Secure Collaborative Learning in Mining Pool via Robust and Efficient Verification

Xiaoli Zhang[1*], Zhicheng Xu[1*], Hongbing Cheng[1§], Tong Che[2], Ke Xu[3],
Weiqiang Wang[4], Wenbiao Zhao[4], Chunping Wang[1], and Qi Li[3]
[1]Zhejiang University of Technology, [2]MILA, [3]Tsinghua University, and [4]Ant Group

***Abstract*—Recently, collaborative learning is proposed to amortize massive computation costs of highly sophisticated artificial intelligence (AI) tasks. To attract lots of participants, researchers investigate blockchains' economic incentives with proof of useful work (PoUW) consensus protocols to motivate substantial numbers of miners in a mining pool to complete AI tasks. However, participants might be untrusted and defraud rewards with as less as possible efforts. In the paper, we propose a robust and efficient proof of learning scheme called `RPoL` that enables pool managers to verify the training integrity of pool workers for secure pooled mining. Specifically, we devise an address-encoded model and employ a commitment-based secure sampling method to prevent malicious participants from abusing well-trained models or evading the sampling-based verification. Besides, we optimize `RPoL` via locality-sensitive hashing (LSH) to achieve communication-efficient verification while tolerating inherent reproduction errors of AI tasks. Furthermore, we conduct theoretical analysis and extensive evaluations. The results demonstrate that `RPoL` preserves high model performance against adversaries with acceptable costs and thus helps the pool win the mining competition among consensus nodes.**

***Index Terms*—Proof of learning, training integrity, collaborative learning, mining pool, PoUW consensus protocols**

I. Introduction

Deep neural networks (DNN) have revolutionized artificial intelligence (AI) recently and significantly impacted various tasks such as image recognition and robotics. Starting from the invention of GPT-3 [1], a major trend of deep learning is that both the models' sizes and the datasets are growing very fast. Training large-scale models like foundation models [2] requires massive computational resources, data, and engineering efforts. A recent report [3] stated that it might soon be out of reach for almost any single research institution in the world. Thus developing novel ways to train AI models in a *collaborative* manner [4], [5] by many parties and wisely assigning the credits to the parties who contributed to the training becomes an urgent research problem.

Public blockchain systems inherently provide attractive economic rewards to incentivize miners to join and run consensus protocols like proof of work (PoW) in the Bitcoin network [6]. While PoW is notorious for serious energy-wasting issues caused by massive hashing operations without real meaningful purposes [7], proof of useful work (PoUW) [8] has drawn widespread attentions from academia and industry [9]–[12]. It recycles energy originally consumed by PoW by executing some useful work on available resources, such as biomedical image segmentation [10] and model training [11]–[13]. These efforts build asymmetric puzzles based on non-convex DNN optimization. Namely, training a DNN model to a target accuracy is computation-intensive, yet testing its accuracy is relatively easy.

Since one miner is hard to complete extremely complex AI tasks with limited resources individually, *mining pools* like AntPool [14] in Bitcoin are proposed to combine many independent computation powers and increase the probability of gaining the mining rewards [15]–[17]. Usually, mining pools enable individuals to join with *no restrictions*. In this work, we leverage the idea of PoUW and mining pools to incentivize lots of (possibly untrusted) entities to collaboratively train complex models. Specifically, the pool manager acquires a DNN training task from blockchains and starts distributed learning among pool workers. Finally, it distributes mining rewards proportionally to workers' contributions, once their trained model is agreed in the network. However, participants in the mining pool are motivated by attractive economic incentives and may cheat for more revenue with less effort. Therefore, it is critical to verify the integrity of workers' local training to secure mining pools in PoUW-based blockchains.

Unfortunately, there are few studies addressing this issue. Several research efforts of AI-based PoUW protocols either only focus on linking adjacent blocks securely via functional encryption scheme [12] or aim to protect data privacy by homomorphic encryption [11]. None of them appear sufficient to address the above integrity problem inside mining pools. Another line of work that verifies computation integrity still cannot be directly applied in our scenario. Computation verification schemes (e.g., zk-SNARK based protocol [18]) and TEE (trusted execution environment) techniques like Intel SGX [19] are able to verify the integrity of DNN training yet incur huge computation overhead or are limited to CPU performance (compared with GPUs). Replay-based verification solutions check proofs by re-executing the computation [4], [20]. However, machine learning models are hard to be exactly reproduced due to hardware parallel computation or low-level libraries [21]. The difference between original training results and re-executed ones is called *reproduction error*, which exists even with identical model initialization and batching strategies [20]. Current replay-based solutions either cannot

*Both authors contributed equally to this research.
§Hongbing Cheng is the corresponding author.

tolerate such errors [4] or are not secure [20] due to their design flaws [22].

In this work, we propose a robust and efficient proof of learning scheme called RPoL. It enables pool managers to ensure the DNN training integrity of pool workers and thus secure the mining tasks against dishonesty. Following the idea of replay-based verification, we reproduce the trained model weights according to the initial model and relevant dataset for verification. As entirely retraining is expensive, *sampling* is applied to trade-off between the confidence of verification with the overhead.

Though promising, implementing the above idea with the guarantees of security, efficiency, and robustness at the same time is still a non-trivial task. First, shrewd workers in the pool may directly steal the global model and claim mining rewards in blockchains, consequently damaging the interests of all entities in the pool. Second, lazy workers may only perform the sampled training steps or use old results to evade the verification. Third, inherent reproduction errors of DNN tasks increase the difficulty of robust and efficient verification in the distributed networks. Because they have to be tolerated in verification by checking the distance between remote model weights and re-executed ones for robustness, which inevitably introduces massive communication overhead. Worse, the errors may vary across different GPUs or training epochs.

In the paper, to overcome the first issue, we devise an address-encoded DNN model that enables the pool manager to encode its own blockchain address into the model. Meanwhile, the address cannot be tampered with such that the mining rewards would be correctly distributed to the addresses embedded in the models of the winning blocks. As for the second problem, we employ a commitment-based delayed sampling scheme that exposes sampling decisions after workers attested to the completion of local training via commitments, similar to [4], [23]. Furthermore, we optimize RPoL by employing a fuzzy matching technique, *locality-sensitive hashing* (LSH), to reduce communication overhead while tolerating inherent reproduction errors of DNN training. More importantly, we propose an adaptive LSH setting strategy to automatically tolerate the constantly changing reproduction errors in multifarious training settings. The strategy practically achieves 0 false negatives, guaranteeing rewards for honesty. Our contributions can be summarized as follows:

- We propose RPoL, a robust and efficient proof of learning scheme for secure pooled mining. RPoL motivates individual workers with economic incentives to join a mining pool while efficiently guaranteeing their training integrity.
- RPoL is secure against shrewd pool workers. It develops an address-encoded DNN model to prevent workers from defrauding rewards via abusing the well-trained models in networks. Meanwhile, it implements a commitment-based secure sampling scheme to defeat biasing behaviors of adversaries.
- We further optimize RPoL by integrating locality-sensitive hashing to reduce communication costs with the robustness guarantees under DNN training reproduction errors. Additionally, the LSH setting is determined in an adaptive manner so as to automatically tolerate unpredictable reproduction errors of various DNN training tasks.
- We conduct theoretical analysis with an economic view that shows RPoL achieves high-confidence training assurance with sparse sampling. We perform thorough evaluations to demonstrate that RPoL preserves high model accuracy under various attack settings with acceptable costs, thus helping the pool win the mining competition among consensus nodes.

II. Preliminaries

A. Distributed Machine Learning

Distributed machine learning is a key tool to resolve large-scale machine learning problems in practice [1], [24]. Its distribution strategies include model and data parallelism with either synchronous or asynchronous model updating strategies. This work focuses on data-parallelism-based distributed learning with synchronous model updating in a mining pool. How to support other learning paradigms will be studied in the future.

In detail, there is one manager denoted as $\mathcal{M}$ and lots of workers denoted as $\mathcal{W}$. The entire dataset D is randomly shuffled and equally divided into sub-datasets D_w for each worker. The global model θ is trained iteratively. At each iteration, e.g., t-th epoch, a worker w ($w \in \mathcal{W}$) retrieves the latest global model θ^t along with hyper-parameters from $\mathcal{M}$, then computes the local model updates $\mathcal{L}_t^w$ over local sub-dataset. Finally, $\mathcal{M}$ updates the global model as:

$$\theta_{t+1} = \theta_t + \eta \sum_{w \in \mathcal{W}} \frac{|D_w|}{|D|} \mathcal{L}_t^w, \tag{1}$$

where η is the global learning rate, $|D|$ donates the data size. The next training iteration proceeds until the trained model achieves a target training/testing accuracy.

B. Reproducibility of Neural Network Models

Deep neural network models are hard to be completely reproduced due to the entropy growth of the training process [20]. A training step usually updates the weights via the algorithm of stochastic gradient descent [25] as follows:

$$\theta_{t+1} = \theta_t - \eta \nabla F(\theta_t) + \epsilon_t, \tag{2}$$

where ϵ_t represents the random noises that lead to inherent *reproduction errors* of DNN training. The errors are slight and arise from the hardware or low-level libraries like cuDNN which cannot be easily removed [21]. We will investigate the reproducibility of DNN models in detail later.

C. Locality Sensitive Hashing

The locality-sensitive hashing (LSH) technique hashes similar inputs into the same buckets with a high probability. It has been widely applied in various scenarios [26]. The definition of an LSH family is as follows [26]:

Definition 1. *A family* $\mathtt{H} = \{\mathtt{h} : X \to Y\}$ *is called* (d_1, d_2, p_1, p_2)*-sensitive for a distance measure* Dis *if for any* $v, q \in X$*,*

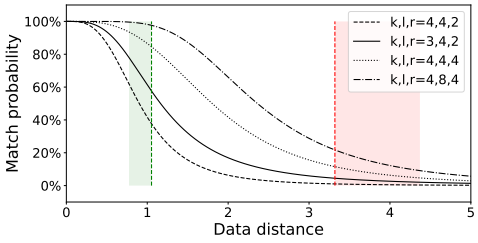


Fig. 1. The relationship between LSH matching probabilities and data distances under varied LSH parameters. The green and red lines show the upper and lower bounds of similar data and dissimilar data, respectively.

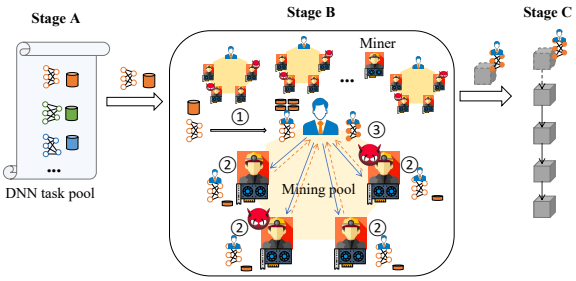


Fig. 2. System architecture and high-level workflow of RPoL.

- *if* $v \in \mathsf{Dis}(q, d_1)$ *then* $\mathtt{Pr}_{\mathtt{H}}[\mathsf{h}(q) = \mathsf{h}(v)] \geqslant p_1$,
- *if* $v \notin \mathsf{Dis}(q, d_2)$ *then* $\mathtt{Pr}_{\mathtt{H}}[\mathsf{h}(q) = \mathsf{h}(v)] \leqslant p_2$,

where $p_1 > p_2$ and $d_1 < d_2$. $\mathsf{Dis}(q, d_i)$ depicts the set of points whose distance from a point called q is not greater than d_i.

There are various LSH constructions using different distance measurement methods [26]. We choose the LSH construction based on p-stable distributions because it measures Euclidean distances that can evaluate the distance of high-dimensional DNN model weights. Briefly, there are l groups, each with k hash functions that map a vector $\mathbf{x}$ into a set of integers. The hash functions are $\mathsf{h}_{\mathbf{a},b}(\mathbf{x}) = \lfloor \frac{\mathbf{a}\cdot\mathbf{x}+b}{r} \rfloor$, where $\mathbf{a}$ is a vector randomly sampled from p-stable distributions, b is a real number selected uniformly from the range $[0, r]$, r is a tunable parameter. The LSH matching probability for vectors with Euclidean distances c is $\mathtt{Pr}_{lsh}(c, r, k, l) = 1 - (1 - p^k)^l$, where p is the collision probability for one hash function. In the end, LSH parameters are $\{r, k, l\}$. They should be carefully tuned based on suitable upper and lower bounds to control the matching probabilities for similar data and dissimilar data (see Fig. 1), respectively.

III. Problem Statement

A. System Setting

As shown in Fig. 2, we consider a PoUW-based blockchain system where both individual miners and mining pools are consensus nodes [9], [12]. At a high level, the consensus nodes in the system are identified by the *blockchain addresses* and *compete with each other* to propose blocks for revenue. They first pull DNN training tasks from a task pool in stage A, then train DNN models with a *training dataset* in stage B, and finally propose a block with the trained model in stage C (within a time limit). The block also contains the address of the proposer for further determining the ownership of mining rewards. Finally, the block including the model with the best generalization performance on a *test dataset* is agreed by other consensus nodes, which further triggers the mining rewards distribution to the block owner. Note that, to prevent the miner/mining pool from directly training models on the test dataset, the test dataset is published in the blockchain system only when a number of trained models are proposed. More details refer to the work [12].

In contrast to individual miners, a mining pool includes two types of entities: a pool manager $\mathcal{M}$ and a set of pool workers $\mathcal{W}$. Usually, pool workers have limited computation and communication resources and join the mining pool so as to obtain a relatively steady stream of revenue than individual mining [16]. The manager is responsible for coordinating pool workers $\mathcal{W}$ to solve sophisticated DNN tasks. Essentially, the manager starts an iterative distributed learning procedure among workers. It first initializes the training task, for example, randomly shuffles the dataset and divides them into sub-datasets for pool workers (see the step ① in Fig. 2). At each training iteration, e.g., t-th epoch, pool workers retrieve the latest global model and some auxiliary parameters from the manager, generate local model updates (i.e., the step ②), then send the results to $\mathcal{M}$ for model aggregation (i.e., the step ③). If the global model achieves the target training performance, $\mathcal{M}$ publishes the model and triggers the further consensus process. Finally, the mining revenues are sent to the pool manager via the blockchain address, once the proposed block is agreed. The manager further distributes rewards to the workers according to their contribution.

B. Threat Model

In a mining pool, we consider the threat model as follows:

- The pool manager $\mathcal{M}$ is considered to be *honest* from the perspective of gathering computation resources from pool workers for mining profits. *Inside the mining pool*, he faithfully coordinates the DNN tasks among pool workers without any malicious disruptions for the purpose of proposing a winning block. We emphasize that the pool manager might still take dishonest behaviors, similar to individual miners in PoUW-based blockchains. The rational incentives of pool managers include reporting more work than actually performed [9], stealing honest miners' models as their own [27], selfish mining [28], tampering with an old block to launch double spending attack [12], and so on. All those threats are addressed by existing PoUW consensus mechanisms [9], [12], [27], which is complementary to our focus.
- The pool workers $\mathcal{W}$ in the pool are *dishonest* and have economic incentives to defraud rewards with as less as possible efforts. For example, they may perform less training, directly fabricate model updates, or upload previous results (i.e., replay attacks). Worse, they may steal the collaboratively trained model and propose the block individually, *Thus, our security objective is to prevent dishonest pool workers from receiving more or equal revenues by investing less effort than honest*

ones. Additionally, the concerns regarding data privacy or data quality are not our focus and can be orthogonally addressed via existing privacy-preserving solutions [11], [29], [30].

IV. Design Overview

We propose a robust and efficient proof of learning scheme under mining pool environments. It ensures the training integrity of pool workers efficiently while tolerating inherent DNN reproduction errors. To verify the remote computation, we follow the line of duplicated execution [4], [20]. Briefly, given the initial model weights, final weights, and a dataset, a pool worker needs to offer a sequence of intermediate model weights to prove the training integrity (in step ② in Fig.2). Accordingly, the manager verifies the proofs by repeating each step of gradient descent from the initial weights to the final results (in step ③). Since full re-execution involves expensive computation overhead, the pool manager only checks a small portion of the training steps of pool workers to obtain tunable confidence of verification.

Challenges. Though the idea of duplicated execution for verification is not sophisticated, making it work with the guarantees of security, efficiency, and robustness is challenging:

• The global model may be abused by pool workers to claim rewards in PoUW-based blockchains. Such misbehaviors would harm the interests of all involved parties in the pool. Regarding this, the traditional countermeasure in PoW-based blockchains is to include the address of the pool manager in the hashing puzzles, yet fails to work in DNN training tasks. In the paper, we develop an *address-encoded DNN model* to secure the collaborative learning based mining. Here, when initializing the training task in the pool, the pool manager appends a non-trainable address-encoded mapping layer in front of the original model. The layer incurs no information loss for preserving model performance theoretically. More importantly, it is tamper-resistant and can be verified by consensus nodes in networks based on the blockchain address of the pool manager (see Sec. V-A).

• Directly exposing sampling decisions would lead to dishonest workers passing the verification by merely executing sampled steps or using old results. To tackle the issue, we adopt the *"commit-and-prove"* paradigm for secure sampling. Essentially, workers require to publish a succinct commitment demonstrating all training steps are executed, before receiving sampling decisions from the manager. Meanwhile, we employ a *mini-batch stochastic-yet-deterministic gradient descent* algorithm to defeat replay attacks. Randomly chosen data constitute a batch to compute the gradients as the traditional stochastic gradient descent algorithm does [25]. Especially, our data selection method not only makes the results of different training steps differentiable, but also is deterministic which can be checked by the manager (see Sec. V-B).

• The inherent DNN reproduction errors increase the difficulty of robust and efficient verification in the context of distributed networks. Intuitively, to tolerate *slight* reproduction errors for benign training, the verification requires checking distances between remote results with re-executed ones. However, transferring both the input and output model weights for each sample incurs huge communication costs, especially in wide-area networks. To optimize the process, we introduce the LSH technique to hash over model parameters and leverage *LSH-based fuzzy matching* to tolerate reproduction errors efficiently. In this way, a worker only sends the input model weights and the LSH hash of the output for each sample, consequently reducing communication overhead by nearly 50%. Furthermore, we provide an *adaptive LSH calibration strategy* that constantly tunes LSH parameters by taking both system performance and security into account. The design achieves experimentally 0 false negative, guaranteeing rewards for honesty (see Sec. V-C).

V. Design Details

In the section, we introduce the design details of `RPoL` that enable a pool manager to ensure the training integrity of pool workers in a secure, efficient, and robust manner.

A. Address-encoded DNN Model

To prevent pool workers from stealing the global model, we design an address-encoded mapping layer whose neural network weights are deterministic functions of the pool manager's address. Thus, the manager's identifier associated with the submitted model is publicly verifiable by all consensus nodes in blockchains. More importantly, it preserves the original information in the input data without any loss and resists address-replacing attacks.

In detail, each consensus node (e.g., the pool manager in a mining pool) generates an address-encoded mapping layer (AMLayer) based on its blockchain address and adds it in front of the model task before training. AMLayer is created as a one-layer neural network and is first initialized by a pseudo-random function with the address as the seed.

A main challenge of embedding a layer with non-trainable, predefined neural network weights is that the layer may cause information loss, making the upper layers unable to perform their jobs. To overcome this issue, we make the layer an invertible 1-1 mapping. More precisely, we set the layer as a residual layer and impose the constraint that AMLayer satisfies *Lipschitz continuity* with the theoretical guarantee of negligibly impacting performance [31]. Namely, for any input data pair x_1 and x_2, the inequality holds:

$$||\mathsf{AMLayer}(x_1) - \mathsf{AMLayer}(x_2)||_2 \leq c||x_1 - x_2||_2, \quad (3)$$

where $||\cdot||_2$ denotes the 2-Norm and $c < 1$ is a scaling coefficient. To achieve this, the consensus node adjusts the weights of AMLayer by estimating the maximum singular value $\tilde{\sigma}$ of the weights via the power iteration [32] and performing spectral normalization as follows:

$$\tilde{W} = \begin{cases} cW/\tilde{\sigma} & c/\tilde{\sigma} < 1 \\ W & c/\tilde{\sigma} \geq 1 \end{cases} \quad (4)$$

With a well-set AMLayer, the consensus node begins to train the model while keeping AMLayer fixed. Once the model achieves the target performance, it submits the model (along with c) to the blockchain network. Finally, other consensus nodes can verify the correctness of the model owner by re-calculating the AMLayer's weights based on the blockchain address of the entity that submitted the model. The mining rewards would be sent to the address encoded in the model.

B. Commitment-based Secure Sampling

Given an address-encoded DNN model, we describe how to defend against pool workers' various cheating behaviors. Briefly, a pool worker is required to train the model using the mini-batch stochastic-yet-deterministic gradient descent algorithm to avoid replay attacks. Then, it sends the local results with a commitment indicating that all training steps have finished. Finally, the manager exposes sampling decisions and verifies the proofs for samples, thereby preventing adversaries from evasion via sampling bias.

Mini-batch stochastic-yet-deterministic gradient descent algorithm. Before local training starts, the worker w retrieves from $\mathcal{M}$ the latest global model weights, e.g., θ^t, the training hyper-parameter ζ^t, and a nonce N_t^w, where t denotes the training epoch. In each training step, e.g., m, w uses a pseudo-random function PRF with the nonce N_t^w, i.e., $\mathsf{PRF}(N_t^w \times m + n) \mathsf{mod} |D_w|$, to select the n-th data from the sub-dataset D_w. Such a deterministic selection guarantees that identical data can be picked by $\mathcal{M}$ in verification. Finally, all selected data are fed into the gradient descent algorithm to optimize the model.

Proof and commitment generation. To construct proofs and the commitment for local training, w stores raw model parameters every i training steps as training proofs (denoted as proof) in each epoch. Notably, i represents the *checkpoint* interval, which is tunable to decrease storage overhead. At the end of the t-th epoch, w generates the commitment denoted as commit which satisfies two requirements: 1) commit includes the proofs of all checkpoints; 2) arbitrary proofs can be verified via commit. Therefore, commit can be constructed via a set of hash values of training proofs in order, or the root of a Merkle hash tree where leaf nodes are training proofs in order [33]. Here, we adopt the former construction as an example.

Verification. Once receiving commitments from pool workers, the manager samples the training checkpoints and requests the related proofs for all checkpoint samples from the workers for verification. For each sampled checkpoint, e.g., c_i, the requested proof (e.g., proof_{c_i}) is the input and output model weights for the checkpoint (e.g., $\{\theta_{c_i}, \theta_{c_{i+1}}\}$).

As an example, the verification for the sampled checkpoint c_i proceeds in two steps: 1) $\mathcal{M}$ checks the validity of model parameters in poof_{c_i} by calculating their hash digests and comparing with those in commit; 2) $\mathcal{M}$ re-executes the mini-batch stochastic-yet-deterministic gradient descent function based on the model initialized via θ_{c_i} and compares whether the distance between the re-calculated results (e.g., $\theta'_{c_{i+1}}$) and those (e.g., $\theta_{c_{i+1}}$) in proof_{c_i} is less than a predefined threshold (for the purpose of tolerating the DNN slight reproduction errors). Other sampled checkpoints are verified in the same manner. Finally, if all sampled checkpoints are successfully verified, the manager ensures the worker has honestly performed the sub-task in the epoch with high confidence.

C. LSH-based Optimization

We optimize RPoL by leveraging LSH-based fuzzy matching for reduced communication costs with the tolerance of DNN slight reproduction errors. We first illustrate how to apply LSH in verification for efficiency and robustness, then analyze how to set LSH configurations by taking system performance and security into consideration, and finally elaborate an adaptive calibration strategy to handle unpredictable changes of reproduction errors in the iterative learning process.

Applying LSH in verification. To avoid always transferring both input and output model weights for each sample, we leverage the LSH technique to generate the hash digests of proofs in the commitment. Essentially, for one sampled checkpoint, only the input model weights are sent to the manager. The manager then checks the consistency between the LSH value of the local outputs with that in the commitment, thus no longer always retrieving the output weights from the worker.

Note that the LSH technique only achieves probabilistic fuzzy matching as described in Sec. II-C and might lead to mismatched LSH digests between original weights and those with marginal reproduction errors. To prevent false negatives incurred by LSH, we develop a *double-check strategy* in which $\mathcal{M}$ requests one more time from w the raw model weights for the LSH-unmatched checkpoints and compares the distance of two weights. Such a double-check strategy effectively reduces false negatives (due to LSH fuzzy matching) yet at the cost of additional communication overhead. We emphasize that it is an important feature of mining pools in order to guarantee rewards for honesty. Fortunately, we find that the double-check strategy is rarely triggered for benign results in Sec. VII-D. Below we describe how to set LSH parameters in the system.

Setting LSH under considerations of RPoL performance and security. As shown in Fig. 1, LSH settings are determined by acceptable upper and lower distance bounds for similar data and dissimilar data, respectively. Here, we represent the former as α, i.e., *the distance threshold for tolerating most reproduction errors*. The latter is denoted as β, called *the distance threshold for differentiating unrelated models*. The inequality holds: $\alpha < \beta$. We identify the submitted results with reproduction distances in the range $[0, \beta)$ as benign results, while those in the range $[\beta, \infty)$ are spoofing weights.

Because of the LSH feature of fuzzy matching, there remain *LSH matching fail* for remote benign model weights with reproduction distances in $[0, \beta)$ (denoted as false negatives incurred by LSH) and *LSH matching pass* for remote spoofing weights with spoof distances in $[\beta, \infty)$ (represented as false positives caused by LSH). FNR_{lsh}/FPR_{lsh} are defined as the expectations of LSH matching fail/pass rates for honest/dishonest participants, respectively:

$$\begin{cases} FNR_{lsh} = \int_0^{\beta} p_{repr}(c)(1 - \mathsf{Pr}_{lsh}(c, r, k, l))dc, \\ FPR_{lsh} = \int_{\beta}^{\infty} p_{spoof}(c)\mathsf{Pr}_{lsh}(c, r, k, l)dc, \end{cases} \quad (5)$$

where $p_{repr}(c)$ and $p_{spoof}(c)$ are the probability density function (PDF) for varied reproduction and spoof distances, respectively. $\mathsf{Pr}_{lsh}(c, r, k, l)$ is the LSH matching probability for two vectors with distance c, $\{r, k, l\}$ are LSH parameters.

The goal of setting LSH is to make $\mathsf{max}(FNR_{lsh})$ and $\mathsf{max}(FPR_{lsh})$ as small as possible under the constraint of acceptable computational costs (denoted as K_{lsh}). To find $\mathsf{max}(FNR_{lsh})$ and $\mathsf{max}(FPR_{lsh})$, we consider a near worst case where all honest workers' reproduction errors are equal to α. The setting works under the assumption that the probability of reproduction errors falling between α and β is near 0. The assumption is rational via further experimental validation (see Fig. 5). Meanwhile, spoof distances are set to be equal to β (as the worst case). Consequently, we derive the following multi-objective optimization goal for LSH setting:

$$\begin{cases} \min\limits_{r,k,l} 1 - \mathsf{Pr}_{lsh}(\alpha, r, k, l), \\ \min\limits_{r,k,l} \mathsf{Pr}_{lsh}(\beta, r, k, l), \end{cases} \text{Constraint: } k \cdot l \leq K_{lsh}. \quad (6)$$

Given α and β, we can employ a simple additive weighting method [34] to obtain the optimal LSH parameters.

Adaptive strategy for LSH calibration. The estimation of α and β depends on the maximum DNN reproduction error. We conduct extensive experiments in Sec. VII-C to investigate reproduction errors with varied GPU models, training datasets, optimizers, and training epochs. From the experiments, we found that reproduction errors exist even for the same tasks on the same GPUs and increase as GPU performance improves. Interestingly, reproduction errors for tasks using i.i.d sub-datasets are near and follow *a normal distribution* on the same GPU hardware. Additionally, the errors vary for different epochs or optimizers, yet the above results still hold for the same optimizer and the same epoch. More detailed experimental results will be given in Sec. VII-C.

Given the above empirical evidence, α and β (used to set LSH for all sub-tasks) can be estimated by measuring the maximum reproduction error of a sub-task training on the i.i.d dataset in each epoch. As an example, α is set as the measured maximum reproduction error plus the standard deviation in the normal distribution. Since β is used to differentiate unrelated models, we set β by enlarging α, e.g., $\beta = x \times \alpha + y$, where x and y are tunable for the pool manager. The effectiveness of all relevant configurations is thoroughly validated in Sec.VII-D. The experiments demonstrate that with a proper magnification of α (near to the maximum error) to determine β, we achieve 0 false negative, guaranteeing the rewards for benign workers.

We devise an adaptive strategy for the pool manager to continuously calibrate LSH parameters, because reproduction errors of DNN training vary across different epochs. In detail, $\mathcal{M}$ equally partitions the whole dataset into $(n+1)$ i.i.d sub-datasets and maintains one sub-dataset locally, n is the number of pool workers. Before starting one epoch, $\mathcal{M}$ executes the sub-task twice on the current top-2 best-performant GPUs, so as to try to get as large reproduction errors as possible. Note that such GPU information can be obtained from pool workers' registration information. Then, $\mathcal{M}$ calculates the reproduction errors for all checkpoints of the current sub-task and further determines α and β according to the measured maximum error. Finally, $\mathcal{M}$ generates the optimal LSH parameters and distributes them to pool workers for producing LSH-based commitment.

VI. Security Analysis

We analyze the correctness and soundness of RPoL.

Theorem 1 (Correctness)**.** *For all training epochs (e.g., t), all model updates $\mathcal{L}^t$ and corresponding proofs $\mathcal{P}(\mathcal{L}^t)$ produced by honest workers, our verification procedure obtains* $\mathsf{Verify}(\theta^t, \mathcal{L}^t, \mathcal{P}(\mathcal{L}^t)) \rightarrow 1$.

Proof. The correctness of RPoL follows from the double-check strategy that guarantees verification success for honest training. ■

Theorem 2 (Soundness)**.** *For all DNN models θ, all dataset D, all training epochs (e.g., t), all model updates $\mathcal{L}^t_{\mathcal{A}}$ and proofs $\mathcal{P}(\mathcal{L}^t_{\mathcal{A}})$ generated by an adversary $\mathcal{A}$, the following inequality holds,*

$$\mathsf{Pr}(\mathsf{Verify}(\theta^t, \mathcal{L}^t_{\mathcal{A}}, \mathcal{P}(\mathcal{L}^t_{\mathcal{A}})) \rightarrow 1) < Pr_{err}, \quad (7)$$

where Pr_{err} is the soundness error.

Proof. For one sampled checkpoint, the probability that an attacker passes verification is $p^1_{attacker} \leq h_{\mathcal{A}} + (1 - h_{\mathcal{A}})Pr^{\mathcal{A}}_{lsh}$, where $Pr^{\mathcal{A}}_{lsh}$ is the probability of LSH matching pass for $\mathcal{A}$ and $Pr^{\mathcal{A}}_{lsh} \leq \mathsf{Pr}_{lsh}(\beta)$ (see Sec. V-C). $h_{\mathcal{A}}$ is the honesty ratio equal to the proportion of honestly trained checkpoints. Accordingly, for q randomly sampled checkpoints, the evasion probability (also called the soundness error Pr_{err}) is $p^q_{attacker} \leq (h_{\mathcal{A}} + (1 - h_{\mathcal{A}})\mathsf{Pr}_{lsh}(\beta))^q$. Thus, to maintain a given soundness error Pr_{err}, we can set the number of samples as:

$$q \geq \frac{\log Pr_{err}}{\log(h_{\mathcal{A}} + (1 - h_{\mathcal{A}})\mathsf{Pr}_{lsh}(\beta))}. \quad (8)$$

When $Pr_{err} = 1\%$ and $\mathsf{Pr}_{lsh}(\beta) = 5\%$, we need 3 and 47 samples for $h_{\mathcal{A}} = 10\%$ and $h_{\mathcal{A}} = 90\%$, respectively. ■

Actually, all workers join the mining pool because of economic benefits. We further formalize the soundness goal from an economic view and investigate what settings can diminish the benefits of attackers.

Theorem 3 (Soundness from an economic view)**.** *For all $h_{\mathcal{A}}$, there exists a valid q ($q \geq 1$) such that $G_{\mathcal{A}} \leq 0$ holds, where q is the number of randomly sampled checkpoints, $G_{\mathcal{A}}$ is the expected economic net gains of $\mathcal{A}$ for one submission.*

Proof. $G_{\mathcal{A}}$ equals the total rewards minus total computation and communication costs (including those incurred by the double-check strategy). We define the following parameters:

1) the reward for one successfully verified submission of each epoch is 1; 2) the computation costs for one honest submission is C_{train}; 3) the computation costs of spoofing attacks in an epoch is C_{spoof}; 4) the communication costs for one set of model weights is C_t. We derive $G_{\mathcal{A}}$ as:

$$G_{\mathcal{A}} \leq p^q_{attacker} - \{h_{\mathcal{A}} \cdot C_{train} + C_{spoof} + q \cdot C_t + q \cdot C_t \cdot (h_{\mathcal{A}} \cdot (1 - \mathsf{Pr}_{lsh}(\alpha)) + (1 - h_{\mathcal{A}})(1 - \mathsf{Pr}_{lsh}(\beta)))\}. \tag{9}$$

To keep $G_{\mathcal{A}} \leq 0$ always true, it suffices to ensure: $\mathsf{max}(G_{\mathcal{A}}) \leq 0$. We observe that, for any $\mathsf{Pr}_{lsh}(\alpha)$, $\mathsf{Pr}_{lsh}(\beta)$, and $h_{\mathcal{A}}$, the inequality $\frac{\partial G_{\mathcal{A}}}{\partial C_t} < 0$ always holds where $\frac{\partial G_{\mathcal{A}}}{\partial C_t} = -q \cdot (2 - \mathsf{Pr}_{lsh}(\beta) - h_{\mathcal{A}} \cdot (\mathsf{Pr}_{lsh}(\alpha) - \mathsf{Pr}_{lsh}(\beta)))$. Thus, $G_{\mathcal{A}}$ reaches the maximum when $C_t = 0$. We fix $C_t = 0$ in Eq. (9) as:

$$\mathsf{max}(G_{\mathcal{A}}) = p^q_{attacker} - h_{\mathcal{A}} \cdot C_{train} - C_{spoof} \leq 0$$
$$\Rightarrow \log_{p^1_{attacker}} p^q_{attacker} \geq \log_{p^1_{attacker}} (h_{\mathcal{A}} \cdot C_{train} + C_{spoof}) \tag{10}$$

Finally, we derive the lower bound for q:

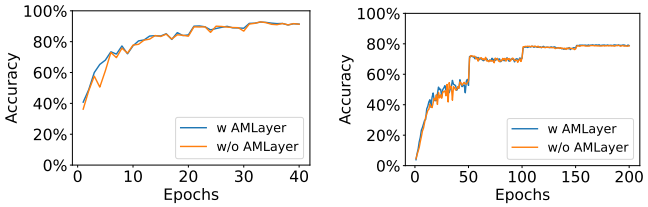
$$q \geq \frac{\log\left(h_{\mathcal{A}} \cdot C_{train} + C_{spoof}\right)}{\log\left(h_{\mathcal{A}} + (1 - h_{\mathcal{A}})\mathsf{Pr}_{lsh}(\beta)\right)} \tag{11}$$

As an example, we set $\mathsf{Pr}_{lsh}(\beta) = 5\%$. According to a report [7] that states the ratio of electricity costs to the total Bitcoin incomes for miners is 88% in 2022, we set the C_{train} as 0.88 while the reward of one successful submission is 1. Since C_{spoof} is negligible compared with C_{train} and negatively correlated with q, we set $C_{spoof} = 0$ for obtaining a larger q. In this case, if $h_{\mathcal{A}} = 10\%$, we require 2 samples; if $h_{\mathcal{A}} = 90\%$, we need 3 samples at least. Note that, when $q = 3$, the soundness error is about 74.12% which is relatively high according to Theorem 2. However, for an attacker, the probability of winning the mining rewards is only 0.74, while the computation costs are larger than 0.9 times those of one honest worker. Thus, such a sampling strategy effectively makes the net gains of attackers lower than honesty, shattering the attacker's motivations. ■

VII. Implementation and Evaluation

In this section, we prototype RPoL and estimate its effectiveness and efficiency via extensive experiments. In particular, the experiments include the following four aspects:

1) We evaluate the performance of the address-encoded design in terms of its influence on model performance and its ability against address-replacing attacks. (Sec. VII-B)
2) We estimate various factors affecting DNN reproduction errors, which provide important insights into configuring LSH as specified in Sec. V-C. (Sec. VII-C)
3) We evaluate how the adaptive LSH calibration strategy affects the performance and security of RPoL. (Sec. VII-D)
4) We evaluate the impact of RPoL on the ability to propose winning blocks for mining pools and assess the system overhead both inside mining pools and in the blockchain system. (Sec. VII-E)



(a) ResNet18 CIFAR10 (b) ResNet50 CIFAR100

Fig. 3. Testing accuracy curves of DNN tasks with and without AMLayer.

A. Implementation and Setup

Pool worker side: We implement all modules at the pool worker side using Python: 1) we use Pytorch 1.11[1] to implement the p-stable LSH with support of GPU acceleration for high computational efficiency; 2) we slightly modify the model training algorithms by performing stochastic-yet-deterministic data selection and adding the operations of saving checkpoints and calculating their LSH values at certain intervals.

Pool manager side: We prototype RPoL at pool manager side using Python. The distributed machine learning is implemented using Pytorch. We develop the following modules: 1) address-encoded DNN model generation; 2) LSH calibration (including the estimation of α and β and calculation of optimal LSH parameters); 3) commitment-based verification with double-check strategy. The communication channels between the manager and workers are established via TLS.

We build a prototype of a mining pool as one consensus node in the blockchain. It includes one manager and 10 workers to collaboratively perform the AI mining task, all equipped with NVIDIA GeForce RTX 3090 GPU (abbreviated G3090). We sample 3 checkpoints in each epoch to offset the benefits of attackers as demonstrated in Sec. VI. In the experiment, we mainly perform two DNN tasks, i.e., ResNet18[2] on CIFAR-10[3] and ResNet50 on CIFAR-100. Both CIFAR-10 and CIFAR-100 include images with the size of $32 \times 32 \times 3$. By default, the ResNet18 is trained for 40 epochs, while the ResNet50 (as a more complex task) is trained for 200 epochs. The batch size is 128. The optimizer is SGDM with the learning rate as 0.1, momentum as 0.9. The checkpoint interval at the worker side is 5. Unless otherwise stated, the above default settings are used in our experiment.

B. Performance of Address-encoded DNN Model

We evaluate whether AMLayer affects model performance, and meanwhile, whether it can effectively defend against address-replacing attacks. In front of each model, an AMLayer is added with the scaling coefficient c as 0.5. In particular, AMLayer is generated as a residual block with one convolutional layer. Its input channels are 3, output channels are 64, kernel size is 3×3, padding and stride are 1. We demonstrate the effectiveness of AMLayer via comparison experiments using the above two DNN tasks with and without AMLayer.

[1]https://pytorch.org
[2]https://doi.org/10.1109/CVPR.2016.90
[3]https://www.cs.toronto.edu/~kriz/cifar.html

TABLE I
PERFORMANCE EVALUATION OF AMLAYER WITH TWO TASKS: A(RESNET18 ON CIFAR-10), B(RESNET50 ON CIFAR-100)

Task		One-epoch training time	Accuracy	Accuracy (w Attack)
A	Origin	31.43s	92.67%	-
	AMLayer	32.54s	92.33%	$24.54\% \pm 5.13\%$
B	Origin	60.00s	79.38%	-
	AMLayer	60.75s	78.95%	$6.23\% \pm 2.92\%$

Impact on model performance: As shown in Fig. 3, the testing accuracy curves with and without AMLayer are near for both ResNet18 and ResNet50. Meanwhile, Table. I shows that the final model accuracy with AMLayer for ResNet18 and ResNet50 is 0.34% and 0.22% less than that without AMLayer, respectively. Such differences are negligible. More importantly, since all PoUW miners require to use the AMLayer in the consensus process, the small reduction in model accuracy is inevitably introduced for all miners and thus would not influence their competition in proposing winning blocks.

Additionally, we measure the one-epoch training time with and without AMLayer. As shown in Tab. I, the average one-epoch training time with AMLayer is only magnified by 3.5% for ResNet18 and 1.2% for ResNet50.

Ability against address-replacing attack: To evaluate the security of AMLayer, we replace the original AMLayer of a well-trained model with other *AMLayer*'s encoding other addresses (denoted as address-replacing attacks), and estimate the predictive accuracy of the attack models. In the experiment, we use some random addresses and generate 10 *AMLayer'* for each model.

Table I records the average inference accuracy of the attack models and presents the standard deviations. We can see that the accuracy drops significantly for both ResNet18 and ResNet50, reducing by about 67.79% and 72.72%, respectively. That means if an attacker wants to steal a well-trained model by tampering with the AMLayer using his address, he would lose the mining competition compared with the honest consensus node.

In conclusion, AMLayer only introduces a marginal reduction in model accuracy and is lightweight without significant inflation on training time. Besides, AMLayer can effectively resist address-replacing attacks.

C. Estimation of Reproducibility of DNN training

We analyze the factors, including GPUs, optimizers, training epochs, checkpoint intervals, and DNN models. All reproduction errors are calculated using the Euclidean distance of two model weights. Besides the two AI tasks on G3090, we estimate the impact of different GPU hardware on training reproducibility via three other widely used hardware environments: 1) Alibaba ECS gn7i-c8g1.2xlarge instance with A10 GPU, abbreviated GA10; 2) Alibaba ECS gn5-c4g1.xlarge with P100 GPU, abbreviated GP100; 3) Alibaba ECS gn6i-c4g1.xlarge with T4 GPU, abbreviated GT4. These GPUs are sorted in a descending order in terms of GPU performance of FP32 computing capacity[4], concretely, G3090 with 35.7TF, GA10 with 31.2TF, GP100 with 10.6 TF, and GT4 with 8.1TF.

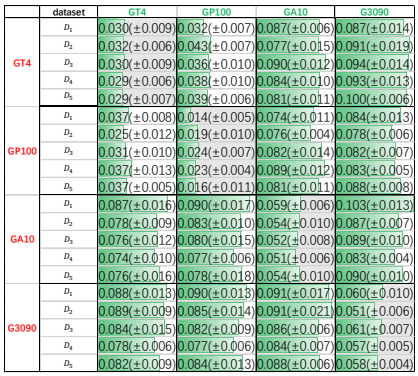


Fig. 4. Influences of varied GPU models and training dataset on reproduction errors when training ResNet18 on CIFAR10.

We first assess the influences of varied *GPU models*, *training dataset*, and *optimizers* on training reproduction errors in one epoch, then estimate the errors in different training epochs.

We randomly shuffle the dataset (CIFAR10) and divide them into five groups for every $10,000$ elements (denoted as D_i, $1 \leq i \leq 5$) to obtain i.i.d sub-datasets. We train ResNet18 on different GPU models using these sub-datasets and archive checkpoints for each task. The checkpoint interval is 5. Note that, for each task on each GPU pair (e.g., GT4 and GP100), we calculate the mean plus standard deviation of all checkpoints' reproduction errors as the maximum value. We define the same tasks by those training the same model with the same dataset.

Observing Fig. 4, we can see the following results. 1) Reproduction errors exist even for the same tasks on the same GPUs, meanwhile, these errors seem to slightly increase as GPU performance improves. 2) Reproduction errors are larger for the same tasks on the different GPUs, compared with those on the same GPUs. Interestingly, the reproduction error appears to be largest when using two different GPUs with the top-2 performance (i.e., G3090 and GA10 in our evaluations). 3) On the same GPUs (or GPU pairs, e.g., G3090 and GA10), reproduction errors for tasks using i.i.d sub-datasets are near, and more importantly, they follow a normal distribution. The latter feature is determined by the Kolmogorov-Smirnov test[5], which is used to statistically test whether samples follow a specific distribution. Moreover, we repeat the experiments using other popular optimizers (RMSprop[6], and Adam[7]) and under various training epochs. We find that the errors are different for different optimizers and also vary for various training epochs, yet the above results still hold inside each epoch with the same optimizer. Additionally, we evaluate the relationship between checkpoint intervals and reproduction

[4]https://cpu-compare.com/benchmark/fp32
[5]https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test
[6]https://pytorch.org/docs/1.11/generated/torch.optim.RMSprop
[7]https://pytorch.org/docs/1.11/generated/torch.optim.Adam

errors. The results show the errors increase linearly as the checkpoint interval grows. The results based on ResNet50 training task are similar to those using ResNet18. Due to space limitations, we omit the related experimental details.

D. Evaluation of Adaptive LSH Calibration

We measure how the adaptive LSH calibration affects the system performance and security via FNR_{lsh} (proportion of honestly trained checkpoints failing in *LSH matching*) and FPR_{lsh} (proportion of the spoofed checkpoints passing *LSH matching*), respectively. Besides the two tasks illustrated in Sec. VII-A, we also train ResNet18 on CIFAR-100 and ResNet50 on CIFAR-10 for a thorough evaluation.

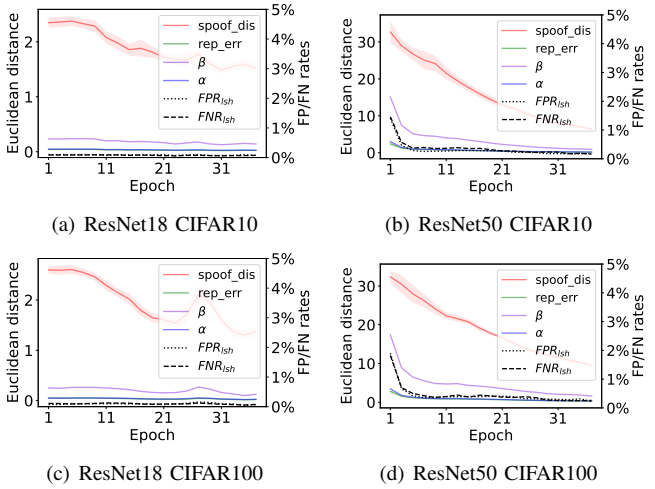
To simulate the procedure of LSH setting as described in Sec. V-C, we separate the dataset into two i.i.d sub-dataset, the one for estimating α and β on the manager's behalf, the other for simulating behaviors of honest/dishonest workers. At the manager side, we use G3090 and GA10 (i.e., the top-2 best performant GPUs in our experiments) to measure as large reproduction errors as possible. α is set as the mean plus standard deviation of all checkpoints' reproduction errors in one epoch. Meanwhile, β is 5α to identify malicious models as an example. Additionally, we use G3090 to verify workers' training and set the LSH parameters with $K_{lsh} = 16$. At the honest worker side, we train the model with the sub-dataset on GA10 that may produce the larger reproduction errors with that on G3090 (used in verification), so as to estimate the FNR_{lsh} in a near worst case.

At the adversary side, we consider a strong attacker `Adv` to evaluate the FPR_{lsh} in the pessimistic case. We assume the attacker simulates the existing DNN optimizers with little computation overhead [25]. It employs momentum to update models via weighted previous model updates heuristically. Essentially, it has honestly generated i checkpoints' model weights in one epoch, denoted as $\{\mathbf{c_1}, \mathbf{c_2}, \cdots, \mathbf{c_i}\}$, and tries to spoof weights for the $(i+1)$-th checkpoint as:

$$\begin{aligned} \mathbf{c_{i+1}} &= \mathsf{Spoof}(\mathbf{c_i}, \mathbf{c_{i-1}}, \cdots, \mathbf{c_1}) \\ &= \mathbf{c_i} + \frac{\sum_{j=0}^{i-2} K_j(\mathbf{c_{i-j}} - \mathbf{c_{i-j-1}})}{\sum_{j=0}^{i-2} K_j}, \end{aligned} \tag{12}$$

where K_j is the coefficient of the differences between $\mathbf{c_{i-j}}$ and $\mathbf{c_{i-j-1}}$. Here, we set the coefficients via exponential descent, i.e., $K_j = \lambda^j$, as an example. Note that $\lambda \in [0, 1]$ is an adjustable parameter. In the experiment, we first honestly train 4 different tasks for 40 epochs using GA10 under the default settings. Then, we spoof the last two third checkpoints in each epoch using the above attack strategy.

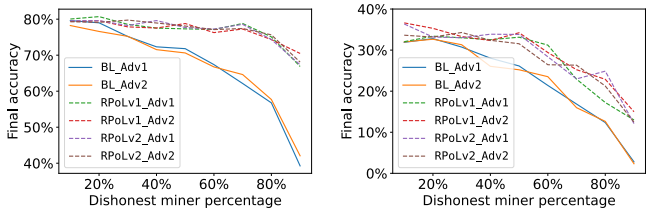
Evaluation results: Figure 5 records the maximum reproduction error and minimum spoof distance among all checkpoints in each epoch. We can see that the reproduction errors and spoof distances vary with different tasks or epochs. The spoof distances decrease as DNN models are trained toward convergence. More importantly, even with the above strong spoofing strategy, the measured reproduction errors using GA10 and G3090 are still far less than the spoof distances.



(a) ResNet18 CIFAR10 (b) ResNet50 CIFAR10

(c) ResNet18 CIFAR100 (d) ResNet50 CIFAR100

Fig. 5. The measured maximum reproduction errors, the spoof distances of `Adv`, the settings of α and β, and FPR_{lsh} and FNR_{lsh} for different tasks.



(a) ResNet18 CIFAR10 (b) ResNet50 CIFAR100

Fig. 6. Testing accuracy curves when training ResNet18 on CIFAR-10 and ResNet50 on CIFAR-100 for 40 epochs under different attack settings.

With the measured reproduction errors, we obtain the corresponding settings of α and β as shown in Fig. 5. Then, we derive the optimal LSH configurations with the theoretical guarantees of $\mathsf{Pr}_{lsh}(\alpha) = 95\%$ and $\mathsf{Pr}_{lsh}(\beta) = 5\%$ (see Fig. 1). Furthermore, we repeat the experiments 50 times and use the above LSH configurations to evaluate FNR_{lsh} and FPR_{lsh} among all honestly trained and spoofed checkpoints, respectively. As shown in Fig. 5, both measured FNR_{lsh} and FPR_{lsh} are lower than the theoretical value of 5%. More importantly, β is always higher than the measured maximum reproduction errors for both ResNet18 and RetNet50 tasks in varied epochs. Thus, our adaptive strategy rarely triggers the double-check strategy for honest submissions in practice and achieves empirically 0 false negative via β. As for security, even if FPR_{lsh} is not 0, our sampling-based verification will further reduce the evasion possibility of attackers. Future work to evaluate error rates of `RPoL` is necessary for highly complex models like GPT-3 [1].

E. Evaluation of RPoL Overall Performance

In the PoUW-based blockchain system, the probability of winning mining rewards increases as the model performance trained in a given time interval grows [11], [12]. Therefore, we evaluate how `RPoL` affects the mining competition via two evaluation metrics: 1) the *inference performance of the global*

model with and without RPoL under different proportions of adversaries in the pool; 2) the *one-epoch training time* of collaborative learning with and without RPoL in the pool. Note that the lower the one-epoch training time, the more training iteration in time duration, thus the more model performance or the faster block production speed. Furthermore, we measure the overhead of RPoL in terms of computation, communication, and storage resources.

In our experiment, there is one manager with a bandwidth of 10Gbps and 10 workers each with a 100Mbps bandwidth in a wide-area network setting. The models in the evaluation are ResNet18, ResNet50, and VGG16[8]. The datasets used include CIFAR-10, CIFAR-100, and ImageNet[9], all are equally distributed to pool workers. α and β are tuned via the method described in Sec. VII-D (i.e., $\mathsf{Pr}_{lsh}(\alpha) = 95\%$, $\mathsf{Pr}_{lsh}(\beta) = 5\%$, $K_{lsh} = 16$). According to the design, we set up two verification schemes: RPoL*v1* (with verification yet without using LSH), and RPoL*v2* (with verification using LSH). The comparison experiments vary with the different evaluation metrics, which will be described below. Additionally, we extrapolate some results for a setting of 100 workers to better analyze the overall overhead of RPoL.

Impact of RPoL *on the model inference performance:* We conduct comparison experiments by building two types of adversaries: Adv1 that directly submits the previous global model without local training in each epoch, and Adv2 that trains only 10% of the training steps in each epoch and fakes the model updates via the spoofing strategy as specified in Eq. (12). There are two baselines without verification: BL_Adv1 including the first type of adversaries and BL_Adv2 containing the second type of adversaries. All workers' results are aggregated in BL_Adv1 and BL_Adv2. In contrast, submissions from dishonest workers successfully detected by RPoLv1 or RPoLv2 are not aggregated into the global model. In the experiment, we conduct two tasks, i.e., ResNet18 trained on CIFAR-10 and ResNet50 on CIFAR-100, all trained for 40 epochs (due to resource limitation). We evaluate model accuracy with and without our verification under different settings where the proportion of dishonest workers in the pool varies from 10% to 90%.

As shown in Fig. 6, RPoLv1 and RPoLv2 under different attack settings always obtain better inference accuracy compared with the baselines without verification. Meanwhile, although RPoLv2 may incur false positives theoretically (caused by LSH), it experimentally obtains the same inference accuracy with RPoLv1. Besides, even in the setting with only 10% adversaries, RPoLv1 and RPoLv2 generate slightly better performant models than those without verification. Such a small advantage in term of model accuracy can still help the mining pool with RPoL win the mining competition against those without verification. Furthermore, the differences in model accuracy between RPoL and baselines increase as the number of attackers grows. This demonstrates that RPoL is significantly important for realistic mining pools, because workers would behave dishonestly if attackers can get rewards as well. Note that, the final accuracy is low for the task of ResNet50 in Fig. 6 because we only train the model for 40 epochs with limited resources. We believe that the accuracy will increase for more epochs.

[8]https://arxiv.org/abs/1409.1556

[9]https://doi.org/10.1109/CVPR.2009.5206848

TABLE II
ONE-EPOCH TRAINING TIME OF DIFFERENT SCHEMES

Task	# of workers	Baseline (Insecure)	RPoLv1	RPoLv2
ResNet50	10	307s	369s	348s
	100	37s	99s	78s
VGG16	10	282s	548s	429s
	100	66s	332s	212s

TABLE III
PERFORMANCE OVERHEAD WHEN TRAINING RESNET50 WITH IMAGENET FOR ONE EPOCH WITH 100 WORKERS

Overhead		Baseline (Insecure)	RPoLv1	RPoLv2
Comp.	$\mathcal{M}$	0 s	180s	240s
	$\mathcal{W}$	30s	30s	30s
Comm.	$\mathcal{M}\&\mathcal{W}$	8.8GB	62GB	35.6GB
Storage	$\mathcal{W}$	0.09GB	4.5GB	5.9GB
Capital Cost	-	$2.13	$8.49	$5.46

Tab. II, the one-epoch training time decreases as the mining pool size grows from 10 to 100, especially for the baseline version. This is because the subtask at the worker side takes less time as the mining pool size grows. For ResNet50 with a size of 90.7MB, the computational cost contributes to a large portion of the epoch time, hence the LSH optimization only slightly improves the overall performance. However, in the training of VGG16 with a size of 527MB, because the communication overhead dominates the total costs, RPoLv2 accelerates the one-epoch training by about 36% compared to RPoLv1 while achieving the same security goal as RPoLv1. This acceleration becomes more prominent as the number of workers grows, and directly facilitates the speed of block proposing in PoUW blockchain systems. Note that the performance can be further boosted with parallel processing on the manager side. In reality, in order to train a very large model, the difficulty level (test set accuracy) should be adjusted to accommodate a reasonable block production time, which is an interesting topic in future work.

Impact of RPoL *on the one-epoch training time:* To assess the overall costs, we simulate heavy AI tasks by training ResNet50 and VGG16 on ImageNet with $1,281,167$ images. All images are equally distributed to workers in the pool. We compare RPoLv1, RPoLv2, and *baseline* (insecure version without verification) under the settings of 10 and 100 workers.

RPoL *verification overhead:* We estimate verification overhead incurred by RPoL in mining pools consisting of 100 workers. Each worker trains ResNet50 on a subset of ImageNet.

Table III shows the evaluation results. At the pool manager

side, the computation overhead of RPoLv1 is higher than the insecure baseline due to the verification. Besides, the computation overhead of RPoLv2 is about 30% higher than RPoLv1, since $\mathcal{M}$ needs to train a local subtask twice for setting LSH in RPoLv2. Note that, the subtask is not useless work whose results can also be aggregated into the global model. At the pool worker side, the computation overhead is dominated by training. Other calculations like LSH-based hashing are very fast (e.g., about 250ms for calculating 50 checkpoints of ResNet50).

Compared with RPoLv1, RPoLv2 is more communication-efficient and about 42% lower than v1. Especially, if excluding the baseline communication costs, RPoLv2 optimizes the communication costs for verification by about 50%. The double-check strategy is not triggered in experiments. In terms of storage overhead, both RPoLv1 and RPoLv2 stores checkpoints for verification, thus incurring more storage overhead than the baseline. Compared with RPoLv1, the storage overhead of RPoLv2 is about 30% higher than v1 due to additional storage for LSH-related parameters.

We further estimate the capital cost of computation, communication, and storage in the mining pool. Assume the mining pool is deployed in the Alibaba cloud[10] where the price for the computation resource of GA10 is $\$1.33$/h (G3090 is not supported in the cloud), for the wide-area communication cost is $\$0.12$/GB, and for the storage cost is $\$5$/100GB per month. The total cost of RPoLv2 is about 35% lower than RPoLv1 since RPoLv2 reduces expensive communication costs. Note that the overhead for verification is acceptable compared with the mining rewards (near $\$133000$ in Bitcoin in 1/2023[11]).

In summary, RPoL helps the pool wins the mining competition via effectively detecting adversaries and preserving high model accuracy with acceptable costs. Additionally, LSH-based optimization reduces the total overhead by about 35% without affecting overall model performance.

VIII. Related Work

PoUW consensus. Recent advances apply useful tasks in PoUW consensus protocols of blockchains. REM [9] allows any entity to execute a useful workload in mining and leverages SGX to offer trustworthy reporting on CPU cycles for security. PoLe [12] studies a proof-of-learning based consensus protocol and prevents block tampering attacks via a functional encryption scheme. PoFL [11] investigates proof of federated learning in mining pools and proposes a privacy-preserving model verification scheme for data privacy. PoNAS [35] attempts to apply the neural architecture search as the PoUW workload in mining pools. Our work is complementary to theirs, as we focus on building training assurance of untrusted participants in a mining pool.

Verification of DNN tasks. Both zk-SNARK based protocol [18] and TEE (trusted execution environment) [36] like Intel SGX [19] can verify the integrity of general computation yet with huge overhead or subtle security issues [37]. Li et. al [38] check the quality of outsourced training via well-crafted training tasks, yet fail to ensure the training integrity in the iterative learning process. PoT [39] verifies the ownership of a well-trained model by carefully checking the properties of a chain of intermediate checkpoints saved during training, which might be insufficient in the collaborative learning scenario. PoL [20] is technically most similar to our work. It builds proof of learning and tolerates reproduction errors via slack parameters. However, PoL incurs huge communication costs for transferring all raw intermediate results. More importantly, PoL only verifies the largest model updates, which can be easily evaded by attackers with deliberately spoofing results. PoL is also demonstrated not secure by a recent work [22].

Secure and robust collaborative learning. Collaborative learning is fragile to model poisoning or backdoor attacks launched by bad participants [40]–[43]. TrustFL [4] requires each participant to equip with TEE so as to guarantee faithful executions. However, it only works for specific models without reproduction errors. Other research studies either detect anomalies via unsupervised learning [40], or perform byzantine-robust federated learning [42], or recover an accurate model by removing discovered anomalies [41]. In contrast, our work ensures the integrity of (possibly lazy) pool workers with various training tasks, which are complementary to the above solutions.

IX. Conclusion

We propose a robust and efficient proof of learning scheme called RPoL to verify the training integrity of workers in a mining pool under PoUW-based blockchain systems. To defeat various adversaries in the pool, RPoL develops an address-encoded DNN model and a commitment-based secure sampling method. Meanwhile, RPoL leverages LSH-based fuzzy matching for robust and communication-efficient verification. We conduct both theoretical analysis and extensive experiments. The results show RPoL can preserve high model accuracy against adversaries with acceptable overhead, thereby facilitating winning the mining competition for the pool.

In the future, we plan to leverage smart contracts to achieve fair exchange between the manager and workers inside the mining pool. Meanwhile, decentralized verification will be implemented to enable multiple workers to securely accelerate the verification in parallel.

Acknowledgement

This work was supported in part by NSFC under Grant 62072407, Grant 62132011, Grant 61825204, in part by Zhejiang Provincial Natural Science Foundation of China under Grant LQ23F020019, in part by the "Leading Goose Project Plan" of Zhejiang Province under Grant 2022C01086, Grant 2022C03139, in part by the National Key R&D Program of China under Grant 2022YFB2701400, and in part by Beijing Outstanding Young Scientist Program under grant BJJWZYJH012019100003011.

[10] https://www.aliyun.com/price

[11] https://www.blockchain.com/explorer/assets/btc

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[3] S. University, "Artificial intelligence index report 2023," 2023. [Online]. Available: https://aiindex.stanford.edu/wp-content/uploads/2023/04/HAI_AI-Index-Report_2023.pdf

[4] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu, "Enabling execution assurance of federated learning at untrusted participants," in *Proc. of IEEE INFOCOM*, 2020.

[5] L. Zhao, J. Jiang, B. Feng, Q. Wang, C. Shen, and Q. Li, "Sear: Secure and efficient aggregation for byzantine-robust federated learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3329–3342, 2021.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

[7] Digiconomist, "Bitcoin energy consumption index," 2022. [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption

[8] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of useful work," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 203, 2017.

[9] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "Rem: Resource-efficient mining for blockchains." in *Proc. of USENIX Security*, 2017.

[10] B. Li, C. Chenli, X. Xu, T. Jung, and Y. Shi, "Exploiting computation power of blockchain for biomedical image segmentation," in *Proc. of IEEE/CVF CVPRW*, 2019.

[11] X. Qu, S. Wang, Q. Hu, and X. Cheng, "Proof of federated learning: A novel energy-recycling consensus algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2074–2085, 2021.

[12] Y. Liu, Y. Lan, B. Li, C. Miao, and Z. Tian, "Proof of learning (pole): empowering neural network training with consensus building on blockchains," *Computer Networks*, vol. 201, p. 108594, 2021.

[13] C. Chenli, B. Li, Y. Shi, and T. Jung, "Energy-recycling blockchain with proof-of-deep-learning," in *Proc. of IEEE ICBC*, 2019.

[14] AntPool, "Technology brings infinite possibilities to the blockchain world," https://www.antpool.com.

[15] M. Rosenfeld, "Analysis of bitcoin pooled mining reward systems," *arXiv preprint arXiv:1112.4980*, 2011.

[16] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin mining pools: A cooperative game theoretic analysis," in *Proc. of AAMAS*, 2015.

[17] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden, "Incentive compatibility of bitcoin mining pool reward functions," in *Proc. of Springer Financial Cryptography and Data Security*, 2017.

[18] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2524–2540, 2021.

[19] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," *HASP@ ISCA*, vol. 11, no. 10.1145, pp. 2 487 726–2 488 370, 2013.

[20] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, and N. Papernot, "Proof-of-learning: Definitions and practice," in *Proc. of IEEE S&P*, 2021.

[21] "Reproducibility." [Online]. Available: https://pytorch.org/docs/stable/notes/randomness.html

[22] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, ""adversarial examples" for proof-of-learning," in *Proc. of IEEE S&P*, 2022.

[23] X. Zhang, H. Duan, C. Wang, Q. Li, and J. Wu, "Towards verifiable performance measurement over in-the-cloud middleboxes," in *Proc. of IEEE INFOCOM*, 2019.

[24] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[26] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[27] C. Chenli, B. Li, and T. Jung, "Dlchain: Blockchain with deep learning as proof-of-useful-work," in *Proc. of IEEE SERVICES 2020*, 2020.

[28] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. of ACM CCS*, 2016.

[29] Q. Song, J. Cao, K. Sun, Q. Li, and K. Xu, "Try before you buy: Privacy-preserving data evaluation on cloud-based machine learning data marketplace," in *Proc. of IEEE ACSAC*, 2021.

[30] B. Tang, H. Kang, J. Fan, Q. Li, and R. Sandhu, "IoT passport: A blockchain-based trust framework for collaborative internet-of-things," in *SACMAT*, 2019.

[31] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," in *Proc. of ICML*, 2019.

[32] R. P. Adams, J. Pennington, M. J. Johnson, J. Smith, Y. Ovadia, B. Patton, and J. Saunderson, "Estimating the spectral density of large implicit matrices," *arXiv preprint arXiv:1802.03451*, 2018.

[33] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. of IEEE S&P*, 1980.

[34] A. Afshari, M. Mojahed, and R. M. Yusuff, "Simple additive weighting approach to personnel selection problem," *International journal of innovation, management and technology*, vol. 1, no. 5, p. 511, 2010.

[35] B. Li, Q. Lu, W. Jiang, T. Jung, and Y. Shi, "A collaboration strategy in the mining pool for proof-of-neural-architecture consensus," *Blockchain: Research and Applications*, vol. 3, no. 4, p. 100089, 2022.

[36] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[37] Z. Zhang, X. Zhang, Q. Li, K. Sun, Y. Zhang, S. Liu, Y. Liu, and X. Li, "See through walls: Detecting malware in sgx enclaves with sgx-bouncer," in *Proc. of ACM AsiaCC*, 2021.

[38] P. Li, Y. Wang, Z. Liu, K. Xu, Q. Wang, C. Shen, and Q. Li, "Verifying the quality of outsourced training on clouds," in *Proc. of Springer ESORICS*, 2022.

[39] Y. Liu, K. Li, Z. Liu, B. Wen, K. Xu, W. Wang, W. Zhao, and Q. Li, "Provenance of training without training data: Towards privacy-preserving dnn model ownership verification," in *Proc. of ACM Web Conference*, 2023.

[40] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211*, 2020.

[41] X. Cao, J. Jia, Z. Zhang, and N. Z. Gong, "Fedrecover: Recovering from poisoning attacks in federated learning using historical information," in *Proc. of IEEE S&P*, 2023.

[42] X. Cao, M. Fang, J. Liu, and N. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. of NDSS*, 2021.

[43] Z. Zhang, K. Xu, Q. Li, X. Liu, L. Li, B. Wu, and Y. Guo, "Seccl: Securing collaborative learning systems via trusted bulletin boards," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 47–53, 2020.