ORIGINAL ARTICLE
Networks

Proof of Training (PoT): Harnessing Crypto Mining Power for Distributed AI Training

Peihao Li[1*]

[1]Co-founder at Intellichain Solutions KFT, 1054 Budapest, Hungary

Correspondence
Peihao Li, Intellichain Solutions KFT, Honvéd utca 8. 1. em. 2. ajtó, 1054 Budapest, Hungary
Email: peihao.li@kaust.edu.sa

Funding information

In the midst of the emerging trend of integrating artificial intelligence (AI) with crypto mining, we identify three major challenges that create a gap between these two fields. To bridge this gap, we introduce the proof-of-training (PoT) protocol, an approach that combines the strengths of both AI and blockchain technology. The PoT protocol utilizes the practical Byzantine fault tolerance (PBFT) consensus mechanism to synchronize global states. To evaluate the performance of the protocol design, we present an implementation of a decentralized training network (DTN) that adopts the PoT protocol. Our results indicate that the protocol exhibits considerable potential in terms of task throughput, system robustness, and network security.

KEYWORDS
proof of training, AI, blockchain, hash power, distributed network, consensus mechanism

1 | INTRODUCTION

1.1 | Motivations

Crypto mining is the process of creating and adding new blocks to a blockchain network through the use of various consensus mechanisms based on different resources (mining rigs, staked tokens etc..), with Proof of Work (PoW) being the most commonly used [21, 24]. In a blockchain network built on the PoW consensus mechanism, miners compete to create the subsequent valid block by being the first to solve a cryptographic puzzle, earning a reward for their efforts. The consensus algorithm, which integrates an appropriate rewards distribution system, is the core of a blockchain

[*]Equally contributing authors.

network. The most prominent blockchain projects in the crypto industry, such as Bitcoin (BTC) and Ethereum (ETH), uses the PoW consensus mechanism, with the latter recently shifted to Proof of State (PoS) [17]. According to bitcoin energy consumption analysis [5, 8], the yearly electricity consumption of Bitcoin mining exceeds that of United Arab Emirates (119.45 TWh) in 2021 and Sweden (131.79 Twh) in 2022. The majority of the energy consumed is dedicated to solving cryptographic puzzles. While this process enables trustless consensus, it does not offer any additional practical benefits. In fact, the apparent lack of a theoretical upper bound on the energy consumption of the PoW mechanism has raised global concerns, leading to the development of alternative consensus mechanisms, such as PoS, and changes in institutional policies. For instance, Tesla announced in 2021 that it would no longer accept BTC due to climate concerns [23].

Crypto mining is a rapidly changing industry. In 2022, Ethereum transitioned from the energy-intensive Proof of Work (PoW) consensus mechanism to an alternative called Proof of Stake (PoS), in response to growing environmental and energy concerns. Consequently, this change led to a substantial reduction in power demand, ranging from 99.84% to 99.9996% [14]. Ethereum's reduction in energy consumption could be comparable to the electrical power needs of a nation like Ireland or even Austria, whose advancement has a significantly positive impact on environmental sustainability. However, it has also resulted in a substantial amount of unused hashrate, equivalent to 1,126,674 GH/s [3], which now lacks a specific application. This brings the potential for miners to shift their computational resources from crypto mining to other areas like internet of things (IOT) and data services [2, 1]. This transition can remain fully within the blockchain space, by using these resources to run processes hosted on decentralized blockchain-based networks.

Meanwhile, with the integration of artificial intelligence (AI) into various sectors of the economy, the demand for computational resources to fuel this machine intelligence is experiencing rapid growth. Training a model like ChatGPT incurs expenses exceeding $5 million, and operating the initial ChatGPT demo costs OpenAI approximately $100,000 per day prior to the surge in its current usage [13]. Due to the extensive number of neural parameters and significant GPU hours required, the high computational demands of model optimization present substantial challenges for academic researchers and small-scale enterprises, limiting the widespread use of artificial intelligence technologies.

It is therefore unsurprising that an increasing number of crypto miners are exploring ways to utilize their existing computation infrastructures to contribute to the advancement of AI, redirecting its previously mining-focused computational resources for machine learning and other high-performance computing (HPC) applications, as demonstrated by Hive Blockchain. The company's long-term HPC strategy involves shifting from Ethereum mining to HPC applications, including artificial intelligence, rendering, and video transcoding, with an anticipated revenue generation of approximately $30 million per month.

Considering the developments mentioned, we believe that the emerging trend of combining and integrating these resources has the potential to significantly enhance the development process of AI tools in both technical and financial aspects. This would provide AI tool developers with a more affordable plan to monetize their innovations, including simplified training and marketplace access. Instead of exclusively commercializing their creations through major technology corporations, developers have the opportunity to contribute to the decentralization of technology by shifting their assets from centralized entities to a global commons. In the long run, this new direction is anticipated to yield significant societal benefits by optimizing resource allocation and minimizing costs.

1.2 | Challenges

Despite the considerable potential, the decentralization of software and hardware underlying AI remains in its early stages, due to the absence of well-developed consensus frameworks. Several pioneering studies have innovatively proposed new consensus schemes based on training machine learning models [4, 2, 6, 19, 10]. However, a notable gap exists between the theoretical foundations of these frameworks and their practical implementations. SingularityNET and FetchAI [4, 2] present a general high level framework but without technical details clearly shown. Coin.AI [6] further addressed this issue by proposing Proof of Useful Work (PoUW). However, they do not have customized AI training task, which can greatly reduce their network efficiency in serving clients, restricting their applicability to a limited range of business models. Authors in [19] further addressed this issue by incorporating features of customized clients. The design's limitation is mainly the inherent flaw in its blockchain structure, where the inclusion of test data within a block's body can rapidly consume the storage capacity of consensus nodes.

While Proof of Work (PoW) has been proven to be quite secure and effective since the launch of Bitcoin and Ethereum, an industry-level consensus mechanism explicitly designed for decentralized AI training remains absent. In general, we identify the following major challenges currently hindering the progress and realization of a decentralized AI utility network:

1. **Reliable validation mechanism.** Although resource consuming, PoW exhibits favorable time complexity for validation, ensuring efficient processing within the system. Upon mining a block, the network can efficiently verify its validity and append it to the local chain with ease. Another benefit of PoW is its determinacy in the global state, which guarantees that if a node is honest and abides by the complete set of rules within the system, it will consistently achieve the same state at a specific timestamp, consequently validating the system with confidence. However, in the context of decentralized machine learning, it is inherently challenging to ascertain whether a miner has genuinely performed its task as required. This is because when using different GPUs to perform the same AI training task with the same optimizer and dataset, it is still possible to obtain completely different results. Factors such as parallelism, random seeds, and rounding errors can all lead to differences in results, thus posing significant challenges for implementing validators within the network. Consequently, it is impossible for an entity to provide verifiable evidence that they have executed the necessary tasks to train a model by following the PoW-like consensus mechanism.
2. **Ownership protection from model-stealing attacks.** In decentralized AI training, once a trained model is released publicly in the network by a miner to claim network rewards, it will be broadcasted by other nodes either unaltered or manipulated (i.e., the model is stolen and the attacker claims ownership) until it fully propagates throughout the network. The model's actual owner may need to prove that they trained the model as a means to claim ownership. Proof of Learning (PoLe) [19] introduced an anti-theft scheme utilizing inner product-based functional encryption (IPFE) and IPFE with function-hiding (IPFE-FH). However, the problem of guaranteeing that the data node receives the complete model information remains unaddressed and requires further exploration. Ideally, it would be preferrable that the network receives full model info before applying the validation process.
3. **Absense of efficient consensus protocols for delivering services.** Upon successfully developing a consensus mechanism for decentralized AI training, it is of great interest to subsequently integrate it within a practical blockchain framework. According to the FLP impossibility theorem, which states that in an asynchronous distributed system where at least one process can fail, it is impossible to create a consensus algorithm that guarantees both safety and liveness at the same time [16], which is why most blockchain systems adopt synchronous consensus mechanisms. However, storage and bandwidth can be quite expensive in such systems since the system always store n replicas of the global states. Therefore, we require the protocol to store only the necessary states, minimizing storage requirements. Given the rapid evolution of AI models, integrating the entire system into a layer-1 (L1) blockchain solution[1] may not be the most optimal approach [19, 6]. Such systems typically maintain

[1] A Layer-1 (L1) blockchain refers to the foundational layer of a blockchain network, which consists of the base protocol that governs the consensus mechanism,

a consistent block production rate[2], thus ensuring a stable transaction throughput capacity. However, in a decentralized AI training system, the workload dynamically fluctuates in response to market supply and demand. There may be periods when the system experiences inactivity due to a lack of incoming training jobs, during which the majority of nodes become stale without a flow of rewards. In such a system, the primary objective is to generate valuable AI models, with transaction validation serving as a secondary function. A well-constructed framework should address these aspects by dynamically adjusting system workload according to the influx of training jobs, enabling seamless system upgrades over time, and ensuring the ease of use and security for users' assets. Such a protocol is currently lacking in the industry.

2 | PROOF OF TRAINING

Our primary objective is to establish a robust consensus protocol called proof of training (PoT) that lays the foundation for harnessing the power of crypto mining for distributed artificial intelligence (AI) training. The development of this protocol is crucial for enabling the efficient and secure utilization of computational resources across a decentralized network, with the ultimate goal of advancing AI model training. In this section, we will concentrate on the functions and utilities of the protocol, abstracting away from specific network designs. A comprehensive discussion of network realization can be found in Section 3.

2.1 | Notations

We denote the set of n aggregator nodes running the global ledger $\mathcal{L}$ by $\boldsymbol{\mathcal{A}} = \{\mathcal{A}_i\}_{i=1}^{n}$ where $\mathcal{A}_i$ represents each aggregator node, coordinating the client $\mathcal{C}$, the service provider $\mathcal{P}$ and the protocol validator $\mathcal{V}$. We denote all participants in the network by $\boldsymbol{\mathcal{N}} = \{\boldsymbol{\mathcal{A}}, \mathcal{C}, \mathcal{P}, \mathcal{V}\}$, with each individual denoted as $\mathcal{N}_i$. We let $\mathcal{S}_{\mathcal{N}}$ denote participant-specific security variables, including components for asymmetric encryption. We let $\mathcal{S}_{\mathcal{N}}[\mathrm{pk}]$ denote the public key of node $\mathcal{N}_i$, and $\mathcal{S}_{\mathcal{N}}[\mathrm{sk}]$ denote the corresponding private key.

We use the notation $\mathcal{M}$ to denote an AI model and $\boldsymbol{\mathcal{M}}$ to denote the full set of n_s models generated in a given specification where $\boldsymbol{\mathcal{M}_s} = (\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_{n_s})$. Specifically, we use the notation $\mathcal{M}_{\mathcal{C}}$ to denote the model supplied by a client for the network to train, typically with a certain initialization. Additionally, we introduce $\mathcal{D}_{\text{train}}$ to represent the training data and $\mathcal{D}_{\text{test}}$ for the test data.

We denote VRF_Model$(\mathcal{M}, \mathcal{D}_{\text{test}}) \rightarrow (\text{score})$ as the validation function, the purpose of which is to validate the model. Generally the model validation function VRF_Model[3] is specified by the client $\mathcal{C}$.

We let $\sigma_{\mathcal{N}}(m) = \mathrm{Sig}_{\mathcal{S}_{\mathcal{N}}}(m)$ denote a signature on message m with respect to $\mathcal{S}_{\mathcal{N}}$, i.e., using corresponding private key $\mathcal{S}_{\mathcal{N}}[\mathrm{sk}]$. Let VRF_Sig$(\mathcal{S}_{\mathcal{N}}[\mathrm{pk}], \sigma, m) \rightarrow \{0, 1\}$ denote a corresponding signature verification algorithm. Specifically we define $\sigma_{\mathcal{P}}^{\mathcal{M}} = \mathrm{Sig}_{\mathcal{S}_{\mathcal{P}}}(\mathcal{M})$ as the model signature message of service provider $\mathcal{P}$ on its generated model $\mathcal{M}$.

transaction processing, and data storage. This layer is responsible for the core functionality of the blockchain system and provides the infrastructure for building additional layers or applications on top of it.

[2] The block production rate in a blockchain refers to the frequency at which new blocks are generated and added to the blockchain. For example, TRON (TRX) network has a fast block production rate, with a new block being produced every 3 seconds.

[3] Two crucial properties are required here: **simplicity and certainty**. The computation complexity of VRF_Model should be $O(1)$, and the output score of the computation should be identical across different nodes, as long as $\mathcal{N}$ honestly perform the function.

2.2 | Consensus Assumptions

In this paper, we employ the term "global ledger" (in uppercase), denoted by $\mathcal{L}$, to refer to the fundamental data structure maintained by PoT protocol in order to support the specific services it offers. While blockchains are one method for implementing a reliable ledger, there are alternative approaches as well. We anticipate that PoT protocol implementations will utilize Byzantine Fault Tolerant (BFT) systems for their underlying ledgers, which significantly predate blockchains like EOS.io [15]. For the sake of convenience, we utilize BFT-type notation and properties throughout this paper, though we stress that PoT implementations can be realized using permissionless consensus protocols as well. We view a ledger generally as having a few key properties:

- *Append-Remove:* Data, once added, can be removed but cannot be modified.
- *Public:* The contents are accessible to everyone, which are consistent across time.
- *Available:* The ledger can always be written to by authorized writers and read by anyone in a timely way.

A wide variety of modern BFT protocols are supported in the PoT protocol. The exact choice will depend on trust assumptions and characteristics among the network nodes. The PoT protocol could in principle be implemented in a highly performant permissionless blockchain or in an adaptive and scalable layer-2 or blockchain system[4].

2.3 | Protocol Overview

(proof-of-training) A PoT scheme enables an efficient service provider $\mathcal{P}$ to convince the network of aggregator nodes $\mathcal{A}$ that $\mathcal{P}$ has trained the model $\mathcal{M}_C$, given by a client C, with validations from $\mathcal{V}$. It also enables the selection of the winner who generated the best model $\mathcal{M}_{\text{optimum}}$. A PoT protocol is characterized by a tuple of polynomial-time algorithms:

$$(\text{Claim, Validate, Verify, Finalize})$$

- PoT.Claim generates the claim message for a trained model trained via the initial model $\mathcal{M}_C$ and data $\mathcal{D}_{\text{train}}$ given by a client C. The service provider trains the model and saves the outputs for further processing. PoT.Claim is employed to generate model ownership claim messages and broadcast models, which are subsequently used for claiming rewards. Furthermore, it supplies information necessary for executing PoT.Validate and PoT.Verify. This process might rely on third-party services, such as model storage and parameter setup.
- PoT.Validate evaluates the models claimed by service providers and subsequently broadcasts a validation message to the network. This message includes the model's performance score and the identity of the service provider, thereby providing an evaluation of their contribution.
- PoT.Verify checks whether a validation from $\mathcal{V}$ is correct. PoT.Verify can be run by any node $\mathcal{N}$ (either a participant or validator) in the network to determine whether a certain validator has correctly validated a model, thereby convincing the global ledger $\mathcal{L}$ that the global states are correct. It's important to note that any incorrect states that are successfully challenged will be corrected, with significant economic incentives awarded to the challengers, which further ensures the safety of the protocol.

[4]A Layer 2 (L2) in blockchain refers to a secondary protocol or framework built on top of an existing blockchain, primarily aiming to enhance the network's scalability, efficiency, and transaction throughput. Layer 2 solutions leverage the security and decentralization of the underlying blockchain (Layer 1), while offloading a portion of the computational workload to a separate network or system. This enables faster and cheaper transactions, as well as more complex operations, without burdening the base layer. Examples of Layer 2 solutions include state channels, sidechains, and rollups.

- PoT.Finalize is run by the aggregators based on global ledger $\mathcal{L}$ to finalize rewards distribution. It summarizes all the validated models and corresponding validators which validated them. The optimum model's owner shall receive the majority of the rewards while the validators which validated the model receive the rest of the rewards to incentivize active participation and honest validation.

2.4 | Practical PoT Construction

In a practical PoT scenario where a client C aims to train a model with data $\mathcal{D}$, the protocol requires that C makes the initial model (potentially with initialized model parameters) $\mathcal{M}_C$ and training data $\mathcal{D}_{\text{train}}$ publicly accessible at time t_0[5]. The protocol also requires C to specify the duration of training time ΔT_{train}, after which the test data $\mathcal{D}_{\text{test}}$ shall be released by client C for validation and verification purposes. Once the current timestamp t satisfies the condition $t > t_0 + \Delta T_{\text{train}}$, the network rejects new incoming model signatures $\sigma_{\mathcal{P}}(\mathcal{M}_{\text{output}})$. Meanwhile, a service provider $\mathcal{P}$ will broadcast the generated model $\mathcal{M}_{\text{output}}$ corresponding to $\sigma_{\mathcal{P}}(\mathcal{M}_{\text{output}})$ broadcasted earlier. $\boldsymbol{\mathcal{M}}^{\boldsymbol{\sigma}}_{\text{output}}$ aggregates model signatures generated by all service providers which validators execute validation function with.

Generate a Claim

PoT.Claim$(\mathcal{M}_C, \mathcal{D}_{\text{train}}, \mathcal{S}_{\mathcal{P}}) \rightarrow (\sigma_{\mathcal{P}}(\mathcal{M}_{\text{output}})_{t_1}, \left(\mathcal{M}_{\text{output}}\right)_{t_2})$, where $\mathcal{S}_{\mathcal{P}}$ denotes participant-specific security variables for $\mathcal{P}$. $\mathcal{M}_{\text{output}}$ is the **latest generated model**[6] by $\mathcal{P}$ based on $\mathcal{M}_C$ and $\mathcal{D}_{\text{train}}$ within ΔT_{train} specified by C. We use t_1 and t_2 to denote two separate timestamps in the process, which indicate the broadcasting times of the content, with the following condition:

$$t_0 < t_1 < t_0 + \Delta T_{\text{train}} < t_2 \tag{1}$$

Once the current timestamp t of the global ledger $\mathcal{L}$ satisfies $t > t_0 + \Delta T$, the network rejects further model signature messages and $\mathcal{P}$ starts to broadcast $\mathcal{M}_{\text{output}}$ corresponding to the previous model signature message $\sigma^{\mathcal{M}}_{\mathcal{P}}$.

- `INPUTS:`
  - initial model $\mathcal{M}_C$
  - data $\mathcal{D}_{\text{train}}$
  - key parameter $\mathcal{S}_{\mathcal{P}}$
- `OUTPUTS:` model signature message $\sigma_{\mathcal{P}}(\mathcal{M}_{\text{output}})_{t_1}$, generated model $\left(\mathcal{M}_{\text{output}}\right)_{t_2}$

Validating the Models

PoT.Validate$(\sigma^{\mathcal{M}}_{\mathcal{P}}, \mathcal{M}, \mathcal{D}_{\text{test}}, \mathcal{S}_{\mathcal{P}}[\text{pk}], \mathcal{S}_{\mathcal{V}}) \rightarrow (\sigma_{\mathcal{V}}(\pi^{\mathcal{D}}_{\mathcal{V}})_{t_4}, \left(\pi^{\mathcal{D}}_{\mathcal{V}}\right)_{t_5})$, where $\mathcal{M}$ is a model the validator $\mathcal{V}$ newly receives. $\mathcal{S}_{\mathcal{V}}$ denotes participant-specific security variables for $\mathcal{V}$. $\mathcal{D}_{\text{test}}$ denotes the testing data released by the client at t_3, and we use t_4 and t_5 to denote two separate timestamps in the process, with $\Delta T_{\text{validate}}$ specified by the client or default values in the protocol:

$$t_0 + \Delta T_{\text{train}} \leq t_3 < t_4 \leq t_3 + \Delta T_{\text{validate}} < t_5 \tag{2}$$

[5] t_0 will be set by the primary of aggregator nodes following the PBFT synchronization protocol.

[6] As long as $t_1 < t_0 + \Delta T_{\text{train}}$ satisfies, the service provider $\mathcal{P}$ will keep optimizing the model $\mathcal{M}_{\text{output}}$ and once better model $\mathcal{M}^*_{\text{output}}$ is generated, $\mathcal{P}$ will send $\sigma_{\mathcal{P}}(\mathcal{M}^*_{\text{output}})_{t_1}$ to replace the previous model signature message $\sigma^{\mathcal{M}}_{\mathcal{P}}$.

Utilizing the VRF_Model function, validators can effectively evaluate the performance metrics of each model. The output $\pi_{\mathcal{V}}^{\mathcal{D}}$ refers to the validation message provided by the validator, which contains meta data such as the key and score of $\mathcal{M}$ and identity of $\mathcal{P}$.

- `INPUTS:`
  - model signature message $\sigma_{\mathcal{P}}^{\mathcal{M}}$
  - generated model $\mathcal{M}$
  - data $\mathcal{D}_{\text{test}}$
  - public key of service provider $\mathcal{S}_{\mathcal{P}}[\text{pk}]$
  - key parameter $\mathcal{S}_{\mathcal{V}}$
- `OUTPUTS:` validation signature message $\sigma_{\mathcal{V}}(\pi_{\mathcal{V}}^{\mathcal{D}})_{t_4}$, validation message $\left(\pi_{\mathcal{V}}^{\mathcal{D}}\right)_{t_5}$

Verifying the Validations

PoT.Verify$(\pi_{\mathcal{V}}^{\mathcal{D}}, \mathcal{D}_{\text{test}}) \rightarrow \{0,1\}$, which checks whether a validation from $\mathcal{V}$ is correct. PoT.Verify can be run by any node $\mathcal{N}_i$ (either a participant or validator) in the network and convince the global ledger $\mathcal{L}$ whether a certain validator has correctly validated a model. If not, the node will send $\text{Sig}_{\mathcal{S}_{\mathcal{N}}}(c_{\pi}^{\mathcal{V}})$ to the network along with a challenge message $c_{\pi}^{\mathcal{V}}$, which other participants can verify$(\sigma_{\mathcal{N}}(c_{\pi}^{\mathcal{V}})_{t_6}, \left(c_{\pi}^{\mathcal{V}}\right)_{t_7})$. We denote t_6 and t_7 as two seperate timestamps and we use $\Delta T_{\text{Challenge}}$ to denote the client specified or protocol default challenge period, which satisfies:

$$t_5 < t_6 \leq t_3 + \Delta T_{\text{validate}} + \Delta T_{\text{Challenge}} < t_7 \tag{3}$$

If the challenge is successful, the challenged validator $\mathcal{V}$ will be penalized, and the challenger $\mathcal{N}_i$ will be rewarded by receiving part of the penalization.

- `INPUTS:`
  - validation message $\pi_{\mathcal{V}}^{\mathcal{D}}$
  - data $\mathcal{D}_{\text{test}}$
- `OUTPUTS:` verification boolean value $b : \{0,1\}$, challenge message $(\sigma_{\mathcal{N}}(c_{\pi}^{\mathcal{V}})_{t_6}, \left(c_{\pi}^{\mathcal{V}}\right)_{t_7})\&\&(\neg b)$

Distributing the Rewards

After the challenge period of a client's order, PoT.Finalize$(\boldsymbol{\mathcal{M}}, \boldsymbol{\pi}) \rightarrow (\mathcal{M}_{\text{optimum}}, \mathcal{S}_{\mathcal{P}_{\text{optimum}}}[\text{pk}], \boldsymbol{\mathcal{V}})$ is run by the global ledger $\mathcal{L}$ to finalize reward distribution. $\boldsymbol{\mathcal{M}}$ is the vector containing all the validated models $(\mathcal{M}_1, \mathcal{M}_2, \cdots)$. $\boldsymbol{\pi}$ is the vector of global validation messages indicating the performance of different models in $\boldsymbol{\mathcal{M}}$. $\mathcal{M}_{\text{optimum}}$ and $\mathcal{P}_{\text{optimum}}$ are the optimum model (with the highest score) and its corresponding owner after sorting operations performed by $\mathcal{L}$. $\boldsymbol{\mathcal{V}}$ is the vector containing addresses of all the corresponding validators $(\mathcal{S}_{\mathcal{V}_1}[\text{pk}], \mathcal{S}_{\mathcal{V}_2}[\text{pk}], \cdots)$ which validated $\mathcal{M}_{\text{optimum}}$. The owner $\mathcal{P}_{\text{optimum}}$ shall receive the majority of the rewards while the validators $\boldsymbol{\mathcal{V}}$ receive the rest of the rewards to incentivize active participation and honest validation.

- `INPUTS:`
  - validated models $\boldsymbol{\mathcal{M}}$
  - global validation messages $\boldsymbol{\pi}$
- `OUTPUTS:` optimum model $\mathcal{M}_{\text{optimum}}$, owner's public key $\mathcal{S}_{\mathcal{P}_{\text{optimum}}}[\text{pk}]$, corresponding validators $\boldsymbol{\mathcal{V}}$

Fig. 1 presents a brief overview of the data flow between different participants within the protocol. 'StorageM',

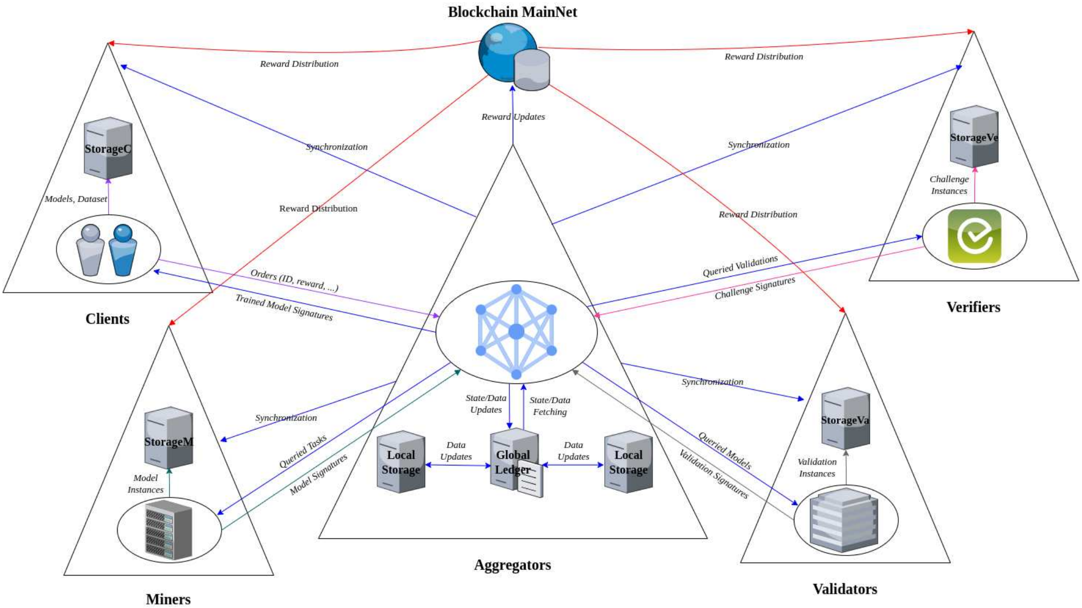

FIGURE 1 Brief overview of data flow between protocol participants. Data flows are represented by arrows, each labeled with the corresponding operation and color-coded based on the participant interaction. Access to different storage components by various participants is omitted for simplicity, whereas all participants can access any storage through a query link.

'StorageC', 'StorageVe', and 'StorageVa' represent storage resources provided by miners, clients, verifiers, and validators, respectively. These can be either centralized storage services or IPFS [7]. It's the participants' responsibility to ensure that this stored data remains accessible to other network participants, while also meeting a certain bandwidth limit required by the protocol implementation. If this is not guaranteed, the system will invoke a voting process that may render the participant invalid. Fig. 2 provides an illustration of the protocol by showcasing a sequential diagram that describes the protocol logic and data flow across different phases of a complete task cycle.

2.5 | Miscellaneous Notes

- *Network Securities and Cryptoeconomic Aspects:* The decentralization of AI model generation across various nodes requires robust security measures, particularly when some nodes may be compromised or corrupted. Ensuring that nodes have a financial incentive to act honestly is crucial in maintaining the integrity of the system. One such method is staking, which requires nodes to place deposits of utility tokens, with the potential for confiscation in cases of misbehavior. This incentive design has already been successfully employed in numerous blockchain implementations, as evidenced by the literature [20]. We require the **aggregator nodes**, which maintain the global ledger $\mathcal{L}$, to stake a significant amount of utility tokens in order to become an aggregator node. Misbehavior will result in the loss of their staked tokens. In this way, we can ensure the security of the L1 layer of the protocol.
  In addition to the aggregators, the appropriate number of tokens that different roles in the network should stake depends on various factors, such as the value of the tokens, the expected rewards, the risk of penalties, and

the overall economic model of the network. Here are some suggestions to help determine the staking amounts for different roles: **Service providers** should stake an amount that reflects their commitment to providing quality services and generating accurate models. The staking amount should be high enough to discourage fraudulent behavior with slows down the validation process of the network but not too high to create a barrier to entry for genuine providers. **Validators** should stake an amount that demonstrates their commitment to performing honest and accurate validations. The staking amount should be substantial enough to prevent validators from approving fraudulent claims or models, yet not so high as to create transaction friction that prevents honest validators from participating. **Verifiers** should stake a relatively smaller amount compared to service providers and validators, as their primary role is to verify the validators' work, which is generally expected to be accurate.

- *Concurrent Roles*: It is possible for various nodes to assume different roles and responsibilities simultaneously in order to maintain the integrity and efficiency of the system, as the network can make better use of available resources. For example, nodes with high GPU power can act as both validators and service providers. In general, it is expected that any node within the network would be capable of performing verification, as this process can be efficiently optimized. It is **absolutely** essential to apply the verification algorithm to a model associated with two or more clusters of validations, as at least one or more clusters of validations are guaranteed to be incorrect. Meanwhile, a model linked to a single cluster of validations is **highly likely** to have been correctly validated.
- *Validation Definiteness*: The validation process must yield consistent results, ensuring that for a given model and test data, the output remains **constant** across honest nodes with varying settings. This requirement eliminates any potential confusion in both validation and verification procedures. Consequently, it is recommended that the PoT implementation itself always supply the validation function, ensuring adaptability and upgradability within the system. Clients should not be allowed to provide their own validation functions for their models to avoid inconsistencies. Instead, they should be given options to select from available validation functions. To accommodate a wide range of use cases, it is crucial for the network to be compatible with most mainstream models, such convolutional neural networks (CNN) and Long Short-Term Memory Network (LSTM). This can be achieved by incorporating the validation functions for these models into the system's foundational layer, thus ensuring a consistent and reliable validation process across all nodes.
- *Commitment Scheme*: A commitment scheme[7] in the context of decentralized training systems enables participants to commit to a generated model while keeping it hidden from others, with the ability to reveal the committed model later. Such commitment schemes are designed so that a participant cannot claim the model without committing to it at an earlier timestamp than that of the real owner (in the global ledger $\mathcal{L}$). This approach has important applications in PoT protocol implementations, including model ownership claim/verification, and rewards distribution. Recall PoT.Claim algorithm, interactions in the commitment scheme take place in two phases:
  1. The **commit phase**: During this phase, a participant trains a model and commits to it by broadcasting its signature to the network.
  2. The **reveal phase**: In this phase, the participant reveals the trained model by sharing it with the network, allowing other participants to validate its performance and verify the ownership claim.

  Given the commitment scheme, malicious service providers are theoretically unable to steal any models, as they do not possess the model's signature in the global ledger during the model revealing phase, when the network stops accepting new model signatures.

[7] The concept of commitment schemes was formalized by Gilles Brassard, David Chaum, and Claude Crépeau [9] as part of numerous zero-knowledge protocols for NP.

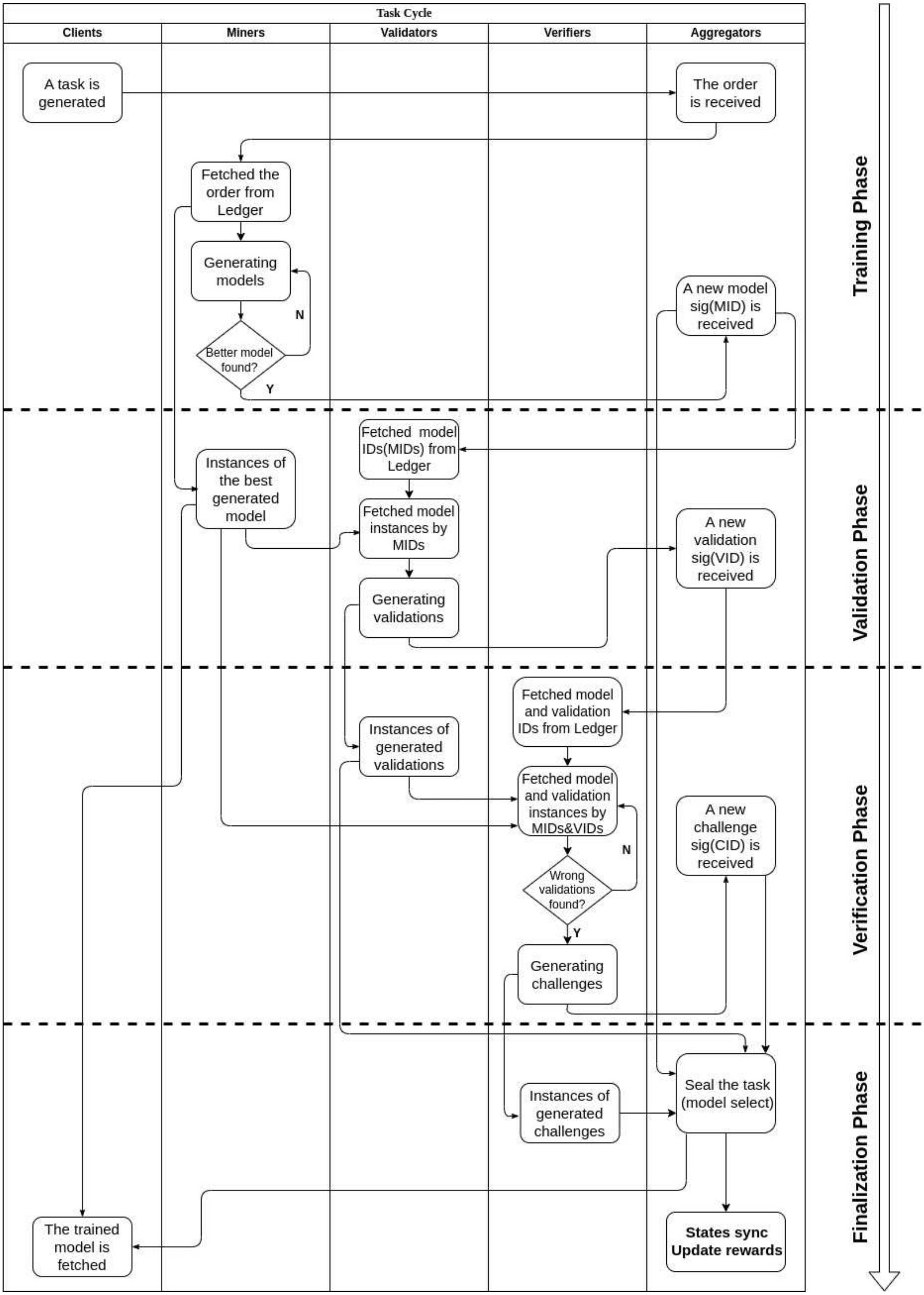

FIGURE 2 Sequential diagram describing protocol logic and data flow in different phases of a task cycle. It contains the training phase $[t_0, t_0 + \Delta T_{\text{train}}]$, the validation phase $[t_0 + \Delta T_{\text{train}}, t_3 + \Delta T_{\text{validate}}]$, the challenge phase $[t_3 + \Delta T_{\text{validate}}, t_3 + \Delta T_{\text{validate}} + \Delta T_{\text{Challenge}}]$, and the finalization phase. The time instances are defined in Eqs. 1, 2, and 3.

3 | PROTOCOL IMPLEMENTATION WITHIN NETWORK ARCHITECTURE

We present an in-depth exploration of the proof-of-training (PoT) protocol within a peer-to-peer network architecture through the design and implementation of a decentralized training network (DTN). The DTN aggregates models offered by multiple independent service providers, and the network participants self-coordinate to provide the best models to clients. This coordination is decentralized and does not require trusted parties. The secure operation of the system is ensured by the PoT protocol, which verifies that operations are correctly carried out by network participants.

3.1 | The DTN Construction

In decentralized service networks, blockchains fulfill two roles: they serve both as registers of cryptocurrency ownership and as foundations for decentralized services. In our system, the registration of participants and distribution of rewards happen on-chain, whereas the actual execution of training, validation, and other model-related computations occur off-chain due to the inherent costs and limitations of on-chain operations. On-chain operations are not only slow and expensive, but also restricted, unable to benefit from real-world data and various functionalities that simply can't be accomplished on-chain. These functionalities include diverse forms of computation, fast data distribution between miners and clients, and flexible infrastructure upgrades, among other features.

To effectively leverage the potential of this decentralized network for AI training, a two-layer architecture is implemented: the on-chain component (SC), which records the value flow in the network, and the off-chain component (exec), consisting of a set of protocols running on the DTN where utilities are performed. By securely integrating the on-chain functionality with the vast array of off-chain services offered by the DTN, it can exhibit the robustness and upgradability that traditional Layer 1 solutions often lack. In the L1-L2 design, the protocols and infrastructures primarily operate off-chain in the decentralized network, whereas token utilities such as transfer and withdrawal operate on Layer 2 of any mainstream blockchain. This setup allows the system to continuously update with additional features and utilities, while keeping the network's assets and user experience unaffected. Further details are depicted in Fig. 3.

3.1.1 | Network

The DTN is a decentralized training network that is *publicly verifiable* and designed on *incentives*. Clients pay a network of miners[8] for model generation and retrieval. Miners compete to train the best models in exchange of payments. Miners receive their payments only if the network has verified that their service was correctly provided.

Definition Our DTN scheme is a tuple of algorithms run by clients and network participants:

$$(\text{Put}, \text{Get}, \text{Manage})$$

- `Put(order)` $\rightarrow$ `OID`: Clients execute `Put` to submit a training order under a unique identifier `OID` (order ID). The training order includes all information necessary for service providers to execute the training task.
- `Get(OID)` $\rightarrow$ `model`: Clients execute `Get` to retrieve trained models that are stored using `OID`, upon task completion.

[8] The terms 'miners', 'service providers' and 'validators' can be used interchangeably in this section.

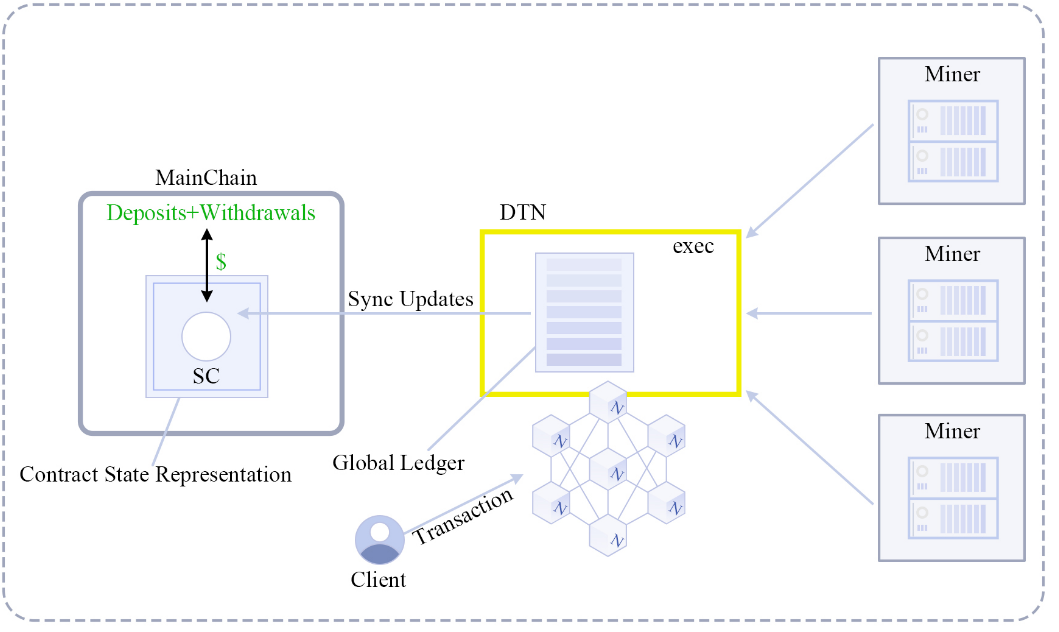

FIGURE 3 Conceptual figure depicting on-chain / off-chain components in the DTN, which consists of two major components: an on-chain component SC , resident on a mainstream blockchain, and an off-chain component exec that executes on a DTN. The DTN serves as a bridge between the two components as well as connecting the system-level contract with off-chain resources such as service providers, validators, decentralized storage, etc.

- Manage: `Manage()`: The network of participants coordinates via `Manage` to: control the available computational resources, validate the service offered by providers and repair possible faults. The `Manage` algorithm is mostly run by a network of aggregator nodes.

3.1.2 | The Global Ledger and Data Storage

In our decentralized training network, the Global Ledger $\mathcal{L}$ plays a key role as a system record, logging all essential network interactions. The ledger contains three key components: the *orders record*, the *task cycle data*, and the *node info*. The *orders record* logs all orders placed by clients within the network, each containing the specific task details requested by a client; including the required model, data, and associated rewards. The *task cycle data* records the metadata of tasks that have undergone the full cycle of model generation and validation within the network; including the generated model signatures, related validation outcomes, and potential challenges. The *node info* section saves the details of all registered nodes (miners and validators) within the network, including their reputation and performance history. Collectively, these components of the ledger boost the network's performance by ensuring all operations are traceable and accessible in a timely manner. The *aggregator node*, tasked with the responsibility of publishing multi-signature transactions on the blockchain and updating contract states, plays a central role in managing ledger data and global states. Through the application of the Practical Byzantine Fault Tolerance (PBFT) algorithm [11], it effectively maintains, updates, and synchronizes the Global Ledger $\mathcal{L}$. Besides storing and managing a synchronized copy of the global ledger, the aggregator nodes also act as data access points for other network participants. They provide on-demand access to the global ledger, ensuring its data is always available for different network operations.

Clients in the network are responsible for providing training and test data links, while miners must supply model instances. These data must be consistently accessible throughout the task cycle. Failure to comply with this requirement can lead to order or model claim invalidation through a community voting process. It is the participants' responsibility to download the necessary data to their local storage for efficient training and validation processes.

3.1.3 | Economics and Cryptoeconomics

To encourage correct behavior in the DTN, the system implements a cryptoeconomic incentive model. Each node is required to deposit a certain amount of tokens into a smart contract as a stake during registration. This staked amount acts as a financial commitment and failure to comply the rules may result in the lost of staked tokens.

The staking system also provides protection against Sybil attacks. By introducing a cost for network participation, the system discourages entities from creating multiple nodes with the intention of disrupting network operations. This cryptoeconomic model incentivizes nodes to act in the best interests of the DTN, thereby enhancing its security and overall efficiency.

Client Economics

The design of our network's economic system ensures that clients' tasks are handled with precedence, proportional to the average rewards offered over time. This approach prevents an overload of low-reward tasks that could strain the network's computational resources. Moreover, it incentivizes miners to prioritize tasks that yield higher returns, thereby optimizing the network's efficiency.

3.2 | Data Structure

The Decentralized Training Network (DTN) utilizes several primary data structures for operation purposes.

Orders

An *order* in the context of our DTN is a declaration of intent to request a service. Clients issue *orders* to the network to request services, and miners compete to provide the best services.

Claims

A *claim* in our DTN is a commitment made by a miner to deliver a trained model. Miners broadcast their claims to the ledger, which allows them to start competing rewards. A claim consists of the signature of the trained model and the model itself after t_2, following the PoT protocol's requirements in Eq. 1

Models

The *models* is a mapping between a model identifier (MID) and its corresponding model instances, which is built by using information extracted from *claims*. This data structure increase the system's efficiency by directly associating a model's link with its identifier, enabling quick look-ups and access.

Validations

A *validation* is the result of an evaluation process carried out by a validator to compute the performance of a trained model in the network. The validator uses the validation function and testing data as input parameters to this process. Upon completing the evaluation, the validator broadcasts the validation message to the network. This message comprises a validation signature at t_3 which serves as a seal of the validation, and a validation instance that details the

TABLE 1 Core Data Structures in our DTN scheme

Data Structures	
Order	**Network Record Table**
O^i := ⟨reward, type, time, link⟩	table: $\{O^i \rightarrow O^i$'s cycle data, ...$\}$
• reward: the economic incentive provided to the miners for training a model.	*order's cycle data* := ⟨*order*, phase, mList, vList, cList⟩$_{\text{order}}$
• type: the kind of model that is to be trained.	• *order*: the original order of the model.
• time: the Unix time instances including the training time t_0 and the validation time t_2.	• phase: An enum indicating the current phase of the task. It can be either 'model generation phase' or 'validation phase'.
• link: the link specific to model's training/testing data and metadata necessary for the task (such as initial model parameters).	• mList: list of generated models and its corresponding validations and challenges (if any) $\{\mathcal{G}_1..\mathcal{G}_{\text{m}}\}$.
Orders $\left(O^1..O^n\right)$	$\mathcal{G}_i$:= ⟨MID, vList, cList⟩$_{O^i}$
• O^i, current orders from txPool.	• MID: the hash of the generated model instance.
Validation	• vList: the list of validations corresponding to the generated model instance $\{\mathcal{V}_1, \mathcal{V}_2..\}$.
validation message := ⟨MID, score, vStake⟩$_{\mathcal{M}_i}$	– $\mathcal{V}_i$:= ⟨*validation message*, sig⟩$_{O^i}$
• MID: the hash of the model instance generated for the order's request.	– sig: the signature of the *validation message*.
• score: the model's performance metrics.	• cList: the objections raised against a existing validation identified by VID $\{c_1, c_2..\}$.
• vStake: the amount of stakes a validator is willing to commit to support a particular validation message.	– c_i := ⟨MID, VID, sig⟩$_{\mathcal{V}_i}$
Validations $(\mathcal{V}_1..\mathcal{V}_n)$	– VID: the hash of the validation being challenged.
• $\mathcal{V}_i$, current validations from txPool.	– sig: the signature of the *challenge message*.
Challenge	**Global Ledger**
challenge message := ⟨VID, cStake⟩$_{\mathcal{V}_i}$	$\mathcal{L}$:= ⟨models, txPool, table, nInfo⟩
• VID: the hash of the original validation for a model.	• models: the maps between a MID and its model instances, where $\mathcal{M}_{\text{MID}}$ = models[MID]
• cStake: the amount of stakes a verifier is willing to commit to support a particular challenge message.	• txPool := ⟨*orders*, *claims*, *validations*, *challenges*⟩
Challenges $(c_1..c_n)$	• table: the network record table.
• c_i, current challenges from txPool.	• nInfo: the node info structure containing a node's registration metadata and reputation.

model's performance metrics at t_5, following the PoT protocol's requirements in Eq. 2.

Challenges

A *challenge* includes a digital signature at t_6 and a challenge message at t_7, following the PoT protocol's requirements in Eq. 3. The digital signature is generated by the challenger signing the hash of the challenge message. The challenge message itself holds the specific validation being challenged and the amount of staked tokens backing the challenge.

Network Record Table

The *Network Record Table* functions as a key-value database. The table's structure is designed to map the hash of each *order* to a list of data structures which contains the following components: the original *order* issued by the client, the *phase* indicating the current phase of the task (as defined in Fig. 2), the *ModelList* comprising generated models related to the order with each model containing the *ValidationList* detailing evaluations carried out on the model and the *ChallengeList* capturing any objections raised against existing validations.

3.3 | Protocol Implementations

In this section, we focus on the operations carried out by various participants - clients, the network, and the miners. We illustrate the process flow of of different algorithms.

3.3.1 | Client Cycle

We give an overview of the client cycle.

1. **Put**: *The client orders model training service.*
   Clients can train their models by paying service providers with DTN utility tokens. A client initiates `Put` by submitting an order to the network. Subsequently, service providers have the freedom to decide whether they wish to compete for this order, which they can do by submitting claims, along with generated models, to the network. Clients have the flexibility to determine the amount of training time by modifying the 'time' variable in their orders. A longer training time may potentially yield higher accuracy in the resulting models.
2. **Get**: *Client retrieves model from the network.*
   Clients can retrieve any model stored in the DTN by fetching model links from the network. A client initiates `Get` by submitting an API request to one of the aggregator nodes. This node then retrieves the link from its local database. When the best model generated by the miners is found, the client receives a notification (with the model link) from the network. It is the miners' responsibility to ensure that their model links are always live to avoid penalties from the network.

3.3.2 | Mining Cycle (for service providers)

We give an overview of the mining cycle of service providers. Service providers earn rewards by competing to generate the model with highest score in the validation evaluation.

1. **Register**: Service Providers pledge their computational resources to the network. This is done by depositing collateral, via a transaction in the network, using `Manage.RegisterResource`. This collateral is locked in for the time intended to provide the service, and is returned upon request of the service provider if the provider decides to stop committing to the network, using `Manage.UnRegisterResource`. Once the service provider is registered, they can start generating model claims which will be added to the global ledger.

   `Manage.RegisterResource/UnRegisterResource`
   - `INPUTS:`
     - current global ledger $\mathcal{L}_t$
     - registration request `register`
   - `OUTPUTS:` current global ledger $\mathcal{L}_{t'}$
2. **Fetch Orders**: Service providers can fetch training orders from the network. Once registered, service providers specify how many orders they would like to fetch using the `Manage.FetchOrders` function. Upon executing this function, the corresponding number of training orders are fetched from the global ledger, sorted by the average

reward per unit of training time, and sent to the service provider. These orders contain details about the model training tasks, including the necessary data, the model to be used, and the amount of training time required. Once fetched, service providers can freely decide which orders they want to handle based on their available computational resources and other preferences.

> `Manage.FetchOrders`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
>   - number of orders to fetch `nOrders`
> - `OUTPUTS:` fetched orders `fetchedOrders` (sorted by average reward per unit of training time)

3. **Compete Rewards**: Service providers compete for the client's order by executing the training with the provided training data. They aim to generate a model with higher training accuracy. Once a new, more accurate model is generated, it is broadcasted to the network. This new model replaces any previous models with lower performance for that specific order. This process continues until the specified training time has elapsed. Subsequently, the validation process begins for that order[9].

> `Manage.Claim`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
>   - the generated `POT.claim`
> - `OUTPUTS:` updated global ledger $\mathcal{L}_{t'}$

4. **Sending Models**: Service Providers are responsible for ensuring the availability of links to generated model instances throughout the full mining cycle. This is done through the `Get.SendModel` function. If a service provider fails to maintain the availability of these model links, the network may invalidate the model, which will result in the service provider not receiving the rewards.

> `Get.SendModel`
> - `INPUTS:`
>   - model ID `MID`
>   - model link `mLink`
> - `OUTPUTS:` success status `sStatus`

3.3.3 | Mining Cycle (for verifiers)

We give an overview of the mining cycle of verifiers[10] . Verifiers earn rewards by challenging the wrong validations in the validation phase of an order.

1. **Register**: Verifiers pledge their computational resources to the network. This is done by depositing collateral, via a transaction in the network, using `Manage.RegisterVerifier`. This collateral is locked in for the time intended to provide the service, and is returned upon request of the verifier if the verifier decides to stop participating in

[9] While the `Manage.Claim` function describes the process of competing for a single order, it's important to note that service providers can execute this function in parallel to compete for multiple orders simultaneously. This parallelism allows service providers to optimally utilize their computational resources and maximize their potential rewards.

[10] It's worth noting that both service providers and validators can take on the role of a verifier, as long as they have sufficient computational resources. This overlap of roles allows for increased flexibility and efficiency in the network, as entities with more resources can contribute more significantly to the network's operations.

the network, using `Manage.UnRegisterVerifier`. Once the verifier is registered, they can start challenging the validations in the validation phase of an order.

> `Manage.RegisterVerifier/UnRegisterVerifier`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
>   - registration request `register`
> - `OUTPUTS:` current global ledger $\mathcal{L}_{t'}$

2. **Fetch Validations**: Verifiers can fetch validations for a specific order from the network. Once registered, verifiers specify which order they would like to fetch validations for by providing the `OID` of the order using the `Manage.FetchValidations` function. Upon executing this function, the corresponding validations for that order are fetched from the global ledger and sent to the verifier. Once fetched, verifiers can determine whether any of these validations should be challenged by executing the `POT.verify` function on them.

> `Manage.FetchValidations`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
>   - order ID `OID`
> - `OUTPUTS:` fetched validations `fetchedValidations` for the specific order

3. **Challenge Validations**: Verifiers challenge the validations in the validation phase of an order by executing the `Manage.Challenge` function. If a validation is found to be incorrect, the verifier earns the reward associated with the successful challenge.

> `Manage.Challenge`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
>   - the incorrect validation $\mathcal{V}$
> - `OUTPUTS:` updated global ledger $\mathcal{L}_{t'}$

3.3.4 | Network Cycle

We give an overview of the network cycle.

1. **Refresh**: The global ledger will repeatedly refresh orders in the transaction pool and corresponding data structures. For instance, it will check if an order has transitioned from the training phase to the validation phase. If so, the global ledger updates the state variable and starts accepting validations and challenges, while miners begin sending models. This refresh uses the Unix timestamp as a reference and updates the system variables according to the PoT protocol.

> `Manage.Refresh`
> - `INPUTS:`
>   - current global ledger $\mathcal{L}_t$
> - `OUTPUTS:` updated global ledger $\mathcal{L}_{t'}$

2. **Update**: The global ledger periodically commits the system states to smart contracts on the main chain, updating the global ledger accordingly. When orders pass both the validation and challenge phases, they are marked as

'sealed' and include the information of the winning miner, before being placed into a pending rewards queue. At each update cycle, aggregator nodes coordinates to multisign a transaction to the main chain, which updates the smart contracts, enabling miners to receive their rewards. Simultaneously, the global ledger removes the orders, along with their corresponding models and validations, once rewards have been distributed. This process is designed to save space in local storage. It's worth noting that the length of the update cycle is determined by a voting process among DTN nodes.

`Manage.Update`
- `INPUTS:`
  - current global ledger $\mathcal{L}_t$
- `OUTPUTS:` updated global ledger $\mathcal{L}_{t'}$

TABLE 2 Example execution of the DTN, grouped by network participants and sorted chronologically by row

Client	Network	Miner	
`PutOrders`(..,O_i)		`CompeteOrders`(..,O_{selected})	Put
		`Validate`(..,$\mathcal{M}_i$)	
	`AllocRewards`(..)	`Challenge`(..,$\mathcal{V}_i$)	
	`AddOrders`(..,O_{seal})		
		`GetModels`(..,oID)	Get
`GetModels`(..,oID)			
		`SendModels`(..,mID)	
	`TrackDeliver`(..)		
		`Register`(..)	Manage
	`Refresh`()		
	`Update`()		
		`UnRegister`()	

3.4 | Guarantees and Requirements

The DTN is designed to ensure *integrity*, *retrievability*, *incentive compatibility*, *public verifiability* and *flexibility* in the network, as detailed below:

- *Achieving Integrity:* Models, orders, and validations are identified by their respective cryptographic hashes (IDs). Clients only need to keep the hashes of the order and model to retrieve the model and verify the integrity of the content received. Network participants can utilize these hashes to reference specific data structures, thus simplifying the input requirements for function calls.
- *Achieving Retrievability:* All participants are required to stake tokens to register in the network to join mining, creating a strong incentive for them to consistently keep data available. If a client is unable to fetch the model they paid for, they may initiate an irretrievable report. The network will then call upon other participants to verify the report. Once a significant number of participants (with the total number of registered tokens supporting their consensus) verify and agree, the service provider will be penalized for failing to provide the model instance.
- *Achieving Incentive Compatibility:* Miners and validators are rewarded for the computation resources they pro-

vide. Those who fail to fulfill their commitments or submit incorrect proofs are penalized, incentivizing honest participation in the network.

- *Achieving Public Verifiability and Auditability:* All network participants, including miners and validators, have the ability to verify the validity of validations stored in the global ledger. They are economically incentivized to audit all work on the network, as successful challenges can earn rewards through utility tokens taken from dishonest or malicious validators. Unsuccessful challenges, however, result in a loss of collateral for the challenger. This system encourages positive behavior while simultaneously preventing any wrongdoing in the network, thus enhancing the autonomous feature of the system.
- *Flexibility:* The network utilizes a community Decentralized Autonomous Organization (DAO)[12] to decide on critical system parameters, such as the length of the challenge period, among others. This mechanism allows the system to adapt and evolve over time in response to the needs of the community and the growth of the business. This flexibility, combined with the network's robust design, creates a strong foundation for a secure, efficient, and user-responsive DTN.

4 | SIMUILATIONS

4.1 | Global Ledger Synchronizations

In the first part, we primarily focus on the synchronization of the transaction pool (txPool) within the global ledger. The txPool holds the most recent transactions and provides all necessary information for the global ledger to reach global states. The network of aggregator nodes maintains this global ledger, assembling incoming transactions from various network participants and synchronizing them with the global txPool.

The synchronization mechanism we implement is the Practical Byzantine Fault Tolerance (PBFT) algorithm, which enforces consensus among nodes regarding the pool of transactions, thus ensuring consistent data synchronization across the network. The efficacy of this synchronization mechanism, especially in real-world scenarios, is crucial to our system's performance and throughput. Therefore, we will conduct a series of simulations to evaluate the effectiveness of our PBFT-based synchronization within the global ledger.

4.1.1 | A Localhost Network Analysis

During our simulation, we used *crypto/x509* and *encoding/pem*, for facilitating the digital signatures and *SHA-256* for hashing algorithm. The source code implementing PBFT algorithm can be accessed on the author's GitHub page, for accommodating future improvements and extensions.

- *SHA-256 Hash:* For any hashing needs, the SHA-256 algorithm is used which produces a 256-bit (32-byte) hash.
- *RSA-2048 Signature:* RSA-2048 is used for signatures, meaning the size of a signature would be equal to the size of the key, i.e., 2048 bits or 256 bytes.
- *String Fields:* Assuming a UTF-8 encoding which is common in Go, a string uses 1 byte per character for most common characters, although some characters can use more.

As shown in Table 3, we analyze the approximate size of orders, validations, and challenges, depending on their respective fields. The order structure, consisting of reward, type, time, link, and a signature fields, costs approximately 312 bytes plus the size of varying fields. The Validation structure is made up of MID, score, vStake, and signature fields,

Algorithm 1 Practical Byzantine Fault Tolerance (PBFT)

1: **Phase 1 [Request]:** *Client*, *Miners*, *Validators*, *Verifiers* sends a request to the Primary aggregator node.
2: **Phase 2 [Pre-Prepare]:** The Primary node broadcasts a PRE-PREPARE message to all the Aggregator nodes.
3: **Phase 3 [Prepare]:** The Aggregator nodes validate the PRE-PREPARE message, and upon validation, they broadcast a PREPARE message to all the Aggregator nodes.
4: **Phase 4 [Commit]:** After receiving $2f$ PREPARE messages from different nodes, the Aggregator nodes broadcast a COMMIT message.
5: **Phase 5 [Reply]:** After receiving $2f + 1$ COMMIT messages from different nodes, the Aggregator nodes apply the operation and send a REPLY message to the *Client*, *Miners*, *Validators*, *Verifiers*.
6: **Phase 6 [Result Acceptance]:** The *Client*, *Miners*, *Validators*, *Verifiers* accept the operation result after receiving $f + 1$ identical REPLY messages from different Aggregator nodes.

costing approximately 304 bytes. The Challenge structure includes VID, cStake, and a signature fields, costing around 296 bytes. These sizes are necessary considerations when simulating the system's throughput, as they affect aspects such as ledger synchronization speeds and bandwidth requirements.

TABLE 3 Size in bytes of orders, validations and challenges

Data Structure	Field	Size (bytes)	Notes
Order	reward	8	Size of float64
	type	varies	Assuming 10 bytes for a string of length 10
	time	8	Size of int64
	link	varies	Assuming 30 bytes for a string of length 30
	sig	256	Size of RSA-2048 signature
	Total	~312	Plus the size of varying fields
Validation	MID	32	Size of SHA-256 hash
	score	8	Size of float64
	vStake	8	Size of float64
	sig	256	Size of RSA-2048 signature
	Total	~304	
Challenge	VID	32	Size of SHA-256 hash
	cStake	8	Size of float64
	sig	256	Size of RSA-2048 signature
	Total	~296	

We categorize networks into three sizes: small, medium, and large. Small networks consist of up to 10 nodes, used in cases such as sample or demo networks. Medium-sized networks, with 10 to 30 nodes, represent moderately distributed systems that could span across several geographical regions or countries. Large networks, with more than 30 nodes, represent global L1-L2 systems such as Chainlink [22]. For our analysis, we focus on the PBFT's synchronization time within the designed DTN structure, excluding considerations of network connection and data transfer latencies.

In Table. 4, we analyze how variations in network size and message size affect synchronization time. By adjusting these parameters, we can measure the capacity of our system to handle varying client request loads. For instance, simulations might involve synchronizing a single order with 10 validations (totaling 3432 bytes), or ten orders each with 10 validations (totaling 6160 bytes), or one order with 100 validations (totaling 30512 bytes), among other scenarios.

TABLE 4 Synchronization times for different message sizes and network sizes (in seconds).

Scenario (Message Size)	Network Size		
	Small (10 nodes)	Medium (30 nodes)	Large (100 nodes)
1 order, 10 validations (3432 bytes)	0.0387	0.132	0.463
1 order, 50 validations (15412 bytes)	0.0374	0.106	0.418
1 order, 100 validations (30512 bytes)	0.0466	0.149	0.373
10 orders, 50 validations (15640 bytes)	0.0452	0.113	0.407
10 orders, 100 validations (30740 bytes)	0.0338	0.125	0.392

The experiments were conducted on a 64-bit Ubuntu 22.04.2 LTS system powered by the 12th Generation Intel® Core™ i7-12700T processor with 20 cores, and equipped with 32GB memory.

As seen in Table 4, the network size significantly affects the synchronization time across all scenarios. As the number of nodes in the network increases, so does the synchronization time, requiring more time to update nodes in larger networks.

Despite differences in message size, the impact on synchronization time appears relatively complex. In fact, the system shows considerable efficiency in handling large packages, regardless of the number of orders and validations. This suggests that, without considering network latency, the system is designed to efficiently manage substantial volumes of transactions simultaneously. Thus, given our PBFT-based design, we can conclude that the network size plays a more substantial role in influencing synchronization time than the message size. Meanwhile, the network handles different message sizes effectively and robustly.

4.1.2 | A Real Network Analysis

Apaft from the theoretical txPool synchronization time analyzed in the previous section, we introduce a more comprehensive simulation of the PBFT synchronization algorithm. This simulation is designed to emulate real-world network conditions in distributed consensus scenarios. It considers the importance of variable network conditions, particularly network latency and bandwidth limitations, as these significantly impact the performance of a distributed system.

In real-world scenarios, nodes within a distributed network are typically spread across different geographical regions, each subject to unique network conditions. These variations in network latency and bandwidth can greatly influence the performance of the consensus algorithm. Consequently, it's crucial to incorporate these parameters into the network simulation, providing a more realistic analysis of the consensus algorithm's performance.

Analyzing recent data trends, global latency times and packet delivery rates can serve as reliable reference points for our simulation inputs. Data from May 2023 reveals average latency times of around 29ms for regional round trips within North America, 15ms for those within Europe, and 71ms for transatlantic round trips. These are general trends and the actual times can fluctuate based on a number of factors, including the specific locations within the regions, the network conditions, and time of day[11]. For transpacific and other international round trips, latency values are typically slightly higher than 300ms, but still within acceptable ranges for efficient network performance. These average latency and packet delivery figures provide us with a solid basis to input realistic and relevant values into our simulations. Regarding the bandwidth, slow networks are classified as those with bandwidths less than 1 Mbps, medium networks range from 1 Mbps to 100 Mbps, and fast networks are those with bandwidths greater than 100 Mbps.

We emulate a global network with varying network sizes, ranging from 10 to 50 nodes. The network latency varies between 30ms and 300ms, reflecting typical delay times both within a country and for transpacific connections. We further adjust the actual bandwidth limit, testing slow, medium, and fast speeds, although the theoretical bandwidth could be significantly higher. Additionally, we alter the size of the synchronized message (number of orders) from 100 orders to 10,000 orders to examine the performance metrics.

TABLE 5 Synchronization times for different message sizes, network sizes, and bandwidth limits (in seconds).

Scenario (Message Size)	Network Size (nodes)	Bandwidth Limit		
		Slow (0.1 Mbps)	Medium (30 Mbps)	Fast (125 Mbps)
100 transactions	Small (10 nodes)	8.609	1.494	1.497
100 transactions	Medium (30 nodes)	8.707	1.685	1.755
1000 transactions	Medium (30 nodes)	73.536	1.682	1.833
100 transactions	Large (50 nodes)	8.697	1.842	1.752
200 transactions	Large (50 nodes)	15.984	1.908	1.893
5000 transactions	Large (50 nodes)	37.532	1.767	1.678
10000 transactions	Large (50 nodes)	-	7.215	2.074

The experiments were conducted on a 64-bit Ubuntu 22.04.2 LTS system powered by the 12th Generation Intel® Core™ i7-12700T processor with 20 cores, and equipped with 32GB memory.

The results in Table 5 illustrate the impact of different message sizes, network sizes, and bandwidth limits on synchronization times. The message contains a bundle of transactions where each transaction can be identified as either an order, a validation, or a challenge, all of which have approximately similar sizes. As the message size increases, particularly under bandwidth-constrained conditions, synchronization times increase significantly. This effect is less apparent under high bandwidth conditions, indicating that adequate network bandwidth can guarantee high network throughput and robustness. Furthermore, the network size does not dramatically affect the synchronization times for small message sizes. Given the results, we emphasize the importance of sufficient bandwidth in the implementations of a Decentralized Training Network (DTN). Also considering the multi-sig process of the aggregator nodes, we don't suggest the number of nodes be large enough because that will complicate the DAO election process. As a result, a network size of 30-50 nodes and an average bandwidth requirement of 30 Mbps for aggregator nodes are suggested in the DTN implementation.

[11] https://www.verizon.com/business/terms/latency/

4.2 | Mining Rewards Distribution

In this section, we analyze the process of mining reward distribution on the Binance Smart Chain (BSC), known for its affordability with lower transaction costs compared to other blockchains. Rewards, generated by sealing orders, are distributed to accounts that could be owned by miners, verifiers or aggregators.

As the multi-signature setup, adopting the (k, n) configuration ensures robustness in our operations, even in the face of up to f faulty nodes. We maintain the capacity to approve and execute transactions as the choice of k ensures a consensus requirement.

Aggregator nodes, using the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm, collate the rewards for each account and update this information within the smart contracts. To safeguard the integrity of the distribution process, a multi-signature system is employed during the transactions. The associated cost of updating an account is computed as: Cost = Gas Price × Gas Cost × Token Price.

Function	Gas Cost	Executions	Tokens	Cost
proposeReward	86,875	Once	0.000396	$0.098
confirmReward	45,371	k times	0.000207	$0.051
executeReward	161,888	Once	0.000739	$0.18

TABLE 6 Summary of function executions on BSC Mainnet. Note that the 'Cost' and 'Tokens' values are calculated based on the current token price and gas price. As of June 20 2023, the gas price on BSC is 4.562 Gwei, and the token price is $246.2.

In Table 6, we present a detailed cost analysis associated with executing key functions for reward distribution on the Binance Smart Chain (BSC) mainnet. Each function's cost is calculated in tokens and their corresponding USD value. By summing the costs of proposing, confirming (with 30 confirmations), and executing a reward, we can estimate the total cost our network spends to distribute rewards to a single account. Given the gas price on BSC (4.562 Gwei as of June 20) and the token price of $246.2, this aggregate cost is approximately 0.007582 tokens or $1.87 per account. The estimation can help us in understanding the scalability and economic feasibility of implementing reward distribution mechanisms on Layer 2 networks on the BSC. Moreover, it's necessary to note that these costs may fluctuate due to variations in gas and token prices.

5 | DISCUSSIONS

5.1 | Protocol Capacity and Scalability

The previous sections have described the proof-of-training (PoT) protocol along with its implementations and simulations in a decentralized training network (DTN). While implementations in DTN can vary, the network's performance can approximately represent the protocol's performance, since the underlying structure and program logic remain consistent. The aggregator nodes serve as a platform in the system, coordinating clients, miners, and validators, enabling self-governance to initiate, process, and finalize services. While the actual influx of transactions (including orders, claims, validations, and challenges) will largely depend on the customer base and total hash power of the network, the processing power of the global states maintained by the aggregator nodes can be analyzed.

According to the simulation, the network exhibits favorable results in synchronizing system tasks and operations.

By considering internet geographical latency and setting specific bandwidth limits and a certain number of aggregator nodes, the network can synchronize thousands of transactions every second. Given the approximate sizes of the order, claim, validation, and challenge respectively, we can infer that the protocol can manage at least 10-100 models every second.

A significant advantage of the design is the allocation of computation-heavy tasks and storage to network participants. This strategy prevents the overconsumption of global storage, which could potentially be expensive, considering that updating global states is a synchronous process. The global ledger only stores order, model, and validation information, the sizes of which are in the unit of kilobytes. Meanwhile, processing them requires a computational complexity of $O(1)$ or $O(n)$. This approach enables the system to handle an empirically unlimited number of task requests and model finalizations simultaneously.

5.2 | Protocol Security

In most blockchain protocols, the security of a protocol is guaranteed by cryptoeconomics, i.e., attacking the system is more costly than complying with it. Similarly, in the proof-of-training (PoT) protocol, one would need to obtain more tokens than the counterparties to initiate attacks, which can often prove quite expensive. Unless the potential rewards are substantial, there is little incentive for someone to attack the protocol. Even in high reward instances, they attract more attention from miners and validators in the network. Consequently, the tokens committed to the task increase significantly, raising the cost of any potential cheating attempts.

Another possible attack scenario involves tampering with the `Manage.Update()` process in the aggregator nodes, allowing hackers to withdraw all tokens from the rewards contract. To compromise the multi-sig design of the PoT protocol, the miners would need a (k/n) portion of the total staked tokens by the aggregator nodes. We call this *Linear staking impact*, meaning that to be successful, an attacker must have a budget B greater than a (k/n) portion the combined staked tokens of all aggregator nodes. More precisely, we mean that as a function of k, $B(k) = dk$ in a network of n aggregator nodes, each with a fixed staked amount d. Given our requirement for aggregator nodes to stake a significant amount of tokens to act as network coordinators, a hacker would need at least 10% of the total circulation if 20% of tokens are held by the aggregator nodes (assuming $k = 18$ and $n = 30$). Therefore, the cost of such an attack is generally much higher than the tokens in the reward contract.

As shown in Table 6, it costs an aggregator an average of $1.87 to finalize an order and update it on the blockchain mainnet. So, how do we incentivize them to cover this cost? Regarding the economic incentives for aggregator nodes, the PoT protocol suggests two possible approaches. The network can periodically issue new tokens to reward aggregators. However, this method would introduce an annual inflation in token value based on the reward rate over time. An alternative approach is to tax each sealed order by a certain percentage (r). As long as the cumulative taxes exceed the cost of updating transactions, the aggregators will make a profit. This profit provides a strong incentive for the aggregators to perform diligently and honestly in their role as an aggregator node.

5.3 | Protocol Advantages

We believe the protocol's major advantage lies in its consensus mechanism design and optimized data structure, which provide significant capacity and scalability benefits compared to other solutions in this field. With this protocol design, the network coordinator, which maintains the global ledger and global states, is relieved from handling large data storage or heavy computation tasks inherently in most AI training processes. These tasks are delegated to participant nodes with sufficient resources. Participants are given strong cryptoeconomic incentives to act honestly and diligently, resulting in a system that can largely self-govern, thus enhancing the protocol's capacity and scalability. Participants are regulated by a voting mechanism. If, for example, any participant fails to provide storage and bandwidth for an instance download, they may be penalized by other nodes on the network and potentially lose their staked tokens during the voting process. The protocol can therefore ensure that participants remain committed to their orders and services, guaranteeing system liveliness.

Another major advantage of the protocol over others lies in the design of its L1-L2 system structure, which ensures the easy upgradability of the system. AI is a rapidly shifting industry with new types of models being developed on a daily basis. The protocol uses Layer-2 (on-chain) applications to deposit, withdraw, and transfer users' assets, while most operations are carried out on Layer-1 (off-chain) for upgradability purposes. For any new models, we can integrate them into the system by asking miners and aggregators to upgrade to the latest version of the `exec`. Then, clients will be able to specify new model types in their orders. Theoretically, the system can include any type of AI model into the L1 infrastructure, given that there is a valid validation function for that model which meets the protocol's requirements mentioned in section 2.

Question: Can the protocol handle training task of Large Language Models (LLM) such as chatGPT?

In the PoT protocol, although a 'miner' denotes a single node, there can actually be many GPU cards behind that node, as seen in the case of mining pools. Hundreds or even thousands of miners can join a mining pool to receive rewards. Given the significant computing power of a mining pool, it is much more likely to receive rewards in a competitive process. These rewards are then evenly distributed among mining pool participants based on their contributed computing power. A significant advantage of a mining pool is its reliability: unlike a single mining entity, which can become faulty at any time, the mining rigs gathered in a pool are typically more reliable. Consequently, they can handle more complex training algorithms like those described in [18], particularly when dealing with large models. It's clear that mining pools are capable of handling large language models (LLMs) with billions of parameters. We believe that, with this component taken into consideration, the protocol can certainly handle LLM training tasks. This can be achieved by specifying detailed parameters on the client side and offering proportionate rewards to miners based on their contributions.

6 | CONCLUSIONS AND FUTURE WORKS

In conclusion, our work successfully bridges the emerging gap between artificial intelligence (AI) and crypto mining by addressing three major challenges that are currently keeping these two fields apart. The proof-of-training (PoT) protocol combines the strengths of both AI and blockchain resources, thereby enhancing the potential of both.

The capacity, scalability, upgradability, and security attributes of the protocol have been rigorously evaluated and discussed throughout this study. By innovatively integrating a delicate system design and robust economic incentives, our solution circumvents common drawbacks of blockchain technology such as high storage and computation costs and limited network data access, while bolstering its strengths, such as security and user accessibility. We believe the protocol can be a game changer in the industry, providing individuals with affordable and straightforward access to resources which were previously exclusive to large companies and enterprises.

One aspect not covered in this paper is the execution of experiments involving the interaction between clients and miners with actual tasks being resolved. This is mainly because any simulation in this aspect would merely represent a specific case of the system's capacity and throughput. However, to analyze the protocol from a financial perspective, we set it as part of our future works: To engage the current hash power in the market by introducing network utility tokens and implementing a complete version of the DTN, which would enable a detailed analysis of the system's performance on real-world tasks, leading to further developments and understanding of the PoT protocol.

Code Availability

The source code used in this study for the implementation of the proof-of-training (PoT) protocol and the decentralized training network (DTN) is available for review, use, and modification under the terms of the MIT License. You can access the repository at: `https://github.com/P-HOW/proof-of-training`.

References

[1] 2018. *Ocean Protocol: A Decentralized Substrate for AI Data and Services*. Accessed: [Date Accessed].

[2] 2019. *Fetch.AI Whitepaper*. Accessed: [Date Accessed].

[3] 2023. *Ethereum Network Hashrate Chart*. Accessed: [Date Accessed].

[4] 2023. *Singularitynet*. Accessed: 2023-04-05.

[5] H. Alshahrani, N. Islam, D. Syed, A. Sulaiman, M. Al Reshan, K. Rajab, A. Shaikh, J. Shuja-Uddin, and A. Soomro, *Sustainability in blockchain: A systematic literature review on scalability and power consumption issues*, Energies **16** (02 2023), 1510.

[6] A. Baldominos and Y. Saez, *Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning*, Entropy **21** (2019).

[7] J. Benet, *IPFS - content addressed, versioned, P2P file system*, CoRR **abs/1407.3561** (2014).

[8] J. Boyle, 2021. *BTC Mining Used More Electricity than Sweden*. Accessed: [Date Accessed].

[9] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci. **37** (1988), 156–189.

[10] F. Bravo-Marquez, S. Reeves, and M. Ugarte, *Proof-of-learning: A blockchain consensus mechanism based on machine learning competitions*, 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), 2019, pp. 119–124.

[11] M. Castro and B. Liskov, *Practical byzantine fault tolerance*, OSDI **99** (1999), 173–186.

[12] S. Cohney, J. Sklaroff, and D. Wishnick, *The dao controversy: The case for a new legal framework for daos*, Stanford Technology Law Review **24** (2021).

[13] Cointelegraph, 2023. *Cryptocurrency miners may lead the next stage of ai*. Accessed: [Date Accessed].

[14] A. De Vries, *Cryptocurrencies on the road to sustainability: Ethereum paving the way for bitcoin*, Patterns **4** (2023), 100633.

[15] EOS.IO, 2017. *Eos.io technical white paper*.

[16] M.J. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM **32** (apr 1985), 374–382.

[17] E. Foundation., *Ethereum 2.0 Specifications*. Accessed: [Date Accessed].

[18] M. Langer, Z. He, W. Rahayu, and Y. Xue, *Distributed training of deep learning models: A taxonomic perspective*, IEEE Trans. Parallel Distrib. Syst. **31** (2020), 2802–2818.

[19] Y. Liu, Y. Lan, B. Li, C. Miao, and Z. Tian, *Proof of learning (pole): Empowering neural network training with consensus building on blockchains*, Comput. Networks **201** (2021), 108594.

[20] C. Nguyen, H. Dinh Thai, D. Nguyen, D. Niyato, H. Nguyen, and E. Dutkiewicz, *Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities*, IEEE Access **PP** (06 2019), 1–1.

[21] T. Nguyen and K. Kim, *A survey about consensus algorithms used in blockchain*, J. Informat. Process. Syst. **14** (01 2018), 101–128.

[22] N. Sergey and S. Johnson, 2017. *Chainlink: A decentralized oracle network*. Accessed: yyyy-mm-dd.

[23] Swiss Government, 2021. *Federal Act on the Adaptation of Federal Law to Developments in Distributed Electronic Register Technology*. Accessed: [Date Accessed].

[24] W. Wang, H. Dinh Thai, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D.I. Kim, *A survey on consensus mechanisms and mining strategy management in blockchain networks*, IEEE Access **PP** (01 2019), 1–1.