

TeleSparse: Practical Privacy-Preserving Verification of Deep Neural Networks

Mohammad M Maheri
Imperial College London

Hamed Haddadi
Imperial College London

Alex Davidson
LASIGE, Universidade de Lisboa

ABSTRACT

Verification of the integrity of deep learning inference is crucial for understanding whether a model is being applied correctly. However, such verification typically requires access to model weights and (potentially sensitive or private) training data. So-called Zero-knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) would appear to provide the capability to verify model inference without access to such sensitive data. However, applying ZK-SNARKs to modern neural networks, such as transformers and large vision models, introduces significant computational overhead.

We present TeleSparse, a ZK-friendly post-processing mechanisms to produce practical solutions to this problem. TeleSparse tackles two fundamental challenges inherent in applying ZK-SNARKs to modern neural networks: (1) Reducing circuit constraints: Over-parameterized models result in numerous constraints for ZK-SNARK verification, driving up memory and proof generation costs. We address this by applying *sparsification* to neural network models, enhancing proof efficiency without compromising accuracy or security. (2) Minimizing the size of lookup tables required for non-linear functions, by optimizing activation ranges through neural *teleportation*, a novel adaptation for narrowing activation functions' range.

TeleSparse reduces prover memory usage by 67% and proof generation time by 46% on the same model, with an accuracy trade-off of approximately 1%. We implement our framework using the Halo2 proving system and demonstrate its effectiveness across multiple architectures (Vision-transformer, ResNet, MobileNet) and datasets (ImageNet, CIFAR-10, CIFAR-100). This work opens new directions for ZK-friendly model design, moving toward scalable, resource-efficient verifiable deep learning.

KEYWORDS

deep learning, verifiable neural network inference, zero-knowledge proof, machine learning, sparsification, teleportation

1 INTRODUCTION

Deep learning models have had considerable success across various machine learning (ML) tasks in recent years. The proliferation of large-scale deep learning models, characterized by their parameter-intensiveness (i.e. in the millions), has become commonplace. However, as the number of parameters in these models increases, several challenges pertaining to trustworthiness in ML arise.

- (1) Due to the escalating number of parameters and computational demands for training or inferring the model, consumers often opt to outsource these computations to service providers, a practice known as ML-as-a-service (MLaaS). Consequently, ensuring the integrity of predictions becomes crucial, especially in cases where consumers lack trust in the reliability of the model provider.

- (2) Training large deep neural networks (DNN) often involves vast amounts of privacy-sensitive or proprietary data, and demands substantial computational resources. As a result, the trained model becomes valuable intellectual property for the trainer. Publishing model weights for auditing purposes [78] is typically impractical. This creates a need for methods to verify specific model properties, while keeping the weights private from the verifier.
- (3) Where client data is highly sensitive, model providers may opt to send the model to client devices for local inference (e.g., large language models). However, this introduces the risk of intentional or unintentional deviations from correct computations. Such deviations may result from attacks on vulnerable client devices or malicious clients attempting to benefit from producing incorrect model predictions.

Consequently, ensuring the correctness of model inference becomes paramount for model providers or any other third-party seeking to trust the model's output predictions. In all three above scenarios, the integrity of model inference is critical, necessitating the implementation of verifiable model inference mechanisms.

ZK-SNARK. Verifying the integrity of inference while preserving privacy of model weight is a critical challenge, as previous research has shown that revealing model weights can expose privacy of the training data through attacks like membership inference [13, 51, 73]. Cryptographic primitives, particularly zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARKs) [8, 38, 61], have emerged as a solution to verifying inference outputs in a privacy-preserving manner [7, 36, 50, 55, 75]. Most machine learning applications require succinctness in zero-knowledge proofs (ZKP), especially when verifiers operate on resource-constrained devices like edge or mobile platforms. While recent ZK constructions, such as those proposed in [52], aim to reduce the high prover costs associated with ZK-SNARKs, they often lack the compact proof size essential for ML applications. The SNARK-based approach holds promise due to: i) its ability to generate succinct proofs, typically verifiable in under 1 second; and ii) the verification process can be conducted by any third party solely by accessing the generated proof, demonstrating public verifiability and eliminating the need for interaction despite methods based on interactive ZK protocol [74]. While ZK-SNARK presents a promising avenue for verifiable deep models, it necessitates significant computational resources on the prover's side, limiting its application primarily to toy example deep learning models. To mitigate this challenge, various cryptographic techniques have been explored, including: i) quantization NN to low precision integers (8bit or 16bit) [23]; ii) utilization of lookup tables for non-linear functions [36]; and iii) efficient representation of relations for convolution equations [50, 55, 75] and attention mechanism in transformer architecture [67]. However,

there has been limited exploration into post-processing deep machine learning models to make them ZK-friendly, in order to reduce the required resources for proof verification.

Computational overheads. With the rise of parameter-intensive neural networks, large-scale models are increasingly employed in various ML applications, such as vision and language models. The cryptographic optimizations mentioned can decrease memory usage and proof generation time for ensuring the integrity of model inferences. However, with significant overheads to naively composing such proofs with such models still present, it is necessary to develop innovative approaches beyond simply using ZK-SNARKs.

Inspired by the emerging trend of tiny ML [53, 81], compression of models appears promising in this regard. However, such techniques should possess specific properties. Firstly, they should be lightweight, ensuring that model compression requires less computation than the reduction of resources needed for proof generation, resulting from said compression. Secondly, compressing the model weights and/or activation values should preserve its performance as much as possible, in terms of maintaining its overall accuracy. If compression compromises model performance, the proof generation step becomes futile. Finally, compression techniques should be adaptable to zero-knowledge proving systems, reducing the required resources without compromising the security guarantees of the proving system.

Our work. In this study, we investigate the root causes of proof generation overhead in ZK-SNARK systems (both in terms of memory usage and computational time) when applied to modern deep learning architectures, such as transformers [71]. We identify two primary challenges. First, the growing complexity of modern architectures like transformers, which contain significantly more parameters than earlier models, results in a corresponding increase in the number of constraints required by ZK-SNARK backends for verification. Second, handling non-linear functions in DNNs poses a challenge for ZK systems, as these functions cannot be directly encoded in arithmetic circuits. In sum-check based proving systems, this limitation is addressed by using bit decomposition and polynomial approximations of non-linear functions [28, 55, 75]. However, these methods introduce computational overhead and can reduce model accuracy due to the inherent approximation of complex functions [23, 67]. In particular, polynomial approximations are not fully precise representations of the original functions. Recent advances in verifiable machine learning models [36, 67] — utilizing lookup tables to handle the non-linear activation functions used in DNNs — still face significant challenges, due to the need for large lookup tables to cover the broad input ranges of activations. These lookup tables introduce considerable overhead, leading to slower proof generation, increased memory demands for the prover, and larger proof sizes. This challenge is particularly acute in transformer architectures, which are extensively employed in large language models (LLMs) [63, 76] and modern vision models such as Vision Transformers (ViTs) [18]. These models are known to produce outlier activation values [3, 9, 68], exacerbating the issues associated with lookup table size and efficiency.

To address the first problem, we carefully adopt a lightweight pruning technique that suits the constraints and requirements of ZK-SNARK verification, in a resource-efficient manner. This approach

reduces the number of constraints necessary for circuit verification by *sparsifying* the model. In other words, allowing inference verification on a pruned model that meets both computational and memory efficiency demands essential for zero-knowledge proofs.

To address the second, we identify symmetric neural network configurations that minimize the range of activation values, thereby reducing the size of the necessary lookup tables. This approach significantly decreases the computational and memory overhead associated with proof generation: a crucial step toward more efficient and scalable ZK-SNARK verification for modern neural network architectures. We achieve this symmetry discovery by formulating an optimization problem over neural teleportation [2], a technique initially developed to accelerate and study neural network training, here adapted to optimize the activation input range. This novel application of neural teleportation allows us to efficiently manage activation ranges in models, maintaining both verification speed and resource efficiency in zero-knowledge contexts.

To implement ZK-SNARKs under the outlined constraints, we leverage Halo2 [82], a recent proving system that provides a highly efficient and flexible backend for ZK applications. Halo2 is a ZK-SNARK constructions that builds on the principle of recursive proof composition and operates without a trusted setup [10], making it particularly suitable for layer-wise DNNs. Moreover, its flexible arithmetization with lookup argument capabilities [26] handles non-linear DNN functions efficiently. Building on this framework, TeleSparse achieves about 67% reduction in proof generation time and a 46% reduction in prover memory usage, with a minimal 1% performance loss on the CIFAR-100 [42] dataset. This carefully chosen pruning strategy is post-processed, making it compatible with any pre-trained transformer model without requiring modifications to the training pipeline and adding less than 1% memory overhead.

Contributions. We present the following key contributions:

- We bridge the gap in adapting modern DNN architectures like transformers to ZK-SNARKs, by post-processing pre-trained models for scalability. This approach identifies two key factors to reduce overheads: minimizing circuit constraints in ML models, and optimizing lookup table ranges.
- We introduce model sparsification to efficiently generate ZK-SNARK proofs, specifically by devising a method that leverages sparsity to reduce the number of circuit constraints. This approach not only enhances proof efficiency but also maintains the security of the underlying proving system, which we formalize in our methodology.
- We present a novel application of neural teleportation to optimize activation ranges, reducing lookup table sizes. This approach, originally used for improving training convergence, proves effective across a wide range of models, including transformer-based architectures.
- We implement TeleSparse on diverse architectures (Vision Transformers, ResNet, MobileNet) and datasets (CIFAR-10, CIFAR-100, ImageNet), demonstrating its efficiency, scalability, and accuracy compared to state-of-the-art methods.

Overall, our contributions are orthogonal to existing ZK-SNARK research that focuses on optimizing the arithmetization of DNN layers (like CNNs and transformers) [55, 74] and parallelizing proof generation using GPU resources [67]. Instead, our work explores

how to make DNN models themselves ZK-friendly—specifically adapting models to be computationally efficient and aware of ZK system requirements. This approach offering a synergistic pathway toward scalable and ZK-efficient ML.

On top of our stated results, we believe that our work can open a new path for designing DNN training or post-training, to make the output model more ZK-friendly. The code is available at this link.

2 RELATED WORKS

2.1 Verifiable deep model inference

The field of secure inference in machine learning (ML) has seen rapid growth in protecting various aspects: i) privacy of inputs, ii) privacy of model parameters, and iii) integrity of model computation against potential malicious actors. Prior efforts to address these critical issues have employed techniques such as multi-party computation (MPC), homomorphic encryption (HE), or zero-knowledge (ZK) proofs. MPC methods distribute computation across multiple parties to prevent the exposure of inputs (e.g., model weights and ML model’s input) [40, 43, 66, 83]. However, these approaches require synchronous interaction among parties, which is often impractical in many ML scenarios. Additionally, most MPC protocols only consider semi-honest adversaries [40, 43, 66, 83], making them susceptible to parties deviating from the protocol. Another approach involves leveraging HE, which allows computation on encrypted data. However, HE does not provide verification for the integrity of ML computations. Moreover, HE entails high computational costs, rendering it impractical for deep models, particularly parameter-intensive models [49, 57]. As the demand for publicly verifiable proof in ML computations rises in various scenarios, efforts have pivoted towards generating ZK proof of ML computations.

Recent research has explored using ZK proving systems to introduce verifiability in the inference of machine learning models. An initial study [23] utilized Groth16 [30] for verifying neural network inference. To mitigate the significant overhead of proof generation, several studies have introduced novel computation representations, such as Quadratic polynomial programs [41, 50]. These representations are tailored to model-specific architectures, particularly Convolutional Neural Networks (CNNs) [6, 20, 50, 55] and Transformers [67]. These approaches are constrained by their focus on a specific model architecture, and most of them [6, 20, 23, 50, 55] struggle with inefficiency in handling non-linear functions (e.g., common activation functions like ReLU), due to the limitations imposed by the underlying constraint systems of their proving systems. They introduce additional overhead because of the need for bit-decomposition or approximating activation functions with polynomial representations, both of which are ineffective for modern activation functions like Gaussian error linear unit (Gelu) [33]. To address the non-linearly functions challenge, [36] employed the more recent proving system Halo2 [82] and adopted Plonk arithmetization [26] to utilize lookup tables for non-linear functions. Alternative lookup proofs [31, 58] reduce computational costs but differ from PLONK arithmetization. Specifically, they do not ensure constant proof size (succinctness) or maintain non-interactivity, which would allow for minimal communication between the prover and verifier. As noted, previous research efforts attempted to represent

neural network model computations efficiently based on the constraint systems provided by their proving mechanisms. Although they have investigated efficient ZK proving by employing cryptographic techniques, they overlooked making DNNs ZK-friendly. To address this gap, we propose post-training on the deep model to reduce both proving time and memory consumption significantly.

2.2 Sparse training/inference

Prior research [24, 35, 62] has shown that a small subset of a fully trained DNN is sufficient to represent the learned function. This property of DNNs allows for the development of efficient architectures, reducing computational and memory requirements during training (sparse training) [35, 48, 54, 64, 70, 84] or through post-training techniques [45, 47, 59, 69]. Sparse training methods focus on decreasing computational load and model footprint during training, requiring optimized algorithms from the early stages of the training process. On the other hand, post-training pruning techniques prune unnecessary weights from a well-optimized model (dense model) using calibration data. These methods aim to reduce inference time and model size while preserving the model’s performance.

Pruning techniques are typically classified into structured and unstructured methods. Structured pruning [22, 59, 79] generally removes entire neurons, channels, or filters, which can lead to a significant drop in accuracy and often requires extensive fine-tuning to recover performance. In contrast, unstructured pruning [5, 46, 77] targets the removal of individual weights, better preserving the model’s performance. However, unstructured pruning requires careful hardware implementation to reduce inference delay effectively.

Although the benefits of sparsification for reducing model size and improving hardware efficiency have been explored, sparse neural networks have not yet been utilized for efficient ZK-proof model verification. In our work, we propose a method to leverage sparsification to reduce the overhead of ZK-SNARK proof generation. Additionally, TeleSparse supports unstructured sparsification, crucial for maintaining the model’s performance. To the best of our knowledge, this is the first study to explore pruning methods that specifically reduce memory consumption and CPU computation while generation ZK-SNARK proofs for sparse DNN models.

2.3 Neural Network Teleportation

Continuous symmetries in neural networks refer to the invariance in the parameter space, where certain transformations of the network’s weights do not affect its output. These symmetries emerge as a result of overparameterization, where multiple distinct weight sets can represent the same model function [29]. Such symmetries have been observed in networks with homogeneous activation functions [4, 19], as well as in other architectural components, such as softmax and batch normalization [44]. Investigating these symmetries has contributed to improved training optimization and better generalization.

Neural teleportation, explored through quiver representation theory [1], exploits symmetries in the loss landscape. It moves network parameters to a different point with the same objective value, enabling faster convergence in gradient-based optimization.

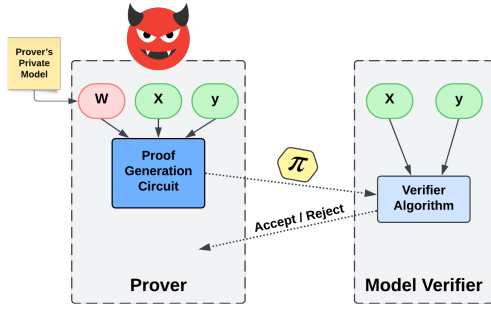


Figure 1: System diagram of ZK-SNARK DNN inference.

This helps traverse the loss landscape efficiently by moving between different symmetric configurations of the model [2].

Several works have expanded on neural teleportation and continuous symmetries. For example, [85] proposed a symmetry teleportation algorithm that not only searches for optimal teleportation destinations but also leverages symmetries to maximize the gradient, thereby accelerating gradient descent. Additionally, they developed a general framework based on equivariance to analyze the loss landscape and the dimensions of minima induced by symmetries.

In this work, we propose a novel application of neural teleportation to minimize the range of inputs to activation functions. This approach effectively mitigates outlier activation inputs [9], which leads to large lookup tables in ZK proof generation. Furthermore, we extend the concept of neural teleportation to modern neural networks that use non-scale-invariant activation functions, such as GELU, which have been less explored in prior research.

3 SYSTEM AND THREAT MODEL

Notation. The mathematical notations are listed in Appendix C.

The goal of our system, shown in Figure 1, is to ensure verifiable and privacy-preserving inference in DNN models. The system involves two primary parties:

- (1) **Prover (P):** Owns a private deep learning model with parameters (W). The prover computes the model’s output (y) for a given input (X) and generates a proof (π) to demonstrate the correctness of the computation, i.e., $y = f(X; W)$, securely and privately.
- (2) **Verifier (V):** Receives the proof (π) and verifies that $y = f(X; W)$ without learning anything about W , ensuring the integrity and confidentiality of the model.

The private deep learning model inference can be represented as $y = f(X; W)$, where:

- X : Public input to the ZK circuit, representing the input data to the DNN.
- y : Public input to the ZK circuit, corresponding to the output of the DNN computation.
- W : Private input to the ZK circuit, representing the parameters (weights) of the DNN.

In pursuit of this objective, we utilize ZK-SNARK [8], a cryptographic protocol that enables Prover P to generate a proof π . This proof enables a Verifier V to ascertain, with only the knowledge

of π , y , and X , that the Prover possesses certain parameters W satisfying $y = f(X; W)$. Following the Halo2 protocol, which is an instantiation of a ZK-SNARK proving system, our system inherits all security properties of Halo2 [8, 36], including Knowledge Soundness, Zero-knowledge detailed in Appendix A.

The proving system operates under the security assumption that adversaries are computationally limited [11]. Even if adversaries deviate from the proving protocol, they cannot generate a valid proof for an incorrect computation. This ensures that malicious adversaries cannot compromise privacy (zero-knowledge property) or undermine the knowledge soundness of the ZK-proving system. We discuss the potential threats arising within our system and security model below in Section 3.1. Regardless, note that this represents a stronger security assumption compared to the honest-but-curious threat model commonly adopted in differential privacy (DP) and multi-party computation (MPC) methodologies.

As in prior works [30, 36, 75], we assume that the model architecture is public (known to the verifier), while the model weights remain private. This assumption aligns with the trend toward open-source models [36]. Such an approach is particularly relevant for practical applications where model providers aim to protect the intellectual property of their weights while enabling the verification of computations. Applications are detailed in Appendix B.

3.1 Potential Threats

The system addresses the following threats:

- **Privacy Threats:** Ensures that W (the model’s parameters) remains private and that V learns nothing beyond y . However, since the verifier observes the proof, an adversary could attempt to extract information about W from it, posing a potential privacy risk. We analyze this leakage and its implications in Section 7.
- **Integrity Threats:** Prevents adversaries from generating valid proofs (accepted by V) for incorrect computations of f or tampering with X or y .

3.2 Design Goals

The primary objective of our approach is to make ZK-SNARKs practical and scalable for verifiable inference on modern DNNs. As noted in Section 2.1, ZK-SNARKs enable efficient verifiable computation due to their ability to generate succinct proofs, which drastically reduces the verifier’s computational load. This efficiency, however, comes at the cost of substantial prover overhead in terms of required memory and proof generation time. For instance, proving even a relatively small model, like the Tiny Vision Transformer (Tiny-ViT) [18], requires over 10TB of memory, which exceed the capacity of most practical systems.

To overcome the challenges associated with ZK-SNARKs for DNNs, we have outlined several key objectives that collectively enable practical and scalable verifiable inference on modern DNNs:

- (1) **Reducing the Number of Circuit Constraints (G_1):** Modern DNNs are typically parameter-intensive, resulting in an enormous number of operations, often in the billions for models such as vision transformers. To construct a ZK-SNARK, each operation within the neural network must be “arithmetized,” or converted into circuit constraints, which

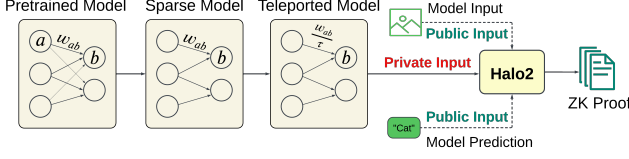


Figure 2: System overview: the neural network weight is fed as the private input to the Halo2 circuit. Input and output of the neural network is public input to the circuit.

the prover must satisfy to generate a valid proof. However, as the number of operations—and consequently, the number of constraints—increases, so do the memory usage and proof generation time required by the prover. Our approach addresses this by leveraging neural network sparsification to reduce the number of constraints associated with DNNs. Specifically, we devise a way to efficiently prove statements about sparse DNNs, preserving predictive performance while significantly reducing prover memory and computation costs. Detailed implementation of this approach is covered in Section 4.

- (2) Reducing Lookup Table Argument Overhead (G_2): Non-linear functions commonly used in DNNs, such as activation functions, present a significant challenge in ZK proof generation. These functions often need to be approximated through large lookup tables to fit within a ZK circuit framework. As prior work has shown, non-linear layers in models like ResNet-101 can consume up to 80% of the prover’s computational load due to extensive lookup table operations [31, 74]. Although Halo2 provides efficient support for lookup arguments, our analysis reveals that non-linear functions still account for a substantial portion of the prover’s computation and increase proof size. To address this, we propose an optimized neural network teleportation detailed in Section 5.

3.3 Methodology Overview

This section presents our methodology overview, outlining how each design goal from Section 3.2 is addressed. Section 4 describes our approach to reducing the number of circuit constraints by applying sparsification techniques to DNNs (design goal 1). We formalize the constraint-reduction process and its security analysis plus leverage two state-of-the-art post-pruning methods to illustrate its effectiveness in our experiments. In Section 5, we use Neural Network Teleportation to constrain the range of activation functions, achieving design goal 2. The overview of the system is shown in Figure 2, and the steps of the proposed TeleSparse method for producing a ZK-friendly DNN are outlined in Algorithm 1.

4 SPARSIFICATION

As noted in 3.2, we aim to optimize the required resource of the ZKP generation process by eliminating unnecessary constraints to achieve design goal 1. In the context of Halo2, we provide an example of one simple linear layer (fundamental layer in deep learning

Algorithm 1: Overview of Methodology for Efficient Zero-Knowledge Proof Generation

Input: Neural network weights W , input X , output y

Output: Valid zero-knowledge proof of computation represented by π aligning with design goals (G_1, G_2)

Sparsification (Addressing G_1):

Generate sparsified weights W_{sparse} by solving:

$$W_{\text{sparse}} \leftarrow \arg \min_W \sum_{i=1}^L \left\| f(x; W^{(i)}) - f(x; \tilde{W}^{(i)}) \right\|^2$$

▷ guided by pruning criteria (Eq. 8).

Teleportation (Addressing G_2):

Optimize CoB τ^* :

$$\tau^* \leftarrow \arg \min_{\tau} \mathcal{L}(\tau) \quad \text{▷ where } \mathcal{L}(\tau) \text{ is defined in Eq. 14;}$$

(Solved by Eq. 16, Eq. 17, Eq. 18).

Apply τ^* to W_{sparse} (Eq. 9, Eq. 10) to compute W_{sparse}^t .

ZK-SNARK Proof Generation:

- (1) Convert the model operations into Halo2 constraints (Eq. 2);
- (2) Eliminate constraints for indices where $W_{\text{sparse}}^t = 0$ to reduce circuit size.
- (3) Generate the zk-proof:

$$\pi \leftarrow \text{Halo2.Prove}(W_{\text{sparse}}^t, X, y)$$

models) which is matrix-vector multiplication to show how sparsification of the model (matrix in this example) could reduce the number of required constraints without damaging security properties of ZK-SNARK. Specifically, we prove that omitting constraints corresponding to zero entries in the matrix (deep model weights) does not compromise the soundness of the proving system. We then present the two post-pruning methods employed in our experiments to complete the path towards reducing prover overhead by sparsification without degrading the accuracy of the DNN model.

4.1 Preliminaries

Halo2 Proving System. Halo2 employs an extended form of the PLONK arithmetization [26], referred to as the “PLONKish”, which structures constraints in a polynomial format and introduces custom gates, lookup arguments, and permutation constraints. This framework is designed to operate over a matrix structure with cells, rows, and columns, where the rows correspond to elements in a multiplicative subgroup of a finite field F_q , typically sized as a power of two ($q = 2^k$).

Each circuit in Halo2 is represented by a matrix, where columns are categorized into:

- **Fixed Columns (Fixed):** Columns with values fixed at circuit synthesis time, same across all proofs. Commitments to these columns are included in the verification key [82].

- **Advice Columns (Advice):** Columns where the prover assigns values during proof generation. These values are private and known only to the prover.
- **Instance Columns (Instance):** Columns representing public inputs to the circuit, provided by the verifier and fixed at proof time. These columns facilitate the verification of statements involving known values.

In addition to these columns, polynomial commitments are utilized to secure the integrity of the data within these columns. A commitment scheme, such as KZG commitments [37], is employed to bind the polynomials that represent each column.

During proof creation, the prover sets up a matrix with *advice*, *instance*, and *fixed columns* and populates each cell with field elements, denoted $A_{i,j}$ for advice or $F_{i,j}$ for fixed values in j -th row and i -th column. To commit to these values, the prover constructs Lagrange polynomials of degree $n - 1$ for each column. For advice and fixed columns, the Lagrange polynomials $a_i(X)$ and $q_i(X)$ are interpolated over the evaluation domain of size n , where ω is the n -th primitive root of unity:

$$a_i(X) \text{ such that } a_i(\omega^j) = A_{i,j},$$

$$q_i(X) \text{ such that } q_i(\omega^j) = F_{i,j}.$$

Commitments to these polynomials are then made as:

$$A = [\text{Commit}(a_0(X)), \dots, \text{Commit}(a_i(X))],$$

$$Q = [\text{Commit}(q_0(X)), \dots, \text{Commit}(q_i(X))], \quad (1)$$

where Q is established during key generation, while A is produced by the prover and sent to the verifier.

Halo2 enforces three primary types of constraints:

- **Custom Gates:** These define polynomial constraints within rows, enabling expressions such as multiplication and addition to be enforced across specific rows. The generic form of constraints in PLONK (and similarly used in Halo2) is formulated as:

$$a_i(X) \cdot q_A(X) + b_{i'}(X) \cdot q_B(X) + a_i(X) \cdot b_{i'}(X) \cdot q_M(X) + c_{i''}(X) \cdot q_C(X) = 0, \quad (2)$$

where $a_i(X)$, $b_{i'}(X)$, and $c_{i''}(X)$ represent the polynomial assignments in advice columns i, i', i'' of the arithmetic matrix, and $q_A(X)$, $q_B(X)$, $q_M(X)$, and $q_C(X)$ are the corresponding polynomials which are stored in fixed columns.

- **Permutation Arguments:** These allow cell values to match across different locations in the matrix, using randomized polynomial constraints for multi-set equality checks. Permutation arguments enable values to be "copied" within the circuit.
- **Lookup Arguments:** Lookup arguments constrain a k -tuple of cells $(A_{1,j}, \dots, A_{k,j})$ in the same row j such that for a disjoint set of k columns, these cells match the values in some other row j' . We can enforce the following constraint:

$$(A_{1,j}, \dots, d_{k,j}) = (A_{1',j'}, \dots, A_{k',j'}),$$

ensuring $(A_{1,j}, \dots, A_{k,j})$ lies within the lookup table defined by those k columns [36, 82].

The formal ZK-SNARK properties guaranteed by Halo2 and the commitment schema are detailed in Appendix D.

4.2 Matrix-Vector Multiplication in Halo2

In this section, we formulate circuit constraints for sparse model weights, aiming to reduce the overall number of required constraints. Without loss of generality, we focus on a fully connected (linear) layer in neural networks. In this setting, the model parameters (weights of the linear layer) are represented by a matrix $W \in \mathbb{R}^{d_1 \times d_2}$, and the input to the layer (the output from the previous layer) is represented by a vector $I \in \mathbb{R}^{d_2}$. The output of the layer is denoted by vector $O \in \mathbb{R}^{d_1}$, calculated as:

$$O = W \cdot I$$

Each component o_i of the output vector o is computed as:

$$o_i = \sum_{j=1}^{d_2} W_{i,j} \cdot I_j \quad (3)$$

By leveraging the sparsity of W , we can reduce the number of constraints in the circuit, thereby decreasing the computations required for verification. In the following we explore how it would be possible for Halo2 to do that efficiently without compromising the soundness of Halo2 mentioned in Equation [21].

Circuit Representation. To construct a ZKP over the matrix-vector multiplication of Equation (3), we should describe the equation using Halo2 circuit constraints using the generic constraints form presented in equation 2.

To provide proof generation on the Linear Layer, as shown in figure 3, we put the model weights $W_{i,j}$ in a fixed column of the Halo2 table named F_0 and input of the current layer (I_j) to an advice column named A_0 . To provide the outputs we consider two advice columns A_1 and A_2 such that for all rows $i \in d_2$ the following equation should be kept for table cells:

$$F_{0i} * A_{0i} + A_{1i} = A_{2i} \quad (4)$$

Note that these constraints followed the form of plonkish constrained previously mentioned in 2. By interpolation of the polynomial function, these constraints will be enforced by the following polynomial equation which should be validated by the prover:

$$A_0(X) \cdot Q_{F_0}(X) + A_1(X) - A_2(X) = 0 \quad (5)$$

As illustrated in Figure 3, copy constraints are applied between advice columns A_1 and A_2 . These constraints ensure that the accumulated value in row i of A_2 is correctly propagated to row $i + 1$ in A_1 , preserving value integrity across the table. Furthermore, commitments to the fixed column, $\text{Commit}(Q_{F_0}(X))$, as well as the advice columns, $\text{Commit}(A_1(X))$ and $\text{Commit}(A_2(X))$, are included in the verification key and proof, respectively.

So far, we have shown how a dense linear neural network layer is represented using Halo2 constraints. Next, we explore reducing constraints—rows in the table—when model weights are sparse, enhancing efficiency without compromising Halo2's soundness.

Handling Zero Entries. Based on equation 3, when $W_{i,j} = 0$, the term $W_{i,j}I_j = 0$ does not contribute to the corresponding output o_i while preserving the correctness of the equation. Therefore,

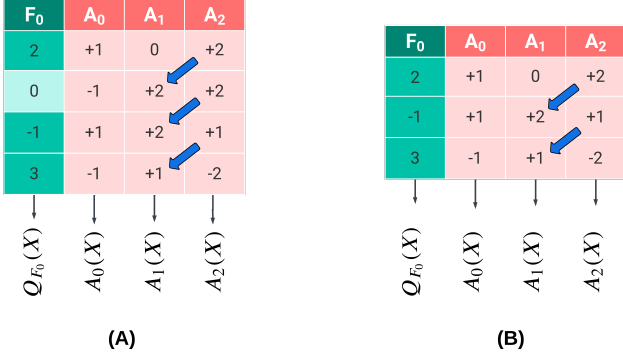


Figure 3: (A) represent a Halo2 circuit without removing sparse weights constraints while (B) represents it after removing zero-weight entries.

we can eliminate constraints involving zero entries so as not to include constraints for terms where $W_{i,j} = 0$. In particular, we could eliminate the row t of the table in equation 4 where F_{0t} is equal to zero. This reduction decreases the number of constraints and computations required during proof generation. In the following, we formalize this constraint reduction.

THEOREM 4.1. *In a matrix-vector multiplication circuit in Halo2, where the matrix W is stored in fixed columns, eliminating constraints corresponding to zero entries $W_{i,j} = 0$ does not compromise the soundness of the proving system.*

The proof of Theorem 4.1 can be found in Appendix E.

4.3 DNN Sparsification

The goal of model sparsification, given a dense model with parameters $W \in \mathbb{R}^d$, is to identify a subset of weights that can be zeroed out while still maintaining the model’s overall performance. We focus on *post-training sparsification methods*, meaning the model has already been trained, and pruning is applied afterwards. This allows us to preserve the original training process while optimizing the model for ZKP generation. Model pruning is performed offline before zkSNARK proving, independently of the ZK circuit inputs. By reusing a single pruned model for multiple proofs, per-run CPU benchmarks are not a bottleneck, so we exclude their costs. We leverage two state-of-the-art pruning methods, **RD_PRUNE** [77] and **CAP** [46]. These methods offer efficient execution even on CPUs, enabling the sparsification of large models in a matter of seconds, without significant loss of model accuracy.

Pruning techniques typically compute an importance score for each weight and then remove the least important weights based on this score. The pruning objective can be summarized as:

$$\min_W \sum_{i=1}^L \mathcal{L}(f(W^{(i)})) + \lambda \|W^{(i)}\|_0 \quad (6)$$

where $W^{(i)}$ represents the weights of layer i , \mathcal{L} is the loss function, and $\|W^{(i)}\|_0$ imposes a sparsity constraint by penalizing the number of non-zero weights. The loss function $\mathcal{L}(f(W^{(i)}))$ measures

the error or deviation of the model’s predictions $f(W^{(i)})$ from the true outputs, quantifying the model’s performance while excluding the pruned weights. By minimizing this loss, the model seeks to retain performance even after pruning. Several metrics have been explored to measure weight importance in prior works. Weight magnitude is one of the most commonly used criteria [27, 86] where in each iteration, weights with the lowest magnitude are pruned. Recent approaches have utilized second-order information about the loss to determine the importance of each weight, resulting in better preservation of the model’s performance especially in transformer-based architectures [17, 46, 72, 80].

Building on these ideas, RD_PRUNE and CAP introduce further refinements by treating the model as a sequence of blocks, thereby making the pruning process layer-wise and computationally efficient. CAP, in particular, uses an approximation of second-order information as its pruning metric, allowing it to maintain model accuracy more effectively. In contrast, RD_PRUNE leverages output distortion as its pruning metric, focusing on the reconstruction of the network’s output behaviour rather than relying on the weight magnitudes or approximated second-order information.

The first method introduces the concept of *distortion*, which quantifies the impact of pruning on the model’s output. The pruning problem is framed as an optimization problem, where the objective is to minimize the overall distortion across layers. This can be expressed as:

$$\min \sum_{i=1}^L \left\| f(x; W^{(i)}) - f(x; \tilde{W}^{(i)}) \right\|^2 \quad \text{s.t.} \quad \sum_{i=1}^L \frac{\|\tilde{W}^{(i)}\|_0}{\|W^{(i)}\|_0} \leq R \quad (7)$$

where \tilde{W} and $\left\| f(x; W^{(i)}) - f(x; \tilde{W}^{(i)}) \right\|^2$ represent the pruned model weights and the distortion at layer i after pruning respectively. R is the sparsity ratio of the entire network. RD_PRUNE uses *dynamic programming* to solve this optimization problem, determining the optimal pruning for each layer. The method follows a *one-shot pruning* setting to fine-tune the pruned model by small calibration samples $\mathcal{D}_{\text{calib}}$ (e.g., 1024 samples) to retain the model performance.

CAP (Correlation-Aware Pruning). The second method takes a more sophisticated pruning criterion based on *weight correlations*. This method uses a *block-wise Hessian update*, which makes it computationally efficient, as it approximates the Hessian matrix in a block-diagonal form, significantly reducing the computational complexity of the pruning process. CAP can prune models in a *zero-shot setting*, meaning no additional training data is required after pruning, further enhancing its efficiency, especially in large models like ViT. In particular, CAP minimizes the following objective:

$$\min_W \frac{1}{2} \sum_{i=1}^L \left(W_i^T H_i^{-1} W_i \right)$$

where H_i is the Hessian matrix for block i . By using a block-diagonal approximation of the Hessian, CAP iteratively updates the pruning scores for each block, ensuring fast computation while maintaining high accuracy. Within each block, CAP calculates the importance of the i -th weight as:

$$\rho_i = \frac{w_i^2}{2 [H_L^{-1}(w, \mathcal{D}_{\text{calib}})]_{ii}} \quad (8)$$

Weights with the smallest importance scores ρ_i are pruned iteratively. To mitigate the impact of pruning, the optimal update for the remaining weights in the block is computed using:

$$\delta w = - \frac{w_i}{[H_L^{-1}(w)]_{ii}} H_L^{-1}(w, \mathcal{D}_{\text{calib}}) e_i$$

This iterative pruning process repeats across all blocks, progressively pruning the least important weights. By using this block-wise approach, CAP effectively balances computational efficiency with model accuracy, making it well-suited for large-scale model pruning without requiring retraining epochs or additional data.

5 TELEPORTATION

Deep neural networks (DNNs), especially architectures like modern ViTs that utilize Layer Normalization, often produce outlier values in the inputs to activation functions due to the normalization process [3, 16, 56]. In most of the recently utilized ZK-proving systems for DNNs, constraints on activation functions are typically represented using lookup tables [36, 67, 82, 87]. Outlier values broaden the input range of activation functions, necessitating larger lookup tables. This expansion increases proving time, memory usage, and proof size as shown in experiment section. Our methodology aims to mitigate these outlier values and optimize the activation input range by leveraging neural teleportation, thus enhancing the efficiency of the verification process. Neural teleportation is a mathematical framework that transforms network parameters while preserving the network's function, thereby reducing the range of activation inputs without affecting the network's outputs.

Neural teleportation assigns positive scaling factors, or **Change of Basis (CoB)** scalars $\tau_j^{(i)} > 0$, to each layer i and neuron j in that layer, where $i \in \{1, \dots, L\}$ and L is the total number of layers. The weights $w_{j,k}^{(i)}$ connecting neuron j in layer i to neuron k in layer $i+1$ are transformed as [2, 85]:

$$v_{j,k}^{(i)} = \left(\frac{\tau_k^{(i+1)}}{\tau_j^{(i)}} \right) w_{j,k}^{(i)} \quad (9)$$

The activation functions $f_j^{(i)}$ at each neuron are adjusted to:

$$g_j^{(i)}(x) = \tau_j^{(i)} f_j^{(i)} \left(\frac{x}{\tau_j^{(i)}} \right) \quad (10)$$

An activation function f is **positive scale-invariant** if:

$$f(cx) = cf(x), \quad \forall x \in \mathbb{R}, \quad \forall c > 0 \quad (11)$$

This property ensures that scaling the input by a positive factor c scales the output by the same factor, which is critical for maintaining consistency in neural transformations. A common example is the ReLU function, $f(x) = \max(0, x)$, widely used due to its simplicity and effectiveness in DNNs. For such functions f , any transformed activation simplifies back to the original, preserving the output of the network:

$$g_j^{(i)}(x) = \tau_j^{(i)} f_j^{(i)} \left(\frac{x}{\tau_j^{(i)}} \right) = f_j^{(i)}(x) \quad (12)$$

Ensuring the network's output stays the same after transformation. Our goal is to optimize the CoB scalars $\{\tau_j^{(i)}\}$ to minimize the range of scaled pre-activation inputs $\frac{z_j^{(i)}}{\tau_j^{(i)}}$ across all activation functions in different layers. Specifically, we aim to minimize the difference between the maximum and minimum scaled pre-activation inputs within each layer (or block) i :

$$\min_{\tau > 0} \sum_{i=1}^L \left(\max_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) - \min_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) \right) \quad (13)$$

which $z_j^{(i)}$ representing input of j -th neuron in the layer i .

The CoB constraints, derived from Theorem 2.1 in previous work [2], are applied as follows:

- (1) **Input, Output, and Bias Layers:** CoB scalars $\{\tau_j^{(i)}\}$ are assigned if layer i corresponds to input, output, or bias layers.
- (2) **Residual Connections:** For layers i and j connected by residual paths, $\{\tau_j^{(i)}\}$ are assigned.
- (3) **Convolutional Layers:** Neurons within the same feature map share the same τ_i .
- (4) **Batch Normalization Layers:** CoB scalars are applied to parameters γ and β , but not to the running mean and variance.

To achieve a reduced range of activation function inputs, We define the objective function $l(\tau)$ as:

$$l(\tau) = \sum_{i=1}^L \left(\max_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) - \min_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) \right) \quad (14)$$

Then our optimization problem summarized as:

$$\min_{\tau > 0} l(\tau) \quad (15)$$

Optimizing the CoB scalars reduces the range of activation inputs within each layer, thereby minimizing the impact of outlier values and reducing the ZK lookup table overhead. Despite the significant reduction in parameters $\tau_j^{(i)}$ to optimize compared to model weights (e.g., 10K vs. 5 million in Tiny ViT), the optimization process in Equation (13) presents two key challenges:

- **Non-Differentiable Objective:** The max and min functions introduce discontinuities in the gradient with respect to τ . As a result, traditional gradient-based optimization methods are ineffective at such points, necessitating alternative approaches.
- **Interdependent Scaling Factors:** The interconnected nature of neural networks means that adjusting one scaling factor $\tau_j^{(i)}$ influences other activations. This interdependence causes the indices j for \max_j and \min_j to vary dynamically, depending on τ , the input data, and the network

weights. Such variations further complicate optimization, as the maxima and minima are not fixed during the process.

To overcome these challenges, we use Coordinate Gradient Estimation (CGE) [12], a zero-order optimization method. Zero-order methods approximate gradients using function evaluations, avoiding explicit gradient computations. The general form of CGE is:

$$\hat{\nabla}_{\tau} \ell(\tau) = \sum_{i=1}^d \left[\frac{\ell(\tau + \mu \mathbf{e}_i) - \ell(\tau)}{\mu} \mathbf{e}_i \right] \quad (16)$$

where $\mu > 0$ is a small perturbation scalar. \mathbf{e}_i is the i -th standard basis vector in \mathbb{R}^d . Applying this to our optimization problem, we approximate the gradient with respect to each $\tau_j^{(i)}$ as:

$$\hat{g}_j = \frac{\ell(\tau + \mu \mathbf{e}_j) - \ell(\tau)}{\mu} \quad (17)$$

Using this estimated gradient, we perform gradient descent updates on the CoB scalars:

$$\tau_j^{(i)} \leftarrow \tau_j^{(i)} - \eta \hat{g}_j \quad (18)$$

where $\eta > 0$ is the learning rate, and \hat{g}_j is the j -th component of $\hat{\nabla}_{\tau} \ell(\tau)$. Since $\tau_j^{(i)} > 0$, after each update, We project $\tau_j^{(i)}$ onto the positive orthant.

5.1 Advantages of Zero-Order Methods

Zero-order methods offer several advantages that make them particularly well-suited for the defined optimization task:

- **No Need for Backpropagation:** Zero-order methods do not require gradient computations via backpropagation, which can be memory and computationally intensive for large models.
- **Handling Non-Differentiable Functions:** Zero-order methods are suitable for optimizing non-differentiable functions, making them appropriate for our problem.
- **Parallelization:** The gradient estimation procedure can be parallelized since each coordinate perturbation is independent. This enables teleportation optimization to be completed in a reasonable time frame compared to the proof generation, as explained in the experimental section.

5.2 Extension to Non-Scale-Invariant Activation

In Equation (12), we assumed that the activation functions used in the DNN are positive scale-invariant. However, many modern DNNs employ activation functions that are not strictly scale-invariant, such as the Gaussian Error Linear Unit (GELU). The GELU activation function is defined as:

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2} \left(1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right) \quad (19)$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, and erf is the error function.

Assuming that such non-scale-invariant activation functions are scale-invariant can lead to changes in the function that the neural network represents, thus altering its function. To address this challenge, we consider two important aspects:

1. **Approximate Scale-Invariance at Extremes:** Non-scale-invariant activation functions like GELU become approximately scale-invariant for large positive or negative inputs. Specifically, as $x \rightarrow \infty$ or $x \rightarrow -\infty$, the GELU function behaves similarly to the ReLU or linear functions, respectively, which are scale-invariant.

2. **Minimizing Approximation Error:** To reduce the effect of the approximation error introduced by assuming scale invariance, we add a reconstruction term to the optimization problem in Equation (13). This term penalizes the difference between the outputs of the original model and the teleported model, ensuring that the teleported model remains close in function space to the original.

Incorporating these considerations, we update our optimization problem to include the reconstruction term:

$$\min_{\tau > 0} \sum_{i=1}^L \left(\max_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) - \min_j \left(\frac{z_j^{(i)}}{\tau_j^{(i)}} \right) \right) + \lambda |f(\theta, \mathbf{1}) - f(\theta, \tau)|^2 \quad (20)$$

- $f(\theta, \mathbf{1})$ represents the original neural network function evaluated with scaling factors $\tau = 1$ (i.e., no scaling applied).
- $f(\theta, \tau)$ represents the teleported neural network function evaluated with the optimized scaling factors τ .
- $\lambda > 0$ is a regularization hyperparameter that balances the trade-off between minimizing the activation input range and preserving the original network function.

The reconstruction term $|f(\theta, \mathbf{1}) - f(\theta, \tau)|^2$ quantifies the difference between the outputs of the original and teleported models, thereby controlling the approximation error due to the non-scale-invariant activation functions.

The algorithm outlined in Algorithm 2 guides the optimization process towards convergence on CoB scalars, effectively reducing the activation input ranges. This reduces the impact of large lookup tables, enhancing resource efficiency as shown in the experiments.

6 EXPERIMENTS

6.1 Implementation

To demonstrate the efficiency of TeleSparse in reducing resource consumption for ZKP generation, we conducted experiments on both dense and sparse versions of DNN models. Our focus is on assessing the resource usage of the prover and verifier under each configuration, specifically observing how the proposed post-processing of TeleSparse impacts computation time and memory demands. To measure that, we employed the EZKL toolkit [87], a state-of-the-art tool for generating ZK-SNARK proofs for DNN models, which is based on Halo2. This toolkit utilizes fixed-point quantization, akin to previous methods exploring ZK-SNARKs of DNNs [36].

6.2 Relevance of Selected state-of-the-art

In order to benchmark TeleSparse against previous State-of-the-art works considering verifiable inference by ZK-SNARK, we selected the zero-knowledge proving system outlined in [36] as a reference. This system, based on Halo2, supports a more diverse range of modern DNN architectures rather than previous works, which aligns with our goal of evaluating scalability across various architectures. We chose not to include CNN-specific methods such as [50, 55] in our evaluation, as they do not generalize to the types of DNNs

Algorithm 2: Optimization teleportation parameters via CGE

Input: Initial scaling factors $\tau = \{\tau_j^{(i)}\}$, learning rate η ,
 perturbation size μ , max iterations N
Output: Optimized scaling factors τ^*
for $n = 1$ **to** N **do**
 Evaluate objective function $l(\tau)$ using Eq. (20).;
 foreach coordinate $\tau_j^{(i)}$ (*parallelizable*) **do**
 Compute gradient estimate:
 Evaluate $l(\tau + \mu \mathbf{e}_j)$

$$\hat{g}_j = \frac{l(\tau + \mu \mathbf{e}_j) - l(\tau)}{\mu}$$

 Update scaling factors:

$$\tau_j^{(i)} \leftarrow \tau_j^{(i)} - \eta \hat{g}_j$$

 Apply constraints:

$$\tau_j^{(i)} \leftarrow \max(\tau_j^{(i)}, \epsilon)$$

 Enforce CoB constraints (see Section 3).;
 if Convergence criteria met (e.g., $\|\hat{\nabla}_\tau l(\tau)\| < \delta$) **then**
 break;
return Optimized scaling factors $\tau^* = \tau$;

considered in our study. Since our experiments were conducted in a CPU-only environment, we did not include GPU-dependent frameworks, such as the system proposed in [67], as they would not provide a directly comparable assessment of CPU performance. Additionally, as discussed in Section 1, TeleSparse is designed to be complementary to previously proposed ZKP systems that are tailored ZKP for specific architectures. TeleSparse can be applied on top of existing specialized frameworks to achieve even more optimized performance.

6.3 Results

For consistency, all experiments were conducted on a virtual machine running Linux (kernel 4.18.0), equipped with 32 virtual CPU cores, 1 TB of memory, and 2 GB of swap space. The fixed-point quantization scale was set at 2^{12} to balance performance and memory usage. Increasing this scale would enhance model accuracy but also increase memory usage during proof generation due to the increasing the required number of field values and larger lookup tables required for non-linear functions in the proving system [36, 87].

To evaluate the effectiveness of TeleSparse, we conducted experiments on three distinct setups: MobileNetV1 [34] on the CIFAR-10 dataset [42] and ResNet-20 [32] on the CIFAR-100 dataset, as detailed in Table 1, and the ViT model [18] on the large-scale ImageNet dataset [14], as presented in Table 2. This selection includes varied architectures and datasets, providing a comprehensive evaluation of our method’s adaptability and efficiency.

As shown in Table 1, we present the proving time, peak memory usage of the prover, and proof size for both MobileNetv1 on CIFAR-10 and ResNet-20 on CIFAR-100, showcasing the effectiveness of the proposed method. With a 50% sparsity ratio and applying the

optimized teleportation, our approach balances model performance (detailed in Table 3) and the reduction of resource consumption. Importantly, the sparsity consideration is unique to TeleSparse as prior works like [36] are not designed for sparse models and therefore do not benefit from the associated resource reductions.

For MobileNetv1, TeleSparse reduces average memory usage by 59.2% and proving time by 54.0% compared to EZKL without the proposed post-processing. The proof size and verification time are reduced by 30.0% and 22.9%, demonstrating a comprehensive improvement in computational and bandwidth usage of the verifier. For ResNet-20, memory usage decreases by 66.8%, proving time by 45.6%, proof size by 30.6%, and verification time by 19.4%. The improvements are due to sparsification (Section 4) and teleportation (Section 5), with their individual effects analyzed in the ablation study in Table 4.

In contrast to prior work focusing on small datasets and simplified model architectures, our experiments utilize the large-scale ImageNet dataset [14] and the ViT transformer-based [18] architecture for a thorough evaluation. Given that ViT’s computational demands are substantial—with approximately 31 times more FLOPs compared to Resnet-20—we employed circuit splitting technique [87] (detailed in F) to make proof generation practical, reducing memory overhead for the prover while maintaining model integrity. In terms of splitting the model, we split the model into $M = 24$ parts for the ViT model. This decision was motivated by two key factors: Firstly, splitting each layer into two components (self-attention and MLP) is advantageous as the intermediate model output is relatively smaller compared to other splitting options, thereby reducing the overhead of commitment to intermediate results. Secondly, the primary factor influencing proving resource usage is the logarithm of the number of rows in the halo2 proving table [65]. By splitting the model such that it features a roughly similar number of constraints across all parts, ensuring each component has an identical logarithmic number of rows. Consequently, memory usage is distributed more uniformly across circuit parts, thereby reducing the maximum memory size required during proof generation. Moreover, if proof generation exhibits similar time across circuit components, parallelization becomes more efficient, and the time required is equivalent to generating only one proof. Although the experimental results presented do not include parallelization, it’s worth noting that parallel proof generation could be achieved because each segment operates with distinct inputs and independently.

Table 2 highlights the effectiveness of TeleSparse in reducing proving time and memory usage on splitted circuits of the ViT model. Each column shows the corresponding metric averaged across M parts of the model. TeleSparse effectively reduces memory usage in transformer architectures too, achieving a notable 45.08% reduction even when applied to model splitting. However, proof generation for ViT demands 6 times more memory than MobileNet-v1, likely due to ViT’s computational intensity, with approximately 5.5 GFLOPs, making it unsuitable for resource-constrained devices. On the positive side, verification time remains efficient, increasing only 1.44 times compared to MobileNet-v1, highlighting the scalability of the verification process despite the proving overhead.

Existing approaches [21, 75] are not directly applicable with Transformers and modern CNNs like MobileNet. [75] performed less favorably compared to ZKML [36], as shown in Table 2 of the

Model	Dataset	Framework	Avg Memory Usage (GB) \pm Std	Avg Proving Time (s) \pm Std	Verification Time (ms)	Proof Size (KB)
MobileNetv1	CIFAR-10	ZKML (tensorflow)	148.3 \pm 10.2	1439	23.8	106
		EZKL (pytorch)	139.0 \pm 14.2	779	3.5	100
		TeleSparse (sparsity=50%)	56.7 \pm 3.0	358	2.7	70
		Reduction (%)	59.2%	54.0%	22.9%	30.0%
Resnet-20	CIFAR-100	ZKML (tensorflow)	128.0 \pm 7.5	1055	20.1	89
		EZKL (pytorch)	120.2 \pm 5.5	564	3.1	85
		TeleSparse (sparsity=50%)	39.8 \pm 3.1	307	2.5	59
		Reduction (%)	66.8%	45.6%	19.4%	30.6%

Table 1: Comparison of various frameworks across models and datasets, with reduction percentage achieved by TeleSparse compared to prior work. The standard deviation is computed by repeating the experiment five times.

Framework	Mem. Usage (GB)	Proving (s)	Verification (ms)	Proof Size (KB)
EZKL (pytorch)	650	1087	4.5	1134
TeleSparse (50%)	357	869	4.1	1076
Reduction (%)	45.08%	20.05%	8.89%	5.12%

Table 2: Comparison of frameworks for Tiny-ViT on ImageNet. Each column represents the average across $M = 24$ model splits, the reduction is calculated relative to EZKL.

ZKML paper, while our results compare favorably. To have a comparison with TeleSparse, we tested [75] on LeNet-5 using similar hardware. Compared to [75] (127.2s) and [21] (11.6s), TeleSparse achieved $7.4s \pm 0.07$, demonstrating superior efficiency.

The quantization applied to weights and activation values within the ZK proving system, combined with the sparsification techniques we employed, would decrease model accuracy within the circuit. To evaluate that, Section 6.4 examines the trade-offs in accuracy. Additionally, Section 6.5 analyzes the individual contributions of each component of TeleSparse to the resource reductions presented in the tables. Notably, the resource usage for teleportation, included in Tables 1 and 2, remains minimal. Even for ViT, teleportation requires less than 2% of the memory needed for proof generation, and the total teleportation time for all MLP parts (to which teleportation exclusively applies) is 50 seconds, constituting less than 0.1% of the overall proof generation time for the model.

6.4 Accuracy

To assess the impact of quantization and post-processing introduced by TeleSparse on model performance over the ZKP system, we evaluate both the original floating-point model accuracy and its quantized model (and post-processed) accuracy provided by the ZKP framework in Table 3.

By selecting to be 2^{12} , the results show that the model retains strong performance even with pruning. The difference between the full-precision model and the ZK model is approximately 0.8% in accuracy in the dense model and 0.9% in the sparse model, showing that the ZK framework could maintain accuracy close to the full-precision model. Although introducing 50% sparsity reduces the ZK accuracy by about 1.03%, it enables significant resource savings,

Model	Dataset	Sparsity	Full Precision Model Accuracy (%)	ZK Accuracy (%)
ResNet-20	CIFAR-100	0%	68.7	67.9
		50%	68.1	67.2
		Reduction (%)	0.87%	1.03%
Tiny-ViT	ImageNet	0%	72.2	71.4
		50%	71.1	70.7
		Reduction (%)	1.1%	0.7%

Table 3: Impact of sparsification and ZK circuit quantization on accuracy.

such as a 66.8% reduction in memory usage as illustrated in Table 1. This balance between minor accuracy loss and substantial resource reduction highlights the effectiveness of sparsity and teleportation in optimizing models for both efficiency and verifiability.

6.5 Ablation Study

To mitigate the computational overhead of ZK proof generation, particularly for the prover, we incorporated sparsification and teleportation into our design. To evaluate the individual contributions of each module, we present the ablation study results in Table 4.

The initial assumption may be that the main overhead in ZKP generation stems from either the large number of constraints or the extensive range of lookup tables. However, the results in Table 4 show that combining teleportation and sparsification yields the most substantial improvements. In the ResNet-20, the combination reduces memory usage by 66.9% and proving time by 45.6%. Each post-processing method alone has a smaller impact, highlighting the complementary benefits of combining these techniques.

Applying only sparsification reduces memory usage and proving time, particularly in ResNet-20, with reductions of 49.5% and 34.8%, respectively, and similar benefits for MobileNetv1. Teleportation, while providing moderate improvements, is more effective for models like ResNet-20 with more lookup arguments, reducing its memory usage by 38.9% and proving time by 23.8%. However, its impact on MobileNetv1 is negligible, likely because the primary bottleneck in that specific model lies in the number of constraints rather than lookup overhead. Combining teleportation with sparsification significantly outperforms sparsification alone, further reducing

Model	Dataset	Post-Processing (Method)	Memory Usage (GB)	Proving Time (s)	Verification Time (ms)
ResNet-20	CIFAR-100	No Post-Processing	120.2	564	3.1
		Teleportation	73.4 (38.9%)	430 (23.8%)	2.9 (6.5%)
		Sparsification 50%	60.7 (49.5%)	368 (34.8%)	2.5 (19.4%)
		Both Sparse and Teleported	39.8 (66.9%)	307 (45.6%)	2.5 (19.4%)
MobileNetv1	CIFAR-10	No Post-Processing	139.0	779	3.5
		Teleportation	138.3 (0.5%)	757 (2.8%)	3.4 (2.9%)
		Sparsification 50%	85.3 (38.6%)	473 (39.3%)	2.8 (20.0%)
		Both Sparse and Teleported	56.7 (59.2%)	358 (54.0%)	2.7 (22.9%)
Tiny-ViT	ImageNet	No Post-Processing	1002.2	541	4.2
		Teleportation	961.5 (4.1%)	421 (22.2%)	4.2 (0.0%)
		Sparsification 50%	838.3 (16.4%)	310 (42.7%)	3.8 (9.5%)
		Both Sparse and Teleported	814.5 (18.7%)	256 (52.7%)	3.7 (11.9%)

Table 4: Comparison between the effectiveness of each post-process on the different computation resources and both the prover and verifier. The reduction percentage is computed relative to the baseline model with no post-processing. The metrics for the Tiny-ViT model correspond only to the MLP parts, as teleportation affects only the activation functions in these parts.

memory usage by up to 34.4% and proving time by 24.3% compared to sparsification alone. Verification time, being less computationally demanding sees smaller reductions across configurations. Nevertheless, The combined approach still manages modest improvements, such as 19.4% for ResNet-20 and 22.9% for MobileNetv1. To demonstrate the effectiveness of the optimized teleportation, we show the distribution difference between the teleported and original model activation ranges in Appendix G.

6.6 Granular Sparsity

To analyze the impact of the sparsity ratio on prover computations, we conduct experiments varying the sparsity ratio from 0% to 75% on the ResNet model, as shown in Figure 4.

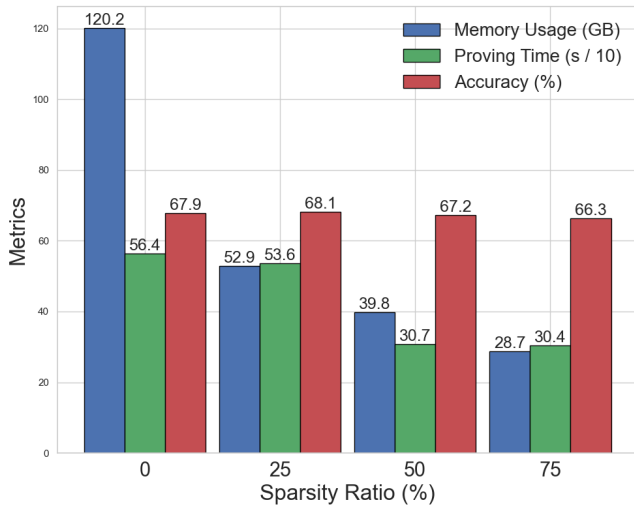


Figure 4: Effect of sparsity ratio on memory usage, proving time, and top-1 accuracy (equal to average accuracy due to balanced classes) for ResNet20 on CIFAR-100.

Table 2 demonstrates the impact of sparsity on memory usage, proving time, and accuracy for ResNet20 on CIFAR-100. Sparsification leads to a significant reduction in memory usage, with a decrease from 120.2 GB at 0% sparsity to 28.7 GB at 75%. Proving time also reduces, dropping from 564 seconds to 304 seconds, though the reduction is less substantial compared to memory usage. Interestingly, at 25% sparsity, the top-1 accuracy improves to 68.1% from 67.9% in the dense model, likely due to reduced quantization error as weights and activations with larger magnitudes, which are more robust to quantization, are preserved [39, 69]. However, accuracy decreases at higher sparsity levels, reaching 66.3% at 75%. These results highlight that sparsification not only optimizes resource efficiency but can also enhance accuracy under moderate sparsity levels, though trade-offs emerge as sparsity increases.

7 PRIVACY ANALYSIS

To conclude our analysis, we prove security of our approach against a malicious verifier that observes the model inference outputs. Our method for proving security is based on the standard secure computation framework of the real-/ideal-world paradigm. Specifically, we construct an indistinguishable (polynomial-time) simulation of the real protocol, given access only to an ideal functionality F that on model input X , computes the corresponding inference output y . Our proof requires that the simulator has access to the sparsity ratio of model weights, which can be observed via the sparsification process in the verification key, and is therefore a form of leakage. In Section 7.3, we examine the possibility of removing this leakage through additional privacy measures that could form the basis of future work. In the following, we represent the original model weights before pruning (sparsification) as W^t and the model weights after applying the pruning algorithm as W_{sparse}^t .

7.1 Verifier View

The verifier has access to the neural network’s input X , output y , ZK circuit embedded in the verification key (containing commitments

to fixed columns Q in Halo2), and the proof π (with commitments to advice columns A). However, the actual weight values W_{sparse}^t remain fully hidden, regardless of whether the model is sparse or dense. Moreover, each fixed column in Halo2 mixes weights, selectors, and other constants, concealing individual weight locations. Although Figure 3 illustrates the weights in an ordered manner for clarity, in practice, their positions are neither predefined nor known to the verifier.

Impact of sparsification on verification key. As part of the sparsification process, certain constraints are removed to reduce proving costs and improve efficiency. This affects the circuit (verification key) and thus the generated proof, as shown in Figure 3, where the sparse case (B) has fewer constraints than the dense case (A). Since the verification key is visible to the verifier, they observe a commitment to Q_{F_0} in the proof and commitments to columns A_0 , A_1 , A_2 in the verification key, which shrink with sparsification. An adversarial verifier can distinguish between the verification keys of the dense and sparsified models by comparing their sizes, potentially estimating the sparsity ratio. Therefore, we must provide the sparsity ratio (R_I) as known leakage to the simulator algorithm when eventually constructing our proof of security. Potential measures for weakening this assumption are covered in Section 7.3.

Note that reducing activation ranges via teleportation (Section 5) does not provide additional leakage. Teleportation only restricts the input domain of neurons without altering the network function, thus not revealing additional information about the model weights.

7.2 Proof of Privacy

We now proceed to prove the privacy of the construction against a malicious verifier, who is trying to learn the values of the model weights W^t . Our proof follows as a consequence of the security properties of the underlying ZK proof system.

Formal privacy guarantee. The main privacy guarantee of our system is given in Theorem 7.1; the proof follows in Appendix H.

THEOREM 7.1. *For any witness W_{sparse}^t produced by the TeleSparse algorithm with sparsity ratio R_I and for any given input X , there exists a simulator \mathcal{S} which, given the sparsity ratio R_I , the input X , and access to the ideal functionality F (without knowledge of the original witness W_{sparse}^t), constructs a simulated view view' that is indistinguishable from the real protocol view view . In particular, the simulated view view' is computationally indistinguishable from the view produced during an execution of the real protocol with witness W_{sparse}^t .*

7.3 Preventing Sparsification Ratio Leakage

As previously discussed in Section 7, privacy loss occurs if the attacker accesses the original model’s architecture and the proving system fails to protect the number of removed constraints corresponding to the pruned weights (sparsity ratio). To mitigate this privacy leakage, we propose adding dummy constraints to the circuit. The idea is to provide a noisy approximation of the number of constraints in the sparsified model. Such constraints would have no impact on the eventual computation, and thus the accuracy of the model inference procedure would go unharmed.

On the other hand, the drawback is that the proof overhead increases proportionally with the number of added dummy constraints, and thus the proving and verification time would be computationally more expensive to execute. Furthermore, analysis of the optimal number of dummy constraints to add while still maintaining both high performance and reducing privacy leakage would require careful consideration of the noise introduced. This would likely require incorporating some notion of differential privacy into the security model, and would also require a method for allowing the PPT simulator to sample from the noisy distribution of sparsification ratios when constructing the security proof.

While the current privacy analysis provides strong justification — and that knowing the sparsification ratio is highly unlikely to impact privacy — we believe that investigating these directions further would be valuable future work. In particular, such investigation would explore the extent to which the approximation of the sparsity ratio is feasible, and to quantify how beneficial the proposed mitigation is in maintaining high performance.

8 CONCLUSION AND FUTURE WORK

In this paper, we make substantial progress toward reducing the computational overhead of applying ZK-SNARKs to large-scale neural networks. By focusing on post-processing techniques, we introduce methods that improve ZK compatibility without modifying model architecture or training pipelines. Through neural network sparsification and an innovative adaptation of neural teleportation, TeleSparse reduces both the number of circuit constraints and the size of lookup tables for non-linear functions. The method reduces prover memory usage by 67% and proof generation time by 54%, with minimal accuracy loss, making ZK-SNARK verification more practical for modern deep learning models. We conducted extensive experiments demonstrating the applicability of TeleSparse across various datasets and model architecture, including CNNs and transformers, to validate effectiveness in diverse settings.

Future research could extend our sparsification and activation optimization techniques for compatibility with other ZK proving systems, beyond Halo2 [82], thereby expanding their applicability across a broader range of ZK systems. Furthermore, although our experiments involve splitting the model and generating ZK-SNARK proofs for each part separately, we focus on selecting the split point based on the structure of transformers. However, determining the optimal split point is essential for minimizing the computational overhead of the prover. Additionally, exploring nested proof aggregation could significantly enhance scalability, especially in resource-constrained environments like edge devices. Other paths forward include exploring methods to quantify DNN operation costs in ZK systems and better understand the interplay between sparsity, memory usage, and proof generation time for various DNN operations. This would enable more targeted sparsification, addressing the operations with the highest computational overhead in ZKP systems. Finally, while we discuss the privacy implications of sparsification (Section 7), further research into the detected leakage risk and mitigation strategies in Section 7.3 is needed.

9 ACKNOWLEDGMENTS

We wish to acknowledge the thorough and useful feedback from anonymous reviewers and our shepherd. The research in this paper was supported by the UKRI Open Plus Fellowship (EP/W005271/1 Securing the Next Billion Consumer Devices on the Edge) and EU CHIST-ERA GNNs for Network Security and Privacy (GRAPHS4SEC) projects. Alex Davidson’s work was supported by FCT through project “ParSec: Astronomically Improving Parliamentary Cybersecurity through Collective Authorization”, ref. 2024.07643.IACDC, DOI 10.54499/2024.07643.IACDC, and the LASIGE Research Unit, ref. UID/00408/2025 – LASIGE.

REFERENCES

- [1] Marco Armenta and Pierre-Marc Jodoin. 2021. The representation theory of neural networks. *Mathematics* 9, 24 (2021), 3216.
- [2] Marco Armenta, Thierry Judge, Nathan Painchaud, Youssef Skandarani, Carl Lemaire, Gabriel Gibeau Sanchez, Philippe Spino, and Pierre-Marc Jodoin. 2023. Neural teleportation. *Mathematics* 11, 2 (2023), 480.
- [3] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456* (2024).
- [4] Vijay Badrinarayanan, Bamdev Mishra, and Roberto Cipolla. 2015. Symmetry-invariant optimization in deep networks. *arXiv preprint arXiv:1511.01754* (2015).
- [5] Guangji Bai, Yijiang Li, Chen Ling, Kibaek Kim, and Liang Zhao. 2024. Gradient-Free Adaptive Global Pruning for Pre-trained Language Models. *arXiv preprint arXiv:2402.17946* (2024).
- [6] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. 2023. Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1437–1451.
- [7] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*. 781–796.
- [8] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012*, Cambridge, MA, USA, January 8-10, 2012, Shafi Goldwasser (Ed.). ACM, 326–349. <https://doi.org/10.1145/2090236.2090263>
- [9] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2023. Quantizable transformers: Removing outliers by helping attention heads do nothing. *Advances in Neural Information Processing Systems* 36 (2023), 75067–75096.
- [10] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive* (2019).
- [11] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK compilers. In *Advances in Cryptology—EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I* 39. Springer, 677–706.
- [12] Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Jiancheng Liu, Konstantinos Parasyris, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. 2023. Deepzero: Scaling up zeroth-order optimization for deep model training. *arXiv preprint arXiv:2310.02025* (2023).
- [13] Ana-Maria Cretu, Daniel Jones, Yves-Alexandre de Montjoye, and Shruti Tople. 2024. Investigating the Effect of Misalignment on Membership Privacy in the White-box Setting. *Proceedings on Privacy Enhancing Technologies* 2024, 3 (July 2024), 407–430. <https://doi.org/10.56553/popets-2024-0085>
- [14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
- [15] Jacob Dexe, Ulrik Franke, and Alexander Rad. 2021. Transparency and insurance professionals: a study of Swedish insurance practice attitudes and future development. *The Geneva Papers on Risk and Insurance. Issues and Practice* 46, 4 (2021), 547.
- [16] Yifu Ding, Haotong Qin, Qinghua Yan, Zhenhua Chai, Junjie Liu, Xiaolin Wei, and Xianglong Liu. 2022. Towards accurate post-training quantization for vision transformer. In *Proceedings of the 30th ACM international conference on multimedia*. 5380–5388.
- [17] Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems* 30 (2017).
- [18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
- [19] Simon S Du, Wei Hu, and Jason D Lee. 2018. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. *Advances in neural information processing systems* 31 (2018).
- [20] Yongkai Fan, Kaile Ma, Linlin Zhang, Xia Lei, Guangquan Xu, and Gang Tan. 2024. ValidCNN: A large-scale CNN predictive integrity verification scheme based on zk-SNARK. *IEEE Transactions on Dependable and Secure Computing* (2024).
- [21] Yongkai Fan, Binyuan Xu, Linlin Zhang, Jinbao Song, Albert Zomaya, and Kuan-Ching Li. 2023. Validating the integrity of convolutional neural network predictions based on zero-knowledge proof. *Information Sciences* 625 (2023), 125–140.
- [22] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16091–16101.
- [23] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive* (2021).
- [24] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).
- [25] Olive Franzese, Ali Shahin Shamsabadi, and Hamed Haddadi. 2024. OATH: Efficient and Flexible Zero-Knowledge Proofs of End-to-End ML Fairness. *arXiv preprint arXiv:2410.02777* (2024).
- [26] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for ocumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).
- [27] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574* (2019).
- [28] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems* 30 (2017).
- [29] Grzegorz Gluch and Rüdiger Urbanke. 2021. Noether: The more things change, the more stay the same. *arXiv preprint arXiv:2104.05508* (2021).
- [30] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology—EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II* 35. Springer, 305–326.
- [31] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. 2024. Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning. In *33rd USENIX Security Symposium (USENIX Security 24)*. 3819–3836.
- [32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [33] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).
- [34] Andrew G Howard. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
- [35] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. 2020. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems* 33 (2020), 20744–20754.
- [36] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. Scaling up Trustless DNN Inference with Zero-Knowledge Proofs. *arXiv preprint arXiv:2210.08674* (2022).
- [37] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology—ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings* 16. Springer, 177–194.
- [38] Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.
- [39] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629* (2023).
- [40] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.
- [41] Ahmed E Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F Sayed, Elaine Shi, and Nikos Triandopoulos. 2014. {TRUESET}: Faster {Verifiable} Set Computations. In *23rd USENIX Security Symposium (USENIX Security 14)*. 765–780.
- [42] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [43] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.

- [44] Daniel Kunin, Javier Sagastuy-Brena, Surya Ganguli, Daniel LK Yamins, and Hidenori Tanaka. 2020. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics. *arXiv preprint arXiv:2012.04728* (2020).
- [45] Eldar Kurtić, Elias Frantar, and Dan Alistarh. 2024. ZipLM: Inference-Aware Structured Pruning of Language Models. *Advances in Neural Information Processing Systems* 36 (2024).
- [46] Denis Kuznedelev, Eldar Kurtić, Elias Frantar, and Dan Alistarh. 2024. CAP: Correlation-Aware Pruning for Highly-Accurate Sparse Vision Models. *Advances in Neural Information Processing Systems* 36 (2024).
- [47] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems* 35 (2022), 24101–24116.
- [48] Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. 2023. Dynamic Sparse Training with Structured Sparsity. *arXiv preprint arXiv:2305.02299* (2023).
- [49] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*. PMLR, 12403–12422.
- [50] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2024. vcnn: Verifiable convolutional neural network based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing* (2024).
- [51] Klas Leino and Matt Fredrikson. 2020. Stolen memories: Leveraging model memorization for calibrated {White-Box} membership inference. In *29th USENIX security symposium (USENIX Security 20)*, 1605–1622.
- [52] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. Zksql: Verifiable and efficient query evaluation with zero-knowledge proofs. *Proceedings of the VLDB Endowment* 16, 8 (2023).
- [53] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. 2023. Tiny machine learning: progress and futures [feature]. *IEEE Circuits and Systems Magazine* 23, 3 (2023), 8–34.
- [54] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. 2021. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*. PMLR, 6989–7000.
- [55] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2968–2985.
- [56] Yijiang Liu, Huanrui Yang, Zhen Dong, Kurt Keutzer, Li Du, and Shanghang Zhang. 2023. Noisyquant: Noisy bias-enhanced post-training activation quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20321–20330.
- [57] Qian Lou and Lei Jiang. 2021. HEMET: a homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International conference on machine learning*. PMLR, 7102–7110.
- [58] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. 2024. An Efficient and Extensible Zero-knowledge Proof Framework for Neural Networks. *Cryptology ePrint Archive* (2024).
- [59] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems* 36 (2023), 21702–21720.
- [60] Nahema Marchal, Rachel Xu, Rasmi Elasmr, Iason Gabriel, Beth Goldberg, and William Isaac. 2024. Generative AI Misuse: A Taxonomy of Tactics and Insights from Real-World Data. *arXiv preprint arXiv:2406.13843* (2024).
- [61] Silvio Micali. 2000. Computationally sound proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298.
- [62] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. 2017. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119* (2017).
- [63] Alec Radford. 2018. Improving language understanding by generative pre-training. (2018).
- [64] Md Aamir Raihan and Tor Aamodt. 2020. Sparse weight activation training. *Advances in Neural Information Processing Systems* 33 (2020), 15625–15638.
- [65] Tobin South, Alexander Camuto, Shrey Jain, Shayla Nguyen, Robert Mahari, Christian Paquin, Jason Morton, and Alex’sSandy’ Pentland. 2024. Verifiable evaluations of machine learning models using zkSNARKs. *arXiv preprint arXiv:2402.02675* (2024).
- [66] Wenting Zheng Srinivasan, PMRL Akshayaram, and Popa Raluca Ada. 2019. DELPHI: A cryptographic inference service for neural networks. In *Proc. 29th USENIX Secur. Symp.* 2505–2522.
- [67] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkLLM: Zero Knowledge Proofs for Large Language Models. *arXiv preprint arXiv:2404.16109* (2024).
- [68] Mingjie Sun, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. 2024. Massive Activations in Large Language Models. *arXiv preprint arXiv:2402.17762* (2024).
- [69] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).
- [70] Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems* 34 (2021), 24193–24205.
- [71] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
- [72] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. 2019. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International conference on machine learning*. PMLR, 6566–6575.
- [73] Xiuling Wang and Wendy Hui Wang. 2024. GCL-Leak: Link Membership Inference Attacks against Graph Contrastive Learning. *Proceedings on Privacy Enhancing Technologies* (2024).
- [74] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, 501–518.
- [75] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. 2023. pvcnn: Privacy-preserving and verifiable convolutional neural network testing. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2218–2233.
- [76] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.
- [77] Kaixin Xu, Zhe Wang, Xue Geng, Min Wu, Xiaoli Li, and Weisi Lin. 2023. Efficient joint optimization of layer-adaptive weight pruning in deep neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 17447–17457.
- [78] Songkai Xue, Mikhail Yurochkin, and Yuekai Sun. 2020. Auditing ml models for individual bias and unfairness. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4552–4562.
- [79] Lu Yu and Wei Xiang. 2023. X-pruner: explainable pruning for vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 24355–24363.
- [80] Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. 2022. Hessian-aware pruning and optimal neural implant. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3880–3891.
- [81] Syed Ali Raza Zaidi, Ali M Hayajneh, Maryam Hafeez, and Qasim Zeeshan Ahmed. 2022. Unlocking edge intelligence through tiny machine learning (TinyML). *IEEE Access* 10 (2022), 100867–100877.
- [82] Zcash. 2022. *The Halo2 Book*. <https://zcash.github.io/halo2/>
- [83] Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. 2023. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5052–5063.
- [84] Yuxin Zhang, Yiting Luo, Mingbao Lin, Yunshan Zhong, Jingjing Xie, Fei Chao, and Rongrong Ji. 2023. Bi-directional masks for efficient n: M sparse training. In *International Conference on Machine Learning*. PMLR, 41488–41497.
- [85] Bo Zhao, Nima Dehmamy, Robin Walters, and Rose Yu. 2022. Symmetry teleportation for accelerated optimization. *Advances in neural information processing systems* 35 (2022), 16679–16690.
- [86] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).
- [87] ZKnduit. 2023. *EZKL: Efficient Zero-Knowledge Proving Toolkit*. <https://github.com/zknduit/ezkl> Accessed: 2024-11-06.

A ZK PROPERTIES OF HALO2 IN THE PROPOSED SYSTEM

TeleSparse, based on the Halo2 protocol, inherits its security properties including:

- **Succinctness:** Since the generated proof is sent over the internet network, the succinctness of the generated proof is crucial to reduce the overhead of communication costs of verifiable DNN inference. The size of the Halo2 generated proof remains sub-linear relative to the complexity of f . Therefore, as the layers of the model grow, the size of the proof does not change drastically.
- **Knowledge Soundness:** A prover (computationally bounded) is unable to generate proofs for incorrect executions of f .

- **Completeness:** If the prover possesses a valid W and generates the proof based on that, the proof will be successfully verified by the verifier.
- **Zero-knowledge:** The generated proof π does not reveal any information about the private input (W) except that $y = f(X; W)$.

B APPLICATIONS

The threat model considered in this work is suitable for various real-world applications:

- **Verifiable Model Accuracy:** A model provider may want to demonstrate the accuracy of their model on a public dataset without revealing the model weights. The model provider can use ZK-SNARK to prove the accuracy of the model on the dataset, without disclosing any information about the model weights [36, 65]. This ensures transparency and trustworthiness, particularly in scenarios where model reliability is critical, such as insurance. For example, insurance companies could apply ZKPs to verify the accuracy of their models in determining premiums or assessing risks [15], while keeping proprietary data (used for training the model) and model confidential.
- **Model Predictions as a Service:** A model provider can offer predictions as a service, where users submit input data and receive the model's output along with a ZK-SNARK proof, demonstrating that the prediction was generated correctly using the committed model weights [36, 58]. This protects the provider's intellectual property while assuring users of the prediction's integrity.
- **Auditing ML Models:** ZK-SNARK can be employed to audit ML models for fairness and compliance with regulations [25]. An auditor can use ZK-SNARK to verify that a model does not discriminate against certain groups, without needing access to the model weights or the training data. This enables accountability and promotes trust in AI systems.
- **Proof of Ownership:** In cases of model theft or extraction, the true model owner can use ZK-SNARK to prove ownership by committing to the dataset used for training and demonstrating that their model was trained on that specific dataset. This deters theft and helps resolve ownership disputes.
- **Proof of Model Ownership in Generative AI:** In cases where generative AI models create content (e.g., images or text), the model provider can use ZK-SNARKs to verify that specific outputs are indeed generated by their model. By committing to the model's weights, the provider can produce a ZKP confirming that a given output, such as an image, was generated by their model. This proof of generation, without revealing model weight, reinforces ownership and helps discourage unauthorized use of generative AI models [60].

C NOTATION TABLE

For a comprehensive list of mathematical notations used throughout this paper, please refer to the notation table provided in Table 5.

D FORMAL HALO2 AND COMMITMENT SCHEMA PROPERTIES

Let κ be a security parameter, and let $\nu(\kappa)$ be a negligible function. The formal ZK-SNARK properties guaranteed by Halo2 are as follows:

Soundness: A proving system is **sound** if a cheating prover cannot convince the verifier of a false statement except with negligible probability.

Formally, for any efficient prover \mathcal{P}^* and any statement ϕ not in the language L , the probability that \mathcal{P}^* convinces the verifier \mathcal{V} to accept ϕ is negligible:

$$\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(\phi) = 1 \mid \phi \notin L] \leq \nu(\kappa) \quad (21)$$

Correctness: A proving system is **correct** if an honest prover can convince the verifier of a true statement. Formally, for any statement $\phi \in L$ and witness w , the verifier accepts with overwhelming probability:

$$\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(\phi, w) = 1] = 1 - \nu(\kappa) \quad (22)$$

Zero-Knowledge: A proving system is **zero-knowledge** if the verifier learns nothing beyond the validity of the statement. Formally, for every probabilistic polynomial-time verifier \mathcal{V}^* , there exists a simulator \mathcal{S} such that:

$$\text{View}(\mathcal{P}(w), \mathcal{V}^*(\phi)) \approx \mathcal{S}(\phi) \quad (23)$$

Polynomial commitment schemes, such as KZG commitments, are cryptographic primitives used to ensure the integrity and privacy of polynomial evaluations in ZKP systems. These schemes satisfy two crucial security properties: *binding* and *hiding*.

Binding Property: A commitment scheme is **binding** if it is computationally infeasible for the prover to open a commitment to two different values. Formally, let $\text{Comm} : \mathbb{F}^n \rightarrow C$ be a commitment function. The binding property ensures that for any $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^n$ with $\mathbf{v} \neq \mathbf{v}'$, it is computationally infeasible to find r, r' such that $\text{Comm}(\mathbf{v}; r) = \text{Comm}(\mathbf{v}'; r')$.

$$\forall \text{ PPT algorithms } \mathcal{A}, \quad \Pr \left[(c, \mathbf{v}, \mathbf{v}') \leftarrow \mathcal{A}() \mid \text{Comm}(\mathbf{v}; r) = c = \text{Comm}(\mathbf{v}'; r') \wedge \mathbf{v} \neq \mathbf{v}' \right] \leq \nu(\kappa), \quad (24)$$

Hiding Property: A commitment scheme is **hiding** if the committed value remains indistinguishable to any adversary without knowledge of the opening randomness. Formally, for any $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^n$ and independent randomness $r, r' \in \mathbb{R}$, the distributions of $\text{Comm}(\mathbf{v}; r)$ and $\text{Comm}(\mathbf{v}'; r')$ are computationally indistinguishable. Specifically, for any PPT adversary \mathcal{A} , it holds that:

$$\left| \Pr [\mathcal{A}(\text{Comm}(\mathbf{v}; r)) = 1] - \Pr [\mathcal{A}(\text{Comm}(\mathbf{v}'; r')) = 1] \right| \leq \nu(\kappa) \quad (25)$$

E FORMAL PROOF OF SPARSIFICATION SOUNDNESS

We aim to prove that excluding constraints for zero-valued entries does not allow a cheating prover to convince the verifier of a

false statement, i.e., that $O' \neq WI$. This proof relies on the formal properties of Halo2 and commitments, as detailed in Appendix D.

Define support for each i :

$$S_i = \{j \in \{1, \dots, d_2\} \mid W_{i,j} \neq 0\}$$

The constraint for each i should represent the following equation:

$$O_i = \sum_{j \in S_i} W_{i,j} I_j$$

Based on the circuit representation in equation 4, the rows which F_{0i} is zero will be discarded as shown in Figure 3 so the polynomials A_0, Q_{F_0}, A_1, A_2 in equation 5 only interpolate remained cells. Consequently the commitments is computed based on the new polynomial functions. In other words, in the verification key only the commitments to the non-zero weights exist.

To prove the soundness, we consider all possible cheating strategies by the prover to somehow change the output of the linear layer (O_i).

Step 1, Altering the Matrix W : Assume we construct an adversary \mathcal{A} against the binding property of the commitment scheme, utilizing an adversary \mathcal{B} that attempts to alter W while generating a valid proof (accepted by the verifier). Since W is stored in fixed column F_0 , and commitments $\text{Commit}(Q_{F_0}(X))$ are included in the verification key, any attempt to alter W by adversary \mathcal{B} would require finding a new matrix W' such that

$$\text{Commit}(W, r) = \text{Commit}(W', r').$$

If adversary \mathcal{B} could successfully alter W by finding W' and r' , adversary \mathcal{A} could utilize \mathcal{B} to break the binding property of the commitment scheme. This contradicts the assumption that the commitment scheme is secure under standard cryptographic assumptions. Therefore, \mathcal{B} cannot alter W .

Step 2, Altering the Witness Vector I : Given that W cannot be altered, the only remaining strategy for adversary \mathcal{B} is to modify the witness vector I to I' in a way that affects the output O . Again, we construct an adversary \mathcal{A} against the soundness of the copy constraints in Halo2. Adversary \mathcal{A} utilizes \mathcal{B} , which attempts to modify the witness vector I to I' .

However, the circuit copy constraints enforce consistency between layer inputs and outputs. These constraints ensure that the input I_j of the current layer l must exactly match the output O_j from the previous layer $l - 1$. If adversary \mathcal{B} attempts to modify the values of I corresponding to zero entries in the matrix W , this will not affect the output O , as those terms contribute nothing to the summation in Equation (3). Thus, the adversary cannot achieve an invalid output O'_i through these modifications. To have any effect, adversary \mathcal{B} attempt must involve changing values in I that correspond to the non-zero entries in W . Assume that adversary \mathcal{B} successfully modifies I_j to I'_j corresponding to non-zero entries (modifies a value I_j where $j \in S_i$) and then generates a valid proof based on this. Therefore, adversary \mathcal{A} could exploit adversary \mathcal{B} to construct a strategy to break the copy constraints of Halo2. This would lead to a contradiction, as the copy constraints of Halo2 are enforced by the soundness property of the ZK-SNARK system. Consequently, \mathcal{B} cannot successfully alter I without being detected by the verifier.

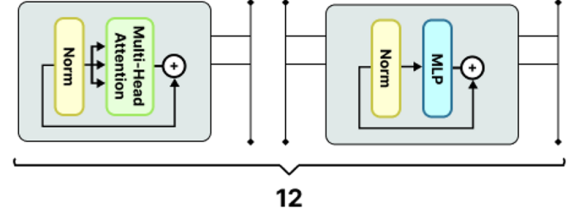


Figure 5: Transformer architecture contains two parts of attention and MLP. We split each part to generate proof in a separate ZK circuits.

Conclusion: If adversary \mathcal{B} cannot alter W due to the binding property of the commitment scheme, and cannot modify I without violating the copy constraints, then no cheating strategy remains. As a result, eliminating constraints for zero entries in fixed columns does not compromise the soundness of the proving system.

F PROOF SPLITTING

To ensure that the ZK-SNARK system remains scalable for use on devices with limited resources, we could leverage Halo2’s recursive proof composition capabilities, allowing complex proofs to be split and incrementally verified. The splitting not only reduces the load on constrained devices but also enhances the overall scalability of the ZK-SNARK verification process.

The total number of constraints regarding a model like a vision transformer is significantly huge. However, by splitting the layers of a DNN (e.g., ViT shown in Figure 5), each part could be proved separately. In particular, we could split a ViT into our desired M parts such that the output of the $(i-1)$ -th part is the input of the i -th part. It is easy to do that in most DNN models including ViT, which consists of 12 layers, and each layer has two main components: self-attention and MLP. For instance, given a sparse ViT, we split the model into $M = 24$ parts, which are non-overlapping. For each part, the corresponding proving key pk_i and the corresponding verification key vk_i are generated by halo2. Also, we should contain the commitment to the output of the $(i-1)$ -th part as the public input to the circuit corresponding to the i -th part. Proof generation for each partition can proceed either in parallel or sequentially, with specific advantages in each approach that impact the scalability of ZK-SNARKs in deep learning contexts. In a parallel setup, proof generation time remains fixed, regardless of the number of partitions M , as each segment can be processed concurrently without increasing overall latency. This parallel configuration is especially beneficial in distributed systems, where multiple nodes can simultaneously handle each partition’s proof, thus minimizing total proving time for large models. Conversely, in a sequential setup, memory requirements remain constant and independent of M , as each partition can be processed one at a time with minimal memory overhead.

This sequential approach is particularly advantageous for memory-constrained environments, where a single proof can be generated step-by-step through the partitions, thus avoiding the exponential memory growth that would typically accompany large models in ZK-SNARK-based proofs for deep networks. This flexibility in proof generation pathways ensures scalability and practicality of ZK-SNARKs for complex architectures like ViTs, where constraint management is critical. Detailed performance measurements of this partitioning approach, specifically on the Vision Transformer (ViT), are provided in the Table 2.

In the verification step, the verification of the i -th part, the verifier should check that the committed output of the $(i - 1)$ -th part is equal to the commitment of the input of the i -th part.

Building on this partitioned proof generation, Halo2 enables proof composition via Nested Amortization [10] and an Accumulation Scheme [82], allowing individual proofs from each partition to be combined into a single recursive proof with constant verification cost. This makes the proof size independent of the partition count. This means that only a single verification check is required, leveraging Halo2’s recursive ZK-SNARKs to verify all parts efficiently. In verification, a single recursive check confirms that the final proof upholds commitments across all partitions 1 to M , implicitly verifying alignment of inputs and outputs between partitions. Proof Composition in Halo2 is completely beneficial for DNNs by keeping the verification cost and proof size constant with respect to M . Proof aggregation is particularly advantageous for applications like on-chain verification, where constant proof size and verification cost significantly reduce computational overhead. However, our experiments focused on partitioned proof generation and generating proof for each part separately, excluding recursive composition, as it lies beyond the scope of this work.

G EFFECT OF TELEPORTATION ON ACTIVATION RANGE DISTRIBUTION

To demonstrate the effectiveness of the proposed optimized teleportation in reducing activation function range values, as discussed in Section 5 and the experiments in Table 4, we present the distribution of activation ranges for 300 CIFAR-100 samples on ResNet-20 in Figure 6. The original network’s activation range is long-tailed, with some outliers exceeding a range of 70. After applying teleportation, the distribution narrows considerably, with reduced variance without extreme values, indicating the success of the teleportation in minimizing the activation range. Specifically, the Mean Activation Loss is reduced from 27.39 (Original) to 16.98 (Teleported), showing a reduction of approximately 38.01%. Similarly, the Standard Deviation of Activation Loss decreases from 5.99 (Original) to 2.64 (Teleported), a reduction of 55.9%, indicating a more compact and consistent distribution after teleportation. As mentioned in Table 4, eliminating outlier values is crucial for reducing the prover’s resource overhead. Outliers can significantly increase the complexity of the lookup constraints in ZK-SNARK proof generation, leading to higher CPU and RAM costs. By minimizing these outliers, the activation range distribution becomes more compact, thus optimizing the prover’s resource utilization and improving the efficiency of the verification process.

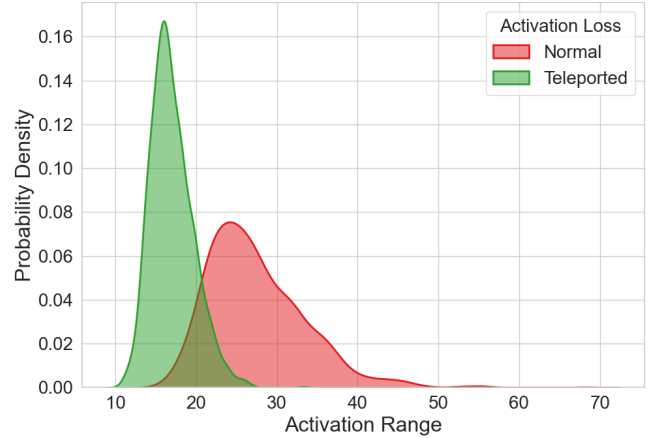


Figure 6: Activation range distribution of 300 CIFAR100 samples on ResNet-20: Red represents the original model’s activation range with outliers, and green shows the distribution after optimized teleportation

H PROOF OF THEOREM 7.1

To prove Theorem 7.1, we must show that the simulator \mathcal{S} can reconstruct all elements of view' such that the verifier cannot distinguish view' from view . Let κ be the security parameter, and let $\text{poly}(\kappa)$ and $\nu(\kappa)$ denote polynomial and negligible functions, respectively. As defined in Section 7.1, the simulated verifier view is: $\text{view}' = (\pi', X, y, vk', R_l)$, where the verification key vk' contains commitments to the fixed columns represented by Q (as shown in Equation (1)). Recall that the simulator has access to an ideal functionality F , which, given the input data X sent by the verifier, outputs the corresponding label y . The simulator does not have access to the true model weights W_{sparse}^t but can generate model weights, denoted as V_{sparse}^t , that produce y given X , with the weights randomly pruned according to the given layer-wise sparsity ratio R_l .

The simulator \mathcal{S} constructs view' as following steps:

- (1) **Verification Key Simulation:** The simulator first generates a verification key vk' by committing to model weights V_{sparse}^t following the defined sparsity ratio, without any knowledge of W_i^t . The simulator then sends vk' to the verifier. For simplicity, assume that the commitment of weights is performed layer-wise. Specifically, for layer l , the commitment to the polynomial $f_l(X)$, which interpolates the remaining weights $V_{\text{sparse}}^{t,(l)}$, is included in the verification key. The position of each weight is not included in the commitment as only the commitment is made to the polynomial $f_l(X)$ of degree $\|V_{\text{sparse}}^{t,(l)}\|_0 - 1$, derived via Lagrange interpolation of the remaining weights. Since the number of values in $V_{\text{sparse}}^{t,(l)}$ matches those in $W_{\text{sparse}}^{t,(l)}$, resulting in equal polynomial degrees, and given the hiding property of the commitment scheme (Equation (25)), the probability that the verifier distinguishes between the simulator and TeleSparse weight commitments are less than $\nu(\kappa)$,

demonstrating the computational indistinguishability of the verification keys corresponding to the two weights.

- (2) *Output Simulation*: Given the input data X provided by the adversarial verifier, the simulator uses the ideal functionality F to compute the corresponding output label y . Therefore, X and y are identical in $view$ and $view'$.
- (3) *Proof Simulation*: Using the simulated proof generation procedure (afforded by the zero-knowledge property of the NIZK proof system):

$$\text{SimProve}(\text{trap}, X, y, vk') \rightarrow \pi'$$

the simulator generates the proof π' and provides it to the verifier. Since π' is a simulated proof constructed using the trapdoor trap , it is computationally indistinguishable from a real proof π . This follows from the fact that Halo2 (the underlying ZK proof system) is a valid NIZK proof system [82] (via the zero-knowledge property, Appendix D). Notably, as the sparsity ratio of V_{sparse}^t is the same as W_{sparse}^t in each layer, the number of constraints in both cases is exactly equal, resulting in the same proof size. \square

Symbol	Description
X	Public input data to the DNN
y	Public output from the DNN computation
W	Weight matrix
π	Proof generated by the prover
$W_{i,j}$	Element at the i -th row and j -th column of weight matrix W
$f(X, W)$	The DNN computation function, taking input X and weights W to produce output y
L	Number of layers in the DNN
W^t	Original model weights before applying the sparsification algorithm
W_{sparse}	Sparse weight matrix after sparsification algorithm
W_{sparse}^t	Sparse and teleported weight matrix
L	Number of layers in the DNN
$W^{(i)}$	Original model weights in i -th layer (or block)
$\tilde{W}^{(i)}$	Transformed (sparsified or teleported) model weights in i -th layer (or block)
ρ_i	Importance of the i -th weight in a block of a neural network, used in the CAP method.
δw	Optimal update for the remaining weights in a block after pruning
H_i	Hessian matrix for block i in a neural network, used in the CAP method.
H^{-1}	Inverse of the Hessian matrix
e_i	The i -th standard basis vector in \mathbb{R}^d , where d is the dimensionality of the vector space.
R	Targeted sparsity ratio of the entire network in the sparsification algorithm
$A_{i,j}$	Value in the j -th row and i -th advice column of the Halo2 table
$F_{i,j}$	Value in the j -th row and i -th fixed column of the Halo2 table
A	Set of commitments to the advice columns in Halo2
Q	Set of commitments to the fixed columns in Halo2
$a_i(X)$	Lagrange polynomial for the i -th advice column
$q_i(X)$	Lagrange polynomial for the i -th fixed column
$Q_{F_0}(X)$	Polynomial representing the fixed column containing model weights (F_0).
S_i	Support set for the i -th row, containing indices of non-zero elements
$\Pr[\dots]$	The probability of the event described within the brackets.
$\langle P^*, V \rangle(\phi)$	The interaction between a cheating prover, P^* , and the verifier, V , on a statement ϕ . soutput is either 1 (accept) or 0 (reject).
ϕ	Statement of the ZK proof generation
κ	Security parameter
$v(\kappa)$	A negligible function, which approaches zero faster than any inverse polynomial.
\approx	Computational indistinguishability (two distributions are so similar that no efficient algorithm can distinguish them).
$S(\phi)$	The output of a simulator S , an algorithm that generates a view indistinguishable from the real view without knowing the witness
ppt	Probabilistic Polynomial-Time
I	Input vector to a layer
O	Output vector from a layer
O_j	j -th element of the output vector O
τ	Change of Basis (CoB) scalar
$\{\tau^{(i)}\}$	Set of CoB scalars for layer i
$\tau_j^{(i)}$	CoB scalar for the j -th neuron in the i -th layer
τ^*	Optimal set of CoB scalars obtained through optimization.
$g_j^{(i)}$	Pre-activation input to the j -th neuron in layer i
$z_j^{(i)}$	Activation output of the j -th neuron in layer i
$\phi(x)$	Activation function in the DNN model
$l(\tau)$	Objective function for teleportation optimization
μ	Small perturbation scalar used in zero-order gradient approximation methods.
λ	Regularization hyperparameter balancing input range minimization and function preservation.

Table 5: Notation Table containing symbols used in the equations