Scalable Smart Contracts for Linear Regression Algorithm

Syed Badruddoja(✉), Ram Dantu, Yanyan He, Abiola Salau, and Kritagya Upadhyay

University of North Texas, Denton, TX 76207, USA
{syedbadruddoja,abiolasalau,kritagyaupadhyay}@my.unt.edu,
{ram.dantu,yanyan.he}@unt.edu

Abstract. Linear regression algorithms capture information from previous experiences and build a cognitive model to forecast the future. The information and the cognitive model representing the history of predicting future outputs must be reliable so that expected results are trusted. Furthermore, the algorithms must be explainable and traceable, making the learning process meaningful and trackable. Blockchain smart contracts boost information integrity, providing trust and the provenance of distributed ledger transactions that support such requirements. Smart contracts are traditionally developed to perform simple transactions with integer operations. However, developing learning algorithms such as linear regression with smart contracts mandates complex computation involving floating-point operations, which are not supported by smart contracts. Moreover, smart contract transactions are expensive and time-consuming. In this work, we propose a novel implementation of smart contracts for linear regression algorithms with fraction-based computation that can train and predict on the Ethereum blockchain. Our smart contract-based training and prediction technique with Solidity programming language produced a similar mean square error to the scikit-learn-based prediction model. Moreover, our design strategy saves training costs for linear regression algorithms through off-chain computations with an optimistic roll-up solution. The off-chain training and on-chain prediction strategy demonstrated in our work will help academic and industry researchers to develop cost-effective distributed AI applications in the future.

Keywords: Blockchain · dApp · Smart Contract · Artificial Intelligence · Multiple Linear Regression · Arbitrum · Ethereum

1 Introduction and Motivation

Fabricated Forecast: Artificial intelligence provides methods to make intelligent decisions for various applications [1]. The models are prepared with well-known algorithms proven to yield high accuracy with many modes of learning techniques. However, one of the crucial problems in recent development involves

W. Meng and W. Li (Eds.): BlockTEA 2022, LNICST 498, pp. 19–31, 2023.
https://doi.org/10.1007/978-3-031-31420-9_2

the trust of the data and model [2]. Data poisoning attacks wreak havoc in applications demanding predictive intelligence where input data, the training model, and output data can be questioned [3]. The training of tampered data fabricates the learning model. The manipulated model forecasts unreliable results. Moreover, the model of training and prediction results are also targets of attack. For example, if the training of the linear regression model is flawed using the tampered dataset in a weather forecast center, the forecast would produce a fake prediction. Therefore, a trusted machine learning model is mandatory to build confidence in the prediction system [5].

Explainable and Transparent AI: The machine learning models predominantly suffer from unclear training methods that make the learning process inexplicable [5]. For instance, in healthcare systems, the severity of diseases (a regression problem) requires investigation of multiple symptoms that mandates explainable features for a complete comprehension of the underlying illness [6]. However, the AI application does not explain the learning process. Moreover, the models also lack provenance to provide proof of learning [7]. Consequently, users sway away from trusting these applications due to low trust and confidence in AI applications. Moreover, AI applications raise ethical concerns about biased models on race, gender, ethnicity, and any feature relevant to the data set. A biased model can predict the wrong regression value and create more discrimination among application users [15].

Smart Contracts Scalability: Blockchain addresses trust, provenance, and explainability of AI [12] application through immutable distributed ledger and integrity feature. It works as a confidence machine for making a consensus-based transaction that is secure and intact. However, smart contracts in blockchain suffer from programming and scalability issues. The solidity programming language [23] in Ethereum blockchain (*one of the popular programming languages for developing DApps*) is a static programming language that denies floating-point computations. Therefore, cognitive smart contracts cannot produce accurate predictions. Moreover, the Ethereum blockchain also has scalability issues with transaction block limit, high computation cost, and delay in the finality of creation of block [9]. For instance, training a linear regression model with iterative optimization [10] requires thousands of iterations and function updates to optimize a model. The delay in the transaction time of the blockchain network deems the training unreasonable, and the higher cost of computations makes it unaffordable.

2 Problem Definition

A tampered linear regression model falsifies predictions, dissuading users from trusting AI applications. Hence, a trustable model is mandatory for reliable forecasts. A trustworthy model mandates untampered data, transparent learning, and explainable predictions. Blockchain smart contracts provide immutable, consensus-based, and tamper-proof transactions that can secure linear regression

models for linear regression models. Moreover, a blockchain distributed ledger provides data provenance, which helps keep the system transparent. However, smart contract languages restrict such learning capabilities due to a lack of floating-point computations. Consequently, the accuracy of learning a multiple linear regression model is unreliable and does not produce the intended regression accuracy for the sake of predictions. Moreover, the scalability of blockchain smart contracts raises concerns about developing such models, as the training of models tends to be very expensive and time-consuming.

3 Our Contribution

- Despite the limitation of the Solidity programming language, we have trained the linear regression model with an iterative optimization method in the smart contract. See Sect. 5
- We have proposed a novel architecture to train linear regression model with layer two blockchain and predict using layer one blockchain. See **Sect.** 5
- We produced comparable training accuracy on blockchain concerning scikit-learn (python machine learning library) based training. See **Sect.** 5
- Our prediction results confirm that smart contracts can predict with high efficacy compared to scikit-learn prediction (Python machine learning library). **See Table** 1, **Fig.** 3 **and** 4.
- We have reduced the cost of training multiple linear regression on blockchain network by 100 times through layer two blockchain scalability solution. **See Table** 2.

4 Literature Review

Blockchain for AI: Blockchain provides enhanced data security for storing sensitive information in diskless environments [13]. The data in the blockchain is digitally signed, ensuring the security of data for AI and enhancing trust. In healthcare systems, for instance, blockchain helps AI with cryptographic security protocols that protect patient data and make a graph database of patient healthcare systems [14]. Ethical concerns and privacy of patient data are two of the main problems when an artificially intelligent application analyzes patient data [15]. Blockchain secures the privacy of the patient data with private key and public key combinations [16] that do not reveal the patient's identity. In addition, blockchain provides automation features that are missing in machine learning applications, eventually improving performance [17]. Such applications are used for fraud detection in financial transactions. Whenever data is exposed to a private authority, it is at risk of exposure depending on the organization's interest. Blockchain helps machine learning applications build a privacy-preserving model for its prediction technique [18]. However, protecting data for integrity and privacy does not guarantee trust in the training and prediction of machine learning models.

Blockchain Integrated AI Applications: *DeepBrainChain* [19] is one of the first frameworks in the industry to run artificial intelligence platforms with blockchain technology. The project reduces the cost of AI tasks with the help of distributed resources and shares the computing load with decentralization but fails to protect AI applications. *CortexAI* [20] is a decentralized AI platform that trains machine learning models offline and predicts online to incentivize the developers and providers of the service. However, online prediction does not use smart contracts and hence lacks trust. *Algorithmia* [21] developed a Danku project that allows anyone to post a dataset and ML model for evaluation and incentivize the model owners. When we train the model outside the blockchain, the data and model become vulnerable to threats and may not be trusted, thus making the platform susceptible to poisoning attacks. Liu et al. [13] discuss the advantage of collaboration between ML and blockchain technology that aids network and communication systems. In this work, blockchain facilitates training data and a sharing model for decentralized intelligence. ML applications can utilize blockchain in communication and networking systems to provide security, scalability, and privacy in intelligent smart contracts. Such promising integration requires the blockchain application to run machine learning algorithms for prediction or classification on distributed ledger platforms. However, the attempt to secure machine learning with blockchain remains unexplored.

Smart Contract Limitations: Solidity suffers from floating point arithmetic operations as they do not allow float division, signed exponents, and other float operations. Fixidity [24], ABDK [25], PrbMath [33], and Decimalmath [34] are some of the libraries trying to implement fixed-point equivalent outcomes, but these libraries increase the transaction cost of float operations along with the integer overflow problem which makes the libraries unreasonable for a training model.

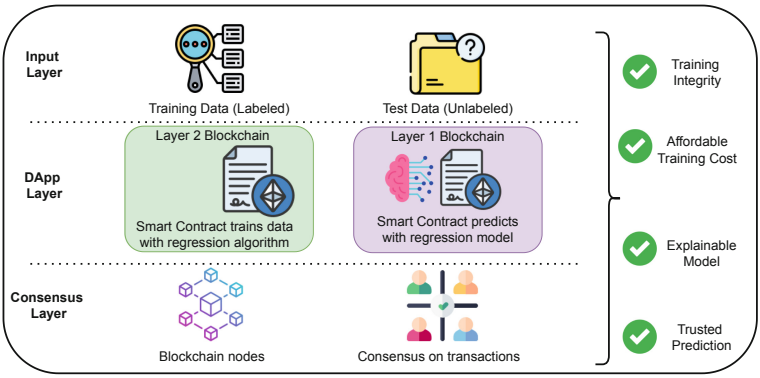

Fig. 1. Design of smart contract-based blockchain application for training multiple linear regression algorithm to optimize learning parameters for prediction purpose

5 Methodology

Design Approach: We developed a fraction-based numerical computation to train a linear regression model with smart contracts to assure integrity, provenance, and trust. We train the model with an optimistic roll-up approach (a layer two scalability method) and predict using the developed model on the blockchain (a layer one blockchain). The iterative optimization of the multiple linear regression method has a gradient descent-based learning approach that learns the parameter with a constant learning rate. The model trained with a smart contract on the Blockchain network produces consensus-based transactions which are highly trusted. Blockchain network and distributed file system together provide a provenance capability of tracing the model. Moreover, this approach also provides explainability of decisions made by the machine learning model that can be updated with a required correction. Figure 1 shows a high-level design of our proposed work. The input layer consists of the dataset. The DApp layer consists of smart contracts deployed on blockchain for training and prediction. The consensus layer computes transaction outputs with verified results.

Layer Two Blockchain for Training: Blockchain provides scalability solutions with an optimistic roll-up, zero-knowledge roll-up, sharding, and sidechains [8]. Although sharding and side chains are layer one scalability solutions of blockchain networks where the transaction delay is similar to the Ethereum network, they provide cheaper transactions. Zero-knowledge roll-up is a layer two blockchain scalability solution that produces complex cryptographic proofs that make the computations highly complex, resulting in inexplicable AI. On the other hand, optimistic roll-up (a layer two scalability solution) assumes that the miners are honest and will produce cheaper transactions with faster outputs, which is ideal for training machine learning algorithms. We chose optimistic roll-up as the blockchain network for training the linear regression algorithm and compared the performance with the Ethereum test network.

Optimistic Roll-up for Off-chain Training: Optimistic roll-ups execute transactions parallel to Ethereum main chain. After all the transactions are complete, the last state change is stored on the main chain [8]. This increases the speed of transactions from 10 to 100 times. "Optimistic" refers to the aggregate of bare minimum information required to be stored without proof, assuming no fraud is committed. Optimism and Arbitrum are two of the platform that implements optimistic roll-ups with layer two blockchain solutions. The proof is provided only when fraud is committed.

Multiple Linear Regression: Multiple linear regression involves learning multiple parameters to form a line of the equation that can best fit a model [10]. Equation 1 shows the prediction formula for linear regression where we have to learn and optimize weights and biases which are $W_1, W_2, ...Wn$ and c. The learning of parameters is performed through the iterative optimization method. The same equation is referred to as $\overline{y}$(referred to as $yhat$) for training purposes.

The $\overline{y}$ is computed repeatedly with updated weights and biases. Multiple linear regression implementation can be detailed at [10]. The next section details how we have implemented iterative optimization with smart contracts.

$$y = W_1x_1 + W_2x_2 + W_3x_3..... + W_nx_n + c \tag{1}$$

Event Flow: Figure 2 shows the event flow of our proposed model, where AI application developers access our smart contracts to train linear regression models on the blockchain network. Later, an AI user access the prediction smart contract to predict the desired outcome through the blockchain network.

Iterative Optimization: The iterative optimization model is implemented with a fraction-based computation to ensure that Solidity smart contract can execute those functions on the Ethereum Platform. Iterative optimization involves 3 steps. Step 1 is to compute $\overline{y}$ with random weight and bias parameters. From step 1, we get the mean square error between $\overline{y}$ and the actual y value. Step 2 involves the derivative computation of weights and biases annotated as *delta_weight* and *delta_bias* concerning the mean square error. Lastly, step 3 involves updating weights and biases with learning rate α [10].

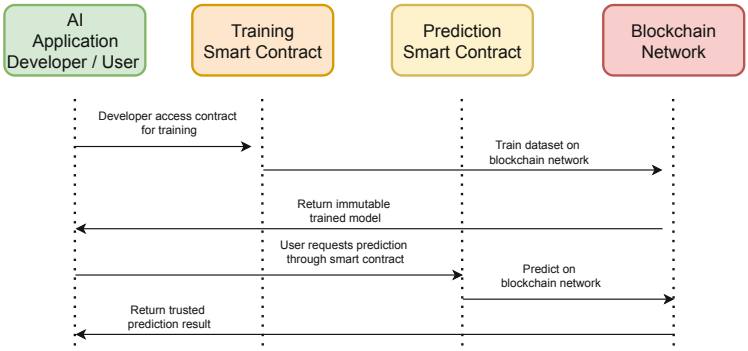

Fig. 2. Event flow of regression model development through smart contracts where training a model and prediction are computed on blockchain networks.

$$get\_y\_hat = \frac{\overline{y\_num}}{\overline{y\_den}} = \frac{weight\_num}{weight\_den} * \frac{x\_num}{x\_den} + \frac{bias\_num}{bias\_den} \tag{2}$$

$$get\_y\_diff = (\frac{\overline{y\_num}}{\overline{y\_den}} - \frac{y_i\_num}{y_i\_den}) \tag{3}$$

$$\begin{aligned} get\_delta\_weights &= \frac{\delta w\_num}{\delta w\_den} \\ &= \frac{1}{N}\sum_{i=0}^{n-1} 2 * \frac{x_i\_num}{x_i\_den}(get\_y\_diff) \end{aligned} \tag{4}$$

$$get\_delta\_bias = \frac{\delta b\_num}{\delta b\_den} = \frac{1}{N}\sum_{i=0}^{n} 2 * (get\_y\_diff) \tag{5}$$

$$\frac{weight\_num}{weight\_den} = \frac{weight\_num}{weight\_den} - \frac{\alpha\_num}{\alpha\_den}.\frac{\delta w\_num}{\delta w\_den} \tag{6}$$

$$\frac{bias\_num}{bias\_den} = \frac{bias\_num}{bias\_den} - \frac{\alpha\_num}{\alpha\_den}.\frac{\delta b\_num}{\delta b\_den} \tag{7}$$

Fraction Transformation for Multiple Linear Regression: We have obtained a fraction-based computational method from the standard iterative optimization method that transfers decimal numbers into a fraction for performing iterative optimization in the solidity smart contract. The Eq. 2 computes $\overline{y}$ values multiplies $weights$ with $features$ and adds biases. The $\overline{y}$ is represented with $\overline{y\_num}/y\_den$ $(numerator/denominator)$. Equation 3 subtracts true y value from $\overline{y}$. The Eq. 4 computes the gradient descent derivative of $weights$ concerning the difference between the true values of training data and $\overline{y}$ values that are computed with Eq. 2. Similarly, Eq. 5 provides the derivative of bias. After completing all the derivative computations the new parameters are updated with Eq. 6 and Eq. 7. All the parameters are computed in fractions with numerator and denominator terms.

6 Experimental Setup

We have considered two datasets for testing our hypothesis. The two datasets are "Diabetes progression" [28] and "Real state valuation" [35] with 442 and 414 samples. The "Diabetes Progression" dataset provides a quantitative measure of diabetes progression concerning age, sex, body mass index, average blood pressure, and six blood serum measurements for 442 diabetes patients. The "Real state valuation" dataset provides price estimates of real estate concerning transaction date, house age, distance to nearest meter station, number of nearby convenience stores, latitude, and longitude with geographical coordinates. These datasets have categorical and continuous variables. Data are pre-processed with label encoders to convert categorical values to continuous variables for smart contract inputs. Moreover, we have deployed the training smart contract on layer two blockchain network (Arbitrum) [27]. The prediction smart contract is deployed in the Ethereum Ropsten test network. To build a comparable analysis, we deployed a linear regression model with the scikit-learn library to record baseline performance accuracy. Scikit-learn [28] provides a set of python standard libraries for various AI algorithms.

7 Performance Evaluation

Prediction Accuracy: The Table 1 provides mean square errors in the prediction of all the datasets for smart-contract-based prediction and python-based

prediction. The prediction error of diabetes and real-state cost are close for both smart contract and python deployment in Table 1, which ensures the reliability of training with smart contracts.

Table 1. Mean square error comparison between smart contract-based and python scikit-learn based prediction

Dataset	MSE in Smart Contract	MSE in Python
Diabetes Progression	2865	2900
Real State Cost	72.67	1.00

The Fig. 3 and 4 provide the comparison of ground truth, library prediction, and smart contract prediction values for the test datasets of diabetes progression and real state costs. The graph in Fig. 3 shows that the scikit-learn-based prediction and smart contract-based predictions are converging. The accuracy of prediction is approximately 95% to the python-library-based function. Furthermore, the graph in Fig. 4 provides another convincing prediction result close to the ground truth that confirms that the mean square error is low.

Cost of Smart Contracts Functions: The smart contract transaction cost for Ethereum Rospten test network follows the formula $transactioncost(Ethers) = gas\_used * (gasprice + basefee))/10^9 Gwei$ [8]. We have plotted the *get_y_hat* function (another name for $\overline{y}$) in Fig. 5 with y-axis showing rise of cost in GWei and x-axis as number of features. It is clear from the graph that the cost of computing $\overline{y}$ forms a linear relationship with a number of features and is predictable. The cost computations of the remaining functions are plotted in Fig. 6. The cost of transaction for get_delta_w, and get_y_diff similarly forms linear relationships

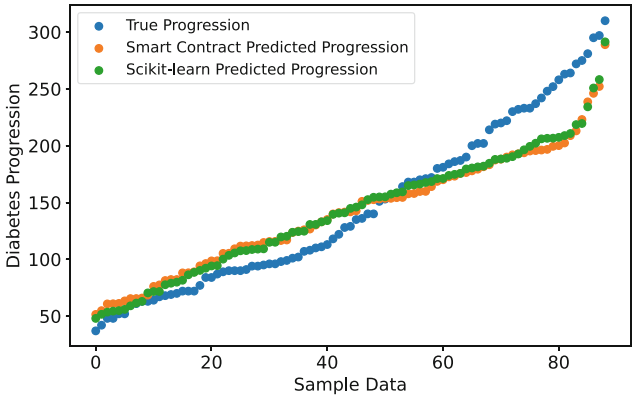

Fig. 3. Comparison of true sorted diabetes progression with predicted progression for scikit learn library and blockchain smart contracts. The scikit-learn and smart contracts produced similar progression estimates.

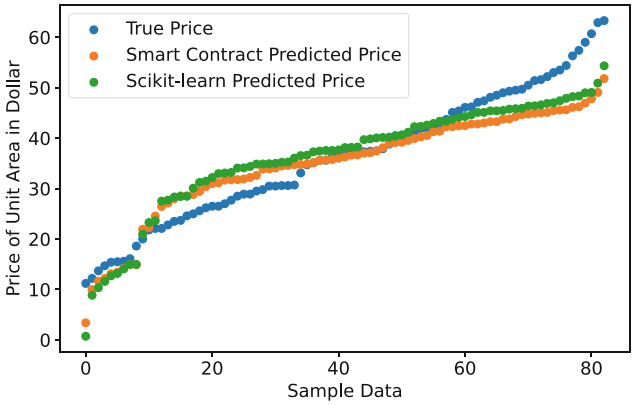

Fig. 4. Comparison of true sorted real state valuation (price) with predicted valuation (prices) for scikit learn library and blockchain smart contracts. The scikit-learn and smart contracts produced similar progression estimates.

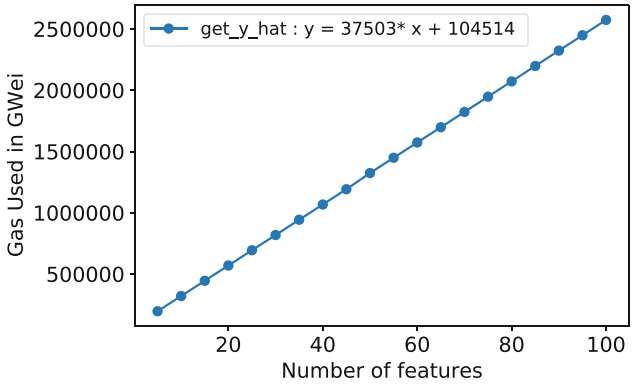

Fig. 5. Shows the cost of computing *get_y_hat* function on Ropsten test network for a rising number of data samples. The relationship is linear for the x-axis and y-axis, with a slope value of 37503.

with rising number of samples as shown in Fig. 6. The slope of get_delta_w is greater than get_y_diff due to the higher number of computations involved in calculating weight parameters.

The Table 2 provides the comparative analysis of the cost of training a single iteration for 353 samples on Rospten and Arbitrum networks. The cost of *get_y_hat* computation is the highest among all the functions for the Ethereum Ropsten network. It computes the $\overline{y}$ values for all the samples. The number of function execution is equivalent to the number of examples in the training dataset. Linear regression requires more than 1000 iterations to optimize weights with iterative optimization methods, and the cost of $\overline{y}$ function is 2587 ethers, US dollar equivalent to 50,590,801, which is drastically high. Conversely, the

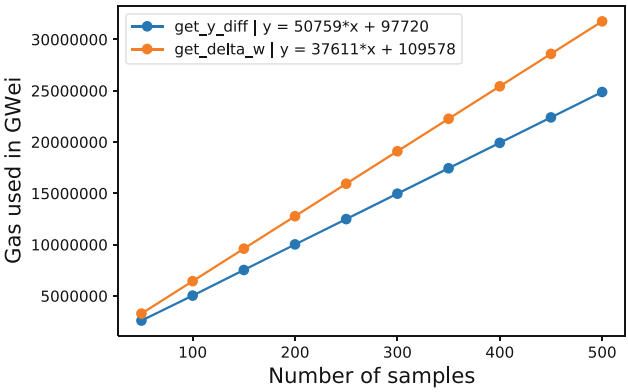

Fig. 6. Shows the cost of computing $get\_y\_diff, get\_delta\_weights, get\_delta\_biasand$ function on Ropsten test network. The slope of get_delta_w is greater than get_y_diff due to the higher number of computations involved in calculating weight parameters.

Table 2. Training cost of a single iteration of multiple linear regression for Diabetes progression dataset with 353 samples of training on Ropsten Ethereum Network and Arbitrum network.

Function Name	Ethereum Cost	Ethereum Time (Seconds)	Arbitrum Cost	Arbitrum Time (Seconds)
get_y_hat	2.587768	4500–6000	0.0066046	300
get_y_diff	0.31607	3–25	0.00080997	1–2
get_delta_w	0.17934	2–30	0.0033095	1–2
get_delta_b	0.00879	3–25	0.00016627	1–2
get_new_weights	0.00451	3–26	0.00000573	1–2
get_new_bias	0.00072	2–34	0.00000571	1–2

Arbitrum network seems to reduce the price more than 100 times and make the cost of training a model on a blockchain platform more affordable. The cost of training the model with 1000 iterations would be approximately 10.016 Arbitrum ethers, equivalent to 0.0033 USD (1 Arbitrum ether = 0.0002933 USD).

Table 3. Shows a rise in the number of operations for fraction-based computations for smart contract functions while compared with non-fraction-based computations.

Function	Decimal Flops	Fraction Flops
get_y_hat	2 nm	6 nm
get_delta_w	2 nm	6 nm
get_delta_b	n	4n

8 Computational Analysis

The number of lines of code and operations in blockchain smart contracts is crucial for the computational cost of functions defined underneath. We have analyzed the difference in the number of computations between fraction and non-fraction-based calculations. We have considered Watkins's "Fundamental of Matrix Computation" book [29] for the analysis of the number of operations involved in our application. Considering a training dataset of $m * n$ matrix, Table 3 shows that the number of computations will increase by approximately 3–4 times when calculations are performed with fractions compared to decimal counts. This rise in the count will impact the cost of smart contract functions.

9 Limitation and Challenges

Block Gas Limit: The gas limit for the Ethereum Ropsten transaction reached block capacity to compute a higher number of computations to develop a linear regression model with a smart contract. For Ropsten, the default gas limit is a hard limit of 4712388 GWei [30] as current information. Due to the number of iterations involved in computing the entire set of functions, the gas capacity could not be controlled. The main Ethereum network has a default block gas limit of 15,000,000 Gwei [31] and can be increased to 30,000,000 Gwei. We assume that the Ethereum main network will allow a higher number of computations due to the higher block gas limit.

10 Conclusion

Smart contracts do not allow floating-point computations for linear regression algorithms. This hinders the development of the AI model in blockchain networks. We have proposed a novel approach to develop a trustable machine learning model with the help of blockchain technology and make the artificially intelligent application more secure. Our work also shows that static smart contracts can be transformed into learning smart contracts by running machine learning algorithms inside the blockchain network. We have deployed a smart contract with a multiple linear regression mechanism to train our models on blockchain and achieved excellent training accuracy concerning mean square error computation. We have also achieved good prediction accuracy for the model learned on-chain. Moreover, our solution minimizes the cost of training linear regression algorithms using optimistic roll-up (layer two blockchain). We have analyzed the cost of training a machine learning model and showed that the optimistic roll-up saves the training cost by more than 100 times. In the future, we aim to develop more AI algorithms using smart contracts with further investigation and analysis.

References

1. Bangbit Technologies: Introduction to artificial intelligence (AI): a deep dive into machine learning & deep learning (2019). https://medium.com/@BangBitTech/introduction-to-artificial-intelligence-ai-a-deep-dive-into-machine-learning-deep-learning-4763e6985344
2. Bantis, A.C.: Is your ML model secure. https://medium.com/slalom-technology/is-your-ml-model-secure-fe10b8589b71. Accessed Sep 2021
3. Pitropakis, N., et al.: A taxonomy and survey of attacks against machine learning. Comput. Sci. Rev. **34**, 100199 (2019). https://doi.org/10.1016/j.cosrev.2019.100199
4. Liao, Q.V., et al.: Introduction to explainable AI. In: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–3 (2021)
5. Kale, A., et al.: Provenance documentation to enable explainable and trustworthy AI: a literature review. Data Intell. 1–41 (2022)
6. Pawar, U., O'Shea, D., Rea, S., O'Reilly, R.: Explainable AI in healthcare. In: 2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), pp. 1–2 (2020). https://doi.org/10.1109/CyberSA49311.2020.9139655
7. Kastner, C.: Versioning, provenance, and reproducibility in production machine learning (2021). https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005
8. Ethereum, W.: Ethereum whitepaper. Ethereum (2014). https://ethereum.org. Accessed 07 July 2020
9. Parizi, R.M., Dehghantanha, A.: Smart contract programming languages on blockchains: an empirical evaluation of usability and security. In: Chen, S., Wang, H., Zhang, L.J. (eds.) Blockchain (ICBC 2018). LNCS, vol. 10974, pp. 75–91. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94478-4_6
10. Neto, J.: Multiple linear regression from scratch using Python (2021). https://medium.com/analytics-vidhya/multiple-linear-regression-from-scratch-using-python-db9368859f
11. Shafiq, A.B.: Which methods should be used for solving linear regression? https://www.kdnuggets.com/2020/09/solving-linear-regression.html
12. Salah, K., et al.: Blockchain for AI: review and open research challenges. IEEE Access **7**, 10127–10149 (2019). https://doi.org/10.1109/ACCESS.2018.2890507
13. Liu, Y., et al.: Blockchain and machine learning for communications and networking systems. IEEE Commun. Surv. Tutor. **22**(2), 1392–1431 (2020). https://doi.org/10.1109/COMST.2020.2975911
14. Campbell, D.: Combining AI and blockchain to push frontiers in healthcare. https://www.macadamian.com/learn/combining-ai-and-blockchain-in-healthcare/
15. Bartoletti, I.: AI in healthcare: ethical and privacy challenges. In: Riaño, D., Wilk, S., ten Teije, A. (eds.) AIME 2019. LNCS (LNAI), vol. 11526, pp. 7–10. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21642-9_2
16. Kumar, R., Tripathi, R.: Secure healthcare framework using blockchain and public key cryptography (2020)
17. Wang, T.: A unified analytical framework for trustable machine learning and automation running with blockchain. IEEE Trans. Big Data **2018**, 4974–4983 (2018). https://doi.org/10.1109/BigData.2018.8622262

18. Kim, H., Kim, S., Hwang, J.Y., Seo, C.: Efficient privacy-preserving machine learning for blockchain network. IEEE Access **7**, 136481–136495 (2019). https://doi.org/10.1109/ACCESS.2019.2940052.27
19. Zou, J., et al.: DeepBrainChain: artificial intelligence computing platform driven by blockchain. White Paper. https://cryptorating.eu/whitepapers/DeepBrain-Chain/DeepBrainChainWhitepaper.pdf. Accessed Nov 2021
20. Chen, Z., Wang, W., Yan, X., Tian, J.: Cortex-AI on blockchain- the decentralized AI autonomous system. White Paper. https://cryptorating.EU/whitepapers/Cortex/Cortex_AI_on_Blockchain_EN.pdf. Accessed Nov 2021
21. Kurtulmus, A.B., Daniel, K.: Trustless machine learning contracts; evaluating and exchanging machine learning models on ethereum blockchain. https://arxiv.org/pdf/1802.10185.pdf
22. Harris, J.D., Waggoner, B.: Decentralized and collaborative AI on blockchain. IEEE Int. Conf. Blockchain **2019**, 368–375 (2019). https://doi.org/10.1109/Blockchain.2019.00057
23. Solidity Programming guide. https://docs.soliditylang.org/en/v0.8.9/. Accessed Sept 2021
24. Fixidity fixed point library for solidity. https://github.com/CementDAO/Fixidity. Accessed Nov 2021
25. ABDK library for solidity. https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/ABDKMath64x64.sol. Accessed Nov 2021
26. Ethereum white paper, "Scaling" (2022). https://ethereum.org/en/developers/docs/scaling/
27. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitron: scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 2018), pp. 1353–1370 (2018)
28. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
29. Watkins: Fundamentals of matrix computations. https://davidtabora.files.wordpress.com/2015/01/david_s-_watkins_fundamentals_of_matrix_computat.pdf
30. Moriya, H.: How to get ethereum block gas limit. https://piyopiyo.medium.com/how-to-get-ethereum-block-gas-limit-eba2c8f32ce. Accessed Dec 2021
31. Notik, D.: Ethereum. https://ethereum.org/en/developers/docs/gas/. Accessed Dec 2021
32. Project Implementation: "Github Source". https://github.com/syber2020/LR-KNN-6950-FA21/tree/master/LR-Python-Web3/MLR
33. PRBMath library. https://github.com/paulrberg/prb-math. Accessed July 2022
34. Decimalmath. https://github.com/alcueca/DecimalMath. Accessed July 2022
35. Yeh, I.C., Hsu, T.K.: Building real estate valuation models with comparative approach through case-based reasoning. Appl. Soft Comput. **65**, 260–271 (2018)