G}_{\mathbf{A}}^{(\ell)'}, \tag{9}$$

$$\mathbf{G}_{\mathbf{A}}^{(\ell)} = 2^R \mathbf{G}_{\mathbf{A}}^{(\ell)'} + \mathbf{R}_{\mathbf{G}_{\mathbf{A}}}^{(\ell)}. \tag{10}$$

It is also important to observe that for the intermediate variables involved in (7), (8), (9) and (10), the values of the tensors each tensor need to be constrained so as to prevent malicious manipulations by the prover. Namely, $\mathbf{Z}^{(\ell)'} \in \left[-2^{Q-1}, 2^{Q-1}\right)^{D^{(\ell)}}$, $\mathbf{B}_{Q-1}^{(\ell)} \in \{0,1\}^{D^{(\ell)}}$, $\mathbf{R}_{\mathbf{Z}}^{(\ell)} \in \left[-2^{R-1}, 2^{R-1}\right)^{D^{(\ell)}}$, $\mathbf{G}_{\mathbf{A}}^{(\ell)'} \in \left[-2^{Q-1}, 2^{Q-1}\right)^{D^{(\ell)}}$, and $\mathbf{R}_{\mathbf{G}_{\mathbf{A}}}^{(\ell)} \in \left[-2^{R-1}, 2^{R-1}\right)^{D^{(\ell)}}$, are bounded and share the same dimension $D^{(\ell)}$. However, in compatibility with our design of FAC4DNN, which is overviewed in Section I-A2 and detailed in IV, the proof is aggregated over multiple training steps and layers. Therefore, in the following discussion, we use the notations without the superscripts (i.e., $\mathbf{A}, \mathbf{Z}, \mathbf{G}_{\mathbf{Z}}, \mathbf{G}_{\mathbf{A}}$ for the input and outputs of both directions of ReLU, and $\mathbf{Z}', \mathbf{B}_{Q-1}, \mathbf{R}_{\mathbf{Z}}, \mathbf{G}'_{\mathbf{A}}, \mathbf{R}_{\mathbf{G}_{\mathbf{A}}}$ as the intermediate values introduced) to represent the stacked tensors over multiple layers, all of which share the same dimension denoted as D.

A. Formulation of zkDL

As an initial step, we presuppose that the proof's execution over other components yields the claimed evaluations of the multilinear extensions on $\mathbf{A}, \mathbf{Z}, \mathbf{G}_{\mathbf{Z}}, \mathbf{G}_{\mathbf{A}}$. This is achieved through the aggregated sumchecks on the other operations in which these tensors participate, as will be elaborated in Section IV. These are represented as $\widetilde{\mathbf{A}}(\mathbf{u}_{\mathbf{A}}), \widetilde{\mathbf{Z}}(\mathbf{u}_{\mathbf{Z}}), \widetilde{\mathbf{G}_{\mathbf{Z}}}(\mathbf{u}_{\mathbf{G}_{\mathbf{Z}}}), \widetilde{\mathbf{G}_{\mathbf{A}}}(\mathbf{u}_{\mathbf{G}_{\mathbf{A}}})$. Thus, by committing to the intermediate values $\mathbf{Z}', \mathbf{B}_{Q-1}, \mathbf{R}_{\mathbf{Z}}, \mathbf{G}'_{\mathbf{A}}, \mathbf{R}_{\mathbf{G}_{\mathbf{A}}}$ and executing the sumcheck protocol on the aggregated versions of (7), (8), (9), (10), the validity of these four equations can be verified with overwhelming probability.

Nevertheless, the validity criteria for the intermediate values also warrant consideration. To ensure complete compatibility with FAC4DNN, which functions over aggregated tensor structures, these intermediate values are collectively represented as a 3D binary tensor— the auxiliary input $\mathbf{aux} \in$

$\{0,1\}^{2\times D\times(Q+R)}$. Here, $\mathbf{aux}_{[0,:,:]}$ and $\mathbf{aux}_{[1,:,:]}$ stand for binary representations of the $(Q+R)$-bit integers in $\mathbf{Z}$ and $\mathbf{G_A}$ respectively, given by:

$$\mathbf{aux}_{[0,:,:]}\mathbf{s}_{Q+R} = \mathbf{Z}, \tag{11}$$

$$\mathbf{aux}_{[1,:,:]}\mathbf{s}_{Q+R} = \mathbf{G_A}. \tag{12}$$

Here, $\mathbf{s}_B = (1, 2, 2^2, \ldots, 2^{B-2}, -2^{B-1})^\top$ facilitates the recovery of a B-bit integer from its binary representations.

Using this configuration, the intermediate variables can be equated as:

$$\mathbf{aux}_{[0,:,R:Q+R]}\mathbf{s}_Q + \mathbf{aux}_{[0,:,R-1]} = \mathbf{Z}', \tag{13}$$

$$\mathbf{aux}_{[1,:,R:Q+R]}\mathbf{s}_Q + \mathbf{aux}_{[1,:,R-1]} = \mathbf{G}'_\mathbf{A}, \tag{14}$$

$$\mathbf{aux}_{[0,:,0:R]}\mathbf{s}_R = \mathbf{R}_\mathbf{Z}, \tag{15}$$

$$\mathbf{aux}_{[1,:,0:R]}\mathbf{s}_R = \mathbf{R}_{\mathbf{G_A}}, \tag{16}$$

$$\mathbf{aux}_{[0,:,Q+R-1]} = \mathbf{B}_{Q-1}. \tag{17}$$

Ensuring (8) and (10) is upheld, while (7) and (9) can be reframed as:

$$\mathbf{A} = (\mathbf{1} - \mathbf{aux}_{[0,:,Q+R-1]}) \odot \left(\mathbf{aux}_{[0,:,R:Q+R]}\mathbf{s}_Q + \mathbf{aux}_{[0,:,R-1]}\right), \tag{18}$$

$$\mathbf{G_Z} = (\mathbf{1} - \mathbf{aux}_{[0,:,Q+R-1]}) \odot \left(\mathbf{aux}_{[1,:,R:Q+R]}\mathbf{s}_Q + \mathbf{aux}_{[1,:,R-1]}\right). \tag{19}$$

Furthermore, to ensure that $\mathbf{aux}$ is genuinely binary, the auxiliary input validity proof (AIVP) must be conducted on

$$\mathbf{aux} \odot (\mathbf{aux} - \mathbf{1}) = \mathbf{0}. \tag{20}$$

Implementing the sumcheck protocol on (11), (12), (18), (19), and (20) is sufficient to ascertain the correctness of the computation for ReLU with respect to $\mathbf{Z}, \mathbf{G_A}, \mathbf{A}$, and $\mathbf{G_Z}$. Practically, zkReLU combines the sumcheck protocols for these equations to further diminish the running times and size of the proof, chiefly by confining the relatively costly proof of evaluation on $\mathbf{aux}$ to a single instance. The full details of zkReLU, including its optimized sumcheck protocol, are given in Appendix A.

IV. FAC4DNN: An alternative arithmetic circuit design for modelling neural networks

In Section III, we operate under the assumption that the sumcheck protocols are executed on the arithmetic components of each neural network layer using the same randomness. At first glance, this might suggest potential security vulnerabilities. In the neural network structure, the output from one layer serves as the input to the subsequent layer; hence, employing identical randomness across two such layers might seemingly compromise security. However, we introduce an alternative framework, FAC4DNN, which stands for a **f**lat **a**rithmetic **c**ircuit **for** a **d**eep **n**eural **n**etwork. The core insight of FAC4DNN is the inherent separation between consecutive layers brought about by non-arithmetic operations such as ReLUs and their backpropagation. Given that tensor data on either side of this separation is secured through commitments, FAC4DNN is not constrained to maintain the traditional sequence inherent to neural networks. Instead, it can aggregate proofs across various layers and training steps, leading to notable optimizations in both the runtime and the proof sizes.

Consider the process of a single training step on a neural network (NN) using ReLU activations, which entails both a forward and backward pass. As highlighted in Figure 2, a break in the arithmetic relations arises at each ReLU activation and its associated backward propagation. Absent the reintegration provided by zkReLU using auxiliary inputs $\mathbf{aux}^{(\ell)}$, the arithmetic operations corresponding to each NN layer ℓ—namely the layer's input $\mathbf{A}^{(\ell-1)}$ (with $\mathbf{A}^{(0)}$ defined as the input data $\mathbf{X}$), the model parameters $\mathbf{W}^{(\ell)}$, the pre-activation $\mathbf{Z}^{(\ell)}$, and their respective gradients $\mathbf{G}_\mathbf{A}^{(\ell-1)}$, $\mathbf{G}_\mathbf{W}^{(\ell)}$, and $\mathbf{G}_\mathbf{Z}^{(\ell)}$—can be validated separately from operations in other NN layers using the sumcheck protocol.

For each training step, we operate under the assumption that both $\mathbf{W}^{(\ell)}$ and $\mathbf{G}_\mathbf{W}^{(\ell)}$ are bound by commitments. Therefore, the declared multilinear extension values for these two tensors can be directly verified through the proof of evaluation. On the other hand, the declared multilinear extension values of the remaining four tensors can initiate the proof of the correctness of ReLU and its backpropagation via zkReLU, as described in Section III-A. Expanding our view to encompass all training steps, the proofs validating parameter updates across varying layers become independent, not only across different layers but also throughout distinct training steps. This independence is upheld by commitments that bind both old and new model parameters and their gradients. Similarly, the proofs verifying arithmetic operations and zkReLU maintain their independence across individual training steps.

The crucial insight regarding FAC4DNN is that the traditionally sequential order of the layers and training steps has been parallelized, leading to a reduction in the circuit depth by a factor of $O(N)$, with N representing the product of the neural network depth and the number of training steps. Notably, this departure from the original sequential order allows the prover to execute the sumcheck protocols for all N parallel components of the circuit using identical randomness, without any interference between components. This concept underpins the design of the aggregated proofs over FAC4DNN.

Aggregating the proof. With the non-arithmetic operations transformed into auxiliary inputs and re-established as arithmetic operations, the validity of the training process becomes synonymous with the correctness of all arithmetic tensor operations across all layers. Given that tensor operations of similar sizes commonly exist in different layers and are repeated across multiple steps, proofs of these operations can be batched across both layers and steps.

Consider N instances of a tensor operation that we want to aggregate, such that each instance (indexed $0 \le n \le N-1$) with K input tensors $\mathbf{X}_k^{(n)}$ for $0 \le k \le K-1$, and one output tensor $\mathbf{Y}^{(n)}$. For each of these $K+1$ types of tensors, the N instances of it are of the same dimensionality. By stacking the N tensors of each type together, we get $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_{K-1}$ and $\mathbf{Y}$, where using the notations of multilinear extensions, each $\widetilde{\mathbf{X}}_k(n, \cdot)$ and $\widetilde{\mathbf{Y}}(n, \cdot)$ can be equated with $\mathbf{X}_k^{(n)}$ and $\mathbf{Y}^{(n)}$, correspondingly. Consider any tensor operations that can

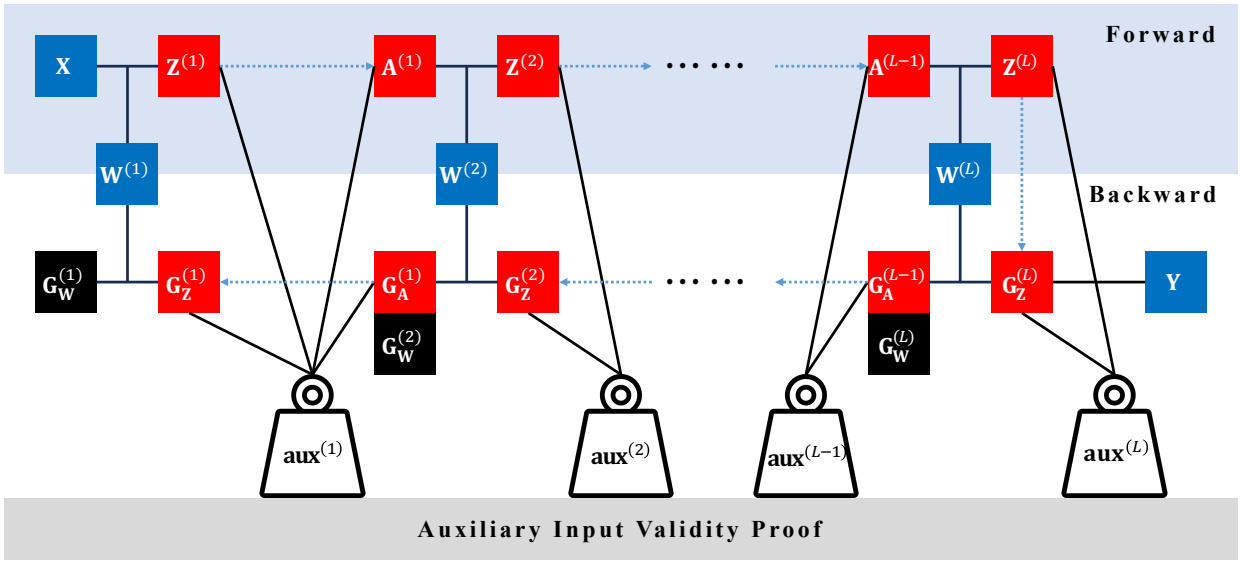


Fig. 2: The configuration of FAC4DNN for each training step: The entire circuit is anchored by the auxiliary inputs $\mathbf{aux}^{(\ell)}$ through arithmetic relations represented by the black lines, which replace the non-arithmetic operations depicted by the blue dash arrows. These non-arithmetic operations include the "comparison-with-0" operation in ReLU and its gradient. The gradients of the model parameters are highlighted within black bounding boxes. Both the data and model parameters are bound by the Pederson commitments and are delineated within blue bounding boxes. The tensors pertinent to zkReLU are bound by the commitment of $\mathbf{aux}$ and are enclosed within red bounding boxes.

be expressed as

$$\widetilde{\mathbf{Y}}(n,\mathbf{i}) = \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} f\left(\widetilde{\mathbf{X}_0}(n,\mathbf{i}_{I_0},\mathbf{j}_{J_0}), \widetilde{\mathbf{X}_1}(n,\mathbf{i}_{I_1},\mathbf{j}_{J_1}), \ldots, \widetilde{\mathbf{X}_{K-1}}(n,\mathbf{i}_{I_{K-1}},\mathbf{j}_{J_{K-1}})\right), \quad (21)$$

where $0 \le \mathbf{i} \le D_{\text{out}}-1, 0 \le \mathbf{j} \le D_{\text{in}}-1$ are indices expressed in binary format, f is a known multivariate polynomial, and $I_k \subseteq [\lceil \log_2 D_{\text{out}} \rceil]$ and $J_k \subseteq [\lceil \log_2 D_{\text{in}} \rceil]$ are subsets of the indices defined by the nature of the operation. For simplicity, we abbreviate each summand on the right-hand side of (21) as $f\left(\widetilde{\mathbf{X}_k}(n,\mathbf{i}_{I_k},\mathbf{j}_{J_k})\right)_{k=0}^{K-1}$. The sumcheck protocol for (21) can be formulated as

$$0 = \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \widetilde{\beta}(\mathbf{u},\mathbf{i}) \left(\widetilde{\mathbf{Y}}(\mathbf{n},\mathbf{i}) - \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} f\left(\widetilde{\mathbf{X}_k}(n,\mathbf{i}_{I_k},\mathbf{j}_{J_k})\right)_{k=0}^{K-1} \right) \quad (22)$$

$$= \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}(\mathbf{u},\mathbf{i}) \left(D_{\text{in}}^{-1} \widetilde{\mathbf{Y}}(n,\mathbf{i}) - f\left(\widetilde{\mathbf{X}_k}(n,\mathbf{i}_{I_k},\mathbf{j}_{J_k})\right)_{k=0}^{K-1} \right), \quad (23)$$

for $\mathbf{u} \sim \mathbb{F}^{\lceil \log_2 D_{\text{out}} \rceil}$ uniformly randomly chosen by the verifier. Writing n in the binary form $\mathbf{n}$, consider the weighted sum of (23) indexed by $\mathbf{n}$ from 0 to $N-1$,

$$0 = \sum_{\mathbf{n}=0}^{N-1} \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}(\mathbf{w},\mathbf{n}) \widetilde{\beta}(\mathbf{u},\mathbf{i}) \left(D_{\text{in}}^{-1} \widetilde{\mathbf{Y}}(\mathbf{n},\mathbf{i}) - f\left(\widetilde{\mathbf{X}_k}(\mathbf{n},\mathbf{i}_{I_k},\mathbf{j}_{J_k})\right)_{k=0}^{K-1} \right), \quad (24)$$

where $\mathbf{w} \sim \mathbb{F}^{\lceil \log_2 N \rceil}$ is the randomness chosen by the verifier, such that running the sumcheck protocol on (24) proves all N instances of (21) simultaneously.

Alternatively, if the preceding execution of the sumcheck protocol produces a claim on the value of $\widetilde{\mathbf{Y}}(\mathbf{w},\mathbf{u})$, which is not necessarily verified against the commitment of $\mathbf{Y}$ through the proof of evaluation, the sumcheck of (21) can be reformulated as:

$$\widetilde{\mathbf{Y}}(\mathbf{w},\mathbf{u}) = \sum_{\mathbf{n}=0}^{N-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}(\mathbf{w},\mathbf{n}) \widetilde{\beta}(\mathbf{u}_{J_\beta},\mathbf{j}_{J_\beta}) f\left(\widetilde{\mathbf{X}_k}(\mathbf{n},\mathbf{u}_{I_k},\mathbf{j}_{J_k})\right)_{k=0}^{K-1}, \quad (25)$$

where $J_\beta \subset [\lceil \log_2 D_{\text{in}} \rceil]$, similar to the I_k and J_k, is contingent upon the nature of the tensor operations. By executing the sumcheck protocol on (25), the uncorroborated claim on the stacked $\mathbf{Y}$ is translated to ones on $\mathbf{X}_k$s. These can be verified either by the proof of evaluation if $\mathbf{X}_k$ is committed, or by invoking (25) again if $\mathbf{X}_k$ is an intermediate value that has not been committed. A comprehensive explanation of the sumcheck protocol's execution on (24) and (25) is provided in Appendix B. A crucial observation is that the sequence in which the operations appear in the original training process is **completely irrelevant** in the aggregated proof of them.

Example IV.1. *In deep learning, matrix product* $\mathbf{Y}^{(n)} = \mathbf{X}_0^{(n)}\mathbf{X}_1^{(n)}$ *and Hadamard product* $\mathbf{Y}^{(n)} = \mathbf{X}_0^{(n)} \odot \mathbf{X}_1^{(n)}$ *are two frequently encountered tensor operations. Here,* $0 \le n \le N-1$ *indicates the indices of the operations, and we intend to batch the proofs of* N *operations for each type together.*

As a simplification from (23)*, these two operations can be proved by running the sumcheck protocol on the following*

equations:

$$\widetilde{\mathbf{Y}^{(n)}}\left(\mathbf{u}_0,\mathbf{u}_1\right)=\sum_{\mathbf{i}}\widetilde{\mathbf{X}_0^{(n)}}\left(\mathbf{u}_0,\mathbf{i}\right)\widetilde{\mathbf{X}_1^{(n)}}\left(\mathbf{i},\mathbf{u}_1\right), \tag{26}$$

$$\widetilde{\mathbf{Y}^{(n)}}\left(\mathbf{u}\right)=\sum_{\mathbf{i}}\widetilde{\beta}\left(\mathbf{u},\mathbf{i}\right)\widetilde{\mathbf{X}_0^{(n)}}\left(\mathbf{i}\right)\widetilde{\mathbf{X}_1^{(n)}}\left(\mathbf{i}\right), \tag{27}$$

where $\mathbf{u}_0,\mathbf{u}_1$ *and* $\mathbf{u}$ *are the random vectors compatible with the respective tensor dimensions in equations* (26) *and* (27)*.*

Following (25)*, the sumcheck for proof of the batched forms of the tensors can then be constructed as:*

$$\widetilde{\mathbf{Y}}\left(\mathbf{w},\mathbf{u}_0,\mathbf{u}_1\right)=\sum_{\mathbf{n}}\sum_{\mathbf{i}}\widetilde{\beta}\left(\mathbf{w},\mathbf{n}\right)\widetilde{\mathbf{X}_0}\left(\mathbf{n},\mathbf{u}_0,\mathbf{i}\right)\widetilde{\mathbf{X}_1}\left(\mathbf{n},\mathbf{i},\mathbf{u}_1\right), \tag{28}$$

$$\widetilde{\mathbf{Y}}\left(\mathbf{w},\mathbf{u}\right)=\sum_{\mathbf{n}}\sum_{\mathbf{i}}\widetilde{\beta}\left(\mathbf{w},\mathbf{n}\right)\widetilde{\beta}\left(\mathbf{u},\mathbf{i}\right)\widetilde{\mathbf{X}_0}\left(\mathbf{n},\mathbf{i}\right)\widetilde{\mathbf{X}_1}\left(\mathbf{n},\mathbf{i}\right), \tag{29}$$

where $\mathbf{w}\sim\mathbb{F}^{\lceil\log_2 N\rceil}$ *is a random vector, and can thus be proved directly without separation into* N *proof instances, as long as the batched tensors* $\mathbf{X}_0,\mathbf{X}_1,\mathbf{Y}$ *are bound by commitments.*

Re-indexing. In neural networks, repetitive tensor operations are common. Nevertheless, not every operation involves the same set of tensors. As a specific example, consider the gradient computation process. The gradient of the input to the first layer, which pertains to the data, remains uncomputed. Contrastingly, for subsequent layers, specifically for $\ell\geq 2$, gradients of the inputs — the activations from the preceding layer — are computed based on model parameters $\mathbf{W}^{(\ell)}$ along with other retained values of the respective layers.

Such an observation implies that, when batching proofs for these operations across the layers excluding the first, the multilinear-extension claim relates to the sequence $(\mathbf{W}^{(2)},\mathbf{W}^{(3)},\ldots,\mathbf{W}^{(L)})$. This differs from the complete sequence $(\mathbf{W}^{(1)},\mathbf{W}^{(2)},\mathbf{W}^{(3)},\ldots,\mathbf{W}^{(L)})$, which finds application during forward propagation encompassing all L layers.

Such a difference highlights the criticality of re-indexing tensors, particularly when batching proofs over various layers and operations in the neural network. Appropriate indexing is imperative for the precise alignment of tensor operations, which in turn ensures the integrity and efficiency of computations along with their verifiable assertions.

Considering a stacked tensor comprising N tensors of dimension D, represented as $\mathbf{X}=(\mathbf{X}^{(0)},\mathbf{X}^{(1)},\ldots,\mathbf{X}^{(N-1)})$, and another set of K tensors, $\mathbf{X}_k(0\leq k\leq K-1)$, that manifest as permutations of $\mathbf{X}$, one can represent $\mathbf{X}_k$ as

$$(\mathbf{X}^{(j_{k,0})},\mathbf{X}^{(j_{k,1})},\ldots,\mathbf{X}^{(j_{k,N_k-1})}),$$

where each $j_{k,i}$ belongs to the range $[N]$. Assuming the prover makes a claim on each $\mathbf{X}_k$, denoted as $\widetilde{\mathbf{X}_k}\left(\mathbf{u}_k,\mathbf{u}\right)$, with different randomness $\mathbf{u}_k\sim\mathbb{F}^{\log_2\lceil N_k\rceil}$ over the added dimension from stacking, and consistent randomness across other dimensions as $\mathbf{u}\sim\mathbb{F}^{\lceil\log_2 D\rceil}$. Given randomness coefficients $r_0,r_1,\ldots,r_{K-1}\sim\mathbb{F}$ as determined by the verifier, the relation

$$\begin{aligned}&\sum_{k=0}^{K-1}r_k\widetilde{\mathbf{X}_k}\left(\mathbf{u}_k,\mathbf{u}\right)\\&=\sum_{k=0}^{K-1}r_k\sum_{j=0}^{N_k-1}\widetilde{\beta}\left(\mathbf{u}_k,j\right)\widetilde{\mathbf{X}_k}\left(j,\mathbf{u}\right)\\&=\sum_{i=0}^{N-1}\left(\sum_{k=0}^{K-1}\sum_{j=0}^{N_k-1}r_k\widetilde{\beta}\left(\mathbf{u}_k,j\right)p_k(i,j)\right)\widetilde{\mathbf{X}}\left(i,\mathbf{u}\right)\end{aligned} \tag{30}$$

is valid, where $p_k(i,j)$ equals 1 if the i-th component of $\mathbf{X}$ matches the j-th element of $\mathbf{X}_k$, and 0 otherwise (both the prover and verifier are privy to p_k). Consequently, by representing i in binary, executing the sumcheck protocol on (30) verifies the reordering of indices of stacked tensors from $\mathbf{X}_k$s to $\mathbf{X}$, while achieving an $O(TD)$ proving time and $O(\log T)$ proof size.

Incorporating zkReLU. In Section III, we transform the correctness verification of ReLU into the validation of a tensor operation involving $\mathbf{Z},\mathbf{A},\mathbf{G_A},\mathbf{G_Z}$, and $\mathbf{aux}$. With the incorporation of $\mathbf{s}_{Q+R}$ and $\mathbf{s}_Q$, this operation can be verified using the sumcheck protocol. Consequently, zkReLU seamlessly integrates into the FAC4DNN framework, enabling aggregation across layers and training steps.

Another approach to positioning zkReLU within the FAC4DNN framework is to acknowledge that the stacked tensors $\mathbf{Z},\mathbf{A},\mathbf{G_A}$, and $\mathbf{G_Z}$ are anchored by the commitment of $\mathbf{aux}$, bypassing their individual commitments. However, to validate the non-arithmetic relations among these tensors, and upon establishing the claimed evaluations on the multilinear extension of these four tensors, the proof of evaluations on $\mathbf{aux}$ must be augmented with the zkReLU sumcheck.

V. Putting everything together

The proof produced by zkDL, employing the zkReLU protocol coupled with the compatible circuit design of FAC4DNN, delivers notable improvements in both computational and communicational efficiency for the prover and verifier, all while meeting security and privacy standards.

In this section, we assume that the training process spans T steps and the neural network includes L layers. Theoretically, maximum proof compression can be attained by amalgamating all the $O(TL)$ repetitive units throughout the entire training process. However, such a method would necessitate the retention of all T training checkpoints. Hence, we adopt a more encompassing premise: proofs from every sequence of T' training steps are collectively processed. We proceed with the assumption that T' divides T; if not, zero padding can be implemented. When $T'=T$, it signifies that proofs from all training steps are aggregated, whereas $T'=1$ indicates the verification of each step separately.

Moreover, in a typical neural network, layers of various types and sizes coexist, with each type linked to a distinct set of tensor operations. This diversity in operations poses challenges when aggregating proofs, even though these proofs

arise from distinct layers and have been rendered independent by FAC4DNN. As an illustration, the specialized sumcheck protocols for convolution and matrix multiplication are profoundly different, complicating their aggregation. In the same vein, aggregating proofs for one large and several smaller FCs is not efficient, as this results in an undue allocation of computational resources to padded zeros.

Therefore, in this discussion, we introduce N_A families of tensor operations, denoted as $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{N_A-1}$. Each family, $\mathcal{A}_i$, includes tensor operations of analogous nature and dimensionality. For instance, matrix multiplications found in several FC layers of similar sizes or zkReLUs present at the output of a group of layers with matching dimensions fall under this category. The term $|\mathcal{A}_i|$ represents the count of operations within $\mathcal{A}_i$ for every training step.

Similarly, we classify all tensors involved in each training step—including the training data and the auxiliary input in zkReLU—into N_T families: $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{N_T-1}$. Each family contains tensors of similar dimensionality and the same nature, such as the weights of several FC layers of corresponding sizes. The symbol $|\mathcal{T}_i|$ denotes the quantity of tensors within $\mathcal{T}_i$ during each training step.

The complete zkDL protocol involving both the prover and verifier is outlined as Protocol 1. The security and overhead analyses for this protocol are presented in Sections V-A and V-B, respectively.

Example V.1 (FCNN)**.** *We analyze a fully connected neural network (FCNN) consisting of L layers. We assume the use of ReLU activation at the output of each hidden layer and the application of the square loss. For every training step, using a data batch represented as $(\mathbf{X} = \mathbf{A}^{(0)}, \mathbf{Y})$, the correctness of the subsequent tensor operations is both necessary and sufficient to ensure the correctness of the training:*

- *Forward propagation through each FC layer:*

$$\mathbf{Z}^{(\ell)} = \mathbf{A}^{(\ell-1)}\mathbf{W}^{(\ell)}, \qquad 1 \le \ell \le L, \tag{31}$$

- *Backward propagation of the square loss function:*

$$\mathbf{G}_{\mathbf{Z}}^{(L)} = \mathbf{Z}^{(L)} - \mathbf{Y}, \tag{32}$$

- *The backward propagation through each FC layer:*

$$\mathbf{G}_{\mathbf{A}}^{(\ell)} = \mathbf{G}_{\mathbf{Z}}^{(\ell+1)}\mathbf{W}^{(\ell+1)^\top}, \qquad 1 \le \ell \le L-1, \tag{33}$$

$$\mathbf{G}_{\mathbf{W}}^{(\ell)} = \mathbf{G}_{\mathbf{Z}}^{(\ell)^\top}\mathbf{A}^{(\ell-1)}, \qquad 1 \le \ell \le L, \tag{34}$$

- *ReLU and its backward propagation, as per zkReLU.*
- *Parameter updates, based on the selected optimizer.*

Each type of tensor operation forms its own distinct family of tensor operations: all instances of (31) *constitute one family, all instances of* (33) *another, and likewise for* (34)*, zkReLU, and parameter updates. Similarly, families of tensors can be delineated based on tensors that share inherent characteristics. This applies to tensors signified by the same notation but differentiated by their superscripts, such as $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L)}$, which represent the parameters across all layers. With a consensus on the formation of families, the prover and verifier can employ Protocol 1 to generate proof for the entire training process, consolidating the proofs for different layers and training steps, thus optimizing computational and communicational expenses.*

A. Security analysis

Protocol 1 achieves perfect completeness and near-certain soundness, ensuring that the proof's acceptance by the verifier is equivalent to the prover's adherence to the Protocol 1 (the training process and the proof combined) almost surely.

Theorem V.2 (Completeness)**.** *For a neural network employing the ReLU activation function, where ReLU and its backpropagation stand as the sole non-arithmetic operations, and given that the unscaled $\mathbf{Z}^{(\ell)}$ and $\mathbf{G}_{\mathbf{A}}^{(\ell)}$ are $(Q+R)$-bit integers for the ReLU activation at every layer ℓ, the verifier, under the semi-honest assumption, accepts the proof with a probability of* 1 *provided the prover strictly adheres to Protocol 1.*

In Appendix C-A, the validity of Theorem V.2 is predicated on the perfect completeness achieved by the sumcheck protocols for zkReLU, the arithmetic operations, and the proofs of evaluations. This ensures that, for a prover fully adhering to Protocol 1, all checks are passed. However, establishing the soundness of Protocol 1 is more intricate. The soundness of zkReLU must capture the actual correctness of the ReLU non-arithmetic operation. Additionally, the soundness of FAC4DNN must encompass the correctness of all aggregated tensor operations.

Theorem V.3 (Soundness)**.** *For a neural network employing the ReLU activation function, where ReLU and its backpropagation stand as the sole non-arithmetic operations, if any tensor operation (including ReLU) is incorrectly computed by the prover, the verifier, operating under the semi-honest assumption, accepts the proof produced by Protocol 1 with a probability of $\texttt{negl}(\lambda)$.*

In addition to fulfilling the completeness and soundness requirements, the zkDL protocol also guarantees zero-knowledge, ensuring that it reveals no information about the training set and model parameters. This property is formalized in Appendix C-C.

B. Overhead analysis

Prover time. Compared to general-purpose ZKP backends, zkDL does not necessitate alterations to the original structure of neural networks to accommodate the internal framework of the ZKP backends. Instead, zkDL aligns seamlessly with the inherent tensor-based architectures of the computations intrinsic to deep learning training. This alignment enables the effective utilization of the pre-existing computational environment tailored for parallel tensor computations. Furthermore, the aggregation methodologies introduced by the design of FAC4DNN overcome the limitations imposed by the sequential arrangement of neural network layers and training phases. This innovation yields a more pronounced parallelism in proof generation than in the training procedure itself. These factors combined contribute to zkDL being the first viable work on verifiable training for large neural networks.

Protocol 1 zkDL

1: **for** $t \leftarrow 0, T', 2T', \ldots, T - T'$ **do**
2: Prover executes training steps $t, t+1, \ldots, t+T'-1$.
3: **for** $i \leftarrow 0, 1, \ldots N_T - 1$ **do** ▷ $\mathbf{S}_i^{(t)}$ is the stack of all tensors in $\mathcal{T}_i$ and in training steps t to $t+T'-1$.
4: Prover computes commitment $\texttt{com}_i^{(t)} \leftarrow \texttt{Commit}\left(\mathbf{S}_i^{(t)}\right)$, and sends $\texttt{com}_i^{(t)}$ to verifier.
5: **end for**
6: **for** $j \leftarrow 0, 1, \ldots, N_A - 1$ **do** ▷ $f_j^{(t)}$ is the aggregation of all operations in $\mathcal{A}_j$ and in training steps t to $t+T'-1$.
7: Prover and verifier execute the sumcheck protocol (24) for the aggregated tensor operation $f_j^{(t)}$.
8: Prover and verifier execute the sumcheck protocol (30), output $\widetilde{\mathbf{S}_i^{(t)}}\left(\mathbf{u}_i^{(t)}\right)$ for each $0 \le i \le N_T - 1$.
9: **end for**
10: **for** $i \leftarrow 0, 1, \ldots N_T - 1$ **do**
11: Prover and verifier executes the proof of evaluations for $\widetilde{\mathbf{S}_i^{(t)}}\left(\mathbf{u}_i^{(t)}\right)$ with respect to $\texttt{com}_i^{(t)}$.
12: **end for**
13: **end for**

Proof size. There are two primary components influencing the proof size of zkDL as delineated in Protocol 1: the commitment size in Line 4 and the sizes of the sumcheck-based proofs in Lines 7 and 8. For each tensor class, $\mathcal{T}_i$, we assume that the size of every tensor within $\mathcal{T}_i$ is $O(\sigma_i)$. Given the square-root growth of the commitment size in Hyrax, the commitment size employing the traditional sequential generation of the proof is $O\left(T\sum_{i=0}^{N_T-1}|\mathcal{T}_i|\sqrt{\sigma_i}\right)$. Conversely, the commitment size for the aggregated tensors is $O\left(\frac{T}{T'}\sum_{i=0}^{N_T-1}\sqrt{T'|\mathcal{T}_i|\,\sigma_i}\right)$, which simplifies to $O\left(\frac{T}{\sqrt{T'}}\sum_{i=0}^{N_T-1}\sqrt{|\mathcal{T}_i|\,\sigma_i}\right)$.

On the other hand, for the sumcheck protocols, proof sizes are generally logarithmic with respect to the complexity of the operation. We represent the complexity of a singular operation in each $\mathcal{A}_i$ as $O(\psi_i)$, rendering the proof size of one operation as $O(\log\psi_i)$. With the traditional sequential proof, the cumulative verification cost would, in a straightforward manner, accumulate to $O\left(T\sum_{i=0}^{N_A-1}|\mathcal{A}_i|\log\psi_i\right)$. However, when using FAC4DNN for proof aggregation, the multiplicative factor of the total aggregated proof instances is replaced by a nearly negligible additive term that is logarithmic in this number. Hence, the refined proof size is $O\left(\frac{T}{T'}\left(N_A\log T' + \sum_{i=0}^{N_A-1}(\log|\mathcal{A}_i| + \log\psi_i)\right)\right)$.

Verifier time. The analysis of the verifier's time closely mirrors the analysis of the proof size. With Hyrax [27], the verifier's time in the proof of evaluation scales as the square root of the committed tensor's size. Consequently, the verifier's time for each tensor class $\mathcal{T}_i$ is reduced by a factor of $O\left(\sqrt{T'|\mathcal{T}_i|}\right)$. On the other hand, given the logarithmic nature of the verifier's time in the sumcheck protocol, the verifier's time for each tensor class $\mathcal{A}_i$ experiences a reduction almost by a factor of $O\left(T'|\mathcal{A}_i|\right)$.

VI. Experiments

We developed zkDL in CUDA. Our **open-source implementation** of zkDL is available at https://github.com/jvhs0706/zkdl-train. To ensure seamless interfacing with tensor structures and their respective specialized aggregated proofs, our approach entailed creating customized CUDA kernels. These kernels cater to tensors, deep learning layers, and the novel cryptographic protocols presented in this research. Our framework builds upon `ec-gpu` [9], which is a CUDA implementation of the BLS12-381 curve, guaranteeing 128-bit security. The efficacy of zkDL was assessed using Example V.1. Given the challenges posed by quantization-induced rounding errors, we implemented a scaling factor of 2^{16}. This assured that every real-number computation within the system was encapsulated in the interval $[-2^{15}, 2^{15})$. Consequently, these numbers were aptly scaled as 32-bit integers. It is pivotal to highlight that throughout our experimental iterations, overflow incidents were conspicuously absent. Our experimental evaluations were orchestrated on a computing node nestled within a cluster, equipped with a Tesla A100 GPU.

Baselines. Earlier research on zero-knowledge verifiable inference did not offer ample techniques to incorporate verifiability into the training phase. In contrast, groundbreaking studies on zero-knowledge verifiable training, such as [36], [6], [31], predominantly concentrated on classical machine learning models, leaving real-world scale deep neural networks untouched. Nevertheless, for a comprehensive illustration of the benefits of aggregating proofs across multiple layers and training steps, we positioned zkDL against the traditional sequential proof generation found in the GKR protocol, which fully adheres to the reverse sequence of training steps, as well as the order of layers navigated throughout the training procedure.

Power of FAC4DNN. To showcase the efficiency of FAC4DNN, which liberates us from the mandate of strictly adhering to the sequential order of computations during proof generation, we embarked on a comparative analysis between zkDL and less compact proof variations. Specifically, we juxtaposed it against **1)** the proof that unwaveringly conforms to a reverse computation sequence, and **2)** a proof that only undergoes aggregation within individual training steps.

The experimental trials were executed on the CIFAR-10 dataset, utilizing an 8-hidden-layer perceptron, activated by ReLU, comprising 1,024 neurons in each layer. Notably, the

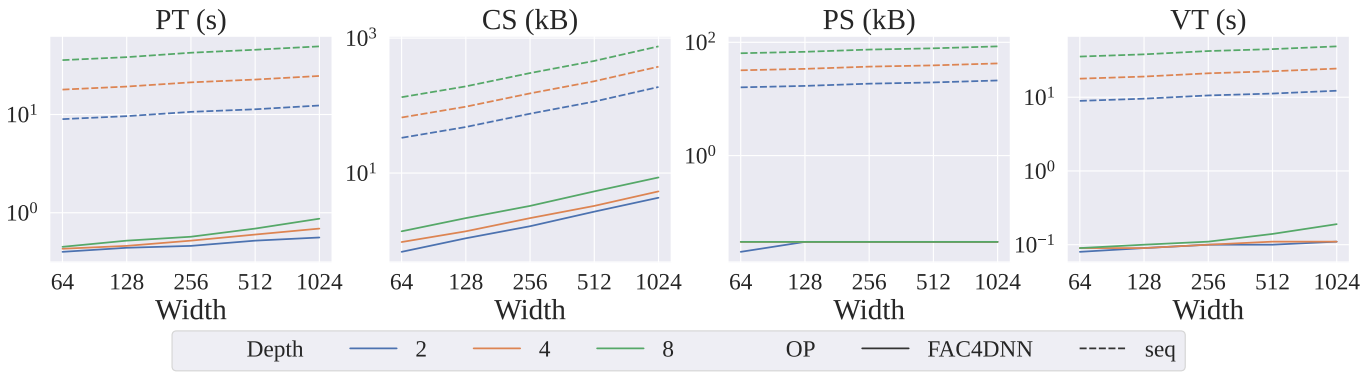


Fig. 3: zkDL's performance under different NN sizes: OP (order of proving), PT (proving time), CS (commitment size), PS (proof size), VT (verifying time).

TABLE I: Per-step computational and communicational costs of zkDL: T' (number of aggregated training steps; `seq` = no aggregation, 1 = within-step aggregation), PT (proving time), CS (commitment size), PS (proof size), VT (verifying time).

T'	BS	PT (s)	CS (kB)	PS (kB)	VT (s)
seq	16	45	510	78	45
	32	47	610	82	46
	64	49	750	85	46
1	16	6.0	230	11	6.0
	32	6.1	260	11	6.2
	64	6.2	270	12	6.3
4	16	1.8	110	3.1	1.8
	32	1.8	120	3.3	1.9
	64	1.9	120	3.4	1.9
16	16	0.83	56	0.81	0.80
	32	0.83	60	0.86	0.82
	64	0.84	61	0.88	0.84
64	16	0.84	28	0.31	0.53
	32	0.83	30	0.32	0.52
	64	0.85	30	0.34	0.56
256	16	0.84	14	0.085	0.30
	32	0.84	15	0.089	0.33
	64	0.84	15	0.091	0.33
1,024	16	0.85	8.2	0.030	0.18
	32	0.85	8.6	0.032	0.19
	64	0.86	8.7	0.033	0.19

input and output layers diverged in size, hosting 3,072 and 10 neurons, respectively. Therefore, the number of parameters in the neural network is as large as 10 millions. For the sake of comprehensiveness, our approach was to modify the aggregation span across different numbers of steps and subsequently record the findings, as tabulated in Table I.

It is pertinent to highlight that in a bid to provide a more lucid perspective on the superiority of zkDL, we have presented the computational and communication expenditures averaged per-step, as opposed to the cumulative costs associated with aggregated units of diverse step counts.

The data presented in Table I reveals a discernible trend: as the number of aggregated training steps increases, various performance metrics of zkDL generally improve. Notably, while FAC4DNN does not offer improvements in the theoretical complexity concerning proving time, the parallel processing capabilities of CUDA still manage to deliver a marked reduction in real-world proving durations. This improvement continues until it plateaus in less than 1 second. This limitation is due to CUDA's memory constraints, after which point data transfers become necessary within every aggregated proof. Furthermore, with sound theoretical underpinnings, as the number of aggregated steps increases, the sizes of the commitments and proofs, as well as the verifier times, decrease significantly, thanks to the design of FAC4DNN.

Further experimental results. In Figure 3, we evaluate the efficiency of zkDL when applied to neural networks of different dimensions. Keeping the number of aggregated steps constant at 1,024 and the batch size at 64, which is consistent with the largest experimental setting shown in Table I, we modify the neural network's width and depth. The outcomes are then contrasted with the fully sequential proof.

In alignment with the data in Table I, zkDL consistently outperforms the sequential proof in all facets. An unusual observation from both Table I and Figure 3 is that the verifying time is not considerably shorter than the proving time, a deviation from what is typically observed in most generic ZKP frameworks. This anomaly can be explained by the verification protocol's predominantly sequential nature, preventing it from fully harnessing the parallel computational capabilities of CUDA. Hence, the development of verification algorithms that are both theoretically and empirically expedient, particularly those optimized for parallel computing environments like CUDA, may forge the path for future advancements in specialized ZKPs for deep learning training.

VII. Related work

Verifiable machine learning inference. Zero-knowledge proof (ZKP) systems have emerged as important solutions to address security and privacy concerns in machine learning. These systems enable the verification of machine learning inference correctness without disclosing the underlying data or model. Notably, zkCNN [21] introduced an interactive proof protocol for convolutional layers, based on the GKR protocol [14] and its refinements [32], [34], [33]. This solution provides zero-knowledge verifiable inference for VGG-scale convolutional neural networks, expanding verifiable computations to modern deep learning. Meanwhile, zk-SNARK-based

inference, represented by ZEN [8], vCNN [20], pvCNN [30], and ZKML [19], Mystique [29], ezDPS [28], focuses on enhancing the compatibility of neural networks with the zk-SNARK backend [23], [15], [10], [3], [27], [2], scaling up non-interactive zero-knowledge inference. Once the committed model is verified to be correctly trained using this work, the verifiable inference can serve as a downstream application.

Verifiable machine learning training. VeriML [36] serves as an initial endeavour in zero-knowledge verifiable training for core machine learning algorithms. Building upon this, works on provable unlearning [6], [31] offer proofs of correct training execution and updates in machine unlearning, advancing past the probabilistic verifications in [11], [25], [16]. Their methodology, however, is restricted to instances where users provide data and fully govern data changes via explicit requests. This restricts proof generation to only data points involved in the aforementioned requests, leaving it ill-suited for our setting where the training data is also considered the intellectual property of the prover. Additionally, the models they support are constrained to limited-scale neural networks while bypassing certain non-arithmetic operations, like replacing ReLU with square activation. Another noteworthy contribution is zkPoT [12], which, based on MPC-in-the-head [17], offers zero-knowledge verifiable logistic regression.

Proof of learning (PoL) [18]. PoL serves as a non-cryptographic-based alternative to verifiable training. However, its probabilistic guarantees render it unsuitable for legitimacy-related settings like zkDL [7], [35]. Additionally, its threat model assumes adversaries to forge proofs by expending less computation resources than training, which does not deter dedicated malicious prover capable of deviating from the prescribed training logic (e.g., planting backdoors) at the cost of equivalent or additional computational power.

VIII. Conclusion

This paper introduces zkDL, the inaugural specialized zero-knowledge proof system tailored for deep learning training. By harnessing the unique computational structure of deep neural networks that enables a compressed proof across data batch, training step and neural network layers, zkDL substantially diminishes the time and communication overheads involved in verifying the genuine execution of deep learning training. Complemented by our pioneering CUDA-based implementation for verifiable deep learning, zkDL adeptly addresses the authenticity concerns related to trained neural networks with provable security guarantees. Experimentally, zkDL enables the generation of complete and sound proofs in less than a second per batch update for an 8-layer neural network with 10M parameters and a batch size of 64, while provably ensuring the privacy of data and model parameters. To our best knowledge, we are not aware of any existing work on zero-knowledge proof of deep learning training that is scalable to million-size networks.

References

[1] Ramy E. Ali, Jinhyun So, and Amir Salman Avestimehr. On polynomial approximations for privacy-preserving and verifiable relu networks. CoRR, abs/2011.05530, 2020.

[2] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. J. Cryptol., 35(3):15, 2022.

[3] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Recursive zk-SNARKs from any additive polynomial commitment scheme. IACR Cryptol. ePrint Arch., page 1536, 2020.

[4] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Electron. Colloquium Comput. Complex., TR17-057, 2017.

[5] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012, pages 90–112. ACM, 2012.

[6] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. Verifiable and provably secure machine unlearning. CoRR, abs/2210.09126, 2022.

[7] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. On the fundamental limits of formally (dis)proving robustness in proof-of-learning. CoRR, abs/2208.03567, 2022.

[8] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: efficient zero-knowledge proofs for neural networks. IACR Cryptol. ePrint Arch., page 87, 2021.

[9] Filecoin. ec-gpu. https://github.com/filecoin-project/ec-gpu, 2023. Accessed: 2023-10-13.

[10] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch., page 953, 2019.

[11] Xiangshan Gao, Xingjun Ma, Jingyi Wang, Youcheng Sun, Bo Li, Shouling Ji, Peng Cheng, and Jiming Chen. Verifi: Towards verifiable federated unlearning. CoRR, abs/2205.12709, 2022.

[12] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. IACR Cryptol. ePrint Arch., page 1345, 2023.

[13] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 4672–4681, 2017.

[14] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008, pages 113–122. ACM, 2008.

[15] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II, volume 9666 of Lecture Notes in Computer Science, pages 305–326. Springer, 2016.

[16] Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified data removal from machine learning models. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 3832–3842. PMLR, 2020.

[17] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007, pages 21–30. ACM, 2007.

[18] Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021, pages 1039–1056. IEEE, 2021.

[19] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless DNN inference with zero-knowledge proofs. CoRR, abs/2210.08674, 2022.

[20] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network. IACR Cryptol. ePrint Arch., page 584, 2020.

[21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In

Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, pages 2968–2985. ACM, 2021.

[22] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. J. ACM, 39(4):859–868, 1992.

[23] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, pages 238–252. IEEE Computer Society, 2013.

[24] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. J. ACM, 27(4):701–717, 1980.

[25] David Marco Sommer, Liwei Song, Sameer Wagh, and Prateek Mittal. Towards probabilistic verification of machine unlearning. CoRR, abs/2003.04247, 2020.

[26] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II, volume 8043 of Lecture Notes in Computer Science, pages 71–89. Springer, 2013.

[27] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA, pages 926–943. IEEE Computer Society, 2018.

[28] Haodi Wang and Thang Hoang. ezDPS: An efficient and zero-knowledge machine learning inference pipeline. Proc. Priv. Enhancing Technol., 2023(2):430–448, 2023.

[29] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021, pages 501–518. USENIX Association, 2021.

[30] Jia-Si Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. pvcnn: Privacy-preserving and verifiable convolutional neural network testing. IEEE Trans. Inf. Forensics Secur., 18:2218–2233, 2023.

[31] Jia-Si Weng, Shenglong Yao, Yuefeng Du, Junjie Huang, Jian Weng, and Cong Wang. Proof of unlearning: Definitions and instantiation. CoRR, abs/2210.11334, 2022.

[32] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III, volume 11694 of Lecture Notes in Computer Science, pages 733–764. Springer, 2019.

[33] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV, volume 13510 of Lecture Notes in Computer Science, pages 299–328. Springer, 2022.

[34] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, pages 159–177. ACM, 2021.

[35] Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. Adversarial examples for proof-of-learning. In 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022, pages 1408–1422. IEEE, 2022.

[36] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. IEEE Trans. Parallel Distributed Syst., 32(10):2524–2540, 2021.

[37] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings, volume 72 of Lecture Notes in Computer Science, pages 216–226. Springer, 1979.

APPENDIX A
ZKRELU

In this appendix, we delve into the specifics of zkReLU. The foundational requirement of zkReLU, stemming from the sumcheck-protocol-based tensor operations discussed in Section IV, encompasses the declared evaluations on both the input and output of ReLU's forward and backward propagations. Specifically, these are $\widetilde{\mathbf{A}}(\mathbf{u_A}), \widetilde{\mathbf{Z}}(\mathbf{u_Z}), \widetilde{\mathbf{G_Z}}(\mathbf{u_{G_Z}}), \widetilde{\mathbf{G_A}}(\mathbf{u_{G_A}})$.

The sumcheck protocol for the forward propagation, denoted as (11) and (18), is given by

$$\widetilde{\mathbf{Z}}(\mathbf{u_Z}) = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u_Z}, \mathbf{i}) \widetilde{\mathbf{aux}}(0, \mathbf{i}, \mathbf{j}) \widetilde{\mathbf{s}_{Q+R}}(\mathbf{j}), \quad (35)$$

$$\widetilde{\mathbf{A}}(\mathbf{u_A}) = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u_A}, \mathbf{i}) (1 - \widetilde{\mathbf{aux}}(0, \mathbf{i}, Q+R-1)) \widetilde{\mathbf{aux}}(0, \mathbf{i}, \mathbf{j}) \widetilde{\mathbf{s}'}(\mathbf{j}), \quad (36)$$

where

$$\mathbf{s}' := \left(0, 0, 0, \ldots, 1, 1, 2, 4, \ldots 2^{Q-2}, -2^{Q-1}\right)^\top.$$

Additionally, the AIVP is expressed as

$$0 = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u}_{\text{bin}}, \mathbf{i} \oplus \mathbf{j}) \left(\widetilde{\mathbf{aux}}(0, \mathbf{i}, \mathbf{j})^2 - \widetilde{\mathbf{aux}}(0, \mathbf{i}, \mathbf{j})\right), \quad (37)$$

with $\mathbf{u}_{\text{bin}} \sim \mathbb{F}^{\lceil \log_2 D \rceil + \lceil \log_2 (Q+R) \rceil}$ being uniformly randomly drawn by the prover.

Analogously, mirroring the construction above for backpropagation:

$$\widetilde{\mathbf{G_A}}(\mathbf{u_{G_A}}) = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u_{G_A}}, \mathbf{i}) \widetilde{\mathbf{aux}}(1, \mathbf{i}, \mathbf{j}) \widetilde{\mathbf{s}_{Q+R}}(\mathbf{j}), \quad (38)$$

$$\widetilde{\mathbf{G_Z}}(\mathbf{u_{G_Z}}) = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u_{G_Z}}, \mathbf{i}) (1 - \widetilde{\mathbf{aux}}(0, \mathbf{i}, Q+R-1)) \widetilde{\mathbf{aux}}(1, \mathbf{i}, \mathbf{j}) \widetilde{\mathbf{s}'}(\mathbf{j}), \quad (39)$$

$$0 = \sum_{\mathbf{i}=0}^{D-1} \sum_{\mathbf{j}=0}^{Q+R-1} \widetilde{\beta}(\mathbf{u}_{\text{bin}}, \mathbf{i} \oplus \mathbf{j}) \left(\widetilde{\mathbf{aux}}(1, \mathbf{i}, \mathbf{j})^2 - \widetilde{\mathbf{aux}}(1, \mathbf{i}, \mathbf{j})\right). \quad (40)$$

To optimize the verification process, the verifier introduces randomness $r, r' \sim \mathbb{F}$ to compress the six sumcheck equations, from (35) to (40). Each equation is multiplied by weights $r^2, r, 1, r'r^2, r'r$, and r' respectively, and subsequently aggregated. This compression technique leads to a succinct proof representation requiring only $3\log_2(D(Q+R)) + O(1)$ field elements.

This procedure creates three intermediate claims about $\mathbf{aux}$: $\widetilde{\mathbf{aux}}(0, \mathbf{v}, \mathbf{w})$, $\widetilde{\mathbf{aux}}(1, \mathbf{v}, \mathbf{w})$, and $\widetilde{\mathbf{aux}}(0, \mathbf{v}, Q+R-1)$ for $\mathbf{v} \sim \mathbb{F}^{\log_2 D}, \mathbf{w}^{\log_2 (Q+R)}$ chosen due to the randomness during the execution of the sumcheck. However, they can be merged into a singular claim, further reducing the proof size to $2\log_2(Q+R) + O(1)$. Consequently, the entire zkReLU proof compression is succinctly captured as $3\log_2 D + 5\log_2(Q+R) + O(1)$ field elements, with one additional proof of evaluation on $\mathbf{aux}$.

APPENDIX B
FAC4DNN

The aggregated proof for tensor operations, specifically the sumcheck on Equation (24), follows Protocol 2. In the same vein, the aggregation of specialized optimized sumcheck protocols, as given by Equation (25), adheres to Protocol 3. In both instances, compression begins over the additional axis resulting from stacking. This process methodically whittles down the proof over the stacked tensor to a singular tensor as shown in (43) and (46). Subsequently, the sumcheck protocol designed for individual tensor operations is directly implemented.

APPENDIX C
SECURITY ANALYSIS

A. Completeness

Proof sketch of Theorem V.2. Note that there are three scenarios in which a semi-honest verifier might reject a proof: the sumchecks in Lines 7, 8, and the proof of evaluations in Line 11.

For the sumcheck concerning aggregated tensor operations in Line 7, as specified in Protocol 2 or 3, regardless of which is invoked, the polynomials f_t, extended by the verifier to g_t, ensure that the sum $g_t(0)+g_t(1)$ comprehensively retrieves the sum from the preceding round. As such, given that the prover adheres to the protocol, it consistently stands that $g_t(0)+g_t(1)$ matches the preceding claim. This ensures that f_t is accepted by the verifier in round t with probability 1. Lastly, owing to the perfect completeness of the sumchecks for singular tensor operations, the verifier will also unconditionally accept the proof presented by an honest prover. Once an honest prover has successfully navigated the prior n rounds and reduced the N operations to one, the probability that the sumcheck protocol for this isolated operation also fails is zero.

Moreover, Equation (30) presents as the inner product of two vectors with length $N-1$, specifically the public input $\left(\sum_{k=0}^{K-1}\sum_{j=0}^{N_k-1} r_k \widetilde{\beta}\left(\mathbf{u}_k, j\right) p_k(i,j)\right)_{i=0}^{N-1}$ and the private input $\left(\widetilde{\mathbf{X}}\left(i, \mathbf{u}\right)\right)_{i=0}^{N-1}$. As a result, the completeness in this phase arises from the perfect completeness of the sumcheck for vector inner-products.

Finally, due to the perfect completeness of the proof of evaluations, and given the declared evaluations — that is, $\widetilde{\mathbf{S}_i^{(t)}}\left(\mathbf{u}_i^{(t)}\right)$ — are correct, the semi-honest verifier accepts the proof of evaluations on all stacked tensors with certainty in Line 11. □

B. Soundness

The proof of soundness (Theorem V.3) relies on Lemma C.1 that captures the soundness of zkReLU:

Lemma C.1 (Soundness of zkReLU)**.** *If the equalities* (11)*,* (12)*,* (18)*,* (19) *and* (20) *hold, then the forward and backward propagation of the ReLU activation is correctly computed as:*

$$\mathbf{A} = \left\lfloor \frac{\mathbb{1}\left\{\mathbf{Z} \geq 0\right\} \odot \mathbf{Z}}{2^R} \right\rfloor, \tag{47}$$

$$\mathbf{G_Z} = \left\lfloor \frac{\mathbb{1}\left\{\mathbf{Z} \geq 0\right\} \odot \mathbf{G_A}}{2^R} \right\rfloor. \tag{48}$$

Proof of Lemma C.1. Consider first the correctness of (20). This ensures that $\mathbf{aux} \in \{0,1\}^{2\times D\times(Q+R)}$. Following from (11), we derive

$$\sum_{j=0}^{Q+R-2} 2^j \mathbf{aux}_{[0,:,j]} - 2^{Q+R-1}\mathbf{aux}_{[0,:,Q+R-1]} = \mathbf{Z}, \tag{49}$$

where the first term on the left-hand side of (49) has a bound between 0 and $2^{Q+R-1}-1$. Consequently, the second term defines the sign of $\mathbf{Z}$, giving $\mathbb{1}\left\{\mathbf{Z}\geq 0\right\} = 1 - \mathbf{aux}_{[0,Q+R-1,:]}$. Applying a similar logic to (12), we deduce that $\mathbb{1}\left\{\mathbf{G_A}\geq 0\right\} = 1 - \mathbf{aux}_{[1,Q+R-1,:]}$.

For the rescaling of $\mathbf{Z}$, we can represent it as:

$$\left\lfloor \frac{\mathbf{Z}}{2^R} \right\rfloor = \left\lfloor \frac{\sum_{j=0}^{R-1} 2^j \mathbf{aux}_{[0,:,j]}}{2^R} \right\rfloor + \sum_{j=0}^{Q-1} 2^j \mathbf{aux}_{[0,:,R+j]} - 2^Q \mathbf{aux}_{[0,:,Q+R-1]}. \tag{50}$$

For each dimension denoted by i, the first term on the right-hand side is 1 precisely when $\mathbf{aux}_{[0,i,R-1]} = 1$. Thus, it is valid that $\left\lfloor \frac{\mathbf{Z}}{2^R} \right\rceil = \mathbf{aux}_{[0,:,R:Q+R]}\mathbf{s}_Q + \mathbf{aux}_{[0,:,R-1]}$. From (18), we derive:

$$\mathbf{A} = \mathbb{1}\left\{\mathbf{Z}\geq 0\right\} \odot \left\lfloor \frac{\mathbf{Z}}{2^R} \right\rfloor = \left\lfloor \frac{\mathbb{1}\left\{\mathbf{Z}\geq 0\right\}\odot \mathbf{Z}}{2^R} \right\rfloor, \tag{51}$$

which leads to (47). Using the same reasoning on (19) confirms the validity of (48). □

Lemma C.1 asserts that, through the integration of $\mathbf{aux}$ and its corresponding auxiliary components, the correctness of ReLU can be equated to the correctness of conventional arithmetic tensor operations. This enables aggregation across layers and training iterations within Protocol 1. Consequently, for soundness, it is only necessary to ensure the correctness of all tensor operations.

Proof of Theorem V.3. A malicious prover may attempt to cheat in Protocol 1 specifically at Lines 7, 8, and 11. We investigate the implications in the reverse order of these steps.

Beginning with the proof of evaluation in Line 11, the commitment scheme's binding properties ensure a soundness error of $\texttt{negl}\left(\lambda\right)$. Thus, if the prover attempts to falsely claim that v is equivalent to $\widetilde{\mathbf{S}_i^{(t)}}\left(\mathbf{u}_i^{(t)}\right)$ for any $v \neq \widetilde{\mathbf{S}_i^{(t)}}\left(\mathbf{u}_i^{(t)}\right)$, the verification passes with a probability no greater than $\texttt{negl}\left(\lambda\right)$.

Next, focusing on Line 7, we consider the case where the prover begins with a false assertion regarding at least one $\widetilde{\mathbf{X}_k}\left(\mathbf{u}_k, \mathbf{u}\right)$ in the sumcheck of (30) but ultimately claims accuracy for the stacked tensor $\mathbf{X}$. Here, the randomness of r_ks implies that (30) is likely not satisfied with a probability of $1-\frac{1}{|\mathbb{F}|}$. Given this, the failure probability of the sumcheck

Protocol 2 Sumcheck on Equation (24)

Require: Prover $\mathcal{P}$, verifier $\mathcal{V}$, $N, D_{\text{in}}, D_{\text{out}}$ are powers of 2 (zero-padding may be applied otherwise)

1: Denote $n := \log_2 N, d_{\text{in}} := \log_2 D_{\text{in}}, d_{\text{out}} := \log_2 D_{\text{out}}$

2: $\mathcal{P}$ sends to verifier the uni-variate polynomial

$$f_0(v) := \sum_{\mathbf{n}' \in \{0,1\}^{n-1}} \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{w}_{[1:n]}, \mathbf{n}'\right) \widetilde{\beta}\left(\mathbf{u}, \mathbf{i}\right) \left(D_{\text{in}}^{-1} \widetilde{\mathbf{Y}}\left(v, \mathbf{n}', \mathbf{i}\right) - f\left(\widetilde{\mathbf{X}_k}\left(v, \mathbf{n}', \mathbf{i}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \right) \tag{41}$$

3: $\mathcal{V}$ computes $g_0(v) \leftarrow \widetilde{\beta}\left(\mathbf{w}_{[0]}, v\right) f_0(v)$, and checks $g_0(0) + g_0(1) = 0$

4: $\mathcal{V}$ sends to $\mathcal{P}$ a uniform random $v_0 \sim \mathbb{F}$

5: **for** $t \leftarrow 1, 2, \ldots, n-1$ **do** ▷ Denote $\mathbf{v}_t := (v_0, v_1, \ldots, v_{t-1})^\top$

6: $\mathcal{P}$ sends to $\mathcal{V}$ the uni-variate polynomial

$$f_t(v) := \sum_{\mathbf{n}' \in \{0,1\}^{n-t-1}} \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{w}_{[t+1:n]}, \mathbf{n}'\right) \widetilde{\beta}\left(\mathbf{u}, \mathbf{i}\right) \left(D_{\text{in}}^{-1} \widetilde{\mathbf{Y}}\left(\mathbf{v}_t, v, \mathbf{n}', \mathbf{i}\right) - f\left(\widetilde{\mathbf{X}_k}\left(\mathbf{v}_t, v, \mathbf{n}', \mathbf{i}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \right) \tag{42}$$

7: $\mathcal{V}$ computes $g_t(v) \leftarrow \widetilde{\beta}\left(\mathbf{w}_{[t]}, v\right) f_t(v)$, and checks $g_t(0) + g_t(1) = f_{t-1}(v_{t-1})$

8: $\mathcal{V}$ sends to $\mathcal{P}$ a uniform random $v_t \sim \mathbb{F}$

9: **end for**

10: $\mathcal{P}$ and $\mathcal{V}$ execute the sumcheck protocol on

$$f_{n-1}(v_{n-1}) = \sum_{\mathbf{i}=0}^{D_{\text{out}}-1} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{u}, \mathbf{i}\right) \left(D_{\text{in}}^{-1} \widetilde{\mathbf{Y}}\left(\mathbf{v}_{n-1}, \mathbf{i}\right) - f\left(\widetilde{\mathbf{X}_k}\left(\mathbf{v}_{n-1}, \mathbf{i}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \right) \tag{43}$$

on (30) is $1 - O\left(\frac{\log N}{|\mathbb{F}|}\right)$, making the overall deceptive success probability $\texttt{negl}(\lambda)$.

Lastly, for any incorrectly computed tensor operation, consider its contribution to the sumcheck protocol (24) observed in Line 7. Consequently, a pair of indices $\mathbf{n}, \mathbf{i}$ exists for which

$$\widetilde{\mathbf{Y}}\left(\mathbf{n}, \mathbf{i}\right) \neq \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{u}, \mathbf{i}\right) f\left(\widetilde{\mathbf{X}_k}\left(\mathbf{n}, \mathbf{i}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1}. \tag{52}$$

By applying the Schwartz-Zippel Lemma, it is deduced that the equality in (24) cannot be maintained with a probability of $1 - O\left(\frac{\log(ND_{\text{in}})}{|\mathbb{F}|}\right)$. In such scenarios, the sumcheck has an upper bound success rate of $O\left(\frac{\log(ND_{\text{in}}D_{\text{out}})}{|\mathbb{F}|}\right)$. Thus, the probability that a deceitful prover triumphs during the sumcheck for combined tensor operations is similarly $\texttt{negl}(\lambda)$. □

C. Zero-knowledge

Theorem C.2 captures the zero-knowledge properties of zkDL, i.e., the execution of zkDL leaks no information about the training data and model parameters:

Theorem C.2 (Zero-knowledge)**.** *Assuming the implementation of the zero-knowledge Pedersen commitment scheme and the zero-knowledge variant of the sumcheck protocol [4], [32], [33], Protocol 1 is zero-knowledge. Specifically, there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ for which the ensuing two views are computationally indistinguishable by any probabilistic polynomial-time (PPT) algorithm, given the public parameters pp (the generators used in the commitment scheme within the context of zkDL):*

Real:

1: $com \leftarrow zkDL\text{-}Commit\left(\mathbf{X} \| \mathbf{y} \| \mathbf{W}_{init}; pp\right)$

2: $\pi \leftarrow zkDL\text{-}Prove(com; pp)$

3: ***return*** com, π

Ideal:

1: $com \leftarrow \mathcal{S}_1\left(1^\lambda; pp\right)$

2: $\pi \leftarrow \mathcal{S}_2(com; pp)$*, with oracle access to the correctness of the training procedure*

3: ***return*** com, π

In the aforementioned setting, $zkDL\text{-}Commit\left(\mathbf{X} \| \mathbf{y} \| \mathbf{W}_{init}; pp\right)$ pertains to the steps of training (Line 2) and making commitments to the tensors (Line 4) in Protocol 1. Meanwhile, $zkDL\text{-}Prove(com; pp)$ refers to the processes of the sumcheck protocols (Lines 7 and 8) and the proof of evaluations relative to the commitments (Line 11).

Proof sketch of Theorem C.2. Firstly, by leveraging the zero-knowledge sumcheck protocols, for each tensor operation $\mathcal{A}_j$, there is a simulator $\mathcal{S}^{\mathcal{A}_j}$ that indistinguishably simulates the execution of the sumcheck protocol with sole reliance on oracle access to validate the correctness of the aggregated operations among them. In contrast, for each tensor class $\mathcal{T}_i$, there exists a pair of simulators, denoted as $\left(\mathcal{S}_1^{\mathcal{T}_i}, \mathcal{S}_2^{\mathcal{T}_i}\right)$. Specifically, $\mathcal{S}_1^{\mathcal{T}_i}$ is tasked with simulating the generation of the commitment, while $\mathcal{S}_2^{\mathcal{T}_i}$ simulates the proof of evaluation, contingent on oracle access to ascertain the evaluation at the precise point the committed tensor is assessed.

Building on this, given that the randomness across all

Protocol 3 Sumcheck on Equation (25)

Require: Prover $\mathcal{P}$, verifier $\mathcal{V}$, $N, D_{\text{in}}, D_{\text{out}}$ are powers of 2 (zero-padding may be applied otherwise)

Denote $n := \log_2 N, d_{\text{in}} := \log_2 D_{\text{in}}, d_{\text{out}} := \log_2 D_{\text{out}}$

$\mathcal{P}$ sends to verifier the uni-variate polynomial

$$f_0(v) := \sum_{\mathbf{n}' \in \{0,1\}^{n-1}} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{w}_{[1:n]}, \mathbf{n}'\right) \widetilde{\beta}\left(\mathbf{u}_{J_\beta}, \mathbf{j}_{J_\beta}\right) f\left(\widetilde{\mathbf{X}_k}\left(v, \mathbf{n}', \mathbf{u}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \tag{44}$$

$\mathcal{V}$ computes $g_0(v) \leftarrow \widetilde{\beta}\left(\mathbf{w}_{[0]}, v\right) f_0(v)$, and checks $g_0(0) + g_0(1) = \widetilde{\mathbf{Y}}\left(\mathbf{w}, \mathbf{u}\right)$

$\mathcal{V}$ sends to $\mathcal{P}$ a uniform random $v_0 \sim \mathbb{F}$

for $t \leftarrow 1, 2, \ldots, n-1$ **do** ▷ Denote $\mathbf{v}_t := (v_0, v_1, \ldots, v_{t-1})^\top$

$\mathcal{P}$ sends to $\mathcal{V}$ the uni-variate polynomial

$$f_t(v) := \sum_{\mathbf{n}' \in \{0,1\}^{n-t-1}} \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{w}_{[1:n]}, \mathbf{n}'\right) \widetilde{\beta}\left(\mathbf{u}_{J_\beta}, \mathbf{j}_{J_\beta}\right) f\left(\widetilde{\mathbf{X}_k}\left(\mathbf{v}_t, v, \mathbf{n}', \mathbf{u}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \tag{45}$$

$\mathcal{V}$ computes $g_t(v) \leftarrow \widetilde{\beta}\left(\mathbf{w}_{[t]}, v\right) f_t(v)$, and checks $g_t(0) + g_t(1) = f_{t-1}(v_{t-1})$

$\mathcal{V}$ sends to $\mathcal{P}$ a uniform random $v_t \sim \mathbb{F}$

end for

$\mathcal{P}$ and $\mathcal{V}$ execute the sumcheck protocol on

$$f_{n-1}(v_{n-1}) = \sum_{\mathbf{j}=0}^{D_{\text{in}}-1} \widetilde{\beta}\left(\mathbf{u}_{J_\beta}, \mathbf{j}_{J_\beta}\right) f\left(\widetilde{\mathbf{X}_k}\left(\mathbf{v}_{n-1}, \mathbf{u}_{I_k}, \mathbf{j}_{J_k}\right)\right)_{k=0}^{K-1} \tag{46}$$

aforementioned simulators remains independent, the overall simulator for zkDL can be architected in the following manner:

1) $\mathcal{S}_1$, realized as a composite of all $\mathcal{S}_1^{\mathcal{T}_i}$s, is responsible for simulating the creation of commitments.
2) With oracle access to the integrity of the comprehensive training process, that is, the correctness of all aggregated tensor operations, $\mathcal{S}_2$ emerges as a combined entity of both $\mathcal{S}^{\mathcal{A}_j}$s and $\mathcal{S}_2^{\mathcal{T}_i}$s. Its role is to simulate proofs of tensor operations' correctness through sumchecks and the subsequent proofs of evaluations.

As a consequence, the composite simulated transcript, owing to the inherent independence between the components generated by these simulators, remains computationally indistinguishable from an authentic transcript. □