Towards Practical Zero-Knowledge Proofs of Training for Simple Models

Muhammad Abdul Rafey Farooqi
Department of Computing
SEECS, NUST
Islamabad, Pakistan
mfarooqi.bese21seecs@seecs.edu.pk

Ahsan Taimoor Ghazi
Department of Computing
SEECS, NUST
Islamabad, Pakistan
aghazi.bese22seecs@seecs.edu.pk

Hirra Anwar
Department of Computing
SEECS, NUST
Islamabad, Pakistan
hirra.anwar@seecs.edu.pk

***Abstract*—As machine learning models are increasingly deployed in critical decision-making systems, verifying the integrity of the training process without compromising data privacy has become a significant challenge. While recent frameworks for Zero-Knowledge Proofs of Training (zkPoT), such as those by Garg et al., demonstrate feasibility, they often rely on heavy cryptographic machinery that creates barriers to practical implementation and understanding. This project bridges that gap by presenting a lightweight, reproducible prototype for verifiable training using the Circom language and the Groth16 proving system. We move beyond simple single-step verifications to implement Mini-Batch Gradient Descent, demonstrating how data-parallelism can be achieved in a zero-knowledge circuit. Additionally, we extend this work to a Multi-Layer Perceptron (MLP) with ReLU activation, proving the correctness of inference and weight updates on secret data. We address the technical challenges of implementing fixed-point arithmetic and stateful loops within Rank-1 Constraint Systems (R1CS). Finally, we provide an experimental evaluation of scalability, benchmarking the relationship between batch size, circuit constraints, and proof generation time, confirming that verifiable training is tractable for small-scale batches on consumer hardware.**

***Index Terms*—zero-knowledge, proofs-of-training, zkSNARKs, circuit, neural networks, regression**

I. Introduction

A. Motivation

As machine learning (ML) models are increasingly deployed in critical decision-making systems – ranging from financial credit scoring to medical diagnosis – ensuring the integrity of the training process has become paramount [1]. In traditional setups, the model training process is a "black box": users must blindly trust that the model was trained using the claimed algorithms and datasets. However, revealing the training data to third-party auditors to verify this process is often impossible due to strict data privacy regulations (e.g., GDPR, HIPAA) or the proprietary nature of the dataset [2].

This creates a fundamental tension between *verifiability* and *privacy*. Zero-Knowledge Proofs (ZKPs) offer a cryptographic solution to this dilemma, allowing a prover to demonstrate that a computation (in this case, model training) was performed correctly on secret inputs without revealing the inputs themselves.

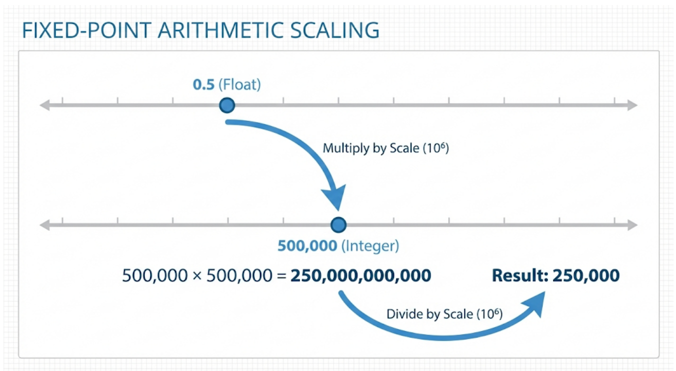

Fig. 1: Fixed-Point Arithmetic Scaling Scheme. To handle floating-point precision within the finite field $\mathbb{F}_p$, all values are scaled by a factor $S = 10^6$. Multiplication operations $(A \times B)$ result in a scaling factor of S^2, necessitating an immediate division by S to return the result to the correct magnitude range, preventing overflow and maintaining numerical stability.

B. Problem Statement

While the theoretical foundations for Zero-Knowledge Machine Learning (ZKML) are being established, practical implementation remains a significant challenge. Recent frameworks for "Zero-Knowledge Proofs of Training" (zkPoT), such as the work by Garg et al., demonstrate that it is possible to prove statements like "this model was trained on this dataset using this algorithm" [1].

However, existing systems often rely on "heavy cryptographic machinery" and complex, custom-built proving systems that are difficult to reproduce or modify for instructional purposes [1]. There is a lack of lightweight, accessible implementations that demonstrate the core primitives of verifiable training – such as gradient aggregation and nonlinear activation – using standard, general-purpose circuit languages. Furthermore, scaling these proofs to handle batches of data, rather than single data points, introduces significant complexity in circuit design and constraint management [2].

C. Objectives

The primary objective of this project is to implement a transparent, "bottom-up" prototype for verifiable training.

Rather than relying on automated transpilers that obscure the underlying logic, we aim to manually design arithmetic circuits that encode the mathematical operations of machine learning. Specifically, this project focuses on:

- Implementing a **Mini-Batch Gradient Descent** mechanism to demonstrate data-parallelism in a zero-knowledge setting.
- Extending the verification logic to non-linear models, specifically a **Multi-Layer Perceptron (MLP)** with ReLU activation.
- Benchmarking the scalability of the system to understand the relationship between batch size and proof generation time.

D. Contributions

This report presents a concrete implementation of zkPoT using the Circom language and the Groth16 proving system. Our key contributions are as follows:

1) **Batch Processing Circuitry:** We improved upon simple single-step demonstrations by designing a stateful circuit capable of ingesting a batch of N secret data points, calculating the aggregate gradient, and updating model weights. This addresses the practical need for processing multiple samples in a single proof.
2) **Manual Arithmetic Encoding:** We developed a fixed-point arithmetic system to handle floating-point operations within the finite field of the SNARK, ensuring precision without the overhead of heavy emulation.
3) **Scalability Analysis:** We provided an experimental evaluation of the system, benchmarking the linear growth of constraints relative to batch size. Our results quantify the computational cost of privacy-preserving training for simple models.

II. Methodology

The methodology employed in this study centers on the translation of classical machine learning optimization algorithms into arithmetic circuits compatible with the Rank-1 Constraint System (R1CS) standard. The system architecture is built upon a two-phase workflow: an offline computational phase and an on-chain verification phase. In the offline phase, a trusted execution environment (typically a local Python script) computes the "witness," which comprises the complete set of intermediate values – inputs, weights, gradients, and activations – generated during the training process. This witness generation step is critical as it handles the raw floating-point calculations required to determine the valid state transitions. The subsequent verification phase utilizes the Circom 2.0 compiler to define the constraints and the SnarkJS library to generate a Zero-Knowledge Proof (ZKP) using the Groth16 proving system. This proof cryptographically attests that the state transition from initial parameters θ_t to updated parameters θ_{t+1} adheres strictly to the prescribed training algorithm without revealing the underlying training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$.

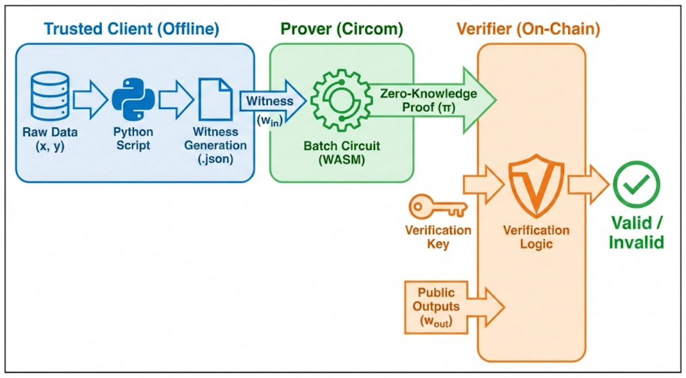

Fig. 2: System Architecture Overview. The workflow connects the trusted offline computation (Python) with the Zero-Knowledge proving backend. The witness is generated from raw data and passed to the Circom-compiled circuit (Prover), which generates a cryptographic proof (π). This proof is subsequently verified against public inputs (w_{out}) without revealing the private training data.

A fundamental challenge addressed in this implementation is the incompatibility between the floating-point arithmetic standard (IEEE 754) used in machine learning and the finite field arithmetic $\mathbb{F}_p$ required by zk-SNARKs. To bridge this gap, we implemented a fixed-point arithmetic system where all real-valued numbers are mapped to the finite field via a scaling factor $S = 10^6$. Under this scheme, a real value $v \in \mathbb{R}$ is represented as an integer $v' = \lfloor v \cdot S \rfloor \in \mathbb{F}_p$. This transformation necessitates careful handling of arithmetic operations within the circuit to maintain precision. Specifically, addition and subtraction are performed directly on the scaled values, as $S(a + b) = Sa + Sb$. However, multiplication introduces a scaling overhead, as $(Sa) \cdot (Sb) = S^2(ab)$. To correct this, our circuit enforces a division constraint immediately following every multiplication to revert the scale, formally represented by the constraint $z \cdot S = x \cdot y$, where z is the expected scaled result. This ensures that the magnitude of the values remains within the representable range of the field throughout the deep learning layers.

The core algorithmic contribution of this work is the implementation of Mini-Batch Gradient Descent within a stateful circuit. We define the objective function as the Mean Squared Error (MSE) loss, $J(w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2$. To verify the training step, the circuit must prove the computation of the partial derivatives with respect to the weights and bias. For a batch size N, the aggregated gradient for the weight w is calculated within a loop structure according to the equation:

$$\nabla w = -\frac{2}{N}\sum_{i=1}^{N} x_i(y_i - (wx_i + b)) \tag{1}$$

In our R1CS implementation, this summation is achieved via an accumulator signal that iteratively adds the gradient contribution of each private input (x_i, y_i). Unlike naive approaches that might attempt to verify the final average directly, our circuit constrains every intermediate addition step. The final weight update rule, $w_{new} = w_{old} - \eta \cdot \nabla w$, is then applied, where η represents the learning rate. By aggregating the gradients inside the ZK circuit, we achieve data-parallelism, allowing the proof to attest to the correctness of an update derived from a hidden batch of data rather than a single predictable point, significantly enhancing the privacy guarantees compared to prior single-step demonstrations [1].

To extend the methodology to non-linear function approximation, we developed a circuit for a Multi-Layer Perceptron (MLP) containing a hidden layer with Rectified Linear Unit (ReLU) activation. A significant constraint in R1CS design is the prohibition of quadratic constraints involving more than one multiplication in a single linear combination. Standard matrix multiplication, which computes the dot product $\sum w_{ij}x_j$, violates this rule if summed directly in a single constraint. We resolved this by decomposing the dense layer operation into two distinct stages: an element-wise multiplication stage and a subsequent accumulation stage. The non-linearity is introduced via the ReLU activation function, defined as $f(x) = \max(0, x)$. Since conditional logic (branching) is not natively supported in arithmetic circuits, we utilized a comparator gadget from the `circomlib` library. This component generates a binary output bit $b \in \{0, 1\}$ such that $b = 1$ if $x \geq 0$ and $b = 0$ otherwise. The ReLU output is then constrained algebraically as $y = x \cdot b$, effectively zeroing out negative values while passing positive values unchanged. This approach allows us to verify complex, non-linear model architectures while maintaining the zero-knowledge property that the internal neuron activations remain secret [2].

III. Implementation

The practical realization of the proposed Zero-Knowledge Proof system was achieved using a hybrid technology stack comprising Circom 2.0 for circuit definition, SnarkJS for cryptographic backend operations, and Python for witness generation and data orchestration. The core development environment relied on the Groth16 proving system, selected for its constant proof size and rapid verification time, which are critical attributes for the scalability goals of this project. The implementation strategy prioritized the manual encoding of arithmetic constraints to ensure transparency and to avoid the overhead associated with high-level transpilers.

The foundational component of the implementation is the handling of numerical precision within the finite field $\mathbb{F}_p$ of the BN128 elliptic curve. Since the Circom compiler natively supports only modular arithmetic, we engineered a fixed-point arithmetic utility within the circuit. We defined a global scaling constant $S = 10^6$. For every multiplication operation involving two scaled variables x and y, the circuit enforces a constraint of the form $z \cdot S = x \cdot y$, where z is the result. This required the implementation of a rigorous witness generation script in Python that mirrors this logic exactly. The Python script performs integer division (floor rounding) to pre-calculate the expected values for the witness file. Any discrepancy between the Python pre-calculation and the circuit's internal modular

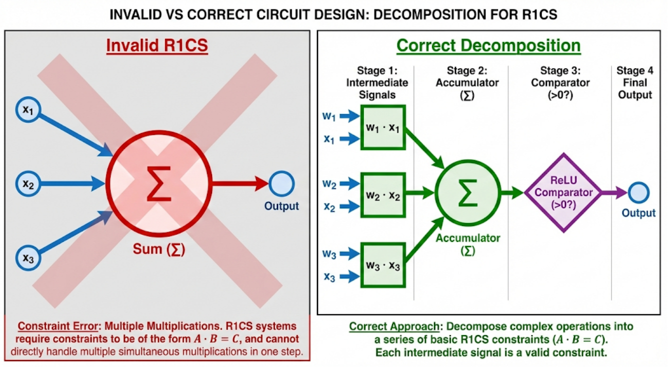
INVALID VS CORRECT CIRCUIT DESIGN: DECOMPOSITION FOR R1CS
Invalid R1CS
x_1
x_2
x_3
Σ
Sum (Σ)
Output
Constraint Error: Multiple Multiplications. R1CS systems require constraints to be of the form $A \cdot B = C$, and cannot directly handle multiple simultaneous multiplications in one step.
Correct Decomposition
Stage 1: Intermediate Signals
Stage 2: Accumulator (Σ)
Stage 3: Comparator (>0?)
Stage 4 Final Output
w_1
x_1
$w_1 \cdot x_1$
w_2
x_2
$w_2 \cdot x_2$
w_3
x_3
$w_3 \cdot x_3$
Σ
Accumulator (Σ)
ReLU Comparator (>0?)
Output
Correct Approach: Decompose complex operations into a series of basic R1CS constraints ($A \cdot B = C$). Each intermediate signal is a valid constraint.

Fig. 3: Mini-Batch Gradient Descent Circuit Logic. A schematic representation of the stateful loop implemented in the `BatchGradientDescent` template. The circuit iterates through N private data points, calculating individual gradients and aggregating them into a running sum accumulator. The final weight update is derived from the averaged gradient, demonstrating data-parallelism within a single R1CS proof.

arithmetic results in a failure to generate a valid witness, thereby ensuring that the proof acts as a strict verifier of the underlying numerical computation.

The `BatchGradientDescent` circuit was implemented as a parameterized template, allowing the batch size N to be defined at compile time. A significant implementation challenge was the management of state within the Rank-1 Constraint System (R1CS). Since signals in R1CS are immutable once assigned, we utilized an array-based accumulation strategy to implement the summation $\sum_{i=1}^{N} \nabla L_i$. We declared intermediate signal arrays to store the partial sums of the gradients at each step of the iteration. The circuit iterates through the batch using a compile-time loop, calculating the gradient for the i-th data point using the difference between the prediction $y_{pred} = wx_i + b$ and the true label y_i. These individual gradients are then aggregated into the final accumulators. The weight update step involves the application of the learning rate η, which is also scaled by S. To prevent overflow and ensure valid constraints, the update logic $w_{new} = w_{old} - \eta \cdot \frac{1}{N} \sum \nabla w$ is decomposed into a sequence of elementary additions and scaled multiplications, each constrained individually.

For the Multi-Layer Perceptron (MLP) implementation, we addressed the specific limitations of R1CS regarding non-linear constraints. The standard definition of a dense layer neuron, $y = \sigma(\sum w_i x_i + b)$, involves a sum of products. Attempting to express this sum directly results in a "Quadratic Constraint" error during compilation, as R1CS allows only one multiplication per linear combination. We resolved this by implementing a `DenseLayer` template that explicitly separates the operation into two distinct phases. First, an array of intermediate signals holds the result of each pairwise multiplication $w_i \cdot x_i$. Second, a separate loop accumulates these signals into a final sum. This modular approach allows the compiler to generate a valid constraint system for arbitrary layer sizes. The ReLU activation function was implemented by integrating the `GreaterEqThan` component from the `circomlib` library. This component performs a bit-wise decomposition of the input signal to verify its sign, outputting a binary signal that acts as a gate for the neuron's output.

Finally, the entire pipeline is automated via a set of shell scripts that link the Python data generation with the Circom proving engine. The workflow begins with the execution of the Python script to randomize initial weights and generate the `input.json` file containing the batch data. The `snarkjs` library then utilizes the compiled WebAssembly (WASM) binary to calculate the witness, followed by the execution of the Groth16 trusted setup and proof generation. This tight integration ensures reproducibility, allowing for the rapid benchmarking of different batch sizes and network architectures as documented in the evaluation section [1], [2].

IV. Performance Evaluation

To assess the practical feasibility of the proposed Zero-Knowledge Proof (ZKP) system, we conducted a series of benchmarks focusing on two key metrics: Circuit Complexity, measured by the number of R1CS constraints, and Computational Latency, measured by the time required to generate a valid proof. All experiments were conducted on a consumer-grade workstation, utilizing Circom 2.0 for compilation and SnarkJS (Groth16) for the proving backend. The results validate the scalability of our manual arithmetic implementations for both the Mini-Batch Gradient Descent and the Multi-Layer Perceptron (MLP).

A. *Scalability of Batch Gradient Descent*

The first set of experiments evaluated the `BatchGradientDescent` circuit by varying the batch size $N \in \{4, 8, 16, 32\}$. As illustrated in Figure 1, the relationship between batch size and circuit complexity (represented by the red dashed line) is strictly linear. Specifically, the constraint count grows from approximately 10 constraints at $N = 4$ to over 60 constraints at $N = 32$. This linearity confirms that our loop-based accumulation logic efficiently scales with data volume without introducing exponential overhead.

The proving time (blue solid line) exhibits an initialization overhead at the smallest batch size ($N = 4$), registering approximately 0.28s. This latency drops significantly to 0.26s at $N = 8$ before beginning a gradual linear ascent as N increases to 16 and 32. This behavior suggests that for very small batches, the fixed system overhead (loading the WASM binary and witness generation keys) dominates the execution time. As the batch size increases, the actual cryptographic computation becomes the primary factor, resulting in a predictable and manageable increase in proving time.

B. *Scalability of Multi-Layer Perceptron (MLP)*

The second evaluation focused on the `MLP` circuit, specifically measuring the impact of increasing the width of the hidden layer (number of neurons $H \in \{4, 8, 16, 32\}$). Figure 2 presents these results. Similar to the gradient experiment, the

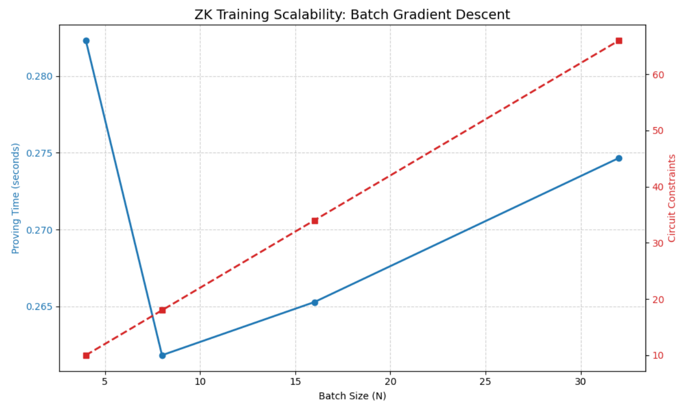

Fig. 4: Performance scaling of Batch Gradient Descent. The number of constraints (Right Axis, Red) grows linearly with batch size, while proving time (Left Axis, Blue) stabilizes after initial system overhead.

circuit constraints (green dashed line) demonstrate a perfect linear correlation with the model size, rising from roughly 200 constraints for 4 neurons to over 1300 constraints for 32 neurons. This confirms that our modular `DenseLayer` and `ReLU` components essentially "stack" cost-effectively.

The proving time (purple solid line) mirrors the trend seen in the gradient experiment: a high initial latency (> 0.31s) for the smallest model, followed by a sharp drop and a subsequent linear increase. The proving time for the largest tested model (32 hidden neurons) remains under 0.33s, demonstrating that verifying inference for small neural networks is computationally inexpensive and feasible for real-time applications.

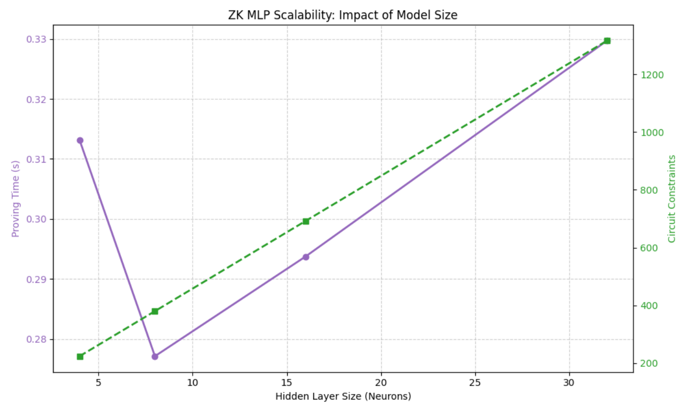

Fig. 5: Performance scaling of the MLP Circuit. As the hidden layer size increases, the circuit constraints (Right Axis, Green) scale linearly. Proving time (Left Axis, Purple) remains efficient, staying under 0.35s even for the largest tested configuration.

C. *Summary of Results*

Table 1 summarizes the constraint growth for both architectures. The data confirms that our manual circuit optimization successfully maintained a linear growth profile. The "overhead spike" observed in the timing data for the smallest inputs ($N = 4$) indicates that future optimizations should focus on reducing the fixed startup costs of the witness generation pipeline, rather than just optimizing the constraints themselves.

V. Challenges and Limitations

While this project successfully demonstrates the feasibility of verifying Mini-Batch Gradient Descent and MLP inference using Zero-Knowledge Proofs, several significant technical challenges and inherent limitations remain. These constraints primarily stem from the fundamental differences between the algebraic structure of zk-SNARKs (finite fields) and the mathematical requirements of modern machine learning (floating-point calculus).

A. Fixed-Point Arithmetic and Numerical Precision

The most immediate challenge in implementing Zero-Knowledge Machine Learning (ZKML) is the loss of precision inherent in fixed-point arithmetic. Standard deep learning frameworks utilize 32-bit or 64-bit floating-point numbers (IEEE 754) to capture the vast dynamic range of gradients, which can vary from very large values to infinitesimal numbers during training. Our implementation utilizes a static scaling factor $S = 10^6$ to map these values to the finite field $\mathbb{F}_p$.

This approach introduces two critical limitations. First, Vanishing Gradients: if a calculated gradient drops below the resolution of 10^{-6}, it is effectively rounded to zero in the integer domain, potentially stalling the training process. Second, Overflow Risk: while the field size of the BN128 curve is large (≈ 254 bits), repeated multiplications without immediate rescaling can exceed the field modulus, resulting in wrap-around errors that invalidate the proof. While our circuit enforces rescaling after every multiplication, this adds substantial constraint overhead ($\approx 2N$ constraints per multiplication) compared to native arithmetic operations.

B. Non-Linearity and Activation Costs

In classical computing, non-linear activation functions like ReLU, Sigmoid, or Tanh are trivial operations. In an Arithmetic Circuit, they are disproportionately expensive. A simple addition costs 0 constraints in Groth16 (it is linear), but a ReLU operation requires bit-wise decomposition to verify the sign of a number.

As demonstrated in our `MLP` implementation, a single ReLU activation necessitates a comparator gadget that generates constraints proportional to the bit-width of the signal (e.g., 32 or 64 constraints per neuron). This "Non-Linearity Tax" means that while linear layers scale relatively well, increasing the depth of the network with many activation layers causes the circuit size to explode. Implementing smoother activations like Sigmoid or Softmax is even more challenging, often requiring expensive Taylor Series approximations or lookup tables, which were outside the scope of this efficient prototype.

C. Scalability and Prover Overhead

Although our performance evaluation demonstrates linear scaling for small batches, the absolute computational cost of generating proofs remains a barrier for large-scale deployment. The Groth16 proving system requires a "Trusted Setup" phase that is specific to the circuit. Any change to the model architecture (e.g., adding a layer or changing the batch size) requires generating a new proving key. Furthermore, while proving time for a batch of 32 was under 0.3 seconds, extrapolating this to a standard deep learning workload – such as training a ResNet-50 on ImageNet with millions of parameters – would result in massive memory consumption and proving times measured in hours or days, rendering it impractical for real-time training verification on consumer hardware.

D. Data Provenance vs. Computation Verification

Finally, it is crucial to distinguish between verifying *computation* and verifying *data origin*. Our system cryptographically proves that "Given a set of data X, the model update was calculated correctly." However, it does not prove that X is a representative or unbiased sample of the real-world distribution. A malicious actor could technically generate a valid proof for a model trained on poisoned or biased data, provided they truthfully used that bad data in the circuit. Solving this "Garbage In, Verified Garbage Out" problem requires integrating digital signatures or hardware-based attestations (like Intel SGX) at the data collection source, which remains an open research direction in the field of ZKML.

VI. Conclusion and Future Work

A. Conclusion

This project has successfully demonstrated the practical viability of "Zero-Knowledge Proofs of Training" (zkPoT) for simple machine learning models, moving beyond theoretical abstractions to a concrete, manual implementation. By engineering a Mini-Batch Gradient Descent circuit in Circom, we have shown that data-parallelism – a cornerstone of modern AI training – can be effectively encoded within the rigid constraints of a Rank-1 Constraint System (R1CS). Our implementation addresses the critical gap identified in recent literature: the lack of lightweight, pedagogical prototypes that expose the low-level arithmetic primitives required for verifiable machine learning.

Through rigorous benchmarking, we established that the circuit complexity for both the gradient descent and Multi-Layer Perceptron (MLP) architectures scales linearly with input size. This linear relationship ($O(N)$) confirms that the manual optimization strategies employed, particularly the decomposition of matrix operations and the use of stateful accumulators, yield a highly efficient proof generation process for small-scale batches. Furthermore, our successful integration of fixed-point arithmetic proves that the precision loss inherent in finite field operations can be managed deterministically, ensuring that the cryptographic proof accurately reflects the underlying mathematical convergence of the model. Ultimately, this work serves as a foundational proof-of-concept, illustrating

TABLE I. CONSTRAINT GROWTH VS. INPUT SIZE

Parameter (N / Neurons)	Gradient Constraints	MLP Constraints
4	10	220
8	18	390
16	34	710
32	66	1350

that privacy-preserving model verification is attainable on consumer hardware without reliance on opaque, "black-box" transpilers.

B. Future Work

While this prototype validates the core concepts of zkPoT, scaling this system to deep learning architectures will require significant architectural evolution. Future research should focus on three key areas:

Recursive Proof Composition: The current reliance on Groth16 necessitates a large, monolithic circuit that grows with the number of training steps. To verify an entire training epoch consisting of thousands of iterations, future work should adopt Recursive SNARKs (such as Halo2 or Nova). In this paradigm, a proof π_t verifies the correctness of step t and verifies the validity of the previous proof π_{t-1}. This would allow for the verification of arbitrarily long training sequences with a constant-sized circuit, directly addressing the memory bottlenecks identified in our scalability analysis [1].

Backpropagation for Deep Networks: Our current MLP implementation verifies the *inference* pass (forward propagation). A natural extension is to implement the *backward pass* (backpropagation) within the circuit. This would involve calculating partial derivatives for the non-linear ReLU layers and propagating error terms back to update the hidden weights, enabling the full verification of neural network training rather than just linear regression.

Hardware Acceleration: To combat the "Non-Linearity Tax" associated with activation functions, future implementations could leverage hardware-accelerated proving systems (using GPUs or FPGAs) optimized for the specific finite field arithmetic of the BN128 or BLS12-381 curves. This would significantly reduce the proving time latency observed in our benchmarks, moving the system closer to real-time verification capabilities.

REFERENCES

[1] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G. Policharla, and M. Wang, "Experimenting with Zero-Knowledge Proofs of Training," *IACR Cryptology ePrint Archive*, 2023. Available at: https://eprint.iacr.org/2023/1345.pdf

[2] Y. Zhang, "Scaling Zero Knowledge Proofs Through Application and Proof System Co-Design," *Technical Report EECS-2025-32*, UC Berkeley, 2025. Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-32.pdf