# Computational Attestations of Polynomial Integrity Towards Verifiable Back-Propagation

Dustin Ray[1] and Caroline El Jazmi[2]

[1] University of Texas Austin, Austin TX 78701, USA,
dustinray@utexas.edu
[2] University of Texas Austin, Austin TX 78701, USA,
eljazmi@utexas.edu

**Abstract.** Recent advancements in machine learning accuracy and utility have been driven by the effective combination of sophisticated models with high-performance computational scaling. As the development of large-scale models shifts away from commodity hardware to outsourced computation, it becomes paramount to ensure that the training process is executed with integrity and transparency. This encompasses verifying that adequate computational resources were expended and that the resulting model is accurate, rather than the product of skipped steps or resource-saving shortcuts by the external provider. Building on our previous efforts, which demonstrated the computational feasibility of using this system to argue correctness for differentially-private linear regression, we extend those results to achieve fully provable back-propagation—a cornerstone operation in modern machine learning training. Our system achieves complete zero-knowledge, revealing nothing about the input data during training, and ensures quantum security by relying on no weak cryptographic primitives. Efficiency is substantially increased through the use of a fixed-point decimal representation, reducing the computational overhead typically associated with floating-point arithmetic. Notably, our solution is doubly efficient, achieving a logarithmic-time verifier and a linear-time prover. Implemented entirely in Rust without reliance on external machine learning libraries, and executed within a cryptographically secure virtual machine, this work represents a significant advancement toward verifiable, secure, and efficient outsourced machine learning computations.

**Keywords:** Machine Learning, Back-Propagation, Zero-Knowledge Cryptography, Probabilistic Checkable Proofs, ZK-STARK.

## 1 Introduction

The scaling of computational resources has enabled the development of increasingly powerful artificial intelligence (AI) models, particularly in machine learning and generative systems. This progress has brought transformative applications across diverse domains, but it has also created a growing dependency on highly specialized hardware, such as graphics processing units (GPUs), to

train these models effectively. Training state-of-the-art models requires immense computational and financial resources, often exceeding the capacity of individual organizations to self-host. The costs associated with training cutting-edge AI models have skyrocketed, with some models requiring millions of dollars in computational expenses [1][2]. As a result, many organizations rely on external service providers that offer access to this specialized hardware and adopt cost optimization strategies to manage expenses [3][4].

As machine learning systems, including generative AI, become integral to modern workflows, their complexity and reliance on vast datasets continue to grow. A Deloitte study [5] found that at least 50% of surveyed organizations planned to integrate machine learning systems into their operations in 2023. Outsourcing model training to external services introduces significant security and trust challenges, particularly when sensitive training data or resource-intensive computations are involved. Ensuring that models are trained correctly, that proper computational resources are used, and that the results reflect honest computations rather than unauthorized optimizations or tampering that could compromise computational integrity, is crucial in maintaining the integrity of outsourced machine learning pipelines. Effective cost management, such as that advocated by major cloud providers [6][7], further emphasizes the need for careful planning and resource allocation in these environments.

Our penultimate result demonstrates how an Machine Learning as a Service (MLaaS) operator can leverage novel cryptographic techniques with minimal assumptions to provide *statements of computational integrity* for critical machine learning operations such as back-propagation. This approach ensures:

- The consumer is convinced with high probability that the computations were executed correctly.
- Verification of the computation requires a proportionally small amount of work on the consumer's end.

We emphasize that for outsourced machine learning operations to be practical, it is essential that the verification process for the consumer remains significantly less resource-intensive than the computational work performed by the MLaaS operator. This research achieves an *asymptotically optimal solution* by enabling double efficiency: the operator performs linear work while the consumer performs logarithmic work. We demonstrate the utility of this approach by proving the correctness of back-propagation during the training of a model on the MNIST handwriting dataset, achieving high accuracy and optimal asymptotic performance. Further details of our design are discussed in subsequent sections.

The organization of this paper is structured as follows: Section 2.1 discusses the fundamental principles of zero-knowledge (ZK) cryptography and computational integrity arguments, and presents a detailed account of our desired protocol attributes, emphasizing the requirements such as completeness, soundness, and succinctness that guide the design of our argument system. This section further explores the minimal hard assumptions necessary for the robustness of the protocol and provides the theoretical backbone for the argument systems employed to verify the integrity of computations in machine learning processes.

Section 4 examines the different implementation approaches towards achieving efficient computational integrity statements in practice. Section 3 describes back-propagation, which is the central computation of our argument system. Section 4 discusses the various approaches used in practice to obtain argument systems that match the protocol description that we lay out in further sections. It acts as a survey of various approaches that are thus far commonly used in the argument-system landscape. Section 5 shifts focus to the empirical evaluation of the proposed system, detailing the experimental setup, methodologies, and the performance metrics used to gauge the efficacy of the argument system in real-world scenarios.

Section 6 reviews related literature, focusing on the closest known result to ours, which employs succinct argument systems for model verification. In contrast, our approach offers stronger zero-knowledge guarantees and improved verifier efficiency, while operating under fewer cryptographic assumptions. We use this result to highlight the contributions and advancements our approach offers over existing methods. The conclusion in Section 7 encapsulates the significance of the research, its implications for privacy-preserving machine learning, and potential future directions.

## 2 Zero-Knowledge Cryptography and Arguments of Computational Integrity

This work builds on an active area of cryptographic research focused on argument systems, particularly those of the probabilistically-checkable and non-interactive variety. The seminal results of [8] introduce an argument system involving interactions between a prover $\mathscr{P}$ and a verifier $\mathscr{V}$. In this framework, $\mathscr{P}$ seeks to convince $\mathscr{V}$ of the truth of a statement, such as the correctness of a computational result obtained from a given input. Rather than requiring $\mathscr{V}$ to re-execute the entire computation to verify correctness, [8] demonstrates that $\mathscr{V}$ can, with extremely high probability, rely on a logarithmic-size, non-deterministic sampling of an argument provided by $\mathscr{P}$ to determine correctness or detect inconsistencies. This protocol, which offers a significant reduction in verifier workload, is formally defined below.

### 2.1 Protocol Definitions: Completeness, Soundness, and Succinctness

With a computation defined as a tuple of algorithms ($\mathscr{P}$ and $\mathscr{V}$), we parameterize an argument system with desired properties and analyze its suitability for outsourced machine learning operations. The desired argument system operates under random oracle assumptions and is defined with the following probabilistic guarantees [35]:

*Completeness*: True statements can always be proven by a prover $\mathscr{P}$ and will be accepted by a verifier $\mathscr{V}$ with probability 1. Formally, for every instance-witness pair $(x, w)$ in a relation $\mathscr{R}$:

$$\Pr\left[\mathscr{V}^p\left(,\mathscr{P}^p(,)\right)=1\right]=1,$$

where the probability $p$ is taken over the randomness of $\mathscr{P}$ and $\mathscr{V}$. This guarantees that a valid computation will be accepted by the verifier with probability 1.

*Soundness*: A prover $\widetilde{\mathscr{P}}$ should not be able to convince the verifier $\mathscr{V}$ to accept a false statement except with negligible probability. Formally, for every instance  not in the language of $\mathscr{R}$, and for every malicious prover $\widetilde{\mathscr{P}}$ submitting at most a polynomial number of queries to a random oracle:

$$\Pr\left[\mathscr{V}^p\left(,\widetilde{\mathscr{P}^p}\right)=1\right]\leq \text{negl}(\lambda),$$

where $\lambda$ is the security parameter. Ben-Sasson *et al.* [35] describe how the soundness parameter directly relates to a configurable $n$-bit security level based on the choice of hash function.

*Succinctness*: The argument system ensures that the computational and communication overhead for verification is minimal compared to the cost of performing the original computation. Specifically, for a computation of size $\mathcal{O}(n)$, the prover $\mathscr{P}$ performs $\mathcal{O}(n)$ work, while the verifier $\mathscr{V}$ performs $\mathcal{O}(\log n)$ work. This property is critical for outsourced machine learning operations, as it allows computationally intensive algorithms, such as back-propagation, to be verified efficiently. For instance, a computation requiring $10,000,000$ steps can be verified with only $\mathcal{O}(log_2(10,000,000)=23)$ queries.

*Minimal Hard Assumptions*: Our protocol is designed to rely on minimal cryptographic hard assumptions wherever possible. Following the results of [35], this requirement is satisfied by using only a secure hash function as the underlying primitive, combined with error-correcting codes. This construction leads to computational integrity statements that are plausibly secure against quantum adversaries. While we do not simulate a quantum adversary in this work, the security of our scheme inherits the post-quantum plausibility from the framework of [35].

*Perfect Zero-Knowledge*: We diverge slightly from the nomenclature of [10] by observing that "computational integrity statements" in our scheme do not strictly require perfect zero-knowledge. While the zero-knowledge property in our results follows trivially from the application of [10] with minimal overhead, it is not a strict necessity for our protocol. This is because the MLaaS operator and consumer are assumed to be the only parties involved in the computation, both with access to the same sensitive data and resulting machine learning model. Nevertheless, the argument remains zero-knowledge by construction and reveals no information that could compromise the confidentiality about the model or dataset used in the computation. This property enables the argument to be safely verified by any public or untrusted party without compromising privacy.

The parameters for our protocol are instantiated using a cryptographic non-interactive argument scheme resembling a *zero-knowledge scalable transparent argument of knowledge* (ZK-STARK) [10], applied to a machine learning algorithm.

### 2.2 Nomenclature

It is common in zero-knowledge (ZK) literature to refer to computational integrity statements as "proofs" and to the party generating the statement as the "prover". When requirements around perfect soundness are relaxed, these statements are better described as *arguments*, since they do not absolutely prove the truth of a statement but instead *argue* its correctness with high probability. Throughout this work, we use the terms "computational integrity (CI) statement" and "argument" interchangeably, as they are functionally equivalent to the concept of a "zero-knowledge proof".

An honest prover holds a complete proof that the computation was executed faithfully according to the specified algorithm and input. To achieve optimal asymptotic complexity, the prover transforms this proof into an argument, which retains the essential guarantees while being computationally efficient. Thus, the term "prover" accurately reflects the role of the party responsible for producing CI statements, and we adopt this terminology consistently throughout the work. In Section 5, we use the phrase "proving the dataset" as shorthand to indicate the creation of a CI statement for the training process over the dataset.

Finally, we adopt the nomenclature of [8], with respect to machine learning, where terms like "learn" and "hypothesis" are used to describe the process of training a machine learning model and the resulting model, respectively. This aligns with the broader "Probably Approximately Correct" (PAC) framework, a foundational perspective in the theoretical study of machine learning.

Table 1: Key Terminology Used in This Work

| **Term** | **Definition** |
|---|---|
| **Proof** | A complete computational demonstration of the correctness of a computation, typically requiring high computational overhead. |
| **Argument** | A probabilistically checkable statement that efficiently argues the correctness of a computation, without requiring a complete proof. Used when perfect soundness is relaxed. |
| **Prover** | The party responsible for generating computational integrity statements (arguments) from a computation. |
| **Verifier** | The party that checks the validity of the argument provided by the prover, ensuring the correctness of the computation. |
| **Computational Integrity (CI) Statement** | An efficient argument that asserts the correctness of a computation, equivalent to a zero-knowledge proof in this context. |
| **Learn** | The process of training a machine learning model using a dataset. |
| **Hypothesis** | The resulting machine learning model after training. |

## 3 Back-Propagation: A Key Component of Machine Learning Training

Back-propagation is a bedrock algorithm in machine learning, enabling the training of neural networks by efficiently computing gradients used for optimization. Originally developed in the 1970s and gaining widespread attention in the 1980s, back-propagation leverages the chain rule of calculus to propagate error gradients from the output layer of a neural network backward through its layers. This process updates the network's weights and biases to minimize the difference between the predicted outputs and the true labels in a dataset.

The algorithm operates in two main phases:

- **Forward Pass**: The input data propagates through the network layer by layer, producing an output. The output is compared to the true label using a loss function, which quantifies the prediction error.
- **Backward Pass**: The gradient of the loss function with respect to each parameter is calculated using the chain rule. These gradients are then used to update the network's parameters via an optimization algorithm, such as stochastic gradient descent (SGD).

Back-propagation is particularly effective in training deep neural networks, where the architecture consists of multiple layers with non-linear activation functions [11]. It has been instrumental in the success of modern machine learning applications, from image recognition to natural language processing. However, its computational intensity poses challenges, particularly when training large-scale models on massive datasets. This has driven the need for optimized implementations and the use of specialized hardware, such as graphics processing units (GPUs) and tensor processing units (TPUs), to accelerate training.

In this work, we demonstrate the use of back-propagation within a cryptographic framework to ensure computational integrity. By constructing arguments of correctness for the back-propagation process, we establish a verifiable framework for outsourcing machine learning training to untrusted settings, ensuring the accuracy and reliability of the resulting model with high confidence.

### 3.1 Mathematical Description of Back-Propagation

Back-propagation leverages the chain rule of calculus to compute the gradients of the loss function with respect to the parameters of a neural network. Consider a neural network composed of $L$ layers, where the output of the $l$-th layer is denoted as $\mathbf{a}^{(l)}$ and the weight matrix and bias vector for that layer are $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$, respectively. The output of the network is given by:

$$\mathbf{a}^{(l+1)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)}\right),$$

where $\sigma(\cdot)$ is the activation function, applied element-wise.

The training process aims to minimize a loss function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$, where $\hat{\mathbf{y}}$ is the predicted output and $\mathbf{y}$ is the true label. The back-propagation algorithm

computes the gradients of $\mathcal{L}$ with respect to the parameters $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ by propagating the error backward through the network.

The error term for the $l$-th layer is defined as:

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}},$$

where $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ is the pre-activation value at layer $l$. Using the chain rule, the error term for layer $l$ can be expressed recursively as:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)}\right)^{\top} \delta^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)}),$$

where $\odot$ denotes the element-wise Hadamard product, and $\sigma'(\cdot)$ is the derivative of the activation function.

The gradients of the loss function with respect to the parameters are given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} \left(\mathbf{a}^{(l-1)}\right)^{\top}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

These gradients are used to update the parameters using an optimization algorithm such as stochastic gradient descent:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}, \quad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}},$$

where $\eta$ is the learning rate.

This iterative process continues until the loss function $\mathcal{L}$ converges to a minimum, producing a trained neural network model.

### 3.2 Challenges of Encoding Back-Propagation in Argument Systems

Encoding back-propagation within traditional argument systems presents significant challenges due to its computational complexity and reliance on operations that do not naturally align with most cryptographic proof frameworks. Back-propagation involves matrix multiplications, gradient computations, and the application of non-linear activation functions, all of which are computationally expensive and involve real-valued arithmetic.

Traditional argument systems, such as those based on zero-knowledge proofs or probabilistic-checkable proofs, typically operate over finite fields, often instantiated as integers modulo a cryptographic prime [35]. This modular arithmetic is inherently discrete and well-suited for applications like arithmetic circuits or field operations. However, back-propagation's reliance on continuous operations, such as multiplication of floating-point numbers and derivatives of activation functions, introduces significant encoding complexity.

To adapt back-propagation for these systems, the following challenges must be addressed:

- **Precision and Representation**: Representing real numbers and their operations in modular arithmetic requires approximations, often implemented through fixed-point representations. This introduces a trade-off between numerical precision and computational overhead [23].
- **Circuit Depth and Size**: Back-propagation's matrix operations and activation function derivatives translate to deep and large arithmetic circuits when encoded. This can lead to prohibitively high proving and verification costs, particularly for deep neural networks [20].
- **Non-Linear Functions**: Non-linear activation functions, such as sigmoid, ReLU, or tanh, are particularly challenging to express as modular arithmetic operations. Approximations or piecewise linear functions are often required, further increasing circuit complexity [23].
- **Interoperability**: Many argument systems assume a structure that is optimal for simple modular arithmetic but not for complex algorithms like back-propagation. Wiring up a back-propagation process as a valid arithmetic circuit often requires extensive custom transformations, which are non-trivial and error-prone [20].

The difficulty of encoding back-propagation as a circuit stems not only from its computational expense but also from a semantic mismatch between machine learning operations and cryptographic circuit models. These challenges highlight the need for optimized frameworks to bridge this gap. In this work, we address these limitations using a custom Rust-based virtual machine that supports a wide range of programming constructs, including fixed-point arithmetic. This design minimizes circuit complexity while maintaining soundness and efficiency, and it enables a more expressive implementation of back-propagation that closely resembles its original algorithmic structure.

## 4 Obtaining Computational Integrity Statements

There are two primary approaches to constructing a protocol that satisfies the requirements outlined in Section 2.1. The first approach involves creating a computational attestation of integrity by encoding the algorithm or computation as a circuit using standardized constraint systems such as Rank-1 Constraint Systems (R1CS) or Algebraic Intermediate Representations (AIR). This is conceptually analogous to hardware design in digital logic, where computations are represented as sequences of gates and wires. This "ASIC-style" approach is prevalent in the zero-knowledge (ZK) literature, largely because the field lacks fully mature general-purpose frameworks, especially for STARK-based systems.

Recent advancements [14] have demonstrated that entire instruction set architectures (ISAs) can be encoded within such circuits. This enables the instantiation of a small virtual machine that can execute instructions as part of a provable circuit. The resulting argument of valid computation demonstrates the correct transition of values stored in the virtual machine's registers according to a given set of instructions. This architecture is commonly referred to as a zero-knowledge virtual machine (ZKVM). A ZKVM can be built on a variety

of argument systems and provides a more flexible and generalizable framework than traditional circuit-based approaches.

For this work, we adopt the RISC-Zero virtual machine [14], which implements an argument system that satisfies the requirements outlined in Section 2.1. Two key features of this ZKVM are critical to our results:

- The RISC-Zero ZKVM accepts arbitrary Rust code as input, substantially reducing programming complexity and iteration time. This approach also supports the development of a broader range of applications due to the expressiveness of high-level languages.
- As a virtual implementation of a RISC-V processor, the RISC-Zero ZKVM is compatible with most Rust crates that can be compiled to the RISC-V instruction set. This compatibility allows for immediate reuse of many libraries and data types from the existing Rust ecosystem.

Certain functions and algorithms exhibit performance characteristics that vary significantly depending on the hardware on which they are executed. Specialized hardware, such as application-specific integrated circuits (ASICs) designed for cryptographic hashing or elliptic curve operations, can achieve runtime performance several orders of magnitude higher than general-purpose central processing units (CPUs).

In contrast, a zero-knowledge virtual machine (ZKVM) simulates an instruction set architecture (ISA), such as RISC-V, as a provable circuit. This simulation introduces overhead, since each instruction must be translated into circuit executions, making it inherently less efficient than a direct ASIC implementation. While this overhead is expected, it represents a trade-off for the flexibility and generality provided by a provable ISA.

To mitigate some of this performance cost, the RISC-Zero ZKVM includes an optional CUDA backend, enabling GPU acceleration of the proving process. On our system, even with a modest laptop GPU, we observe a significant reduction in runtime when this feature is enabled, highlighting the feasibility of optimizing the overhead associated with provable ISA simulations.

## 5 Our Results

This section outlines the experimental setup and presents the results of our study. We run our provable back-propagation algorithm on a single Debian-based laptop equipped with a 13th Gen Intel Core™ i9-13900H CPU, 32 GB of RAM, and an NVIDIA GeForce RTX™ 4070 Laptop GPU with 8 GB of available VRAM. We observe that this relatively accessible and moderately powered and cost-effective hardware is sufficient to carry out our experiments, demonstrating that protocols meeting the attributes defined in Section 2.1 can be reasonably operated outside of high-power and high-cost cloud-based platforms.

For our experiments, we use the MNIST dataset [15], a widely used benchmarking standard in machine learning. The MNIST dataset contains 70,000 grayscale images of handwritten digits, each of size $28 \times 28$ pixels, split into 60,000

training samples and 10,000 test samples. Using our provable back-propagation framework, we train a neural network to classify these digits with validation accuracy exceeding 95% in our most optimized experiments. This setup allows us to benchmark the performance of our cryptographic protocol while leveraging the dataset's standard evaluation metrics to measure accuracy and efficiency.

### 5.1 Training and Experiment Design

The primary computation we are interested in consists of training a neural network designed as a multilayer perceptron (MLP), using back-propagation to learn a hypothesis for classifying the MNIST dataset. The MLP consists of an input layer, one hidden layer, and an output layer, with non-linear activation functions applied to the hidden and output layers. The network is optimized using stochastic gradient descent (SGD), and the back-propagation algorithm computes the gradients needed to update the network's parameters. This setup allows us to evaluate the performance of our cryptographic protocol in the context of a standard machine learning task.

Metrics of interest include "Proof Time vs Dataset Size", "Verification Time vs Dataset Size", and "Model Accuracy vs Dataset Size", each of which serves to quantify different dimensions of system efficiency: computational overhead for the prover, effort required for verification, and the fidelity of model training under cryptographic constraints. We conduct the training over the entire dataset using both the CPU and GPU on our laptop. Due to the current limitations of the ZKVM and our GPU hardware, we process the dataset in batches when using the CUDA backend of the ZKVM. This is necessitated by the limited VRAM on the GPU, which is quickly exceeded during proof generation, as evidenced by the capped batch sizes and runtime increases in Table 2, despite stable epoch counts. However, this batching does not affect the overall runtime of the experiment and allows us to scale the proof generation efficiently.

The ZKVM natively supports all primitive Rust types in a "no_std" environment. Initially, we performed arithmetic operations using the built-in `f32` floating-point types, which offer ease of use for rational arithmetic. However, we observed that adopting fixed-point arithmetic significantly increased the number of samples we could prove per batch when using the GPU. With fixed-point representation, the number of data points processed per batch increased substantially compared to floating-point representation, demonstrating notable efficiency gains. This improvement highlights the importance of carefully selecting numeric representations to optimize performance. The charts below provide an overview of the measurements collected during our study.

Table 2: Training Time and Metrics

| Exp | Batch | Layers | Epochs | Total Time (s) | Epoch Time(s) | Samples/s |
|---|---|---|---|---|---|---|
| 1 | 64 | [784, 64, 10] | 10 | 882.24 | 88.22 | 544.17 |
| 2 | 128 | [784, 128, 64, 10] | 10 | 1898.01 | 189.80 | 252.92 |
| 3a | 256 | [784, 256, 128, 64, 10] | 25 | 10344.87 | 413.80 | 116.00 |
| 3b | 512 | [784, 256, 128, 64, 10] | 25 | 10296.79 | 411.87 | 116.54 |

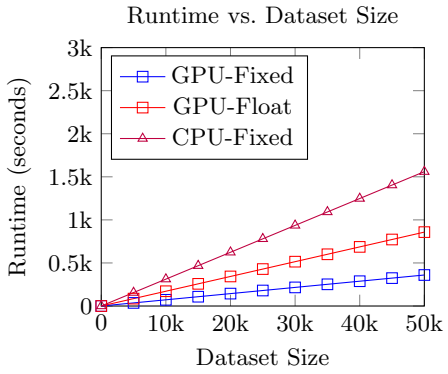



Fig. 1: Runtime vs. dataset size over different data types. We measure the performance of floating-point vs. fixed-point decimals on both the GPU and the CPU hardware.

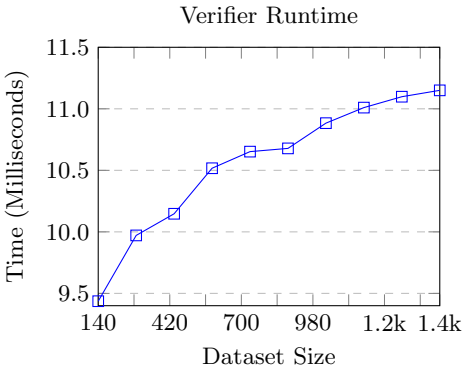
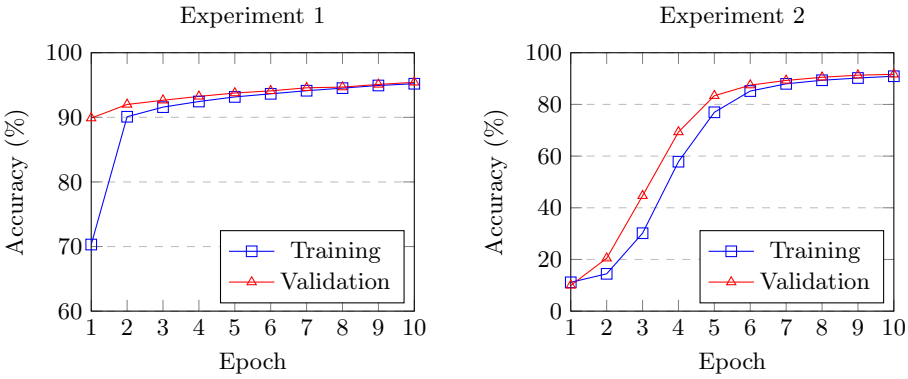


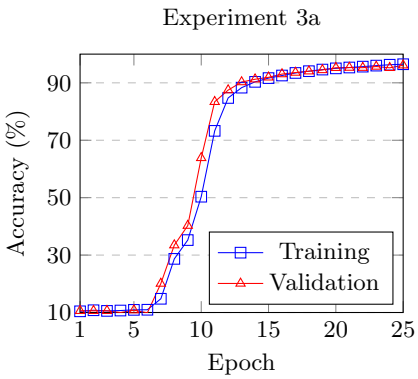

Fig. 2: Verifier runtime over a single GPU-Float batch divided into 10 mini-batches



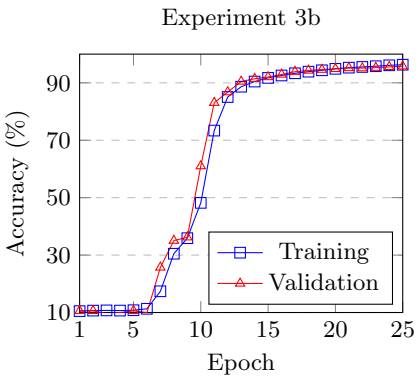

(a) Experiment 1 Accuracy




(b) Experiment 2 Accuracy




(c) Experiment 3a Accuracy




(d) Experiment 3b Accuracy

Fig. 3: Training and Validation Accuracy for Experiments 1, 2, 3a, and 3b

Figure 1 depicts the proving time as a function of dataset size. This approach explicitly demonstrates the linear growth rate of the prover runtime as dataset size increases. The graph suggests that the prover adheres to the protocol specified in Section 2.1. For GPU-based measurements, dataset size limitations due to hardware constraints are accounted for by dividing the dataset into a variety of different sized batches of elements and aggregating batch runtimes to obtain the total runtime for larger datasets.

Lastly, Figure 2 shows the runtime measurements for the verifier. We observe that verifier runtime increases in a roughly logarithmic fashion with respect to dataset size. While verifier runtime measurements exhibit more variability across multiple runs compared to the prover, the observed relationship between runtime and dataset size is consistent with the logarithmic growth expected under the protocol specified in Section 2.1. These results empirically support the protocol's expected verifier complexity, consistent with the ZKVM's design.

The verifier performs a relatively simple computation compared to the prover, which is one of the key highlights of this work. The verifier's task is to confirm the authenticity of the prover's computational integrity (CI) statement with minimal effort. Despite the prover's intensive computation, the verifier requires only a small fraction of work to validate the proof, even for large datasets.

In our experiments, we observe that the verifier maintains a runtime below 12 milliseconds in our experiments, exhibiting only modest growth as dataset size increases, with a slow logarithmic growth rate as the dataset size increases. This efficiency is achieved even when proofs are batched for larger datasets, introducing a constant factor for the consolidation of batched proofs. However, the overall runtime remains extremely low, underscoring the utility of this argument system. By working in cryptographic concert with a powerful prover, the verifier can reliably obtain complex and computationally expensive results with minimal effort.

### 5.2 Loading the Dataset

The RISC-Zero ZKVM includes a simulation of standard input and output, implemented as a communication channel between the verifier and prover (distinct from the cryptographic proof channel of the STARK protocol underlying the argument system). This channel allows the verifier to send data to the prover by serializing messages into byte-oriented format, which the prover then deserializes and *validates*.

In our early experiments, we initially used this channel to load the dataset into the prover. However, we observed that the deserialization process significantly degraded prover performance. The deserialization step, being part of the CI statement, requires the prover to verify that the serialized bytes conform to the expected type, leading to excessive computational overhead because this validation is paid for in ZKVM cycles. As a workaround, we can avoid the I/O channel entirely and instead directly embed the dataset as bytes into the RISC-V executable supplied to the prover.

The prover accesses this data via a simple pointer-cast operation, bypassing the costly deserialization step and eliminating its associated overhead [16], while also removing any security checks that the data has not been tampered with in flight. This approach is secure, as the only proof that the verifier will accept as correct is the one generated by faithful execution of the supplied binary. If the RISC-V program containing the embedded data is maliciously modified in any way, the verifier cannot accept any proof that fails to verify against their own correct version of the program.

By embedding the dataset into the binary, we achieve a substantial performance improvement, reducing the cycles required to load the dataset into the ZKVM's RAM from approximately 232 million cycles to just *640 cycles*. Figure 4 illustrates our optimized method and the resulting dramatic reduction in computational overhead. This technique appears prominently in [16], and shows

how to effectively manage resources within the boundaries of a ZKVM in order to reduce unneeded overhead.

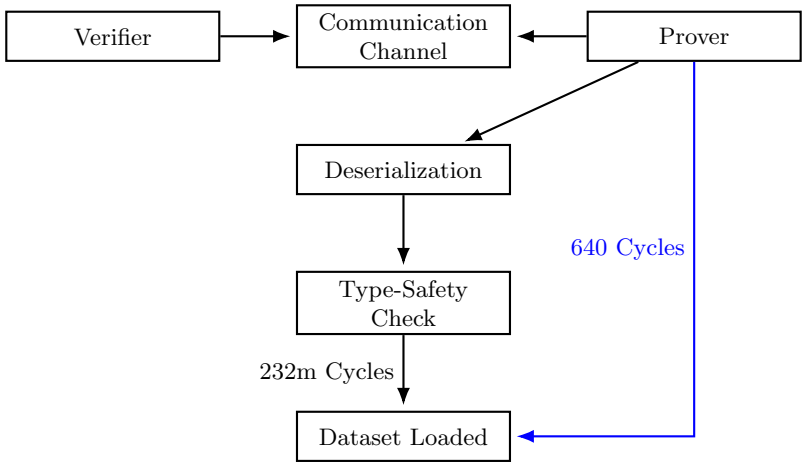



Fig. 4: Optimized Data-Handling Path in the RISC-Zero ZKVM

Rust is a modern programming language designed with a strong emphasis on memory safety and performance. Unlike languages such as C or C++, which require manual memory management, Rust enforces strict ownership and borrowing rules at compile time to prevent common memory-related errors, such as buffer overflows or use-after-free bugs. However, Rust also allows developers to bypass these safety guarantees using "unsafe Rust" when greater control over memory operations is required. While operations in "unsafe Rust" would be considered standard practice in languages like C, Rust's explicit delineation of safe and unsafe code provides additional clarity and helps ensure that developers are aware of potential risks.

The data embedding technique described earlier requires the use of "unsafe Rust," which in some contexts may raise concerns about application security. However, in our setting, we assume that the verifier constructs the binary with due diligence, ensuring both the correctness and memory safety of the embedded dataset. The use of fixed-point arithmetic, which is well-supported in Rust, further enhances performance while maintaining predictable memory layouts [17]. Since the dataset size is fixed and known in advance, it is represented as a contiguous array in memory rather than a dynamically allocated structure, ensuring safe memory access within the bounds defined by the verifier [19].

The prover has no control over the program's memory access patterns or execution flow; any attempt to modify the binary or exceed array bounds would invalidate the CI statement with high probability. Thus, as long as the verifier verifies the memory safety of the binary prior to distribution, this approach does not compromise the program's correct execution. Techniques such as pointer

casting in Rust, which allow for efficient type conversions, are employed in this workflow to optimize performance while adhering to the program's safety guarantees [18].

By embedding the dataset directly into the binary, communication between the prover and verifier is further simplified. The verifier's role is reduced to distributing the compiled binary to any party willing to execute it. Beyond this, no further interaction is required between the verifier and prover, apart from receiving the CI statement confirming that the binary was executed faithfully.

### 5.3 Validation

This section evaluates the performance of the multilayer perceptron (MLP) trained on the MNIST dataset using back-propagation. We focus on metrics such as accuracy, loss, and runtime characteristics to assess the effectiveness and efficiency of the training process. These results provide insights into how the network converges during training and the relationship between the cryptographic framework and the model's performance.

The primary metric for validation is accuracy, measured as the percentage of correctly classified samples in the validation dataset. Over multiple training iterations, the model consistently improves its accuracy, demonstrating the successful application of back-propagation within the cryptographic framework. We obtain various accuracy results which are illustrated in the tables and charts in this section. Ultimately we find ourselves attempting to balance the runtime of the prover with the desired level of accuracy.

We also measure the loss, calculated using the categorical cross-entropy function, to track how well the model fits the training data. As expected, the loss decreases over successive iterations, stabilizing at or above 94% validation accuracy in all test cases, as illustrated in Figures 3a and 3b

Additionally, we analyze runtime metrics, such as proof generation and verification times, across various dataset sizes. The proving time grows linearly with dataset size, as expected from the protocol specifications outlined in Section 2.1. Verification time remains logarithmic with respect to dataset size, with no more than 12 milliseconds required to verify the computation for almost all cases. These runtime metrics demonstrate the scalability and efficiency of the cryptographic proof system.

Overall, the results highlight that the MLP achieves competitive accuracy on the MNIST dataset while leveraging the computational integrity guarantees provided by the ZKVM framework. These metrics validate the feasibility of combining machine learning with cryptographic protocols for secure and efficient training.

### 5.4 Scaling Up

The limitation on batch size within the ZKVM applies only to a single ZKVM instance running on a single GPU. However, this limitation can be mitigated in

a distributed setting where multiple ZKVM instances, each running on separate GPUs, process different batches in parallel. In this setting, the total proving time becomes a function of the number of GPUs in the network, up to an optimal number of nodes. Each ZKVM processes a batch of a certain number of samples, determined as the optimal batch size for our hardware. Given a dataset of 60,000 samples and a network of approximately 100 GPUs, the entire dataset could be proven in significantly reduced time, enabling efficient scalability for large computations.

The highly parallel nature of the ZKVM, when leveraging GPU acceleration, allows for substantial reductions in proving time. By distributing the computation across a network of provers, the impact of per-instance batch size limitations is mitigated through parallel execution. Each GPU processes its assigned batch independently, generating proofs that can then be sent to a central verifier for aggregation.

### 5.5 Aggregating Models from Distributed Provers

In a distributed system of provers running back-propagation on multilayer perceptrons (MLPs), the verifier must aggregate the received models into a single, unified model. This step requires $O(c)$ work, where $c$ is the number of nodes in the network. Assuming that each proof has been successfully verified, the verifier can combine the disjoint models with high confidence in their correctness.

For an MLP, each model is represented by a set of learned weights $\mathbf{W}$ and biases $\mathbf{b}$ for all layers in the network. The aggregated model is obtained by averaging the weights and biases across all models received:

1. Averaging Weights:

$$\overline{\mathbf{W}}^{(l)} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{W}_i^{(l)}$$

where $n$ is the number of models, $\mathbf{W}_i^{(l)}$ is the weight matrix for the $l$-th layer of the $i$-th model, and $\overline{\mathbf{W}}^{(l)}$ is the averaged weight matrix for the $l$-th layer.

2. Averaging Biases:

$$\overline{\mathbf{b}}^{(l)} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{b}_i^{(l)}$$

where $\mathbf{b}_i^{(l)}$ is the bias vector for the $l$-th layer of the $i$-th model, and $\overline{\mathbf{b}}^{(l)}$ is the averaged bias vector for the $l$-th layer.

The final aggregated model, composed of the averaged weights and biases, can then be used for prediction or further analysis. This aggregation step ensures that the distributed computation retains coherence and produces a unified output, leveraging the efficiency and scalability of the distributed ZKVM framework.

## 6 Related Works

This section reviews recent advancements in integrating zero-knowledge proofs (ZKPs) with deep learning, focusing on the employed proof systems, data types, and performance metrics such as proof size, proving time, and verification time.

Recent advancements in verifiable computation have explored the use of circuit-based approaches to encode machine learning computations, including back-propagation, directly into cryptographic circuits. These methods leverage the inherent efficiency of hardware-level designs, akin to ASIC (Application-Specific Integrated Circuit) optimizations. For example, [20] and [21] demonstrate significant performance gains by constructing circuits specifically tailored for deep learning tasks. The circuit-based approach ensures that each computation step is verifiable with minimal cryptographic overhead during proof generation, making it an ideal choice for static and repetitive computations.

However, circuit-based methods come with significant trade-offs. The process of encoding computations into circuits is highly labor-intensive and requires domain expertise to adapt the circuits to new machine learning models or tasks. Additionally, the lack of flexibility makes these approaches less suitable for iterative workflows, such as hyperparameter tuning, model architecture adjustments, or experimenting with novel optimizers. Any change to the underlying computation typically necessitates a complete reimplementation of the circuit, significantly slowing down the development cycle. This rigidity makes these approaches more suitable for fixed-use cases but less viable for dynamic or rapidly evolving systems.

In contrast, our proposed framework leverages a virtual machine-enabled ZK-STARK design to prioritize flexibility and adaptability. Unlike circuit-based methods, our approach abstracts computations into a general-purpose virtual machine that can dynamically handle changes to the machine learning workflow without requiring extensive re-engineering. This flexibility allows for rapid prototyping and seamless integration into real-world systems where iterative updates are common. While circuit-based methods outperform virtual machines in raw performance, our approach provides a more practical and versatile solution for scenarios where adaptability is critical. The trade-off between performance and flexibility highlights the importance of aligning the choice of verifiable computation techniques with the specific requirements of the application.

### 6.1 zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training

Sun et al. present zkDL, a ZKP system tailored for deep learning training. It models both forward and backward propagation within neural networks as arithmetic circuits, effectively handling non-arithmetic operations. The system introduces zkReLU, a specialized protocol optimized for the ReLU activation function, addressing challenges in verifiable training due to its non-arithmetic nature. Evaluations on an 8-layer network with over 10 million parameters and a

batch size of 64 using the CIFAR-10 dataset demonstrate proof generation times of less than 1 second per batch update [20].

### 6.2 zkCNN: Zero-Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy

Liu et al. introduce zkCNN, a ZKP framework designed for convolutional neural networks (CNNs). The scheme employs a sumcheck protocol for proving fast Fourier transforms and convolutions with linear prover time, enhancing efficiency. It supports large CNNs, such as VGG16 with 15 million parameters and 16 layers, achieving proof generation in 88.3 seconds, a 1264× speedup over existing schemes. The proof size is 341 kilobytes, and verification time is 59.3 milliseconds [21].

### 6.3 Zero-Knowledge Proofs of Training for Deep Neural Networks

Abbaszadeh et al. propose a zero-knowledge proof of training (zkPoT) system that enables proving the correct training of a committed model on a committed dataset without revealing additional information. The system offers provable security and privacy guarantees, succinct proof size, efficient verifier runtime, and practical prover efficiency [22].

### 6.4 Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning

Weng et al. present Mystique, a system that facilitates efficient conversions for zero-knowledge proofs, with applications in machine learning. It addresses challenges in verifying non-linear functions and complex operations within neural networks, enhancing the applicability of ZKPs in machine learning contexts [23].

### 6.5 zkLLM: Zero-Knowledge Proofs for Large Language Models

Sun et al. extend their work to large language models with zkLLM, introducing tlookup for efficient handling of non-arithmetic tensor operations and zkAttn for the attention mechanism. This system ensures the privacy of model parameters and enables efficient zero-knowledge verifiable computations over large language models [24].

### 6.6 zkFL: Zero-Knowledge Proof-Based Gradient Aggregation for Federated Learning

Xu et al. develop zkFL, a system that integrates zero-knowledge proofs into federated learning by enabling verifiable gradient aggregation. This approach ensures the integrity of the training process across distributed nodes without compromising data privacy [25].

### 6.7 Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation

Xie et al. introduce Libra, a ZKP system achieving both optimal prover time and succinct proof size/verification time. It features a one-time trusted setup dependent only on the input size, not the circuit logic, and demonstrates significant efficiency improvements in proof generation and verification [26].

### 6.8 ZEN: Optimizing Compiler for Zero-Knowledge Neural Network Inference

Feng et al. present ZEN, an optimizing compiler designed to generate efficient zero-knowledge neural network inference schemes. It focuses on reducing constraint sizes in models like LeNet-Face-Large, addressing computational bottlenecks in ZKP-based neural network verification [27].

### 6.9 Verifiable Evaluations of Machine Learning Models Using zkSNARKs

South et al. explore the use of zkSNARKs for verifiable evaluations of machine learning models, ensuring the correctness of model predictions without revealing sensitive information. Their approach enhances trust in outsourced machine learning services by providing cryptographic guarantees of computation integrity [28].
Collectively, these works advance the integration of zero-knowledge proofs with deep learning, addressing challenges related to computational efficiency, proof succinctness, and the verification of complex neural network operations.

## 7 Conclusion

This work presents a robust and efficient framework for cryptographically proving and verifying the correct execution of complex machine learning training processes. We demonstrate significant advancements in the utility and performance of zero-knowledge virtual machines (ZKVMs) for machine learning applications, achieving significantly reduced proving times for computationally intensive tasks such as back-propagation in training multilayer perceptrons (MLPs) over large datasets. By leveraging modest yet powerful graphics hardware, achieving more than 10× improvement in some configurations compared to CPU-based approaches. Additionally, we describe a scalable framework for distributing proofs across a network of machines, enabling further reductions in proving time for cryptographically secure training.

Our design improves upon the state-of-the-art by achieving an $O(\log(n))$ verifier, reducing verification overhead to logarithmic complexity with respect to dataset size. This advancement makes it feasible for a verifying party to validate complex computations securely and efficiently, even when performed by an

untrusted third party. We also highlight the critical role of fixed-point arithmetic in reducing prover runtimes, showing how transitioning away from IEEE floating-point representations lays the groundwork for realizing increasingly sophisticated operations, including gradient computation and neural network training.

This work demonstrates that machine learning algorithms are well-suited to this form of non-interactive argument system, enabling efficient attestation of computational integrity for high-utility tasks. By introducing asymmetry into the computational process, we effectively leverage powerful hardware located in distributed and potentially untrusted environments. Our results establish a practical and scalable framework for verifying the correctness of machine learning models, providing confidence in their integrity and performance even when sourced from external, untrusted parties. This research paves the way for securely outsourcing advanced machine learning computations while preserving trust and reliability.

Argument systems of this variety facilitate what seems at first to be a counter-intuitive, yet intriguing result; by working in concert with a prover, the verifier obtains assurance of having learned the same model with only $O(polylog(n))$ work with respect to the dataset size [8]. As the dependence on outsourced ML hardware continues to grow, we anticipate the need for secure "co-processing" solutions of this nature to expand in kind. We believe this work shows that state-of-the-art ZKVM constructions are uniquely equipped to play an important role in the growth of privacy-preserving machine learning.

We believe this research represents a significant advancement in the development of secure and efficient cryptographic protocols for machine learning. By demonstrating the practicality of deep learning with ZKVMs, we establish a foundation for extending this framework to more advanced neural network architectures, such as convolutional neural networks (CNNs) and transformers. Future efforts will focus on these architectures, exploring their implementation within the ZKVM and leveraging the framework's scalability to handle increasingly complex and computationally demanding models.

While this work addresses training and verification phases with a high degree of efficiency and security, inference currently necessitates revealing both the model and the input data. Advancements in fully homomorphic encryption (FHE) offer promising directions for achieving fully private inference, enabling fully private inference. Building on recent work, such as [16], which integrates FHE with the RISC-Zero ZKVM, future research will aim to explore the feasibility and cost of integrating provable computations with FHE-enabled machine learning.

All experiments and the construction presented in this work are available in a GitHub repository [36]. The codebase can be readily executed with appropriate hardware, providing a resource for further experimentation and development. This research paves the way for secure, scalable, and practical cryptographic solutions in machine learning, with broad applicability across increasingly complex and advanced neural networks.

## References


1. Visual Capitalist. (n.d.). Visualizing the Training Costs of AI Models Over Time. Retrieved from https://www.visualcapitalist.com/training-costs-of-ai-models-over-time/
2. Moesif. (n.d.). The Ultimate Guide to AI Cost Analysis. Retrieved from https://www.moesif.com/blog/technical/api-development/The-Ultimate-Guide-to-AI-Cost-Analysis/
3. Google Cloud. (n.d.). AI and ML Perspective: Cost Optimization. Retrieved from https://cloud.google.com/architecture/framework/perspectives/ai-ml/cost-optimization
4. Google Cloud. (n.d.). Three Proven Strategies for Optimizing AI Costs. Retrieved from https://cloud.google.com/transform/three-proven-strategies-for-optimizing-ai-costs
5. Deloitte. (2023). Generative Artificial Intelligence. Retrieved from https://www2.deloitte.com/us/en/pages/consulting/articles/generative-artificial-intelligence.html
6. CloudThat. (n.d.). Cost Management for Generative AI Projects on AWS. Retrieved from https://www.cloudthat.com/resources/blog/cost-management-for-generative-ai-projects-on-aws
7. Google Cloud. (n.d.). Machine Learning Performance and Cost Optimization Best Practices. Retrieved from https://cloud.google.com/blog/products/ai-machinelearning/machinelearning-performance-and-cost-optimization-best-practices
8. Goldwasser, S., Rothblum, G., Shafer, J., Yehudayoff, A. (2020). ECCC - TR20-058. Retrieved from https://eccc.weizmann.ac.il/report/2020/058/
9. Ben-Sasson, E., Chiesa, A., Spooner, N. (2016). Interactive Oracle Proofs. Cryptology ePrint Archive, Paper 2016/116. Retrieved from https://eprint.iacr.org/2016/116
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046. Retrieved from https://eprint.iacr.org/2018/046
11. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
12. Weng, C., Yang, K., Xie, X., Katz, J., & Wang, X. (2021). Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 2103–2120).
13. Sun, H., Bai, T., Li, J., & Zhang, H. (2023). zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training. arXiv preprint arXiv:2307.16273.
14. RiscZero. (2023). About. Retrieved from https://www.risczero.com/about
15. LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). The MNIST Database of Handwritten Digits. Retrieved from http://yann.lecun.com/exdb/mnist/
16. Chen, W., & Research Partner, L2IV (@weikengchen). (2024). Tech Deep Dive: Verifying FHE in RISC Zero, Part I. Retrieved from https://l2ivresearch.substack.com/p/tech-deep-dive-verifying-fhe-in-risc
17. Rust Internals Forum. (n.d.). Fixed-Point Arithmetic Support in Rust. Retrieved from https://internals.rust-lang.org/t/fixed-point-arithmetic-support/6110
18. Rust by Example. (n.d.). Casting in Rust. Retrieved from https://doc.rust-lang.org/rust-by-example/types/cast.html

19. Rust Language Reference. (n.d.). Pointer Types in Rust. Retrieved from https://doc.rust-lang.org/stable/reference/types/pointer.html
20. Sun, H., Bai, T., Li, J., & Zhang, H. (2023). zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training. arXiv preprint arXiv:2307.16273. Retrieved from https://arxiv.org/abs/2307.16273
21. Liu, T., Xie, X., & Zhang, Y. (2021). zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. arXiv preprint arXiv:2106.11434. Retrieved from https://eprint.iacr.org/2021/673
22. Abbaszadeh, K., Pappas, C., Katz, J., & Papadopoulos, D. (2024). Zero-Knowledge Proofs of Training for Deep Neural Networks. arXiv preprint arXiv:2401.00162. Retrieved from https://eprint.iacr.org/2024/162
23. Weng, C., Yang, K., Xie, X., Katz, J., & Wang, X. (2021). Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 2103–2120). Retrieved from https://www.usenix.org/conference/usenixsecurity21/presentation/weng
24. Sun, H., Li, J., & Zhang, H. (2024). zkLLM: Zero Knowledge Proofs for Large Language Models. arXiv preprint arXiv:2404.16109. Retrieved from https://arxiv.org/abs/2404.16109
25. Xu, Y., Zhang, J., Zhang, Y., & Wang, X. (2023). zkFL: Zero-Knowledge Proof-based Gradient Aggregation for Federated Learning. In *2023 IEEE Symposium on Security and Privacy (SP)* (pp. 1234–1251). IEEE. Retrieved from https://ieeexplore.ieee.org/document/10535217
26. Xie, T., Zhang, J., Zhang, Y., & Song, D. (2019). Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. Cryptology ePrint Archive, Paper 2019/317. Retrieved from https://eprint.iacr.org/2019/317
27. Feng, Y., Zhang, J., & Sun, H. (2024). ZEN: Optimizing Compiler for Zero-Knowledge Neural Network Inference. In *30th USENIX Security Symposium (USENIX Security 24)*. Retrieved from https://www.usenix.org/conference/usenixsecurity24
28. South, T., Camuto, A., Jain, S., Nguyen, S., Mahari, R., Paquin, C., Morton, J., & Pentland, A. (2024). Verifiable Evaluations of Machine Learning Models Using zkSNARKs. arXiv preprint arXiv:2402.02675. Retrieved from https://arxiv.org/abs/2402.02675
29. Kang, D., Hashimoto, T., Stoica, I., & Sun, Y. (2022). Scaling up Trustless DNN Inference with Zero-Knowledge Proofs. arXiv preprint arXiv:2210.08674. Retrieved from https://arxiv.org/abs/2210.08674
30. Zhang, J., Xie, T., Zhang, Y., & Song, D. (2020). Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy (SP)* (pp. 859–876). IEEE. Retrieved from https://eprint.iacr.org/2019/1482
31. NotebookCheck. (2024). NVIDIA GeForce RTX 4070 Laptop GPU Benchmarks and Specs. Retrieved from https://www.notebookcheck.net/NVIDIA-GeForce-RTX-4070-Laptop-GPU-Benchmarks-and-Specs.675690.0.html
32. Weikeng et al. (2023). emptoolkit. Retrieved from https://github.com/emp-toolkit/emp-zk
33. Weng, C., et al. (2020). Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. Retrieved from https://eprint.iacr.org/2020/925.pdf
34. Zama. (2024). Concrete ML. GitHub. Retrieved from https://github.com/zama-ai/concrete-ml

35. Ben-Sasson, E., Chiesa, A., Forbes, M. A., Gabizon, A., Riabzev, M., & Spooner, N. (2016). On Probabilistic Checking in Perfect Zero Knowledge. *arXiv preprint*. Retrieved from https://arxiv.org/abs/1610.03798
36. Ray, D. (2024). capy2vML: Provably-secure differentially-private machine learning training. GitHub repository. Retrieved from https://github.com/drcapybara/capy2vML