Proof of Training: Obtaining Verifiable ML Models by Delegating Training to a Blockchain Network

Łukasz Krzywiecki[1][0000−0002−5326−3627] and
Gabriel Wechta[2][0009−0009−8560−5300]

[1] Department of Fundamentals of Computer Science, Wroclaw University of Science and Technology, Wroclaw, Poland, lukasz.krzywiecki@pwr.edu.pl
[2] Department of Cryptology, NASK National Research Institute Location, Warsaw, Poland, gabriel.wechta@nask.pl

Abstract. The recent rise of Bitcoin has sparked an unprecedented trend of enthusiasts acquiring expensive hardware for mining. This self-perpetuating race, driven by Bitcoin's PoW consensus mechanism, has led to the creation of massive computational centers dedicated solely to solving impractical hash inversions. However, these computational resources could be redirected toward more meaningful tasks if nodes were properly incentivized.
In this paper, we introduce Proof of Training (PoT), a novel consensus mechanism that offers two key advantages over previous approaches. First, it replaces wasteful computations with ML training. Second, by aligning the inherent distrust between nodes in blockchain networks with distributed model training, PoT not only achieves consensus but also produces a trained model along with proof that it was trained on the client-provided data.
PoT enables clients to hire the blockchain network to provably train arbitrary models using their provided datasets and architectures. Meanwhile, nodes that participate in training are rewarded with PoT cryptocurrency based on their computational contributions.

Keywords: Trustworthy ML · Delegated ML · Blockchain networks · Consensus mechanism.

1 Introduction

In recent years, machine learning (ML) and cryptocurrencies have entered the mainstream spotlight. Both have shown immense potential to transform industries, yet they also face growing criticism for their environmental impact [19,27]. While the computational resources required for ML model training are often justified by the functionality and value they deliver, the same cannot be said for blockchain networks. Consensus mechanisms such as Bitcoin's Proof of Work (PoW) [29] rely on solving cryptographic puzzles, where significant energy consumption serves no purpose beyond guaranteeing the immutability of the transaction ledger, making the process highly wasteful.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

However, recent research suggests that the long-term sustainability of cryptocurrencies must rely on consensus frameworks that preserve computational effort rather than eliminating it altogether [12,35,36]. Therefore, we argue that the solution to blockchain inefficiency is not to remove complex computation but to design a consensus mechanism that harnesses global computing power for a tangible, desirable and practical purpose while also remaining profitable for participants [3,16,21].

One of the key challenges in ML model creation is the significant computational power required for training models. Lately, even moderately complex models often demand more resources than most organizations can provide. As a result, many organizations must shift to Cloud Computing Services (CCS) such as Google Cloud Platform, Amazon AWS, or Microsoft Azure which provide significant computing resources often dedicated to ML training under some fees. However, in general, CCSs do not offer any provable guarantees that user-provided models have been trained honestly, that is without any malicious modifications to architecture or dataset, that includes data mislabeling, injections, dropping, etc.

1.1 Our Contribution

In this work, we introduce a novel consensus mechanism, Proof of Training (PoT), aimed at making cloud-like ML training verifiable and reducing wasteful computation. Unlike existing usability-focused consensus mechanisms, PoT emphasizes real-world functionality and leverages other established and trusted consensus mechanisms. We accomplish those goals through distributed protocol during which nodes independently create proofs that they performed training, while simultaneously verifying that their results align with those of other participating nodes. Each confirmation is recorded on the public blockchain, enabling client and third-party verification of the training process. Once a specified number of confirmations is reached, nodes proceed to the next round, which could be a different stage of the previous model or a completely fresh model. Consequently, PoT can be viewed as a secure multi-party computation alternative for *ML as a Service* and CCS platform providers.

Notably, PoT can facilitate model training from the beginning or verify the correctness of externally trained model, additionaly PoT in the context of blockchain networks ia a comprehensive consensus mechanism, allowing it to be substitute for any existing consensus mechanism in various blockchain networks.

In Section 2 we present ML assumptions and formal notation that allows us to define adn formalize malicious ML training. Section 3 is devoted to the PoT. We begin by highlighting the key aspect of PoT's design, that forces each participant to perform training by itself. Later we move to the design and protocol description. We also discuss each step and explain its impact on the the secuirty of the training.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

1.2 Related Work

Attacks on ML training. Data poisoning within the context of SVMs has been extensively examined in several studies [1,5,10]. Additionally, the authors of [28] investigated contaminated training data, shedding light on how such data can benefit attackers. Subsequent research delved into various other applications of data poisoning, including malware detection [14], sentiment analysis [13], collaborative filtering [6], and attacks on spam filters [11]. In [8], the authors proposed a general optimization framework for offline poisoning attacks, while another study [33] experimented with undermining license plate recognition models. Furthermore, [9] provided a comprehensive review of recent attacks and defenses spanning different data modalities. Lastly, the authors of [18] categorized a wide array of vulnerabilities associated with datasets in their study.

Consensus mechanisms. Nakamoto in his seminal work [29] proposed Proof of Work, the first well-adopted consensus mechanism for permissionless blockchain networks. In response to the recognized limitations of PoW, the authors of [22] introduced a consensus mechanism mitigating wasteful computation called Proof of Stake (PoS). Building upon advancements of PoS, the authors of [4] presented a mechanism called Proof of Activity (PoA) incentivizing nodes to remain online. This is in contrast to PoS where nodes tend to go idle over time. For a comprehensive overview of blockchain technology, particularly focusing on consensus mechanisms, refer to [3,35]. Latest attempts in shifting the attention of blockchain network creators towards useful computation resulted in new cryptocurrencies such as Primecoin [21] and in general frameworks [3,16].

Similar to this work. To our knowledge, only a handful of approaches use AI/ML training as a consensus mechanism foundation: Proof of Learning (PoL) [20,26] and Li's Proof of Training (PoT) [24]. PoL makes nodes compete to create the most effective model for a task and independently determine accuracy and parameters. However, PoL neglects key aspects of permissionless blockchain networks, including transaction time realization (heavily dependent on model architecture) and blockchain data storage (requirements escalate rapidly). Recent work [15] weakens PoL's integrity by presenting a spoofing attack. Li's PoT similarly incorporates competition to improve model quality by rewarding better-performing nodes. However, it gives nodes client-specified time limits for training, causing unpredictable block publication times. Furthermore, it assumes a complicated network architecture, unrealistic in decentralized blockchain networks.

Other works explore blockchain and trustworthy ML training intersection. Navarro et al. [30] proposed methods for trustworthy neural network training via blockchain. Lihu et al. [25] introduced a proof of useful work for AI on blockchain. Our PoT differs through stronger focus on adversarial training integrity and its verification mechanism designed to ensure training integrity. The concept of verifying ML training via partial re-execution has been explored previously, though often outside blockchain contexts. For instance, VerDE [2] employs a similar strategy where trainers share intermediate checkpoints and verifiers re-execute training segments with the largest weight changes.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

2 Malicious ML Training

Our discussion adopts a higher-level perspective, abstracting the ML training process from its specific implementation, thus allowing us to generalize our discourse across various types of models, with the only caveat being that a model use gradient-based learning methodologies. Other types of models may also be used with the PoT after defining an appropriate way of calculating a *training secret* (see Section 3). However, in this work, we limit our discussion to models utilizing gradient-based learning.

We use the notation $x \xleftarrow{\$} X$ to indicate that an element x is sampled uniformly at random from the set X. We use k, n, t to denote numbers from $\mathbb{N}^+$. We denote a sequence $(x_1, x_2, \ldots, x_t)$ as $(x_i)_1^t$ or simply as $x^{(t)}$. Let $I = \{i_1, \ldots, i_k\} \subset \{1, \ldots, n\}$ denote indices. We use A_I to denote subset of A, which elements are in I, that is $A_I = \{a_{i_1}, \ldots, a_{i_k}\} \subset A = \{a_1, \ldots, a_n\}$.

Clean Samples. Let $\mathcal{X}$, $\mathcal{Y}$ denote the set of input and output data, respectively. A process of training a parameterized function $f_S \colon \mathcal{X} \to \mathcal{Y}$ (e.g., a neural network) involves updating its parameters S (weights and biases) while minimizing the loss function L. L measures the difference between the output of the model and the desired output for a given input. Given a set of n samples $\mathcal{D} = \{(x, y)\} \subset \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ represents the input data and $\mathcal{Y}$ the corresponding set of labels (e.g., for the classification problem, $\mathcal{X}$ represents input vectors and $\mathcal{Y}$ the corresponding set of categories). Let $\mathcal{M} \subset \mathcal{D}$ denote the training dataset, and $\mathcal{D} \setminus \mathcal{M}$ denote a test set. The objective is to set the optimal parameters S such that for all fresh samples $(x, y) \in \mathcal{D} \setminus \mathcal{M}$ predicted values $y \leftarrow f_S(x)$ have the lowest possible error in terms of L.

Malicious samples. Let $\hat{\mathcal{X}}$ denote the set of input data corresponding to *incorrect* output data $\hat{\mathcal{Y}}$. Let $\hat{\mathcal{D}} = \{(\hat{x}, \hat{y})\} \subset \hat{\mathcal{X}} \times \hat{\mathcal{Y}}$ denote the set of samples, where $(\hat{x}, \hat{y})$ is a pair of *incorrect* data. Let $\hat{\mathcal{M}} \subset \hat{\mathcal{D}}$ denote the malicious samples used during the malicious training process. Thus $\hat{\mathcal{D}} \setminus \hat{\mathcal{M}}$ will be used to evaluate the *malicious accuracy*.

Regular training. Gradient descent-based training [31] is an iterative process of adjusting weights and biases of a model in the direction of the steepest descent of the loss function gradient $g \leftarrow \frac{1}{b} \nabla_{S_i} \sum_{(x,y) \in \mathcal{B}_i} L(f_{S_i}(x), y)$. Here, we consider a *regular training* (RT) with parameters $(\mathcal{D}, \mathcal{M}, f, S_0, L, (\xi_i)_1^t)$, where: $\mathcal{M} \subset \mathcal{D}$ denotes the training samples, f denotes a chosen architecture of the model, S_0 denotes the initial parameters of the model, L is a loss function, and $(\xi_i)_1^t$ denotes the randomness used. The process iteratively updates $S_{i+1} \leftarrow S_i - \eta \cdot g$, with learning rate η, sampling (x, y) from batches $\mathcal{B}_i \subset \mathcal{M}$ of size b (i.e. $|\mathcal{B}_i| = b$).

We denote $S_t \leftarrow \mathsf{RT}(f_{S_0}, (\mathcal{B}_i)_1^t, (\xi_i)_1^t)$, or $S_t \leftarrow f_{S_0}(\mathcal{B}^{(t)})$ to indicate that the model parameters S_t are obtained by applying the training sequence of t batches $(\mathcal{B}_i)_1^t = \mathcal{B}^{(t)}$, using randomness $(\xi_i)_1^t = \xi^{(t)}$ for all indices $i \in I = \{1, \ldots, t\}$, to the model architecture f with the initial parameters S_0. Moreover let $S_i \leftarrow f(S_{i-1}, \mathcal{B}_i, \xi_i)$ denote a single loop in RT, i.e. processing a single ith batch, where ξ_i denotes all randomness used in ith loop.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

Malicious training. Let $\hat{I} = \{i_1, \dots i_k\} \subset \{1, \dots, t\} = I$. (k,t)-*malicious* training involves the same process as RT, except that the k batches $\hat{\mathcal{B}}_i$ are taken from malicious samples $\hat{\mathcal{M}}$ for some indices $\hat{I}$. Thus we denote $\hat{\mathcal{B}}_{\hat{I}}^{(k,t)} = \mathcal{B}^{(t)} \setminus \mathcal{B}_{\hat{I}} \cup \hat{\mathcal{B}}_{\hat{I}}$ to indicate that some k out of t clean batches $\mathcal{B}_i \in \mathcal{B}^{(t)}$ were replaced with $\hat{\mathcal{B}}_j$ for $j \in \hat{I}$. We denote $\hat{S}_t \leftarrow \mathsf{RT}(f_{S_0^*}, \hat{\mathcal{B}}_{\hat{I}}^{(k,t)}, (\xi_i)_1^t)$, or $\hat{S}_t \leftarrow f_{S_0^*}(\hat{\mathcal{B}}^{(k,t)}, \xi^{(t)})$ to indicate that $\hat{S}_t$ is obtained from initial state S_0^* using the training sequence $\hat{\mathcal{B}}^{(k,t)}$ with k malicious batches and the randomness sequence $\xi^{(t)}$.

Advantage of malicious training. Let $S_t \leftarrow f_{S_0}(\mathcal{B}^{(t)}, \xi^{(t)})$ and $\hat{S}_t \leftarrow f_{S_0^*}(\hat{\mathcal{B}}^{(k,t)}, \xi^{(t)})$ denote clean and malicious training, respectively. Accordingly, let $(x, y) \xleftarrow{\$} \mathcal{D} \setminus \mathcal{M}$ and $(\hat{x}, \hat{y}) \xleftarrow{\$} \hat{\mathcal{D}} \setminus \hat{\mathcal{M}}$ denote sampling clean and malicious data, respectively.

Definition 1 ((α, β)-advantage of Malicious Training). *We say Malicious Training has (α, β)-advantage for:*

$$\alpha = \Big|\Pr[y \leftarrow S_t(x)] - \Pr[y \leftarrow \hat{S}(x)]\Big|,$$
$$\beta = \Pr[\hat{y} \leftarrow \hat{S}_t(\hat{x})].$$

The (α, β)-advantage is deemed successful when α is small and β is sufficiently large. We say that k malicious batches $\hat{\mathcal{B}}_{\hat{I}}$ are *injected* into the training process (instead of clean samples $\mathcal{B}_{\hat{I}}$) to achieve β accuracy on malicious data. From the adversary's perspective, achieving (α, β)-advantage implies that the selection of $\hat{\mathcal{B}}_{\hat{I}}$ and k yields satisfactory accuracy β on malicious data for the resulting model $\hat{S}_t$, simultaneously the resulting model remains α-close to the regular accuracy of S_t trained on clean data.

2.1 Distributed Deterministic Training and Verification

A potential countermeasure against malicious injections involves ensuring that the RT process is deterministically verifiable. Assume we have regular RT training sequence $S_t \leftarrow \mathsf{RT}(f_{S_0}, (\mathcal{B}_i)_1^t, (\xi_i)_1^t)$, which is a series of steps

$$S_1 \leftarrow f(S_0, \mathcal{B}_1, \xi_1), \dots, S_t \leftarrow f(S_{t-1}, \mathcal{B}_t, \xi_t).$$

We can record all the parameters $S_0, \dots, S_t$, batches $\mathcal{B}^{(t)}$ and randomness $\xi^{(t)}$. This preservation enables us to recreate the entire RT process later and verify whether the resulting model matches the one obtained during the initial training.

Distributed verifiable RT. Recreating the recorded RT process is indeed as complex as the initial training, requiring the repetition of all its steps over the recorded intermediate values S_0, $S^{(t)}$, $\mathcal{B}^{(t)}$, $\xi^{(t)}$.

2.2 Distributed Deterministic Training Requirements

Here we define assumptions about the training process that are essential for PoT:

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

- *Sequential*: Training of the model from the beginning to the end can be split into stages where each stage depends on the outcome of the preceding stage.
- *Deterministic*: The operation $S_i \leftarrow f(S_{i-1}, \mathcal{B}_i, \xi_i)$, for the same inputs, will always produce the same result. This in practice enforce usage of the same parameters during training e.g., the same seed, framework version etc.
- *Reproducible*: When the pseudorandom number generator used to obtain ξ is initialized with the same seed, it produces the same $\xi^{(t)}$ values. Due to this and the deterministic assumption, the training process can be replicated on different devices ensuring that the output $S_t \leftarrow \mathsf{RT}(f_{S_0}, (\mathcal{B}_i)_1^t, (\xi_i)_1^t)$ will remain the same.
- *Computationally expensive*: Computing $f(S_{i-1}, \mathcal{B}_i, \xi_i)$ is not trivial. This informal statement aims to limit possible f functions to practical training problems.
- *Well-defined*: f is commonly accepted and widely represented in ML libraries.

From now we are discussing only deterministic training, with randomness ξ deterministically reproduced on all involved nodes, we skip ξ in our notation. Thus, $f(S_{i-1}, \mathcal{B}_i)$ denotes $f(S_{i-1}, \mathcal{B}_i, \xi_i)$.

3 Proof of Training

Broadly speaking, the Proof of Training consensus mechanism allows unambiguous training verification, by constructing a solution to a consensus puzzle alongside recreating the recorded RT.

Training secret. The core building block of PoT is the capability to prove that a node indeed performed training from S_i to state S_{i+1}. This has to be achieved using a value that can be learned *only* during the training process. In the text, we call that value the *training secret* of stage S_i, and in equations, we denote it using ρ_i. Note that all functions depending only on S_{i+1} will not work (for example $\mathcal{H}(S_{i+1})$, where $\mathcal{H}$ is a cryptographic hash function) and the training process has to be somehow entangled, since S_{i+1} has to be publicly available for the protocol to work. Now we explain how such value can be derived.

When a batch $\mathcal{B}_i$ of size b is fed to the model, the training process iterates over inputs and makes predictions. Let $\tilde{y}_j$ be a prediction for input x_j. $\tilde{y}_j$ is compared to the expected output label y_j and an error $l_j = L(y_j, \tilde{y}_j)$ is calculated. At the end of the batch, all samples' errors are accumulated

$$g_i = \frac{1}{b} \sum_{1 \le j \le b} L(y_j, \tilde{y}_j).$$

Based on this accumulated error and learning rate η_i, the update algorithm is used to improve the model's weights w using

$$w \leftarrow w - \eta_i g_i.$$

Intermediate errors l_j are forgotten after accumulation. When $b > 1$, retrieving these errors becomes infeasible, even with access to g_i, $\mathcal{B}$, and both the old and

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

new values of w are available. l_js are essential for constructing the training secret, they must be treated as secrets. Consequently, PoT requires a batch size of at least two.

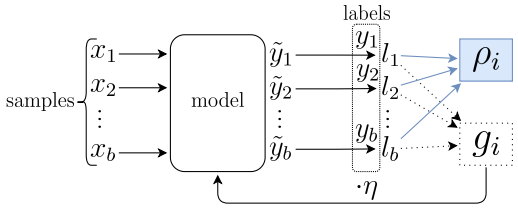

Fig. 1. Graphical representation of the process of incorporating the *training secret* creation into the basic training iteration.

Training secret is calculated in parallel to the training loop (see Figure 1) by hashing the concatenation of current l_j with the hash of the predecessor. This can be expressed by the following equation

$$\rho_i = \mathcal{H}(l_b \parallel \mathcal{H}(l_{b-1} \parallel \ldots \mathcal{H}(l_2 \parallel \mathcal{H}(l_1)) \ldots)). \tag{1}$$

If $\rho_{S_{i+1}}$ is derived during calculating $f(S_i, \mathcal{B}_{i+1})$, we denote that as

$$S_{i+1} \xleftarrow[\rho_{S_{i+1}}]{} f(S_i, \mathcal{B}_{i+1}).$$

Training as sequential steps. Training a single ML model can take anywhere from a few seconds to several days. Since it is crucial to ensure that leader election time remains predictable and depends on the resources a node possesses [35] clearly, a single training process cannot serve as the backbone of a consensus mechanism without modifications.

To solve this problem a PoT user, henceforth called an *employer*, before submitting a training request to the network, is responsible for dividing the training process into sequential steps, with the single-step size being constructed based on functions τ and Δ.

The function τ estimates the total computational effort required to train the entire model on dataset D. One, popular way of doing this relies on analyzing the number of multiply-and-add operations based on the model architecture. Importantly, that metric does not depend on the underlying hardware, resulting in a fair metric for all nodes, irrelevant of the hardware they possess. We utilize the explicit formula provided in [32] as τ.

The function Δ, on the other hand, partitions the dataset into a set $\mathbb{D} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_t\}$, where each $\mathcal{B}_i$ represents a separately downloadable batch of training data. Δ takes two inputs: $\tau(A, D)$, and the number of blocks published in the last 24 hours, similar to how Bitcoin's self-adjusting difficulty function operates (see equation (1) of [34]). Both inputs are used to ensure that the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

expected time for a single training step, $S_{i+1} \leftarrow f(S_i, \mathcal{B}_{i+1})$, remains roughly constant, aligning with the protocol-defined time interval T. It is important to note that such a design does not restrict the model owner's decision-making regarding the number of epochs and the batch size (except the aforementioned $b > 1$). This approach also solves the issue of scaling over time with increasingly large datasets and more complex models. In the case of a large computational load, the dataset would be divided into more batches, effectively resulting in more stages.

Employer's preparation. The employer is responsible for establishing a publicly accessible pull-push repository containing the model architecture A, partitioned dataset $\mathbb{D}$, a seed for deriving $\xi^{(t)}$, and C the number of confirmation votes that employer wishes to receive, all secured with a digital signature. Nodes, after downloading these resources, must verify their authenticity. Additionally, the employer must deposit the fee for node participation into a designated address that is non-retrievable by the employer, but which balance is publically accessible. This fee will later be partially distributed as rewards to nodes that successfully participate in the training process. Note that, although there is no strict authentication mechanism, as this is a permissionless blockchain network, what truly legitimizes the employer is the commitment of coins to the non-retrievable address. The employer then announces their intent to engage the network by broadcasting a message called a *training request*, which includes all necessary data.

3.1 PoT Design

Finally, we present the Proof of Training consensus mechanism. PoT is divided into two phases, namely:

- *Verifiable training*: The first phase is responsible for training the model and creating a *training declaration* message.
- *Block building*: The second phase is responsible for electing a leader based on *block header* and creating the final *wrapped block* that will be appended to the blockchain.

Both phases at the end utilize the PoS mechanism [22], while the second also utilizes the PoA mechanism [4]. Functions D^{td}_{PoS} and D^{bh}_{PoS} are difficulty functions for PoS mechanisms presented below in Algorithm 1 and Algorithm 2, respectively. The workflow of the PoT consensus mechanism is depicted in Figure 2.

Verifiable training phase (Algorithm 1). Steps 1, 2 — nodes independently select stages that meet their preferences. This part depends solely on the strategy of PoT's client implementation. Notably, a node may prefer to participate in the whole training process for some M even though it is not the most profitable strategy at the time to, for example, avoid download bottlenecks. Step 3 — by performing $f(S_i, \mathcal{B}_{i+1})$ node acquires *training secret* $\rho_{S_{i+1}}$. Step 4 — *training declaration* contains:

- $\texttt{ID}_{td}$: Unique identifier of the *training declaration*.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

Algorithm 1 Verifiable training

1: Every node independently selects stage S_i of model M to train, based on overheard training requests and its own preferences.
2: Every node downloads the selected stage S_i and batch $\mathcal{B}_{i+1}$.
3: Every node performs $S_{i+1} \xleftarrow[\rho_{S_{i+1}}]{} f(S_i, \mathcal{B}_{i+1})$ and saves $\rho_{S_{i+1}}$.
4: Node creates *training declaration* message, where `rand` is a random 256-bit string.

$$\textit{training declaration} \coloneqq \{\mathtt{ID}_{td},\ \mathtt{ID}_M, \mathtt{ID}_{S_{i+1}}, \mathcal{H}(S_{i+1}), \mathtt{timestamp},\ \mathcal{H}(\rho_{S_{i+1}} \parallel \mathtt{rand}), \mathtt{rand}, \mathtt{coinstake}, \mathtt{pub\_key}\}$$

5: (*PoS*) Node calculates $h^{td} = \mathcal{H}(\textit{training declaration})$. If $h^{td} \leq D^{td}_{PoS}(\mathtt{coinstake})$ is true, it calculates $\sigma_{td} = \mathrm{Sign}_{\mathtt{priv\_key}}(\textit{training declaration})$, appends it to the message and publishes it, otherwise it increments `timestamp` and starts again.

- $\mathtt{ID}_M$, $\mathtt{ID}_{S_{i+1}}$: Used for model and training stage identification. To reduce message size, a condensed identifier $\mathtt{ID}_{\{M,S_{i+1}\}}$ can be employed (as we do in our implementation, refer to Section 3.1).
- $\mathcal{H}(S_{i+1})$: Allows nodes to verify if S_{i+1} held by them is equal to parameters held by the *training declaration* creator.
- `timestamp`: Used for PoS mechanism.
- $\mathcal{H}(\rho_{S_{i+1}} \parallel \mathtt{rand})$: Known as *secret commitment*. It serves as a zero-knowledge proof of knowing $\rho_{S_{i+1}}$. Note that $\rho_{S_{i+1}}$ is not included in the message. Also, $\mathcal{H}(\rho_{S_{i+1}} \parallel \mathtt{rand})$ reveals no information about $\rho_{S_{i+1}}$. At this point in the protocol, the only way to verify *secret commitment* is to derive $\rho_{S_{i+1}}$ individually by appending the included `rand` to $\rho_{S_{i+1}}$ and hashing them. $\rho_{S_{i+1}}$ will be revealed in Step 2 of Algorithm 2, after which everyone can verify the *secret commitment*.
- `coinstake`: The abbreviation for the coinstake transaction. The idea is borrowed from [22]. This value represents the cumulative age of all coins involved in the coinstake transaction. Coin age determines the difficulty of D^{td}_{PoS}. When a block is published, the age of all coins in the coinstake transaction resets to zero and are back in the possession of the issuer.
- `pub_key`: Public key corresponding to private key `priv_key` used in the signature. Allows verification of σ_{td}.

Step 5 — this step provides resilience against Sybil attacks [35] in the same manner as PoS achieves that. Also, PoS makes shared secret attacks, in which nodes silently cooperate and only one of them performs computation and shares results with others, infeasible. After publishing *training declaration*, a node can decide whether he wants to go back to Step 1 or continue to the next phase. If a node, during the PoS waiting mechanism, receives a block based on the stage that he is currently working on, he was late; he should dump the current state and start over.

Block building phase (Algorithm 2). Step 1 — C is the number of *training declarations* chosen by an employer and fixed for M. Note that the node

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

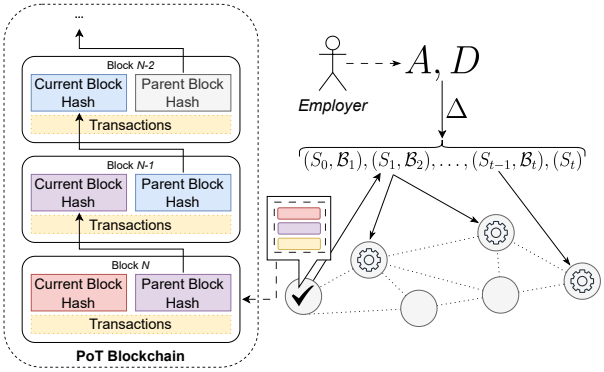

Fig. 2. Typical workflow of PoT consensus mechanism. An employer (right-hand size) publishes architecture A and dataset D. Using Δ, D is divided into $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_t\}$. Participating node selects stage i, downloads $(S_i, \mathcal{B}_{i+1})$, performs $S_{i+1} \leftarrow f(S_i, \mathcal{B}_{i+1})$ and creates *training declaration*. A gear in the node indicates that a node is currently performing training. After a sufficient number of *training declarations* is published, one of the nodes creates *block header*, which leads to the creation of *wrapped block* and (left-hand size) effectively a new block. The elected leader (denoted by a checkmark) broadcasts a new block and uploads a new resulting state to the employer.

has to know $\rho_{S_{i+1}}$ to perform this step. Fields $\texttt{ID}_{bh}$, $\texttt{ID}_M$, $\texttt{ID}_{S_{i+1}}$, `timestamp`, `coinstake`, `pub_key` have the same purpose as in a *training declaration*.

- `parent_block_hash`, `block_index`: The *block header* serves as a vessel for the *wrapped block*, containing mandatory blockchain block fields.
- $\rho_{S_{i+1}}$: The node publishes the *training secret* so that others, who did not compute it, can verify the correctness of *training declarations*. Note that a lazy node cannot intercept $\rho_{S_{i+1}}$ and immediately broadcast their own *training declaration* or *block header* since they have to wait in Step 5 of Algorithm 1 or Step 2 of Algorithm 2.
- $[\texttt{ID}_{td_1}, \texttt{ID}_{td_2}, \ldots, \texttt{ID}_{td_C}]$: Each *training declaration* serves as an independent vote, confirming that the training of the model indeed yields that result. Note that in order to save space, only indices of *training declarations* are sent. If a node does not have access to them, it should request them from some nearby node, similar to the original idea for sharing transactions' data in [29].

Step 2 — serves the same purpose as Step 5 in Algorithm 1. Step 3 — node uploads S_{i+1} (i.e., weights, biases) to the employer. The employer's repository should provide a means for resolving the rare situation in which more than one node will upload S_{i+1} simultaneously. Note that the employer does not have to store every S_{i+1} and may choose to only keep the latest one. Steps denoted with an asterisk (*) are meant to be triggered by the event of receiving a *block header* for any M, and to be performed outside of normal Algorithm 1 or Algorithm 2

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

Algorithm 2 Block building

1: Node after receiving C *training declarations* for the S_{i+1} of M verifies that all meet $D^{td}_{PoS}(\texttt{coinstake})$, have correct signatures, and checks their *secret commitments* against his own *training secret*. If sound, the node creates *block header*, where $[\texttt{ID}_{td_1}, \texttt{ID}_{td_2}, \ldots, \texttt{ID}_{td_C}]$ is a list of *training declarations* identifiers.

$$\textit{block header} \coloneqq \{\texttt{ID}_{bh}, \texttt{ID}_M, \texttt{ID}_{S_{i+1}}, \texttt{parent\_block\_hash}, \texttt{block\_index}, \texttt{timestamp}, \rho_{S_{i+1}}, \texttt{coinstake}, \texttt{pub\_key}, [\texttt{ID}_{td_1}, \texttt{ID}_{td_2}, \ldots, \texttt{ID}_{td_C}]\}$$

2: (*PoS*) Node calculates $h^{bh} = \mathcal{H}(\textit{block header})$. If $h^{bh} \leq D^{bh}_{PoS}(\texttt{coinstake})$ is true it calculates $\sigma_{bh} = \text{Sign}(\textit{block header})$, appends it to the message and publishes it, otherwise it increments `timestamp` and starts again.

3: Node uploads S_{i+1} to the employer.

4: * (*PoA*) Every node verifies C *training declaration* against $\rho_{S_{i+1}}$ included in *block header*. If valid, node acquires K stakeholders by passing $\mathcal{H}(\textit{block header})$ to *follow-the-coin*.

5: * Every online node checks whether it is one of K stakeholders. First $K-1$ stakeholders sign the hash of the *block header* and broadcast their signatures. Kth stakeholder extends the *block header* by creating a *wrapped block*, where $[\texttt{Tx}_1, \texttt{Tx}_2 \ldots \texttt{Tx}_t]$ is a list of network transactions and $[\sigma_1, \sigma_2 \ldots \sigma_{K-1}]$ is a list of $K-1$ stakeholders signatures. Kth stakeholder signs and broadcasts.

$$\textit{wrapped block} \coloneqq \{\texttt{ID}_{bh}, \texttt{Tx}_{\texttt{coinbase}}, [\texttt{Tx}_1, \texttt{Tx}_2, \ldots, \texttt{Tx}_t], [\sigma_1, \sigma_2, \ldots, \sigma_{K-1}]\}$$

6: * Every node after receiving *wrapped block*, checks stakeholder signatures and $\mathcal{H}(S_{i+1})$ against *training declarations* and if sound updates his local replica of the blockchain. Process for S_i is considered finished.

workflows. Steps 4, 5 — typical last PoA steps, tailored for PoT. Step 6 — the *wrapped block* created in Step 5 is appended to the blockchain, thus the global view of the blockchain is updated.

Discussion and security. The main difficulty of providing the service of multi-party verifiable ML model training was assuring that some number of independent nodes would truly train the model without cheating, for example by stating random results, replaying others' messages, or by Sybil attacks.

We argue that introducing *training declaration* message as an additional communication round to a typical consensus mechanism serves as a solution. This communication overhead is justified by the fact that every *training declaration* can be viewed as an independent vote stating "I, `pub_key`, have trained the model M using batch $\mathcal{B}_{i+1}$ from stage S_i to stage S_{i+1}, I know the *training secret* and I present $\mathcal{H}(\rho_{S_{i+1}} \,\|\, \texttt{rand})$ as the proof of that." The more votes S_i has, the more confident the employer can be that the result is correct, hence, the number of votes C is also a parameter of the protocol and the decision about this value belongs to the employer. Correctness of *training declaration* can be verified by every node that has already performed $S_{i+1} \xleftarrow[\rho_{S_{i+1}}]{} f(S_i, \mathcal{B}_{i+1})$, and later, after the block is appended to the blockchain, by anyone, because $\rho_{S_{i+1}}$ is revealed in the *block header*.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

Replay and Sybil attacks are prevented by the fact that every *training declaration* is sealed by the PoS mechanism. Additionally, since PoS checks are performed periodically, the process can run in the background, hence a node does not have to stay idle and can continue to train the next stage or move to another model. Note that forcing the network to append a block with an incorrect *training secret*, so as to perform data injection or just lazily participate in the protocol, is equivalent to performing a 51% attack.

After C *training declarations* are broadcasted to the network, any of the nodes that have already trained the model can start the *block building* phase. Note that since *block building* ends in PoA, there is no race between nodes to build the *block header* [4].

Another thing worth mentioning is that after C *training declarations* for stage S_{i+1} are broadcasted, it does not force the nodes to abandon a prepared *training declaration* which is waiting to meet $D^{td}_{PoS}(\texttt{coinstake})$ or quit training altogether. During PoS in the *block building* phase, nodes can actually increase their chances by taking C-length permutations of the total number of *training declarations* that they have heard. Therefore, the network will continue to perform the *block building* phase until Step 6 is performed by one of the nodes. Such design exponentially increases the probability of meeting $D^{bh}_{PoS}(\texttt{coinstake})$ in Step 2 and thus accelerates finalizing it. At this point in the protocol, the list of C *training declarations* can be thought of as PoW's *nonce* but from a very limited space [29]. This unfortunately opens up the possibility of producing more than one set of stakeholders for given S_{i+1} and M which may result in blockchain forks. Whether the advantages of this functionality outweigh its disadvantages will be determined in future work.

It is important to emphasize that a selfish node after receiving C *training declarations* in Step 2 may opt to delay and attempt to secure a more favorable set of stakeholders. This attack is mitigated by the aforementioned "permutation speed-up" that accelerates after every new *training declaration* gets published, making a selfish strategy extremely risky. Furthermore, if a node aims just to be selected as a Kth stakeholder, this does not provide much profit because PoT incentive design (due to space constraints, incentive design details cannot be included) keeps transaction fees at a low level.

Additionally, PoW's minting and verification times asymmetry (difficult to find, fast to verify) is also present in PoT. The *block building* phase is a time-consuming and depends on parameterized functions Δ, D^{td}_{PoS} and D^{bh}_{PoS} and through them is scaled to meet the expected time T. On the other hand, the block verification method consists of checking C *training declarations* against $\rho_{S_{i+1}}$ and checking K signatures against $\mathcal{H}(\textit{block header})$, thus block verification is a simple process that can be done in constant time.

Absence of the employer. Model training requests are naturally queued, with priority determined by the reward offered by the employer. However, there may be periods when no active training requests are available. To address this, the network includes a dummy model architecture and dataset, whose sole purpose is to provide a minimal framework for nodes to continue participating in the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

protocol and continue handling incoming transactions. In this case, the repository is provided by the organization responsible for maintaining the core implementation of PoT, and no rewards are offered for training the dummy model. Nodes' incentive during this period is limited to coinbase rewards and transaction fees. It is important to note that this situation should naturally resolve itself, as the cost of using the PoT network would decrease in the absence of active training requests.

Hiding the dataset. A critical issue with the PoT protocol is the requirement for employers to make their datasets and architectures publicly available. This imposes a significant barrier, particularly for private enterprises, as it may lead them to refuse to use PoT altogether. Despite this, the core argument of PoT, which asserts the correctness of model training backed by independent votes, necessitates that nodes access the dataset. One potential solution is the utilization of fully homomorphic encryption [7,23], although current high execution times render this approach impractical. Alternatively, employing zero-knowledge systems such as those proposed in [17] could potentially replace the proposed way of computing the training secret as in Equation (1). Nonetheless, it is worth noting that open-access datasets still constitute a significant portion of the datasets in use, especially among independent researchers and small to medium-sized companies, which are the primary target users of PoT services. Furthermore, it is common practice, even in large-budget projects, to begin training models on public datasets and later fine-tune them on private datasets. In such cases, the public portion of the training can be performed using the PoT.

Implementation. To show the feasibility of PoT we implemented a PoC implementation available at `https://github.com/gwechta/proof-of-training`.

4 Conclusion

In this paper, we introduced a novel approach for verifiable ML training in cloud-like environments by harnessing the consensus mechanism of a blockchain network. Our protocol leverages the intrinsic features of blockchain networks, thereby transforming the drawback of wasteful puzzle-solving into a PoT consensus mechanism, serving as a genuine alternative to PoW. We emphasize that our protocol can be run with almost any model and dataset, imposing only minimal restrictions. By distributing the verification process across the blockchain infrastructure, our approach ensures the utilization of dedicated nodes and enables training functionality with ample computational resources. Furthermore, we outlined a threat model and formalized malicious data manipulation attacks on ML models taking into account different forms of attacks as well as we have defined formal notion of describing adversary's advantage, for which our protocol serves as a solution.

References

1. Alfeld, S., Zhu, X., Barford, P.: Data poisoning attacks against autoregressive models. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

(2016)
2. Arun, A., Arnaud, A.S., Titov, A., Wilcox, B., Kolobaric, V., Brinkmann, M., Ersoy, O., Fielding, B., Bonneau, J.: Verde: Verification via refereed delegation for machine learning programs (2025). https://doi.org/10.48550/ARXIV.2502.19405, https://arxiv.org/abs/2502.19405
3. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of useful work (2017), https://eprint.iacr.org/2017/203
4. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: Extending bitcoin's proof of work via proof of stake (2014), https://eprint.iacr.org/2014/452
5. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 (2012)
6. Borrelli, F., Bemporad, A., Morari, M.: Predictive control for linear and hybrid systems. Cambridge University Press (2017)
7. Brand, M., Pradel, G.: Practical privacy-preserving machine learning using fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1320 (2023), https://eprint.iacr.org/2023/1320, https://eprint.iacr.org/2023/1320
8. Chen, Y., Zhu, X.: Optimal attack against autoregressive models by manipulating the environment. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 3545–3552 (2020)
9. Cina, A.E., Grosse, K., Demontis, A., Vascon, S., Zellinger, W., Moser, B.A., Oprea, A., Biggio, B., Pelillo, M., Roli, F.: Wild patterns reloaded: A survey of machine learning security against training data poisoning. ACM Computing Surveys **55**(13s), 1–39 (2023)
10. Cohen, A., Hasidim, A., Koren, T., Lazic, N., Mansour, Y., Talwar, K.: Online linear quadratic control. In: International Conference on Machine Learning. pp. 1029–1038. PMLR (2018)
11. Dean, S., Mania, H., Matni, N., Recht, B., Tu, S.: On the sample complexity of the linear quadratic regulator. Foundations of Computational Mathematics pp. 1–47 (2019)
12. Dotan, M., Tochner, S.: Proofs of useless work – positive and negative results for wasteless mining systems (2020), https://arxiv.org/abs/2007.01046
13. Dua, D., Graff, C., et al.: Uci machine learning repository (2017)
14. Dunning, I., Huchette, J., Lubin, M.: Jump: A modeling language for mathematical optimization. SIAM review **59**(2), 295–320 (2017)
15. Fang, C., Jia, H., Thudi, A., Yaghini, M., Choquette-Choo, C.A., Dullerud, N., Chandrasekaran, V., Papernot, N.: Proof-of-learning is currently more broken than you think (2022). https://doi.org/10.48550/ARXIV.2208.03567
16. Fitzi, M., Kiayias, A., Panagiotakos, G., Russell, A.: Ofelimos: Combinatorial optimization via proof-of-useful-work a provably secure blockchain protocol (2021), https://eprint.iacr.org/2021/1379
17. Garg, S., Goel, A., Jha, S., Mahloujifar, S., Mahmoody, M., Policharla, G.V., Wang, M.: Experimenting with zero-knowledge proofs of training. Cryptology ePrint Archive, Paper 2023/1345 (2023), https://eprint.iacr.org/2023/1345, https://eprint.iacr.org/2023/1345
18. Goldblum, M., Tsipras, D., Xie, C., Chen, X., Schwarzschild, A., Song, D., Madry, A., Li, B., Goldstein, T.: Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. IEEE Trans. Pattern Anal. Mach. Intell. **45**(2), 1563–1580 (2023). https://doi.org/10.1109/TPAMI.2022.3162397
19. Gschossmann, I., van der Kraaij, A., Benoit, P.L., Rocher, E.: Mining the environment – is climate risk priced into crypto-assets? (202),

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15

https://www.ecb.europa.eu/press/financial-stability-publications/macroprudential-bulletin/html/ecb.mpbu202207_3~d9614ea8e6.en.html
20. Jia, H., Yaghini, M., Choquette-Choo, C.A., Dullerud, N., Thudi, A., Chandrasekaran, V., Papernot, N.: Proof-of-learning: Definitions and practice (2021), https://arxiv.org/abs/2103.05633
21. King, S.: Primecoin: Cryptocurrency with prime number proof-of-work. https://primecoin.io/bin/primecoin-paper.pdf (2013)
22. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Self-Published Paper (2012)
23. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network (2021), https://arxiv.org/abs/2106.07229
24. Li, P.: Proof of training (pot): Harnessing crypto mining power for distributed ai training (2023), https://arxiv.org/abs/2307.07066
25. Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., Harvilla, M.: A proof of useful work for artificial intelligence on the blockchain (2020), https://arxiv.org/abs/2001.09244
26. Liu, Y., Lan, Y., Li, B., Miao, C., Tian, Z.: Proof of learning (pole): Empowering neural network training with consensus building on blockchains. Computer Networks **201**, 108594 (2021). https://doi.org/https://doi.org/10.1016/j.comnet.2021.108594
27. Luccioni, A.S., Jernite, Y., Strubell, E.: Power hungry processing: Watts driving the cost of ai deployment? (2023), https://arxiv.org/abs/2311.16863
28. Mei, S., Zhu, X.: Using machine teaching to identify optimal training-set attacks on machine learners. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 29 (2015)
29. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
30. Navarro, E., Standing, K.J., Dagher, G.G., Andersen, T.: Ensuring trustworthy neural network training via blockchain. In: 2023 IEEE 5th International Conference on Cognitive Machine Intelligence (CogMI). pp. 31–40 (2023). https://doi.org/10.1109/CogMI58952.2023.00015
31. Principe, J.C., Euliano, N.R., Lefebvre, W.C.: Neural and Adaptive Systems: Fundamentals Through Simulations. Wiley John + Sons, 1 edn. (December 7 1999)
32. Sevilla, J., Heim, L., Hobbhahn, M., Besiroglu, T., Ho, A., Villalobos, P.: Estimating training compute of deep learning models (2022), https://epochai.org/blog/estimating-training-compute, accessed: 2024-10-08
33. Song, C., Yi, Y., Zhou, T., Yang, J., Liu, L.: Undermining license plate recognition: A data poisoning attack pp. 72–78 (2023)
34. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials **18**(3), 2084–2123 (2016). https://doi.org/10.1109/comst.2016.2535718
35. Wang, W., Hoang, D.T., Hu, P., Xiong, Z., Niyato, D., Wang, P., Wen, Y., Kim, D.I.: A survey on consensus mechanisms and mining strategy management in blockchain networks. IEEE Access **7**, 22328–22370 (2019). https://doi.org/10.1109/ACCESS.2019.2896108
36. Álvarez, I.A., Gramlich, V., Sedlmeir, J.: Unsealing the secrets of blockchain consensus: A systematic comparison of the formal security of proof-of-work and proof-of-stake (2024), https://arxiv.org/abs/2401.14527

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97629-2_15