2019 IEEE International Conference on Blockchain (Blockchain)

DaiMoN: A Decentralized Artificial Intelligence Model Network

Surat Teerapittayanon
Harvard University
Cambridge, USA
steerapi@seas.harvard.edu

H. T. Kung
Harvard University
Cambridge, USA
kung@harvard.edu

***Abstract*—We introduce DaiMoN, a decentralized artificial intelligence model network, which incentivizes peer collaboration in improving the accuracy of machine learning models for a given classification problem. It is an autonomous network where peers may submit models with improved accuracy and other peers may verify the accuracy improvement. The system maintains an append-only decentralized ledger to keep the log of critical information, including who has trained the model and improved its accuracy, when it has been improved, by how much it has improved, and where to find the newly updated model. DaiMoN rewards these contributing peers with cryptographic tokens. A main feature of DaiMoN is that it allows peers to verify the accuracy improvement of submitted models without knowing the test labels. This is an essential component in order to mitigate intentional model overfitting by model-improving peers. To enable this model accuracy evaluation with hidden test labels, DaiMoN uses a novel learnable *Distance Embedding for Labels* (DEL) function proposed in this paper. Specific to each test dataset, DEL scrambles the test label vector by embedding it in a low-dimension space while approximately preserving the distance between the dataset's test label vector and a label vector inferred by the classifier. It therefore allows *proof-of-improvement* (PoI) by peers without providing them access to true test labels. We provide analysis and empirical evidence that under DEL, peers can accurately assess model accuracy. We also argue that it is hard to invert the embedding function and thus, DEL is resilient against attacks aiming to recover test labels in order to cheat. Our prototype implementation of DaiMoN is available at https://github.com/steerapi/daimon.**

***Index Terms*—blockchain, decentralized ledger technology (DLT), neural network, artificial intelligence, distributed machine learning, distance-preserving embedding**

I. Introduction

Network-based services are at the intersection of a revolution. Many centralized monolithic services are being replaced with decentralized microservices. The utility of decentralized ledgers showcases this change, and has been demonstrated by the usage of Bitcoin [1] and Ethereum [2].

The same trend towards decentralization is expected to affect the field of artificial intelligence (AI), and in particular machine learning, as well. Complex models such as deep neural networks require large amounts of computational power and resources to train. Yet, these large, complex models are being retrained over and over again by different parties for similar performance objectives, wasting computational power and resources. Currently, only a relatively small number of pretrained models such as pretrained VGG [3], ResNet [4], GoogLeNet [5], and BERT [6] are made available for reuse.

One reason for this is that the current system to share models is centralized, limiting both the number of available models and incentives for people to participate and share models. Examples of these centralized types of systems are Caffe Model Zoo [7], Pytorch Model Zoo [8], Tensorflow Model Zoo [9], and modelzoo.co [10].

In other fields seeking to incentivize community participation, cryptocurrencies and cryptographic tokens based on decentralized ledger technology (DLT) have been used [1], [2]. In addition to incentives, DLT offers the potential to support transparency, traceability, and digital trust at scale. The ledger is append-only, immutable, public, and can be audited and validated by anyone without a trusted third-party.

In this paper, we introduce DaiMoN, a decentralized artificial intelligence model network that brings the benefits of DLT to the field of machine learning. DaiMoN uses DLT and a token-based economy to incentivize people to improve machine learning models. The system will allow participants to collaborate on improving models in a decentralized manner without the need for a trusted third-party. We focus on applying DaiMoN for collaboratively improving classification models based on deep learning. However, the presented system can be used with other classes of machine learning models with minimal to no modification.

In traditional blockchains, proof-of-work (PoW) [1] incentivizes people to participate in the consensus protocol for a reward and, as a result, the network becomes more secure as more people participate. In DaiMoN, we introduce the concept of *proof-of-improvement* (PoI). PoI incentivizes people to participate in improving machine learning models for a reward and, as an analogous result, the models on the network become better as more people participate.

One example of a current centralized system that incentivizes data scientists to improve machine learning models for rewards is the Kaggle Competition system [11], where a sponsor puts up a reward for contestants to compete to increase the accuracy of their models on a test dataset. The test dataset inputs are given while labels are withheld to prevent competitors from overfitting to the test dataset.

In this example, a sponsor and competitors rely on Kaggle to keep the labels secret. If someone were to hack or compromise

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00026

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

Kaggle's servers and gain access to the labels, Kaggle would be forced to cancel the competition. In contrast, because DaiMoN utilizes a DLT, it eliminates this concern to a large degree, as it does not have to rely on a centralized trusted entity.

However, DaiMoN faces a different challenge: in a decentralized ledger, all data are public. As a result, the public would be able to learn about labels in the test dataset if it were to be posted on the ledger. By knowing test labels, peers may intentionally overfit their models, resulting in models which are not generalizable. To solve the problem, we introduce a novel technique, called *Distance Embedding for Labels* (DEL), which can scramble the labels before putting them on the ledger. DEL preserves the error in a predicted label vector inferred by the classifier with respect to the true test label vector of the test dataset, so there is no need to divulge the true labels themselves.

With DEL, we can realize the vision of PoI over a DLT network. That is, any peer verifier can vouch for the accuracy improvement of a submitted model without having access to the true test labels. The proof is then included in a block and appended to the ledger for the record.

The structure of this paper is as follows: after introducing DEL and PoI, we introduce the DaiMoN system that provides incentive for people to participate in improving machine learning models.

The contributions of this paper include:

1) A learnable Distance Embedding for Labels (DEL) function specific to the test label vector of the test dataset for the classifier in question, and performance analysis regarding model accuracy estimation and security protection against attacks. To the best of our knowledge, DEL is the first solution which allows peers to verify model quality without knowing the true test labels.
2) Proof-of-improvement (PoI), including detailed PROVE and VERIFY procedures.
3) DaiMoN, a decentralized artificial intelligence model network, including an incentive mechanism. DaiMoN is one of the first proof-of-concept end-to-end systems in distributed machine learning based on decentralized ledger technology.

II. DISTANCE EMBEDDING FOR LABELS (DEL)

In this section, we describe our proposed Distance Embedding for Labels (DEL), a key technique by which DaiMoN can allow peers to verify the accuracy of a submitted model without knowing the labels of the test dataset. By keeping these labels confidential, the system prevents model-improving peers from overfitting their models intentionally to the test labels.

A. Learning the DEL Function with Multi-Layer Perceptron

Suppose that the test dataset for the given C-class classification problem consists of m (input, label) test pairs, and each label is an element in $\mathcal{Q} = \{c \in \mathbb{Z} \mid 1 \le c \le C\}$, where $\mathbb{Z}$ denotes the set of integers. For example, the FashionMNIST [12] classification problem has $C = 10$ image classes and $m = 10,000$ (input, label) test pairs, where for each pair, the input is a 28×28 greyscale image, and the label is an element in $\mathcal{Q} = \{1, 2, \ldots, 10\}$.

For a given test dataset, we consider the corresponding *test label vector* $\boldsymbol{x}_t \in \mathcal{Q}^m$, which is made of all labels in the test dataset. We seek a $\boldsymbol{x}_t$-specific DEL function $f : \boldsymbol{x} \in \mathcal{Q}^m \to \boldsymbol{y} \in \mathbb{R}^n$, where $\mathbb{R}$ denotes the set of real numbers, which can approximately preserve distance from a predicted label vector $\boldsymbol{x} \in \mathcal{Q}^m$ (inferred by a classification model or a classifier we want to evaluate its accuracy) to $\boldsymbol{x}_t$, where $n \ll m$. For example, we may have $n = 256$ and $m = 10,000$. The *error* of $\boldsymbol{x}$, or the distance from $\boldsymbol{x}$ to $\boldsymbol{x}_t$, is defined as

$$\mathrm{e}(\boldsymbol{x}, \boldsymbol{x}_t) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(x_i \neq x_{t_i}),$$

where $\mathbb{1}$ is the indicator function, $\boldsymbol{x} = \{x_1, \ldots, x_m\}$ and $\boldsymbol{x}_t = \{x_{t_1}, \ldots, x_{t_m}\}$.

Finding such a distance-preserving embedding function f is generally a challenging mathematical problem. Fortunately, we have observed empirically that we can learn this $\boldsymbol{x}_t$-specific embedding function using a neural network.

More specifically, to learn an $\boldsymbol{x}_t$-specific DEL function f, we train a multi-layer perception (MLP) for f as follows. For each randomly selected $\boldsymbol{x} \in \mathcal{Q}^m$, we minimize the loss:

$$\mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_t) = |\mathrm{e}(\boldsymbol{x}, \boldsymbol{x}_t) - \mathrm{d}(f(\boldsymbol{x}), f(\boldsymbol{x}_t))|,$$

where $\boldsymbol{\theta}$ is the MLP parameters, and $\mathrm{d}(\cdot, \cdot)$ is a modified cosine *distance* function defined as

$$\mathrm{d}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \begin{cases} 1 - \frac{\boldsymbol{y}_1 \cdot \boldsymbol{y}_2}{\|\boldsymbol{y}_1\| \|\boldsymbol{y}_2\|} & \boldsymbol{y}_1 \cdot \boldsymbol{y}_2 \ge 0 \\ 1 & \text{otherwise.} \end{cases}$$

The MLP training finds a distance-preserving low-dimensional embedding function f specific to a given $\boldsymbol{x}_t$. The existence of such embedding is guaranteed by the Johnson-Lindenstrauss lemma [13], [14], under a more general setting which does not have the restriction about the embedding being specific to a given vector.

B. Use of DEL Function

We use the trained DEL function f to evaluate the accuracy or the error of a classification model or a classifier on the given test dataset without needing to know the true test labels. As defined in the preceding section, for a given test dataset, $\boldsymbol{x}_t \in \mathcal{Q}^m$ is the true test label vector of the test dataset. Given a classification model or a classifier, $\boldsymbol{x} \in \mathcal{Q}^m$ is a predicted label vector consisting of labels inferred by the classifier on all the test inputs of the test dataset. A verifier peer can determine the error of the predicted label vector $\boldsymbol{x}$ without knowing the true test label vector$\boldsymbol{x}_t$, by checking $\mathrm{d}(f(\boldsymbol{x}), f(\boldsymbol{x}_t))$ instead of $\mathrm{e}(\boldsymbol{x}, \boldsymbol{x}_t)$. This is because these two quantities are approximately equal, as assured by the MLP training, which minimizes their absolute difference. If $\mathrm{d}(f(\boldsymbol{x}), f(\boldsymbol{x}_t))$ is deemed to be sufficiently lower than that of the previously known model, then a verifier peer may conclude

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

procedure GENERATEDATA($\boldsymbol{x}_t$)
  Pick a random number v in $\{1, 2, \ldots, m\}$
  Pick a random set $\boldsymbol{\mathcal{K}}$ in $\{1, 2, \ldots, m\}^v$
  Initialize $\boldsymbol{x}$ as $\{x_1, x_2, \ldots, x_m\}$ with $\boldsymbol{x} \leftarrow \boldsymbol{x}_t$
  for $k \in \boldsymbol{\mathcal{K}}$ **do**
    Pick a random number c in $\{1, 2, \ldots, C\}$
    $x_k \leftarrow c$
  return $\boldsymbol{x}$

that the model has improved the accuracy of the test dataset. That is, the verifier peer uses $\mathrm{d}(f(\boldsymbol{x}), f(\boldsymbol{x}_t))$ as a proxy for $\mathrm{e}(\boldsymbol{x}, \boldsymbol{x}_t)$.

Note that the DEL function f is $\boldsymbol{x}_t$-specific. For a different test dataset with a different test label vector $\boldsymbol{x}_t$, we will need to train another f. For most model benchmarking applications, we expect a stable test dataset; and thus we will not need to retrain f frequently.

III. Training and Evaluation of DEL

In this section, we evaluate how well the neural network approach described above can learn a DEL function $f : \mathcal{Q}^m \rightarrow \mathbb{R}^n$ with $m = 10,000$ and $n = 256$. We consider a simple multi-layer perceptron (MLP) with 1024 hidden units and a rectified linear unit (ReLU). The output of the network is normalized to a unit length. The network is trained using the Adam optimization algorithm [15]. The dataset used is FashionMNIST [12], which has $C = 10$ classes and $m = 10,000$ (input, label) test pairs. The true test label vector $\boldsymbol{x}_f$ is thus composed of these 10,000 test labels.

To generate the data to train the function, we perturb the test label vector $\boldsymbol{x}_t$ by using the GENERATEDATA procedure shown. First, the procedure picks v, the number of labels in $\boldsymbol{x}_t$ to switch out, and generates the set of indices $\boldsymbol{\mathcal{K}}$, indicating the positions of the label to replace. It then loops through the set $\boldsymbol{\mathcal{K}}$. For each $k \in \boldsymbol{\mathcal{K}}$, it generates the new label c to replace the old one. Note that with this procedure, the new label c can be the same as the old label.

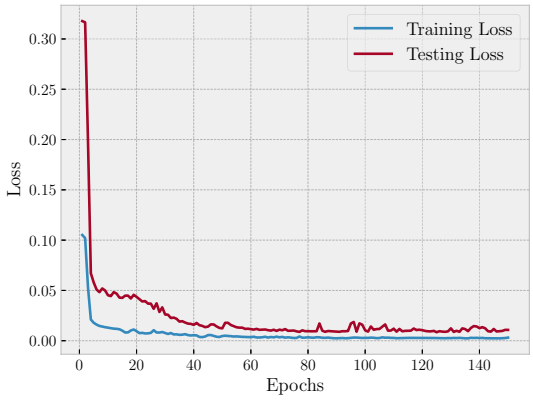

Figure 1: The training and testing loss as the number of epochs increases.

We use the procedure to generate the training dataset and the test dataset for the MLP. Figure 1 shows the convergence of the network in learning the function f. We see that as the number of epochs increases, both training and testing loss decrease, suggesting that the network is learning the function.

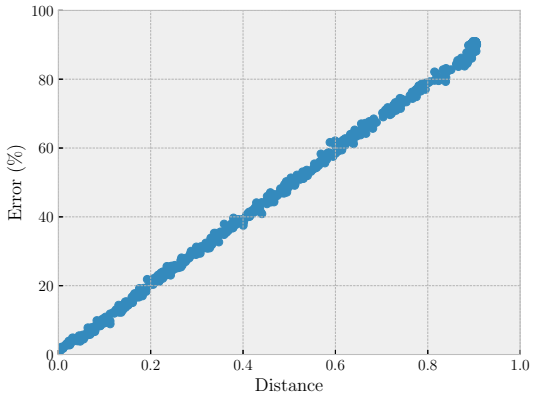

Figure 2: Correlation between error (%) in $\boldsymbol{x}$ with respect to $\boldsymbol{x}_t$ and the distance between them in the embedding space under f.

After the neural network has been trained, we evaluate how well the learned f can preserve error in a predicted label vector $\boldsymbol{x}$ inferred by the classifier. Figure 2 shows the correlation between the error and the distance in the embedding space under f. We see that both are highly correlated.

IV. Analysis on Defense Against Brute-force Attacks

In this section, we show that it is difficult for an attacker to launch a brute-force attack on DEL. To learn about the test label vector $\boldsymbol{x}_t \in \mathcal{Q}^m$ that produces $\boldsymbol{y}_t \in \mathbb{R}^n$ under a known $f : \boldsymbol{x} \in \mathcal{Q}^m \rightarrow \boldsymbol{y} \in \mathbb{R}^n$, the attacker's goal is to find $\boldsymbol{x}$ such that $\mathrm{d}(f(\boldsymbol{x}), \boldsymbol{y}_t) < \epsilon$, for a small ϵ. There are C^m possible instances of $\boldsymbol{x}$ to try, where C is the number of classes. Note C^m can be very large. For example, for a test dataset of 10 classes and 10,000 samples, we have $C = 10$, $m = 10,000$ and $C^m = 10^{10000}$. The attacker may use the following brute-force algorithm:

The success probability (α) of this attack where at least one out of q tried instances for $\boldsymbol{x}$ is within the ϵ distance of $\boldsymbol{x}_t$ is

$$\alpha = 1 - (1 - p)^q \approx pq,$$

where p is the probability that $\mathrm{d}(f(\boldsymbol{x}), \boldsymbol{y}_t) < \epsilon$.

We now derive p and show its value is exceedingly small for a small ϵ, even under moderate values of n. Assume that the outputs of f is uniformly distributed on a unit (n-1)-sphere or equivalently normally distributed on an n-dimension euclidean space [16]. Suppose that $\boldsymbol{y}_t = f(\boldsymbol{x}_t)$. We align the top of the unit (n-1)-sphere at $\boldsymbol{y}_t$. Then, p is the probability of a random vector on a (n-1)-hemisphere falling onto the cap [17] which is

$$p = \mathrm{I}_{\sin^2 \beta}(\frac{n-1}{2}, \frac{1}{2}),$$

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

procedure BRUTEFORCEATTACK($\boldsymbol{y}_t, \epsilon, q$)
  Pick a random set $\boldsymbol{\mathcal{X}}_0$ of q values in $\mathcal{Q}^m$
  for $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}_0$ **do**
    if $\mathrm{d}(f(\boldsymbol{x}), \boldsymbol{y}_t) < \epsilon$ **then**
      return $\boldsymbol{x}$

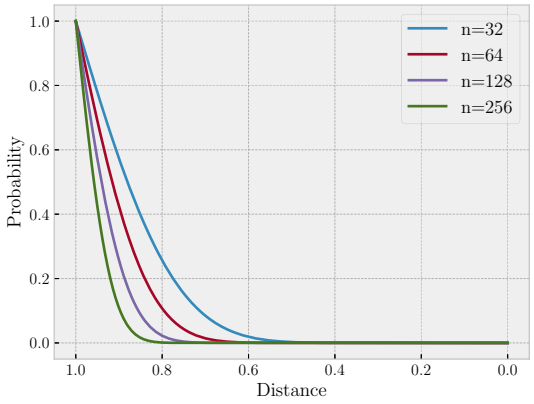

Figure 3: The probability p as distance decreases for varying n.

where n is the dimension of a vector, β is the angle between $\boldsymbol{x}_t$ and a vector on the sphere, and $\mathrm{I}_x(a, b)$ is the regularized incomplete beta function defined as:

$$\mathrm{I}_x(a, b) = \frac{\mathrm{B}(x;\, a, b)}{\mathrm{B}(a, b)}.$$

In the above expression, $\mathrm{B}(x;\, a, b)$ is the incomplete beta function, and $\mathrm{B}(a, b)$ is the beta function defined as:

$$\mathrm{B}(x;\, a, b) = \int_0^x t^{a-1}\,(1-t)^{b-1}\,dt,$$
$$\mathrm{B}(a, b) = \int_0^1 t^{a-1}\,(1-t)^{b-1}\,dt.$$

Figure 3 shows the probability p as the distance (ϵ) from $\boldsymbol{x}_t$ decreases for different values of n. We observe that for a small ϵ, this probability is exceedingly low and thus to guarantee attacker's success ($\alpha = 1$), the number of samples ($q = \alpha/p = 1/p$) of $\boldsymbol{x}$ needed to be drawn randomly is very high. For instance, for a 10% error rate, $\epsilon = 0.10$ and $n = 32$, the probability p is 6.12×10^{-13} and the number of trials q needed to succeed is 1.63×10^{12}. In addition, the higher the n, the smaller the p and the larger the q. For example, for a 10% error rate, $\epsilon = 0.10$ and $n = 256$, the probability p is 3.33×10^{-93} and the number of trials q needed to succeed is 3.01×10^{92}.

V. Analysis on Defense Against Inverse-mapping Attacks

In this section, we provide an analysis on defense against attacks attempting to recover the original test label vector $\boldsymbol{x}_t$ from $\boldsymbol{y}_t = f(\boldsymbol{x}_t)$. We consider the case that the attacker tries to learn an inverse function $f^{-1} : \boldsymbol{y} \in \mathbb{R}^n \rightarrow \boldsymbol{x} \in \mathcal{Q}^m$

procedure GENERATEINVERSEDATANEARBY($\boldsymbol{x}_t, f$)
  $\boldsymbol{x} \leftarrow$ GENERATEDATA($\boldsymbol{x}_t$)
  $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
  return $\{\boldsymbol{y}, \boldsymbol{x}\}$

procedure GENERATEINVERSEDATARANDOM(f, C)
  Pick a random number $\boldsymbol{x}$ in $\{1, 2, \ldots, C\}^m$
  $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
  return $\{\boldsymbol{y}, \boldsymbol{x}\}$

using a neural network. Suppose that the attacker uses a multilayer perceptron (MLP) for this with 1024 hidden units and a rectified linear unit (ReLU). The network is trained using Adam optimization algorithm [15]. The loss function used is the squared error function:

$$\mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}_t) = \|f^{-1}(\boldsymbol{y}) - f^{-1}(\boldsymbol{y}_t)\|_2^2.$$

We generate the dataset using the two procedures: GENERATEINVERSEDATANEARBY and GENERATEINVERSEDATARANDOM shown. The former has the knowledge that the test label vector $\boldsymbol{x}_t$ is nearby, and the latter does not. We train the neural network to find the inverse function f^{-1} and compare how the neural network learns from these two generated datasets.

The GENERATEINVERSEDATANEARBY procedure generates a perturbation of the test label vector $\boldsymbol{x}_t$, passes it through the function f, and returns a pair of the input $\boldsymbol{y}$ and the target output vector $\boldsymbol{x}$ used to learn the inverse function f^{-1}.

The GENERATEINVERSEDATARANDOM procedure generates a random label vector $\boldsymbol{x}_t$ where each element of the vector has a value representing one of the C classes, passes it through the function f and returns a pair of the input $\boldsymbol{y}$ and the target output vector $\boldsymbol{x}$ used to learn the inverse function f^{-1}.

Figure 4 shows the error $\mathrm{e}(f^{-1}(\boldsymbol{y}_t), \boldsymbol{x}_t)$ as the number of training epochs increases. On one hand, using the data generated without the knowledge of the test label vector $\boldsymbol{x}_t$ using the GENERATEINVERSEDATARANDOM procedure, we see that the network does not reduce the error as it trains. This means that it does not succeed in learning the inverse function f^{-1} and therefore, it will not be able to recover the test label $\boldsymbol{x}_t$ from its output vector $\boldsymbol{y}_t$. On the other hand, using the data generated with the knowledge of the test label vector $\boldsymbol{x}_t$ using the GENERATEINVERSENEARBY procedure, we see that the network does reduce the the error as it trains and has found the test label vector $\boldsymbol{x}_t$ from its output vector $\boldsymbol{y}_t$ at around 40 epochs. This experiment gives an empirical evidence that without the knowledge of $\boldsymbol{x}_t$, it is hard to find f^{-1}.

VI. Proof-of-Improvement

In this section, we introduce the concept of *proof-of-improvement* (PoI), a key mechanism supporting DaiMoN. PoI allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a model M improves the accuracy or reduces the error on the test dataset via the use of DEL without the knowledge of the true test label

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

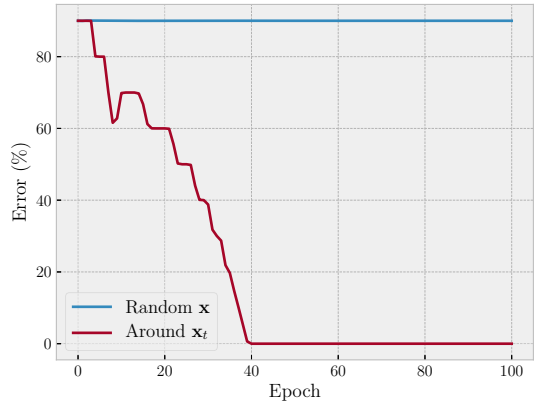

Figure 4: Error $(\mathrm{e}(f^{-1}(\boldsymbol{y}_t), \boldsymbol{x}_t))$ *in percentage as the number of epochs increases for data generated with random* $\boldsymbol{x}$ *and near* $\boldsymbol{x}_t$.

procedure PROVE(M) $\rightarrow \pi_{\mathcal{P}}$
  $\boldsymbol{g} \leftarrow \mathrm{digest}(\mathrm{M})$
  $\boldsymbol{y} \leftarrow f(\mathrm{M}(\boldsymbol{Z}))$
  return $\{\boldsymbol{g}, \boldsymbol{y}, \mathrm{pk}_{\mathcal{P}}\}_{\mathrm{sk}_{\mathcal{P}}}$

vector $\boldsymbol{x}_t$. PoI is characterized by the PROVE and VERIFY procedures shown.

As a part of the system setup, a prover $\mathcal{P}$ has a public and private key pair $(\mathrm{pk}_{\mathcal{P}}, \mathrm{sk}_{\mathcal{P}})$ and a verifier $\mathcal{V}$ has a public and private key pair $(\mathrm{pk}_{\mathcal{V}}, \mathrm{sk}_{\mathcal{V}})$. Both are given our learnt DEL function $f(\cdot)$, $\boldsymbol{y}_t = f(\boldsymbol{x}_t)$, the set of m test inputs $\boldsymbol{Z} = \{\boldsymbol{z}\}_{i=1}^{m}$, and the current best distance d_c achieved by submitted models, according to the distance function $\mathrm{d}(\cdot, \cdot)$ described in Section II.

Let $\mathrm{digest}(\cdot)$ be the message digest function such as IPFS hash [18], MD5 [19], SHA [20] and CRC [21]. Let $\{\cdot\}_{\mathrm{sk}}$ denotes a message signed by a secret key sk.

Let M be the classification model for which $\mathcal{P}$ will generate a PoI proof $\pi_{\mathcal{P}}$. The model M takes an input and returns the corresponding predicted class label. The PROVE procedure called by a prover $\mathcal{P}$ generates the digest of M and calculates the DEL function output of the predicted labels of the test dataset $\boldsymbol{Z}$ by M. The results are concatenated to form the body of the proof, which is then signed using the prover's secret key $\mathrm{sk}_{\mathcal{P}}$. The PoI proof $\pi_{\mathcal{P}}$ shows that the prover $\mathcal{P}$ has found a model M that could reduce the error on a test dataset.

To verify, the verifier $\mathcal{V}$ runs the following procedure to generate the verification proof $\pi_{\mathcal{V}}$, the proof that the verifier $\mathcal{V}$ has verified the PoI proof $\pi_{\mathcal{P}}$ generated by the prover $\mathcal{P}$. The procedure first verifies the signature of the proof with public key $\pi_{\mathcal{P}}.\mathrm{pk}_{\mathcal{P}}$ of the prover $\mathcal{P}$. Second, it verifies that the digest is correct by computing $\mathrm{digest}(\mathrm{M})$ and comparing it with the digest in the proof $\pi_{\mathcal{P}}.\boldsymbol{g}$. Third, it verifies the DEL function output by computing $f(\mathrm{M}(\boldsymbol{Z}))$ and comparing it with the the DEL function output in the proof $\pi_{\mathcal{P}}.\boldsymbol{y}$. Lastly,

procedure VERIFY$(\mathrm{M}, \pi_{\mathcal{P}}, d_{\mathrm{c}}, \delta) \rightarrow \pi_{\mathcal{V}}$
  Verify the signature of $\pi_{\mathcal{P}}$ with $\pi_{\mathcal{P}}.\mathrm{pk}_{\mathcal{P}}$
  Verify the digest: $\pi_{\mathcal{P}}.\boldsymbol{g} = \mathrm{digest}(\mathrm{M})$
  Verify the DEL function output: $\pi_{\mathcal{P}}.\boldsymbol{y} = f(\mathrm{M}(\boldsymbol{Z}))$
  Verify the distance: $\mathrm{d}(\pi_{\mathcal{P}}.\boldsymbol{y}, \boldsymbol{y}_t) < d_{\mathrm{c}} - \delta, \delta \geq 0$
  if all verified **then**
    return $\{\pi_{\mathcal{P}}, d_c, \delta, \mathrm{pk}_{\mathcal{V}}\}_{\mathrm{sk}_{\mathcal{V}}}$

it verifies the distance by computing $\mathrm{d}(\pi_{\mathcal{P}}.\boldsymbol{y}, \boldsymbol{y}_t)$ and sees if it is lower than the current best with a margin of $\delta \geq 0$, where δ is an improvement margin commonly agreed upon among peers. If all are verified, the verifier generates the body of the verification proof by concatenating the PoI proof $\pi_{\mathcal{P}}$ with the current best distance d_c and δ. Then, the body is signed with the verifier's secret key $\mathrm{sk}_{\mathcal{V}}$, and the verification proof is returned.

VII. THE DAIMON SYSTEM

In this section, we describe the DaiMoN system that incentivizes participants to improve the accuracy of models solving a particular problem. In DaiMoN, each classification problem has its own DaiMoN blockchain with its own token. An append-only ledger maintains the log of improvements for that particular problem. A problem defines inputs and outputs which machine learning models will solve for. We call this the problem definition. For example, a classification problem on the FashionMNIST dataset [12] may define an input $\boldsymbol{z}$ as a 1-channel $1 \times 28 \times 28$ pixel input whose values are ranging from 0 to 1, and an output $\boldsymbol{x}$ as 10-class label ranging from 1 to 10:

$$\{\boldsymbol{z} \in \mathbb{R}^{1\times 28\times 28} \mid 0 \leq \boldsymbol{z} \leq 1\},$$
$$\{\boldsymbol{x} \in \mathbb{Z} \mid 1 \leq \boldsymbol{x} \leq 10\}.$$

Each problem is characterized by a set of test dataset tuples. Each tuple $(\boldsymbol{Z}, f, \boldsymbol{y}_t)$ consists of the test inputs $\boldsymbol{Z} = \{\boldsymbol{z}\}_{i=1}^{m}$, the DEL function f, and the DEL function output $\boldsymbol{y}_t = f(\boldsymbol{x}_t)$ on the true test label vector $\boldsymbol{x}_t$.

A participant is identified by its public key pk with the associated private key sk. There are six different roles in DaiMoN: problem contributors, model improvers, validators, block committers, model runners, and model users. A participant can be one or more of these roles. We now detail each role below:

Problem contributors contribute test dataset tuples to a problem. They can create a problem by submitting a problem definition and the first test dataset tuple. See Section VII-B on how additional test tuples can be added.

Model improvers compete to improve the accuracy of the model according to the problem definition defined in the chain. A model improver generates a PoI proof for the improved model and submit it.

Validators validate PoI proofs, generate a verification proof and submit it as a vote on the PoI proofs. Beyond being a verifier for verifying PoI, a validator submits the proof as a vote.

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

Block committers create a block from the highest voted PoI proof and its associated verification proofs and commit the block.

Model runners run the inference on the latest model given inputs and return outputs and get paid in tokens.

Model users request an inference computation from model runners with an input and pay for the computation in tokens.

A. The Chain

Each chain consists of two types of blocks: Problem blocks and Improvement blocks. **A Problem block** contains information, including but not limited to: the block number, the hash of the parent block, the problem definition, the test dataset tuples, and the block hash. **An Improvement block** contains information, including but not limited to: the block number, the hash of the parent block, PoI proof $\pi_{\mathcal{P}}$ from model improver $\mathcal{P}$, verification proofs $\{\pi_{\mathcal{P}}\}$ from validators $\{\mathcal{V}\}$, and the block hash. The chain must start with a Problem block that defines the problem, followed by Improvement blocks that record the improvements made for the problem.

B. The Consensus

After a DaiMoN blockchain is created, there is a problem definition period T_p. In this period, any participant is allowed to add test dataset tuples into the mix. After a time period T_p has passed, a block committer commits a Problem block containing all test dataset tuples submitted within the period to the chain.

After the Problem block is committed, a competition period T_b begins. During this period, a model improver can submit the PoI proof of his/her model. A validator then validates the PoI proof and submit a verification proof as a vote. For each PoI proof, its associated number of unique verification proofs are tracked. At the end of each competition period, a block committer commits an Improvement block containing the model with the highest number of unique verification proofs, and the next competition period begins.

C. The Reward

Each committed block rewards tokens to the model improver and validators. The following reward function, or similar ones, can be used:

$$\mathrm{R}(d, d_c) = \mathrm{I}_{1-d}(a, \frac{1}{2}) - \mathrm{I}_{1-d_c}(a, \frac{1}{2}),$$

where $\mathrm{I}_{\cdot}(\cdot, \cdot)$ is the regularized incomplete beta function, d is the distance of the block, d_c is the current best distance so far, and a is a parameter to allow for the adjustment to the shape of the reward function. Figure 5 shows the reward function as the distance d decreases for different current best distance d_c for $a = 3$. We see that more and more tokens are rewarded as the distance d reaches 0, and the improvement gap $d_c - d$ increases.

Each validator is given a position as it submits the validation proof: the s-th validator to submit the validation proof is given the s-th position. The validator's reward is the model improver's reward scaled by 2^{-s}: $\mathrm{R}(d, d_c)2^{-s}$, where $s \in \mathbb{Z}_{>0}$ is the validator's position, and $\mathbb{Z}_{>0}$ denotes the set of integers greater than zero. This factor encourages validators to compete to be the first one to submit the validation proof for the PoI proof in order to maximize the reward. Two is used as a base of the scaling factor here since $\sum_{s=1}^{\infty} 2^{-s} = 1$.

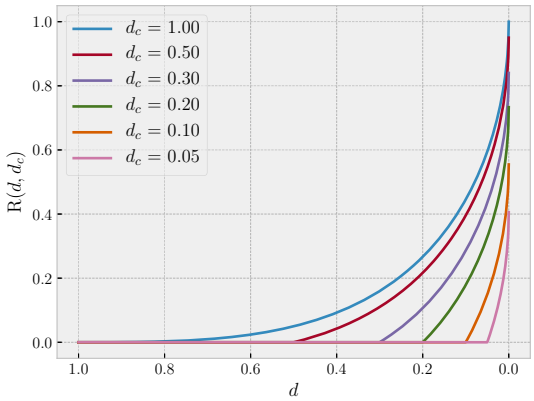

Figure 5: The reward function $\mathrm{R}(d, d_c)$ as the distance d decreases for varying current best distance d_c for $a = 3$.

D. The Market

In order to increase the value of the token of each problem, there should be demand for the token. One way to generate demand for the token is to allow it to be used as a payment for inference computation based on the latest model committed to the chain. To this end, model runners host the inference computation. Each inference call requested by users is paid for by the token of the problem chain that the model solves. Model runners automatically upgrade the model, as better ones are committed to the chain. The price of each call is set by the market according to the demand and supply of each service. This essentially determines the value of the token, which can later be exchanged with other cryptocurrencies or tokens on the exchanges. As the demand for the service increases, so will the token value of the problem chain.

Model runners periodically publish their latest services containing the price for the inference computation of a particular model. Once a service is selected, model users send a request with the payment according to the price specified. Model runners then verify the request from the user, run the computation, and return the result.

To keep a healthy ecosystem among peers, a reputation system may be used to recognize good model runners and users, and reprimand bad model runners and users. Participants in the network can upvote good model runners and users and downvote bad model runners and users.

E. System Implementation

DaiMoN is implemented on top of the Ethereum blockchain [2]. In this way, we can utilize the security and decentralization of the main Ethereum network. The ERC-20 [22] token standard is used to create a token for each

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

problem chain. Tokens are used as an incentive mechanism and can be exchanged. Smart contracts are used to manage the DaiMoN blockchain for each problem. This includes managing the position of each validator as it submits the validation proof.

Identity of a participant is represented by its Ethereum address. Every account on Ethereum is defined by a pair of keys, a private key and public key. Accounts are indexed by their address, which is the last 20 bytes of the Keccak [20] hash of the public key.

The position of the verifiers is recorded and verified on the Ethereum blockchain. As a verifier submits a vote on the smart contract on the Ethereum blockchain, his/her position is recorded and used to calculate the reward for verifier.

The InterPlanetary File System (IPFS) [18] is used to store and share data files that are too big to store on the Ethereum blockchain. Files such as test input files and model files are stored on IPFS and only their associated IPFS hashes are stored in the smart contracts. Those IPFS hashes are then used by participants to refer to the files and download them. Note that since storing files on IPFS makes it public, it is possible that an attacker can find and submit the model before the creator of the model. To prevent this, model improvers must calculate the IPFS hash of the model and register it with the smart contract on the Ethereum blockchain before making the model available on IPFS.

VIII. Discussion

One may compare a DEL function to an encoder of an autoencoder [23]. An autoencoder consists of an encoder and a decoder. The encoder maps an input to a lower-dimensional embedding which is then used by the decoder to reconstruct the original input. Although a DEL function also reduces the dimensionality of the input label vector, it does not require the embedding to reconstruct the original input and it adds the constraint that the output of the function should preserve the error or the distance of the input label vector to a specific test label vector $\boldsymbol{x}_t$. In fact, for our purpose of hiding the test labels, we do not want the embedding to reconstruct the original input test labels. Adding the constraint to prevent the reconstruction may help further defense against the inverse-mapping attacks and can be explored in future work.

Note that a model with closer distance to the test label vector ($\boldsymbol{x}_t$) in the embedding space may not have better accuracy. This results in a reward being given to a model with worse accuracy than the previous best model. This issue can be mitigated by increasing the margin δ. With the appropriate δ setting, this discrepancy should be minimal. Note also that as the model gets better, it will be easier for an attacker to recover the true test label vector ($\boldsymbol{x}_t$). To mitigate this issue, multiple DEL and reward functions may be used at various distance intervals.

By building DaiMoN on top of Ethereum, we inherit the security and decentralization of the main Ethereum network as well as the limitations thereof. We now discuss the security of each individual DaiMoN blockchain. An attack to consider is the Sybil attack on the chain, in which an attacker tries to create multiple identities (accounts) and submit multiple verification proofs on an invalid PoI proof. Since each problem chain is managed using Ethereum smart contracts, there is an inherent gas cost associated with every block submission. Therefore, it may be costly for an attacker to overrun the votes of other validators. The more number of validators for that chain, the higher the cost is. In addition, this can be thwarted by increasing the cost of each submission by requiring validators to also pay Ether as they make the submission. All in all, if the public detects signs of such behavior, they can abandon the chain altogether. If there is not enough demand in the token, the value of the tokens will depreciate and the attacker will have less incentives to attack.

Since we use IPFS in the implementation, we are also limited by the limitations of IPFS: files stored on IPFS are not guaranteed to be persistent. In this case, problem contributors and model improvers need to make sure that their test input files and model files are available to be downloaded on IPFS. In addition to IPFS, other decentralized file storage systems that support persistent storage at a cost such as Filecoin [24], Storj [25], etc. can be used.

IX. Related Works

One area of related work is on data-independent locality sensitive hashing (LSH) [26] and data-dependent locality preserving hashing (LPH) [27], [28]. LSH hashes input vectors so that similar vectors have the same hash value with high probability. There are many algorithms in the family of LSH. One of the most common LSH methods is the random projection method called SimHash [29], which uses a random hyperplane to hash input vectors.

Locality preserving hashing (LPH) hashes input vectors so that the relative distance between the input vectors is preserved in the relative distance between of the output vectors; input vectors that are closer to each other will produce output vectors that are closer to each other in the output space. The DEL function presented in this paper is in the family of LPH functions. While most of the work on LSH and LPH focuses on dimensionality reduction for nearest neighbor searches, DEL is novel that it focuses on learning an embedding function that preserves the distance to the test label vector of the test dataset. For the purpose of hiding the test label vector from verifier peers, it is appropriate that DEL's distance preserving is specific to this test vector. By being specific, finding the DEL function becomes easier.

Another area of related work is on blockchain and AI. There are numerous projects covering this area in recent years. These include projects such as SingularityNET [30], Effect.ai [31] and Numerai [32]. SingularityNET and Effect.ai are decentralized AI marketplace platforms where anyone can provide AI services for use by the network, and receive network tokens in exchange. This is related to the DaiMoN market, where model runners run inference computation for model users in exchange for tokens. Numerai has an auction mechanism where participants stake network tokens to express confidence in their models' performance on forthcoming new

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.

data. The mechanism will reward participants according to the performance of their models on unspecified yet future test data, their confidence, and their stake. While these platforms all use a single token, DaiMoN has a separate token for each problem. It also introduces the novel ideas of DEL and PoI, allowing the network to fairly reward participants that can prove that they have a model that improves the accuracy for a given problem.

X. Conclusion

We have introduced DaiMoN, a decentralized artificial intelligence model network. DaiMoN uses a Distance Embedding for Labels (DEL) function. DEL embeds the predicted label vector inferred by a classifier in a low-dimensional space where its error or its distance to the true test label vector of the test dataset is approximately preserved. Under the embedding, DEL hides test labels from peers while allowing them to assess the accuracy improvement that a model makes. We present how to learn DEL, evaluate its effectiveness, and present the analysis of DEL's resilience against attacks. This analysis shows that it is hard to launch a brute-force attack or an inverse-mapping attack on DEL without knowing a priori a good estimate on the location of the test label vector, and that the hardness can be increased rapidly by increasing the dimension of the embedding space.

DEL enables *proof-of-improvement* (PoI), the core of DaiMoN. Participants use PoI to prove that they have found a model that improves the accuracy of a particular problem. This allows the network to keep an append-only log of model improvements and reward the participants accordingly. DaiMoN uses a reward function that scales according to the increase in accuracy a new model has achieved on a particular problem. We hope that DaiMoN will spur distributed collaboration in improving machine learning models.

XI. Acknowledgment

This work is supported in part by the Air Force Research Laboratory under agreement number FA8750-18-1-0112 and a gift from MediaTek USA.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] "Model Zoo," https://github.com/BVLC/caffe/wiki/Model-Zoo.

[8] "PyTorch Model Zoo," https://pytorch.org/docs/stable/torchvision/models.html.

[9] "Models and examples built with TensorFlow," https://github.com/tensorflow/models.

[10] "Model Zoo," https://modelzoo.co/.

[11] "Kaggle," https://www.kaggle.com.

[12] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[13] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary mathematics*, vol. 26, no. 189-206, p. 1, 1984.

[14] K. G. Larsen and J. Nelson, "Optimality of the johnson-lindenstrauss lemma," in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2017, pp. 633–638.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] A. Stam, "Limit theorems for uniform distributions on spheres in high-dimensional euclidean spaces," *Journal of Applied probability*, vol. 19, no. 1, pp. 221–228, 1982.

[17] S. Li, "Concise formulas for the area and volume of a hyperspherical cap," *Asian Journal of Mathematics and Statistics*, vol. 4, no. 1, pp. 66–70, 2011.

[18] J. Benet, "IPFS-content addressed, versioned, P2P file system," *arXiv preprint arXiv:1407.3561*, 2014.

[19] R. Rivest, "The MD5 message-digest algorithm," Tech. Rep., 1992.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2013, pp. 313–314.

[21] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[22] F. Vogelsteller and V. Buterin, "ERC-20 token standard, 2015," *URL https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md*, 2018.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[24] "Filecoin: A decentralized storage network," *URL https://filecoin.io/filecoin.pdf*, 2017.

[25] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[27] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, "Locality-preserving hashing in multidimensional spaces," in *STOC*, vol. 97. Citeseer, 1997, pp. 618–625.

[28] K. Zhao, H. Lu, and J. Mei, "Locality preserving hashing." in *AAAI*, 2014, pp. 2874–2881.

[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[30] B. Goertzel, S. Giacomelli, D. Hanson, C. Pennachin, and M. Argentieri, "SingularityNET: A decentralized, open market and inter-network for AIs," 2017.

[31] J. Eisses, L. Verspeek, and C. Dawe, "Effect network: Decentralized network for artificial intelligence," *URL http://effect.ai/download/effect_whitepaper.pdf*, 2018.

[32] R. Craib, G. Bradway, X. Dunn, and J. Krug, "Numeraire: A cryptographic token for coordinating machine intelligence and preventing overfitting," *Retrieved*, vol. 23, p. 2018, 2017.

Authorized licensed use limited to: Harvard Library. Downloaded on January 30,2021 at 14:21:27 UTC from IEEE Xplore. Restrictions apply.