Advances in Science, Technology and Engineering Systems Journal Vol. 9, No. 3, 72-83 (2024)

www.astesj.com

Special Issue on Computing, Engineering and Multidisciplinary Sciences

ASTESJ

ISSN: 2415-6698

Deploying Trusted and Immutable Predictive Models on a Public Blockchain Network

Brandon Wetzel, Haiping Xu

Computer and Information Science Department, University of Massachusetts Dartmouth, Dartmouth, MA 02747, USA

A R T I C L E I N F O

Article history:
Received: 25 March, 2024
Revised: 12 May, 2024
Accepted: 29 May, 2024
Online: 16 June, 2024

Keywords:
Machine learning
Predictive model
Public blockchain
Smart contract
On-chain storage

A B S T R A C T

Machine learning-based predictive models often face challenges, particularly biases and a lack of trust in their predictions when deployed by individual agents. Establishing a robust deployment methodology that supports validating the accuracy and fairness of these models is a critical endeavor. In this paper, we introduce a novel approach to deploying predictive models, such as pre-trained neural network models, in a public blockchain network using smart contracts. Smart contracts are encoded in our approach as self-executing protocols for storing various parameters of the predictive models. We develop efficient algorithms for uploading and retrieving model parameters from smart contracts on a public blockchain, thereby ensuring the trustworthiness and immutability of the stored models, making them available for testing and validation by all peers within the network. In addition, users can rate and comment on the models, which are permanently recorded in the blockchain. To demonstrate the effectiveness of our approach, we present a case study focusing on storing vehicle price prediction models and review comments. Our experimental results show that deploying predictive models on a public blockchain network provides a proficient and reliable way to ensure model security, immutability, and transparency.

1. Introduction

Tasks that once required intensive manual labor can now be automated with the aid of artificial intelligence and machine learning (AI/ML) models, guaranteeing efficient completion in just minutes [1]. However, as AI/ML technologies revolutionize every aspect of our lives, ensuring the validity and fairness of these models becomes crucial. For example, in the field of recruitment, many organizations are incorporating machine learning methods into their hiring processes when dealing with high volumes of job applications. To address bias concerns, New York City implemented a groundbreaking law in July 2023 that requires all automated employment decision tools (AEDTs) to undergo bias audits prior to deployment, and the results of these audits must be made available to the public [2]. While the new regulations can help to ensure the trustworthiness of AI/ML models, it is widely recognized that such an approach shifts the responsibility for trust to a centralized regulatory body, which itself cannot guarantee its trustworthiness. An improved solution should allow users to validate the machine learning-based tools on their own and expose every facet of the tools to the public. This transparency would enable users to validate the functionality and efficiency of AI/ML tools and the absence of bias, thus fostering trust in these tools. This need for trust is not limited to AI/ML tools but pervades various domains where intricate algorithms may pose a challenge to human comprehension. Hence, there is a vital necessity to develop a practical and trustworthy mechanism for deploying immutable and publicly accessible AI/ML-based tools, such as predictive models, that can be easily tested and validated by users.

A blockchain network is a peer-to-peer, decentralized ledger that eliminates the need to trust a centralized audit mechanism. Blockchain was initially recognized for its key role in cryptocurrency systems such as Bitcoin and Ethereum, securing and decentralizing transaction ledgers [3], [4]. However, the use of blockchain technology extends far beyond cryptocurrency applications. Blockchain technology can be employed to ensure the integrity and permanence of data in diverse industries, signifying data that is verifiable and untamperable [5]-[7]. A blockchain securely organizes data into linked blocks using cryptographic techniques, ensuring a tamper-proof and immutable record of transactions and smart contracts. The decentralized design of the blockchain network, with its features of immutability, security, and transparency, naturally addresses the issue of public accountability for data stored in the blockchain network. Therefore, if AI/ML-based predictive tools are deployed on a public blockchain, users of the AI/ML systems no longer need to rely on internal auditing for assurance. This paper uses ML-based neural network predictive models as an example to demonstrate how to deploy such models on a public blockchain

*Corresponding Author: Haiping Xu, University of Massachusetts Dartmouth, Dartmouth, MA 02747, Email: hxu@umassd.edu

for predicting vehicle prices, an area that is susceptible to biases driven by company or dealer interests. Machine learning models might inadvertently set prices too high or too low for specific vehicle types in order to optimize profit. The main goal of this paper is to provide a secure way of deploying machine learning tools that support user validation of predictive models to ensure that the models operate in the best interest of the user, rather than serving corporate goals. To this end, the model should receive vehicle parameters and generate price predictions, allowing users to validate the predicted prices even if they do not have specialized knowledge. In addition, users should be able to obtain validation from other users, which could be supported by the establishment of a comprehensive rating system.

In this paper, we present a novel approach to deploying predictive models using smart contracts by storing the parameters of the predictive models on a public blockchain. This method allows a user to recreate the predictive models on a local computer using the stored model parameters. The user can then validate a version of the model by running certain test cases to ensure the required accuracy and fairness of the model predictions. In addition, we define a mutable meta-block (*MB*) attached to the beginning of the blockchain to record indexing information of all models deployed on the blockchain network. The *MB* can be used to efficiently retrieve the predictive models and their review comments, thereby significantly improving the performance of the blockchain network system.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the public blockchain framework that supports the use of smart contracts to store predictive models. It also describes the structural design of the regular blocks for storing predictive models and review comments, and the meta-block for efficiently retrieving models and the review comments. Section 4 describes in detail the procedures for deploying and retrieving predictive models and reviews. Section 5 presents a case study and the analysis results. Section 6 concludes the paper and mentions future work.

2. Related Work

Blockchain is a secure means of storing immutable and protected transactional data through a peer-to-peer decentralized ledger system. Recent advances have extended the traditional application of blockchain in cryptocurrencies to the storage of medical records, including the use of InterPlanetary File System (IPFS) in blockchain based healthcare secure storage solutions [8]. In [9], the authors proposed a blockchain model for creating a secure system for storing and sharing Electronic Health Records (EHRs). Their strategy is to store large amounts of medical data in the cloud, while the blockchain is dedicated to storing metadata related to EHRs. In [10], the authors introduced a storage framework integrating blockchain and IPFS for efficient transaction storage within the blockchain. In their architecture, the primary patient reports are stored in a decentralized off-chain storage system using IPFS, and the blockchain exclusively stores hash values of these reports. This approach effectively reduces the overall block size within the blockchain. In [11], the authors overcame the security concerns of pure cloud storage and provided a robust and scalable solution that utilizes blockchain technology to handle big data storage of healthcare multimedia files. The approach introduces a hierarchical cloud-based blockchain framework for storing large amounts of healthcare data, resulting in significant efficiencies. Similarly, in [12] and [13], blockchain is again used as a step towards digital healthcare by storing medical images and EHRs directly in the blockchain instead of storing their metadata. Their goal in storing healthcare data via blockchain is to apply the security of blockchain to the healthcare system, mitigating the threat of tampering and increasing security. Blockchain technology has also been used in other areas to store data that may be public but must be trusted to be accurate, which is more similar to the intended application of this paper. For example, in [14], the authors proposed a public auditing mechanism for cloud storage systems using blockchain. They demonstrated that the proposed scheme is secure against type I/II/III/IV adversaries. In [15], the authors described a system that utilizes blockchain technology for video surveillance storage and sharing of videos that are encrypted but publicly accessible with a decryption key. Their approach consists of encrypting and storing camera-acquired video off-chain using distributed IPFS and storing the associated metadata in the blockchain. In addition, some other researchers proposed a secure method for sharing sensitive financial data using blockchain [16]. Their methodology involves recording access control rules, hash values, and storage addresses of financial data in the blockchain, while storing the actual financial data in a distributed database external to the blockchain. While blockchain technology has been effective in ensuring the immutability of stored data across various domains, our approach differs from existing methods in that, in addition to storing regular transactional data, we deploy ML-based predictive models in the blockchain network that are publicly available for validation and adoption by all users.

The challenge of bias and fairness in ML models has been a prevalent issue that currently lacks a universally accepted solution. A comprehensive survey on bias and fairness across various ML domains identified many areas of possible bias that can exist in AI/ML models [17]. ML models investigated in a number of real-world commercial use cases have shown that data integrity, learning parameters, and the lack of safeguards against bias can have a significant impact on the fairness of a model. The survey provides a taxonomy for determining the fairness of models and suggests that this is a pervasive problem for which researchers must find practical solutions. Furthermore, as noted in [18], mitigating bias in datasets for supervised ML is a pressing need for the emerging field of ML. Expanding on this idea, some researchers used statistical methods to assess the significant amount of bias in ML algorithms. For example, in [19], gender bias in facial recognition algorithms was measured and found to be prevalent, whereas in [20], the bias in the training data and its effect on the predictive results were detected and analyzed. The above approaches introduce methods for analyzing and assessing the amount of bias in ML models, and generally describe an approximation of the amount of bias in certain areas of ML. However, each of these proposed solutions for fairness and bias assessment requires trust in a centralized organization. In the absence of trust, e.g., when the model organizer has a strong incentive to manipulate the price of an object, a satisfactory solution remains elusive. In contrast, our approach utilizes the inherent security offered by blockchain technology for the storage of ML models. By employing decentralized hosting of ML models within a blockchain network, each user can independently verify the accuracy of the models deployed by different organizations. This approach eliminates the need for users to worry about the fairness and potential bias of the models, as they can perform validation checks on each model individually.

The prevalence of substantial bias in ML models necessitates the need for dedicated research to mitigate bias. While bias mitigation relies on suitable bias assessment methods tailored to each unique problem, distinct solutions are needed. In [21], the authors introduced a novel approach to mitigating bias by visually displaying the attributes of an ML model to indicate bias in an intuitive manner. Subsequently, manual adjustments are required to rectify these attributes by modifying the dataset. In [22], the authors proposed a methodology for assessing fairness specific to dynamic pricing methods, where the fairness constraints on price prediction are assessed by a centralized entity. This provides a valuable mechanism for measuring and enforcing the fairness of model-based price prediction, even when the demand for the item is unpredictable. In [23], the authors described how to mitigate bias in neural network-based ML models trained on image data. The authors showed three main categories of bias mitigation methods, namely data-level techniques to balance the training data, model-level approaches to modify the learning algorithm, and adversarial approaches to identify biases in the data. Unlike the above methods, our approach reduces bias by employing a review system to ensure that users can easily find better-reviewed ML models, which tend to be less biased. Our bias mitigation strategy is centered on minimizing the bias directly encountered by users. Furthermore, our approach is versatile and applicable to a wide range of ML models, bypassing the reliance on statistical rigor and instead emphasizing user satisfaction.

The fusion of blockchain and machine learning is an emerging frontier, with ongoing research underscoring its promising potential. In [24], the authors presented various techniques concerning the preservation and processing of big data necessary for ML models. While this decentralized machine learning approach has advantages in terms of mitigating bias and improving fairness, the proposed solution falls short in describing the potentials for evaluating and validating the models, focusing mainly on the data stored in the blockchain. In [25], the authors described an ML-based approach where training gradients are validated by a decentralized blockchain network. They proposed a decentralized learning framework called LearningChain, where organizations can collaborate on training models by computing gradients together to update specific models. In [26], a similar approach to [25] was used to compute the parameters of an on-chain ML in a decentralized multi-threaded environment. Using association rule mining as an example, the authors showed how blockchain can be used to enable trusted machine learning. In [27], the authors further advanced the idea of using blockchain in the learning process by introducing explainable and traceable algorithms that make the learning process meaningful and trackable. Their study showed that smart contract-based training and prediction techniques produced mean square errors similar to scikit-learn-based prediction models. In [28], the authors analyzed the design space regarding the integration of machine learning into blockchain applications, a concept termed ML on chain. They presented a taxonomy for ML on chain, categorizing existing and prospective approaches based on design attributes and their characteristics. While the above methods inherently offer the fundamental security advantages associated with blockchain technology, our approach is different because we only store the model parameters on the blockchain, excluding the data used in training, as all training is done off-chain. Note that storing training data on a large public blockchain would be costly and impractical. Thus, by focusing on storing only the essential components needed to reconstruct ML models, our approach optimizes deployment efficiency on a blockchain platform. Furthermore, we allow users to review a model to verify its fairness. This enables even non-technical users to make informed decisions about the degree of bias present in the model. Table 1 summarizes the main contributions and novelties of our approach by comparing it with existing approaches in terms of five key features.

Table 1: Comparison with Existing Approaches

Approach	Trustless	Fairness	Scalability	Security	Performance
Bias Mitigation Methods [21]-[23]	No	Yes	N/A	No	N/A
Conventional Blockchain [25]-[28]	Yes	No	No	Yes	No
Our Approach	Yes	Yes	Yes	Yes	Yes

As shown in Table 1, a feature can be supported (Yes) or not supported (No) by an approach, or not applicable (N/A) to an approach. The features we consider include whether the method is trustless, whether the fairness of the model can be verified, whether the method is scalable and secure, and whether the method supports improved performance in terms of efficiently searching for stored predictive models. Methods that focus on bias mitigation are typically centralized [21]-[23], so they do not support trustless computing. Since the fairness constraints of stored models are assessed by centralized entities, no security mechanisms are provided to users. On the other hand, while conventional blockchain approaches support trustless and secure computing [25]-[28], they typically do not support bias mitigation for stored models and are also not scalable due to on-chain training of predictive models. Furthermore, since conventional blockchains do not contain a meta-block, they do not support efficient search for stored predictive models.

3. Deploying Predictive Models on a Public Blockchain

3.1. A Framework for a Public Blockchain with Smart Contracts

Each peer in a blockchain network may have a complete copy of the blockchain with a number of blocks. Figure 1 shows a framework for a public blockchain with smart contracts, as well as the list of transactions and the list of smart contracts stored in block B_2. As shown in the figure, each block in the blockchain consists of three main parts: a block header, a list of transactions, and a list of smart contracts. The block header is defined as a 5-tuple (*BID, HPB, BTS, NTR, NSC*), where *BID* is the block ID, *HPB* is the hash value of the previous block, *BTS* is the timestamp when the block was created, and *NTR* and *NSC* are the number of transactions and the number of smart contracts contained within the block, respectively. Transactions are used to record new transactional information added to the blockchain, such as associated data related to a predictive model and review data of the model provided by users. Each transaction is defined as a 5-tuple (*TID, UID, TTS, DES, DAT*), where *TID* is the ID of the transaction, *UID* is the ID of the user who initiated the transaction, *TTS* is the timestamp when the transaction was created, *DES* is the detailed description of the transaction, and *DAT* is any associated data stored with the transaction.

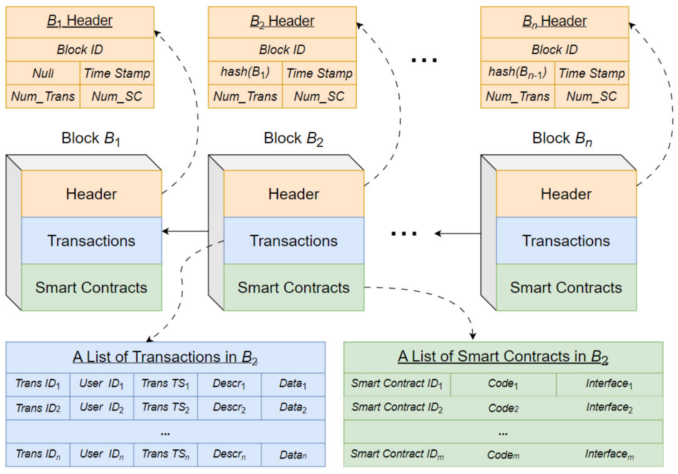

B_1 Header
Block ID
Null
Time Stamp
Num_Trans
Num_SC
B_2 Header
Block ID
hash(B_1)
Time Stamp
Num_Trans
Num_SC
B_n Header
Block ID
hash(B_{n-1})
Time Stamp
Num_Trans
Num_SC
Block B_1
Block B_2
Block B_n
Header
Transactions
Smart Contracts
A List of Transactions in B_2
Trans ID_1 | User ID_1 | Trans TS_1 | $Descr_1$ | $Data_1$
Trans ID_2 | User ID_2 | Trans TS_2 | $Descr_2$ | $Data_2$
Trans ID_n | User ID_n | Trans TS_n | $Descr_n$ | $Data_n$
A List of Smart Contracts in B_2
Smart Contract ID_1 | $Code_1$ | $Interface_1$
Smart Contract ID_2 | $Code_2$ | $Interface_2$
Smart Contract ID_m | $Code_m$ | $Interface_m$

Figure 1: A Framework for a Public Blockchain with Smart Contracts

Smart contracts are initially defined as self-executing code stored in the blockchain and are often seen as the terms of an agreement between two parties. These terms are encoded directly into the smart contract, which is publicly accessible, allowing both parties to fully agree to these terms. In this paper, we broaden the concept of smart contracts to include methods for storing and accessing immutable data on the blockchain. We define a smart contract as a 3-tuple (*SID*, *COD*, *INF*), where *SID* is the identifier of the smart contract, *COD* is the code of the smart contract, which is usually stored as binary code, and *INF* is the interface of the smart contract, which defines the methods that can be invoked. The process of deploying a smart contract involves creating a transaction *T*, where *T.DES* describes the smart contract to be deployed, including the smart contract's location in the block, and *T.DAT* may include any associated data, e.g., a test dataset of a predictive model that is encoded in the smart contract.

Running unverified code, such as smart contracts, on a user's local machine can pose potential security risks. To address this issue, we can assign the responsibility of verifying the trustworthiness of smart contracts on a blockchain to full-node peers. Additionally, executing smart contracts on a virtual machine, such as the Ethereum Virtual Machine (EVM), can help minimize security vulnerabilities by restricting their interactions within the virtual environment. This approach ensures a deterministic execution of smart contracts, as the virtual machine maintains full control over the execution environment.

3.2. Storing Predictive Models Using Smart Contracts

Smart contracts being used as storage for predictive models provide a layer of abstraction that increases the efficiency for users to access and verify predictive models stored on a blockchain network. For example, a predictive model stored with a smart contract can be efficiently extracted by invoking a corresponding method defined in the smart contract. We call a smart contract that stores a predictive model a Model Smart Contract (*MSC*), which contains the serialized parameters of the model and a set of methods that can be used to access and verify the model. Every version of a model must have both a model ID (*MID*) and a version ID (*VID*), where *MID* and *VID* refer to the type of model and an instance of that model, respectively. For example, when *MID* refers to a set of predictive models used to forecast a company's stock price, each model version refers to a specific prediction model. Models sharing the same *MID* can be published by the same publisher at various times or by different publishers, but it is essential for each model to have a unique *VID*.

In our approach, predictive models are trained off-chain and only trained models are stored on-chain. Each peer can extract and recreate an exact copy of the trained model on a local machine by invoking a smart contract method defined in the corresponding *MSC*, which then allows the user to validate and execute the model without having to trust a centralized party. A smart contract containing a predictive model requires an associated transaction to be stored in the same block, documenting its deployment, location, and related data. The review comments of a model are also recorded in transactions but do not require an associated smart contract to access them. Instead, they are stored as plain text in a review transaction, directly accessible to peers. If transaction *T* tracks the deployment of an *MSC*, *T.DES* and *T.DAT* describe the *MSC* and its associated data, respectively. In contrast, if transaction *T* records the review data for a model version, *T.DES* and *T.DAT* contain the *VID* of the model and the corresponding review of the model, respectively. Figure 2 shows *m* versions of a predictive model that are stored on a blockchain network.

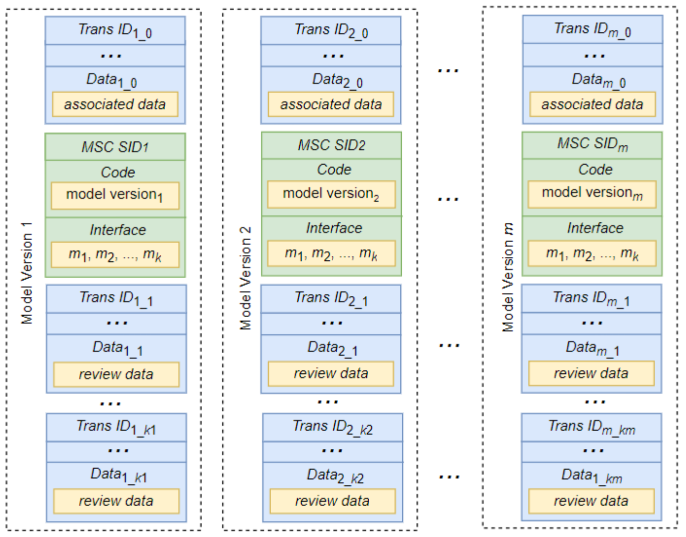
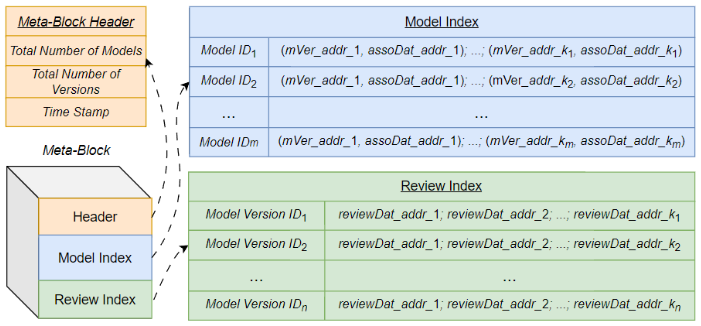


Figure 2: A Predictive Model with Multiple Versions and Review Data

As shown in Figure 2, each model version *VID* is stored in an *MSC* that defines its smart contract ID, the code of the model version, and an interface listing the methods that can be invoked by users. Each *MSC* model version i, where $1 \leq i \leq m$, stored in block *B* is accompanied by a transaction with ID i_0 that describes the model version and is stored in the same block *B*. Suppose model version i has k_i reviews. These reviews are stored in transactions with IDs from i_1 to i_k_i. Note that since the reviews for a model version are uploaded at different times, the review transactions can be stored in different blocks other than block *B*.

3.3. The Structure of a Meta-Block

In order to store indexing information for efficiently finding specific versions of a predictive model and all reviews about the model versions, we introduce the *MB*, which is a mutable meta-block attached to the blockchain. When a new block is approved and added to the blockchain, each peer needs to update its *MB* so that it contains the latest indexing information about the models, versions, and review comments contained in the new block. Figure 3 shows the structure of the *MB* that consists of three parts, namely header, model index, and review index.


Meta-Block Header
Total Number of Models
Total Number of Versions
Time Stamp
Meta-Block
Header
Model Index
Review Index
Model Index
Model ID$_1$ (mVer_addr_1, assoDat_addr_1); ...; (mVer_addr_k$_1$, assoDat_addr_k$_1$)
Model ID$_2$ (mVer_addr_1, assoDat_addr_1); ...; (mVer_addr_k$_2$, assoDat_addr_k$_2$)
... ...
Model IDm (mVer_addr_1, assoDat_addr_1); ...; (mVer_addr_k$_m$, assoDat_addr_k$_m$)
Review Index
Model Version ID$_1$ reviewDat_addr_1; reviewDat_addr_2; ...; reviewDat_addr_k$_1$
Model Version ID$_2$ reviewDat_addr_1; reviewDat_addr_2; ...; reviewDat_addr_k$_2$
... ...
Model Version ID$_n$ reviewDat_addr_1; reviewDat_addr_2; ...; reviewDat_addr_k$_n$
Figure 3: The Structure of a Meta-Block

The *MB* header contains the general information of the *MB*, which is defined as a 3-tuple (*NM*, *NV*, *TS*), where *NM* and *NV* are the total number of models and the total number of versions, respectively, and *TS* is the timestamp when *MB* was last updated. The model index and the review index modules contain indexing information about model versions, associated data and review data in terms of their addresses in the blockchain. The address of a model version is defined as a pair (*BID*, *ISC*), where *BID* is the ID of the block that stores the smart contract containing the model version, and *ISC* is the index of the smart contract within the list of smart contracts in the block. Similarly, the address of the model associated data or review data is defined as a pair (*BID*, *ITR*), where *BID* is the ID of the block that stores the associated data of the model or its review data, respectively, and *ITR* is the index of the transaction that records the information about the model version and review data within the list of transactions in the block. As shown in the figure, the model index module can be implemented in the form of a HashMap, where the key is an *MID* and the output value is a list of pairs (*mVer_addr*, *assoDat_addr*), containing the model version address and the model associated data address, i.e., the address of the smart contract with the predictive model and the address of the transaction that records information about the smart contract, respectively. Likewise, the review index module can also be implemented in the form of a HashMap, where the key is a *VID* of a model version, and the output value is a list of addresses that can be used to locate the transactions that record the reviews about the model version.

4. Deployment and Retrieval of Predictive Models

4.1. Process Overview

To support efficient peer usage of a blockchain network, we require a distinction between full-node peers and regular peers, where a full-node peer contains a full copy of the entire blockchain, while a regular peer is not required to maintain a full copy but can make a request to a full-node peer for information retrieval and uploading. In this paper, we allow regular peers to retrieve a predictive model from a full-node peer and also to send reviews to a full-node peer for publication. Full-node peers have the privilege of deploying models by adding an *MSC* and its accompanying transaction to a new block. When a new block is added, it is broadcast to all full-node peers for updating. Figure 4 shows the process of model deployment and extraction on a blockchain with the *MB* and *m* blocks. As shown in the figure, the blockchain contains models deployed by *n* full-node peers. There are also *r* regular peers, each of which is connected to a full-node peer and can utilize this connection to retrieve a model or post a review for the model. A full-node peer connected to the regular peer can add these retrieval transactions and reviews to a new block. Once the new block contains enough transactions, the full-node peer can broadcast the new block to all full-node peers on the network for approval and adoption.

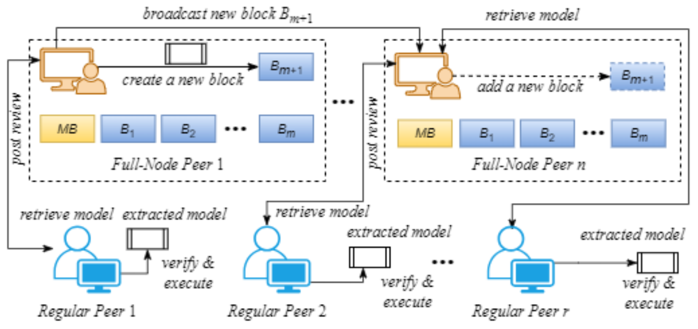


Figure 4: An Overview of the Model Deployment and Extraction Process

4.2. Generating a Meta-Block

The process of generating an *MB* involves searching each block in a blockchain. While searching a block, the addresses of models and reviews are found and added to the *MB*. Each model transaction *tr*, where *tr.DES* describes an *MSC* with an *MID*, must have the accompanying smart contract found in the same block. The address of *tr* and the address of the smart contract described in *tr.DES* are paired and added to the *MB*'s *Model Index* HashMap with *MID* as a key. If the pair represents the first instance of a model, an empty list of addresses is created before the pair of addresses can be added. On the other hand, if *tr* is a review transaction that contains a review on a particular model *VID*, its address is added to the *Review Index* HashMap with *VID* as a key. The complete process to generate an *MB* for a public blockchain is described in Algorithm 1. According to the algorithm, we search for transactions in each block of the blockchain. For each transaction, if it describes the deployment of an *MSC* with *MID* and the model has not been recorded in *MB*, the pair <*MID*, [(*mVer_addr, assoDat_addr*)]> is added to the *Model Index*, where *mVer_addr, assoDat_addr* are the address of the model version and the address of the associated data transaction, respectively. On the other hand, if the transaction is a review transaction for a model version *VID*, the pair <*VID*, [*reviewDat_addr*]> is added to the *Review Index*, where *reviewDat_addr* is the address of review transaction for the model version *VID*. This process is repeated for all transactions in each block of the blockchain until all blocks have been processed. While generating an *MB* on an existing blockchain can be a time-consuming process, this process only needs to be performed once to be usable, thereby significantly reducing the time it takes to search and extract data from the blockchain. Additionally, a peer can download an already existing *MB* from another peer that has the latest *MB*, thus eliminating the need to search through the blockchain to create a new *MB*. Note that when a new block is added to the blockchain, the *MB* can be updated in a similar manner as defined in Algorithm 1.

Algorithm 1: Generating a Meta-Block for a Public Blockchain

Input: A public blockchain *PB* with *n* blocks
Output: A generated *MB*

```
1.  Create an empty MB with Model Index and Review Index HashMap
    $\Pi_M$ and $\Pi_R$, respectively
2.  Initialize the total number of models NM to 0
3.  Initialize the total number of model versions NV to 0
4.  Set the time stamp TS to the current time
5.  for each block B in PB
6.      for each transaction tr in block B
7.          if tr.DES describes a model with MID stored in MSC
8.              if $\Pi_M$ does not contain any versions with key MID
9.                  Add <MID, [(mVer_addr, assoDat_addr)]> to $\Pi_M$
10.                 Increase NM by 1
11.             else Update the list of model versions in $\Pi_M$ with key MID
12.         else if tr.DES describes a review of a model version VID
13.             if $\Pi_R$ does not contain any reviews of VID
14.                 Add <VID, [reviewDat_addr]> to $\Pi_R$
15.                 Increase NV by 1
16.             else Update the list of reviews in $\Pi_R$ with key VID
17. return MB
```

4.3. *Deploying Models and Posting Reviews*

The process of deploying a classifier can only be done by a full-node peer. This process must first check whether the corresponding model version has been deployed. The *MB* is used to ensure that a particular version of a model has not yet been deployed to the blockchain by checking whether the version being deployed has been indexed by the *MB*. Formally, we define a model as a 4-tuple (*MDA, MID, VID, ASD*), where *MDA* is the data required to recreate the model, stored in a standard way of storing model data such as HDF5 [29]; *MID* and *VID* are the model ID and version ID, respectively; and *ASD* is the associated data of model version *VID*. Deploying a model α requires the creation of a transaction T_α and a model smart contract MSC_α, where MSC_α contains a method for extracting model data for the model version. The transaction T_α contains a description of MSC_α including the use cases of the model and the address of MSC_α. T_α must also contain any associated data, ASD_α. T_α and MSC_α are then added to a new block *B* that is being deployed. Deploying review β requires the creation of a review transaction, T_β, where T_β contains the review being posted. A new block *B* may contain multiple models or reviews, as defined by the full-node peer running the algorithm. Once block *B* is ready, it is broadcast to all full-node peers for approval before it can be published and added to the blockchain. If block *B* is successfully published, the *MB* must be updated to index the new data in block *B*. Algorithm 2 describes the process of generating and deploying a new block with multiple new models on a public blockchain. The efficiency of this process is predominantly influenced by the duration needed to broadcast block *B* and the quantity of transactions and smart contracts contained within the new block. Note that the process of publishing a review is similar to the process of deploying a model, but without the need to create an *MSC*. Therefore, in Algorithm 2, we do not show the process of adding a review transaction to the new block *B*. Once the new block *B* is approved and deployed, the *MB* needs to be updated to add the model information and the review data addresses to the HashMaps *Model Index* and *Review Index*, respectively.

Algorithm 2: Generate and Deploy a New Block

Input: A public blockchain *PB* with an *MB;* a list of new models L_{NM}
Output: Boolean value indicating successful or failed deployment

```
1.  Let B be a new block to be generated and deployed
2.  for each predictive model $\alpha$ in $L_{NM}$
3.      if MB contains key $\alpha$.MID in the Model Index HashMap
4.          Let $\Gamma_M$ be a list of deployed models with key $\alpha$.MID
5.          if $\Gamma_M$ contains a model with version ID $\alpha$.VID
6.              continue  // the model version has already been deployed
7.      Create a new model smart contract $MSC\_\alpha$ for model $\alpha$
8.      Add $MSC\_\alpha$ to the list of smart contracts in B
9.      Create a new transaction $T\_\alpha$ with model data $\alpha$.ASD and a
        description of $MSC\_\alpha$
10.     Add $T\_\alpha$ to the list of transactions in B
11. Broadcast B and await consensus approval determination
12. if new block B is not approved
13.     return false  // failed deployment
14. else Update MB to include new information from B
15. return true // successful deployment
```

4.4. *Extracting Models and Reviews*

For a regular peer, the process of extracting a model of *MID* from the blockchain requires a connection to a full-node peer in order to make the request. When a full-node peer receives such a request, it retrieves all its model versions of *MID* and their reviews from the blockchain, and then returns the result as a list of model instances to the regular peer. Algorithm 3 describes the process of extracting all predictive models of *MID* and their reviews from the public blockchain *PB* with an *MB*.

Algorithm 3: Extracting Predictive Models and their Reviews

Input: A public blockchain *PB* with an *MB*; a model *MID* to retrieve all its model versions and their reviews
Output: A list of model instances of *MID* along with their reviews

```
1.  Initialize $L_M$ to an empty list of model instances of MID
2.  if MB contains the key MID in the Model Index HashMap
3.   Let $\Gamma_M$ be a list of (mVer_addr, assoDat_addr) with key MID
4.  else return $L_M$   // $L_M$ is an empty list, i.e., no model is found
5.  for each pair (mVer_addr, assoDat_addr) in $\Gamma_M$
6.    Let the model to which the pair refers be model $\alpha$ with VID_$\alpha$
7.    Create a new empty model instance $\Lambda$
8.    Extract model $\alpha$ stored at mVer_addr
9.    Extract associated data assoDat stored at assoDat_addr
10.   Add model $\alpha$ and associated data assoDat to $\Lambda$
11.   if MB contains the key VID_$\alpha$ in the Review Index HashMap
12.   Let $\Gamma_V$ be a list of reviewDat_addr with key VID_$\alpha$
13.   for each reviewDat_addr in $\Gamma_V$
14.   Extract the review reviewDat stored at reviewDat_addr
15. Attach reviewDat to $\Lambda$
16.   Add $\Lambda$ to the list of model instances $L_M$
17. return $L_M$
```

As shown in Algorithm 3, the full-node peer first searches for *MID* in its *Model Index* HashMap. If the model with *MID* does not exist, an empty list of model instances is returned. Otherwise, each model version must be extracted from the blockchain and added to a list of model instances L_M. For each pair (*mVer_addr*, *assoDat_addr*), both the model α stored at *MSC_addr* and the model data *assoDat* stored at *assoDat_addr* are extracted from the corresponding blocks in the blockchain and added to a model instance *Λ*. Then, the full-node peer searches for *VID_α* in its *Review Index* HashMap. For each review stored at address *reviewDat_addr*, it is extracted from the corresponding block in the blockchain and attached to the model instance *Λ*. Once all review comments have been retrieved, *Λ* is added to the list of model instances L_M. Finally, when all model instances have been retrieved, L_M is returned to the regular peer with the retrieval request. Note that without the *MB*, the extraction process requires traversing the entire blockchain. With the *MB*, it is possible to quickly find the addresses of predictive models and their review comments, thus making the extraction process very efficient.

5. Case Study and Simulation Results

The case study presented in this section demonstrates the efficacy of storing predictive models on a public blockchain. The stored models are multilayer perceptron (MLP) neural network classifiers that predict vehicle prices based on given parameters such as age and mileage. MLP classifiers were chosen due to their relatively large storage requirements compared to other ML methods such as linear regression. By showcasing the successful storage of MLP classifiers on a public blockchain network, we enhance our confidence in the adaptability of our approach to other types of predictive models and domains. This provides a springboard for more ambitious future research efforts. The MLP classifiers used in this case study are deployed by vehicle dealerships with an *MID* for each model, and a *VID* for each version of these models. For simplicity, in this case study, the *MID* and *VID* contain the vehicle model and vehicle year as substrings, respectively. For example, the classifier that predicts the price of a 2011 Toyota Camry, deployed by car dealer *D* on March 15, 2024, may have an *MID* of "Toyota Camry" and a *VID* of "2011_*D*_03152024".

5.1. Environment Settings

We have developed two different modules: one using the Ethereum blockchain [4] and the other providing users with a web interface to interact with the blockchain. To experiment on the Ethereum blockchain, we employed Ganache, a tool commonly used for testing and development [30]. As with many smart contracts on Ethereum, the smart contracts developed in this case study were also coded in Solidity. While this approach is experimentally tested on the Ethereum network, it also involves simulating processes such as connecting to a full-node peer to retrieve data from the blockchain. In addition, since the approval of new blocks takes place through a consensus mechanism [31], we simulated the approval times of new blocks, assuming that the times are normally distributed with appropriate means and standard deviations (*STDs*). In this sense, we collect data points partly from measurements of real-world implementations and partly from simulated values that best fit real-world scenarios.

The web interface was created to allay the concerns of regular users about the complexity of blockchain usage and to establish a secure environment for conducting the case study. Regular users can interact with the user-friendly interface to search for classifiers and their review comments, select suitable classifiers, extract classifiers and execute them on their local machines. It also allows regular users to create, publish, and read reviews comments through the interface. The website interacts with Ethereum through the *Web3.js* tool, facilitating user engagement with the Ethereum nodes. A regular peer can use the website by connecting to a full-node peer and assembling a classifier locally with the support of the *TensorFlow.js* tool. Review comments on a classifier are uploaded and viewed through the website by regular users. Various types of users such as companies or dealers can create and train classifiers with TensorFlow before deploying them to the blockchain through the interface.

5.2. Space Efficiency for Storing Classifiers

Space efficiency is crucial in a public blockchain because the blockchain needs to be stored on every full-node peer. This experiment provides a basis for the expected spatial storage of our proposed solution. In an MLP classifier, parameters are numerical values that quantify the complexity of the classifier. TensorFlow contains efficient methods for storing a classifier that allow us to isolate these parameters. The number of parameters *nP* in an MLP classifier can be calcaulted as in (1).

$$nP = \sum_{i=1}^{n-1}\left(\left(Layer_{i} + 1\right) * Layer_{i+1}\right) \tag{1}$$

where $Layer_i$ refers to the number of neurons in layer *i* of the classifier. The rationale for this equation is that for each neuron in a layer, the number of connections to that node must be the same as the number of neurons in the previous layer plus a connection from a bias neuron. For example, a model with 5 input neurons, 1 output neuron, and 3 hidden layers, each with 128 neurons, would have a total of 33,536 parameters. This quantifies the complexity of the classifier, which allows for comparisons between the different scenarios in this case study.

In our approach, when determining the storage size of a classifier, we consider the size of the classifier itself as well as the overhead of storage, including the storage size of the methods defined in a smart contract and the size of the transactions that track the deployment of the classifier. The storage overhead is usually stable, while the size of the stored classifier can be efficiently measured by the number of bytes stored for its parameters. In addition, we must also account for the number of reviews for each classifier. We define a random number of reviews for each classifier by skewing the normal distribution [32], [33], and define its *pdf* (probability density function) and *cdf* (cumulative density function) using the *pdf* and *cdf* of the normal distribution as shown in (2-5).

$$pdf_{norm}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2}$$

$$cdf_{norm}(x) = \int_{-\infty}^{\frac{x-\mu}{\sigma}} pdf_{norm}(t)dt \tag{3}$$

$$pdf_{skew}(x) = 2 * pdf_{norm}\left(\frac{x-\xi}{\omega}\right) * cdf_{norm}\left(\alpha\frac{x-\xi}{\omega}\right) \tag{4}$$

$$cdf_{skew}(x) = \int_{-\infty}^{\frac{x-\xi}{\omega}} pdf_{skew}(t)dt \tag{5}$$

In (2-5), the parameters μ and σ represent the mean and *STD* of a normal distribution and are set to 0 and 1, respectively. Parameters α, ξ, and ω represent the *shape*, *scale* and *location* of a skew normal distribution and their values are chosen to be 10, 0, and 200, respectively, to better match real-world scenarios. The rational for using this distribution and the chosen parameter values is based on the observations on websites such as Amazon and others with reviews of items, where only a few items are highly popular and reviewed but most items are unpopular and have much fewer reviews. With this understanding, we can adjust the parameter values to create a right-skewed distribution we would find in the real world. The chosen values can produce a highly skewed distribution, similar to the distribution of product reviews on Amazon. This distribution is visualized as in Figure 5, where the distribution has a high variance and a significant skew to the right. The mode can be visually shown to be approximately 70 reviews, although the mean is expected to be larger due to the significant skew.

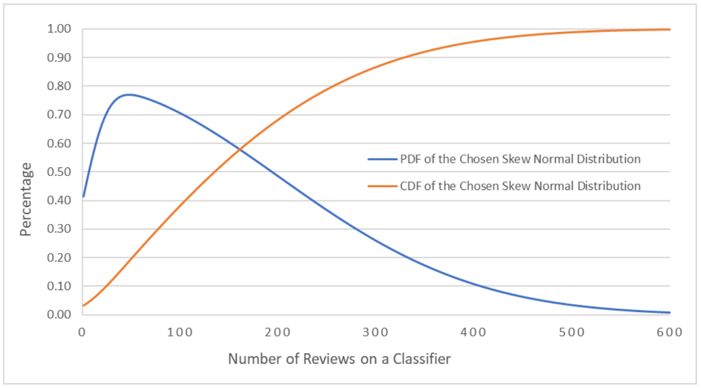


Figure 5: An Example of Skew Normal Distribution of Reviews

The mean, *STD* and skewness of a skew normal distribution can be calculated as in (6-8):

$$mean = \xi + \omega\delta\sqrt{\frac{2}{\pi}}\ , where\ \delta = \frac{\alpha}{\sqrt{1+\alpha^2}} \qquad (6)$$

$$STD = \omega\sqrt{1 - \frac{2\delta^2}{\pi}} \qquad (7)$$

$$skewness = \frac{4-\pi}{2}\frac{(\delta\sqrt{2/\pi})^3}{\left(1-2\delta^2/\pi\right)^{3/2}} \qquad (8)$$

Table 2 shows the resulting mean, *STD*, and skewness of the number of reviews, as well as the *MAX* and *MIN* chosen for the skew normal distribution.

Table 2: Chosen Parameters for the Size of Review Storage

	Number of Reviews[1]	Review Transaction Size (KB)[2]
Mean	159	0.35
STD	121	0.03
Max	600	0.6
Min	0	0.25
Skewness	0.96	0

[1] Number of Reviews follows a skew-normal distribution with skewness of 0.96
[2] Review Transaction Size follows a normal distribution (skewness = 0)

As shown in Table 2, we assume the review transaction size follows a normal distribution with a mean of 0.35KB, a standard deviation of 0.03, and a skewness of 0. Note that review transactions stored on Ethereum require a minimum transaction size, while the maximum size is set to prevent malicious attacks on the blockchain network, such as posting unreasonably large reviews. With a generated random number of reviews for a classifier, we can calculate the total space S_{cla} to store the classifier and its reviews on Ethereum as in (9).

$$S_{cla} = MSCSize(claComp) + RevSizes(numRev) \qquad (9)$$

In (9), *claComp* is the classifier complexity (i.e., the number of model parameters), *MSCSize* is a function that computes the amount of storage space required to store the model's *MSC* in a block, *numRev* is a random variable representing the number of reviews, and *RevSizes*(*numRev*) is the size of *numRev* randomized review transactions. The function *MSCSize* was determined from experimental test data on the Ethereum network, where a linear relationship was found between the complexity of a classifier and the size of the associated *MSC* containing the method of extracting the model parameters of a classifier.

Figure 6 shows the 500 data points generated based on the chosen parameters listed in Table 2 and the storage space required on Ethereum. Each data point is generated by randomizing the model complexity (i.e., a random number of parameters in the model) and yields the storage space required to store the model and the associated transaction describing the model. We further generate the random number of reviews and their review transaction sizes based on the distributions provided in Table 2. The storage sizes of the classifier and all its reviews are summed to yield the total size required to store the model on the Ethereum blockchain network.

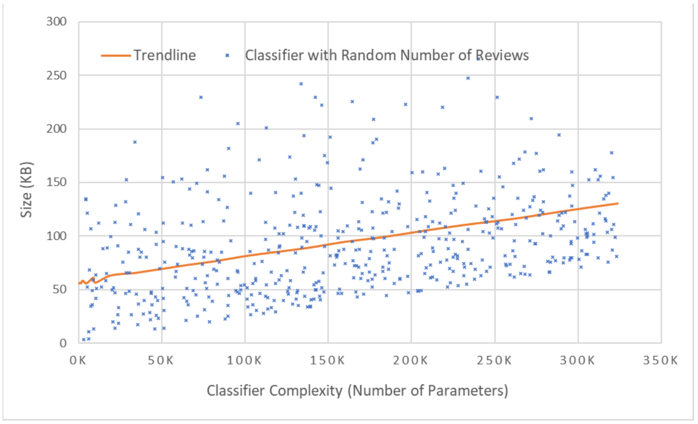


Figure 6: Storage Size of Uploaded Classifier and Their Reviews

As shown in Figure 6, the trendline clearly demonstrates that the average size of storing a model on Ethereum is reasonably small and increases with classifier complexity. While the presence of highly reviewed models puts the trendline slightly above the large clustering of models, the size required to store such classifiers is typically less than 250KB, even at the far end of the range where high-complexity classifiers are accompanied by many reviews. This result is promising and demonstrates both the spatial efficiency and scalability of this approach. For example, if there are 100 car dealerships, each deploying 50 models per year, with approximately 200K parameters in each model, for 10 years, the blockchain storage requirement would be merely 10GB. It is worth noting that while we used Ethereum for our experiments in this paper, our vision for the future is to develop dedicated public blockchain networks for storing specific types of predictive models. Thus, such dedicated public blockchain networks could be scalable and available for a long period of time (e.g., 10+ years). In addition, we could also consider mitigating the scalability issue further by using historical blockchains, such as those discussed in [34].

5.3. Time Efficiency for Deploying a New Block

The amount of time to deploy a new block containing multiple classifiers includes the time to broadcast a new block to other full-node peers and the time to approve the new block using a consensus mechanism. The broadcast time can be estimated based on the broadcast rate and sizes of the classifiers included in the new block. As shown in Table 3, we assume that the size of the classifier is normally distributed with a mean value of 200KB for storing the classifier parameters with different model complexity and the overhead required to store the model on Ethereum. Note that the classifier size distribution requires a maximum value to avoid unrestricted file upload attacks on the blockchain network and a minimum classifier size to avoid deployment of trivial models on the blockchain. Considering the uncertainty of network traffic and the possible bandwidth constraints that may be enforced by the full-node peers, we assume that the broadcast rate and the approval time (i.e., the consensus time) are also normally distributed with the parameters listed in Table 3.

Table 3: Parameters for Deploying a New Block with Multiple Classifiers

	Broadcast Rate (KB/s)	Classifier Size (KB)	Consensus Time (s)
Mean	500	200	12
STD	300	20	2
Max	1,000	1,000	100
Min	200	10	8

Our estimate of the approval time is based on the average time it takes to add a new block to the Ethereum blockchain, which is about 12 seconds [35]. Similarly, the consensus time must be able to time out (e.g., 100 seconds) when there are not enough full-node peers to approve a new block. Since classifiers must be deployed through blocks, the number of transactions or classifiers in each block directly affects the time required for deployment. The time required to broadcast a new block to other full-node peers increases with the number of transactions and classifiers, as well as the sizes of the transactions and classifiers. The time to deploy a new block T_{deploy} can be generated using (10), where *ClaSizes*(*numCla*) returns the size of *numCla* classifiers in the new block, *BCR* is a random broadcast rate, and *ConsTime* is a random consensus time.

$$T_{deploy} = \frac{ClaSizes(numCla)}{BCR} + ConsTime \quad (10)$$

Figure 7 shows the time required to deploy a new block with 1 to 10 classifiers. There are 30 data points for each number of classifiers, each representing a new block, and the variables are chosen randomly according to the distributions defined in Table 3. The trendline in the figure shows how much deployment time is expected to be needed as the number of classifiers increases. With 10 classifiers, the expected deployment time is about 18 seconds, which is a reasonable waiting time for the users. We note that among the above data points, the time used for the approval process accounts for the largest percentage of the total deployment time, while the percentage of the time used for broadcasting new blocks increases slightly with the number of classifiers. Based on the experimental results, it is desirable to include more classifiers in the same block when there is a high demand for uploading classifiers in a short period of time.

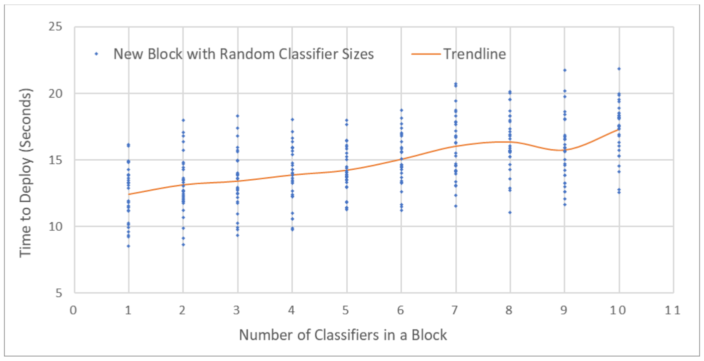


Figure 7: Time to Deploy a New Block

5.4. Generation of a Meta-Block

Another key use case for this approach is the generation of an *MB* by a full-node peer. This process is largely affected by the number of transactions in each block and the total number of blocks in the blockchain. Since there is usually an upper limit to the number of transactions that can be reasonably placed in a block, the time to generate an *MB* would be approximately linear with the number of blocks in the blockchain. In a real-world setting, a full-node peer needs to connect to other full-node peers to verify the consistency of its blockchain before generating an *MB*. The time to connect to other full-node peers for such verification is assumed to be normally distributed with a mean time of 10 seconds and a *STD* of 2 seconds. The time T_{genMB} to generate a meta-block can be calculated using (11), where *GenMB* is the time to generate an *MB* based on experimental tests, and *ConnPeer* is the randomly simulated time to connect to a full-node peer node.

$$T_{genMB} = GenMB + ConnPeer \quad (11)$$

Figure 8 shows the simulation time to generate an *MB* based on the number of blocks in the blockchain.

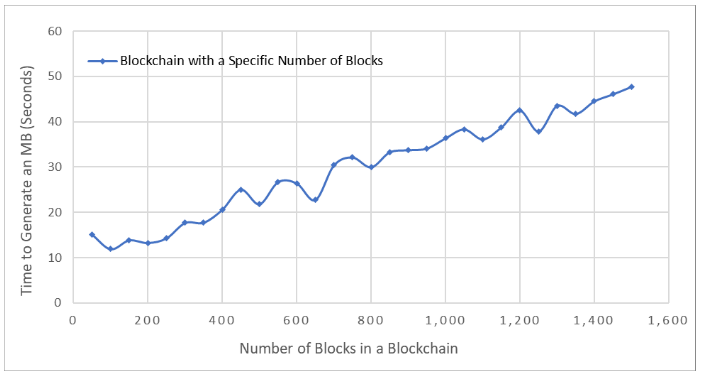


Figure 8: Time to Generate a Meta-Block

Note that Figure 8 only shows up to 1,500 blocks; however, based on the degree of linearity shown in the figure, we can infer how long it takes to generate an *MB* for a blockchain with a larger number of blocks. Additionally, the stochastic impact of the time to connect to a full-node peer is reduced when the number of blocks increases. This is because the connection time is centered on a 10-second average, so as the number of blocks increases, the impact of the connection time becomes smaller compared to the total generation time. For a blockchain with 1,500 blocks, the time required to generate an *MB* in less than a minute is reasonable, and the fact that it only needs to be executed once demonstrates the effectiveness of this approach.

5.5. Analysis of Search and Retrieval Time

The search and retrieval (S&R) time for classifiers and their reviews is the main waiting time for regular users; therefore, we must minimize this time as much as possible. For each retrieval request made by a regular peer, several versions and their review comments must be retrieved and sent back. While the search time through an *MB* is expected to be efficient, regular peers that do not maintain the blockchain need time to connect to a full-node peer, which retrieves the requested data and returns results to the regular peer. Thus, the S&R time $T_{S\&R}$ consists of three components as shown in (12): the time required to connect to a full-node peer from the regular peer (*ConnPeer*), the time required for the full-node peer to search for each requested classifier and its review comments, and the time required to send this data back to the regular peer (i.e., the download time). Note that the download time is based on the total size of the classifiers *ClaSizes*, the total size of all reviews *RevSizes*, and the download rate (*DLR*) for downloading the classifiers and their reviews.

$$T_{S\&R} = ConnPeer + Search(n) + \frac{ClaSizes + RevSizes}{DLR} \quad (12)$$

The search time is measured on an Ethereum blockchain network. According to Algorithm 3, the expected search time *Search*(*n*) is approximately linear as the number of blocks *n* increases. The number of reviews for each model version follows the skew normal distribution defined in Table 2. The size of a classifier, the size of a review and the download rate are randomly generated following the normal distribution defined in Table 4.

Table 4: Parameters for Searching and Retrieving Classifiers

	Review Size (KB)	Classifier Size (KB)	Download Rate (KB/s)	Connection Time (Sec)
Mean	0.1	100	500	10
STD	0.03	50	300	2
Max	0.35	800	1,000	100
Min	0	10	200	1

As shown in Table 4, the size of each review and classifier is significantly different from those listed in Table 2 and Table 3, respectively. This is because in this experiment, we only need to download the reviews and classifiers themselves and not the transactions associated with them. It is important to note that transactions stored with Ethereum usually contain additional information that is not necessary for a regular peer to download. The time to download data is determined by the amount of data downloaded, defined as the last term in (12). In addition, we assume that *ConnPeer* is normally distributed, with a mean time of 10 seconds and a *STD* of 2.

Figure 9 shows the time required to search and retrieve three arbitrarily chosen classifiers and their reviews versus the number of blocks in a blockchain. To demonstrate the effectiveness of our approach, two different search methods are included in the figure: one is our approach using *MB*, and the other is a conventional blockchain method that does not use *MB*. The conventional approach searches for all transactions from the newest block to the oldest block and saves reviews and model data during the search. We conducted 250 experiments with each approach, where a data point represents a request from a regular peer. The S&R time for each data point is calculated using (12), where the search times were measured by simulations on Ethereum.

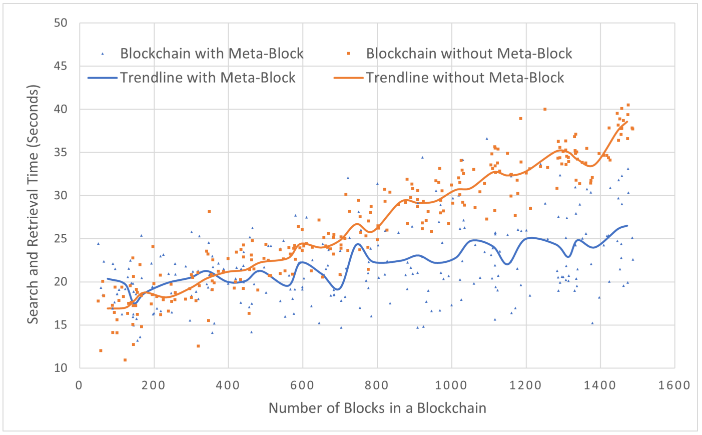


Figure 9: Search and Retrieval Time for Classifiers and Their Reviews

In order to clearly demonstrate the effectiveness of our approach, we show the trendlines for both methods. As can be seen from the trendlines, the average S&R time of the blockchains using *MB* remains relatively stable. The difference between a large number of blocks and a small number of blocks is about 5 seconds, while the conventional method without using *MB* takes significantly longer as the number of blocks increases. Thus, based on the experimental results, our approach greatly improves the ability to search a blockchain compared to a conventional approach without using *MB*.

It is worth noting that the experimental results are based on real-world data collected from the Ethereum blockchain network, as well as simulated data designed to demonstrate the practical implications of the case study. The inclusion of carefully selected distribution values enhances the credibility of the case study and increases confidence in the case study solution.

6. Conclusions and Future Work

As machine learning becomes an increasingly important part of many people's lives, the need to address the issue of trust in these algorithms grows every day. The approach proposed in this paper aims to completely eliminate the need for trust by combining machine learning models with a decentralized public blockchain network. Machine learning models are stored on a blockchain through smart contracts, which contain methods for accessing and validating the stored models. Transactions are used to track the deployment of models and record their reviews. Thus, model users do not need to trust individual organizations, but only peer reviews and their own audits of machine learning algorithms. To further enhance the user experience, we add a meta-block at the beginning of the blockchain to record indexing information for all models and reviews stored on the blockchain. Our experimental results on Ethereum show that our approach is effective and efficient when deploying predictive models on a public blockchain network.

For future work, we plan to develop dedicated public blockchain networks for deploying predictive models. Storage and computation costs can be reduced as the scale of a dedicated public blockchain network is more manageable. The impact of congestion on the blockchain is also a key issue that needs to be addressed [36]. In our current approach, the approval of new blocks and the retrieval of classifiers and their reviews are handled by full-node peers. In future work, we may consider developing efficient load balancing mechanisms for full-node peers [34] to mitigate the impact of blockchain network congestion on system performance. Furthermore, combined with the trustless nature of blockchain, we can bring more systematic changes to the way the machine learning models are created. This could include publishing metrics for training data or goals for the training process on the blockchain. Additionally, training could take place entirely on the blockchain, rather than off-chain as in our approach. While an on-chain approach may lead to scalability issues, it allows users to trust the process of creating the model rather than the model itself. This can be important in situations where the trustworthiness of a model cannot be easily validated by regular users. To elaborate on this approach, a new smart contract could be designed to handle on-chain training, where full-node peers have the ability to define new methods and execute them to train predictive models. Finally, we can consider implementing automated auditing of new AI/ML models [37]. This automated approach helps eliminate undesirable AI/ML models from being stored on the blockchain, thus saving model storage space. Based on previous work [38], new models can be initially deployed in a temporary block, transitioning to a permanent block upon successful completion of auditing. This auditing procedure can be automated, facilitated by classifiers trained on historical data to discern undesirable models.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

We thank the editors and all anonymous referees for spending their valuable time carefully reviewing this paper and making many suggestions for improvement. We also thank the University of Massachusetts Dartmouth for providing financial support to the first author to complete this work.

References

[1] Sarker, "AI-based modeling: techniques, applications and research issues towards automation, intelligent and smart systems," SN Computer Science, **3**(158), 1-20, 2022, doi: 10.1007/s42979-022-01043-x.

[2] NYC 311, "Automated employment decision tools," The Official Website of the City of New York, July 2023. Retrieved on September 1, 2023 from https://portal.311.nyc.gov/article/?kanumber=KA-03552.

[3] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," White Paper, Bitcoin Project, October 2008. Retrieved on January 15, 2023 from https://bitcoin.org/bitcoin.pdf.

[4] V. Buterin, "Ethereum: a next-generation smart contract and decentralized application platform," Ethereum Whitepaper, 2014. Retrieved on May 15, 2023 from https://ethereum.org/en/whitepaper/.

[5] O. Dib, K.-L. Brousmiche, A. Durand, E. Thea, E. B. Hamida, "Consortium blockchains: overview, applications and challenges," International Journal on Advances in Telecommunications, **11**(1&2), 51-64, 2018.

[6] H. Guo, X. Yu, "A survey on blockchain technology and its security," Blockchain: Research and Applications, **3**(2), February 2022, doi: 10.1016/j.bcra.2022.100067.

[7] Thamrin, H. Xu, "Cloud-based blockchains for secure and reliable big data storage service in healthcare systems," In Proceedings of the 15th IEEE International Conference on Service-Oriented System Engineering (IEEE SOSE 2021), 81-89, Oxford Brookes University, UK, August 2021, doi: 10.1109/SOSE52839.2021.00015.

[8] S. Kumar, A. K. Bharti, R. Amin, "Decentralized secure storage of medical records using blockchain and IPFS: a comparative analysis with future directions," Security and Privacy, **4**(5), 1-16, April 2021. doi: 10.1002/spy2.162.

[9] H. Wang, Y. Song, "Secure cloud-based EHR system using attribute-based cryptosystem and blockchain," Journal of Medical Systems, **42**(152), 1-9, July 2018, doi: 10.1007/s10916-018-0994-6.

[10] S. Liu, H. Tang, "A consortium medical blockchain data storage and sharing model based on IPFS," In Proceedings of the 4th International Conference on Computers in Management and Business (ICCMB 2021), 147-153, Singapore, January 2021, doi: 10.1145/3450588. 3450944.

[11] Thamrin, H. Xu, "Hierarchical cloud-based consortium blockchains for healthcare data storage," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), 644-651, Hainan, China, December 2021, doi: 10.1109/QRS-C55045.2021.00098.

[12] S. D. Ashwini, A. P. Patil, S. K. Shetty, "Moving towards blockchain-based solution for ensuring secure storage of medical images," In Proceedings of the 2021 IEEE 18th India Council International Conference (INDICON), 1-5, Guwahati, India, December 19-21, 2021, doi: 10.1109/INDICON52576. 2021.9691516.

[13] S. Ballal, Y. Chandre, R. Pise, B. Sonare, S. Patil, "Blockchain-based decentralized platform for electronic health records management," In Proceedings of the 2023 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS), 1-5, New Raipur, India, October 06-08, 2023, doi: 10.1109/ICBDS58040.2023.10346392.

[14] S. Ajjarapu, S. K. Pasupuleti, "Blockchain based certificateless privacy preserving public auditing for cloud storage systems," In Proceedings of the 2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC), 286-291, Solan, Himachal Pradesh, India, November 25-27, 2022, doi: 10.1109/PDGC56933.2022.10053241.

[15] Y. Jeong, D. Hwang, K. Kim, "Blockchain-based management of video surveillance systems," In Proceedings of the 2019 International Conference on Information Networking (ICOIN), 465-468, Kuala Lumpur, Malaysia, January 2019, doi: 10.1109/ICOIN.2019.8718126.

[16] Z. Su, H. Wang, H. Wang, X. Shi, "A financial data security sharing solution based on blockchain technology and proxy re-encryption technology," In Proceeedings of the IEEE 3rd International Conference of Safe Production and Informatization (IICSPI), 462-465, Chongqing City, China, 2020, doi: 10.1109/IICSPI51290.2020.9332363.

[17] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, A. Galstyan, "A Survey on bias and fairness in machine learning," ACM Computing Surveys, **54**(6), Article No. 115, 1-35, July 2022, doi:10.1145/3457607.

[18] V. N. Mandhala, D. Bhattacharyya, D. Midhunchakkaravarthy, "Need of mitigating bias in the datasets using machine learning algorithms," In Proceedings of the 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), 1-7, Chennai, India, 2022, doi: 10.1109/ACCAI53970.2022.9752643.

[19] M. Atay, H. Gipson, T. Gwyn, K. Roy, "Evaluation of gender bias in facial recognition with traditional machine learning algorithms," In Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 1-7, Orlando, FL, USA, December 2021, doi: 10.1109/SSCI50451.2021. 9660186.

[20] S. Rohani, R. Baeza-Yates, "Measuring bias," In Proceedings of the 2023 IEEE International Conference on Big Data (BigData), 1289-1298, Sorrento, Italy, 2023, doi: 10.1109/BigData59044.2023.10386679.

[21] H. Wang, S. Mukhopadhyay, Y. Xiao, S. Fang, "An interactive approach to bias mitigation in machine learning," In Proceedings of the 2021 IEEE 20th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), 199-205, Banff, AB, Canada, October 2021, doi: 10.1109/ICCICC53683.2021.9811333.

[22] M. C. Cohen, S. Miao, Y. Wang, "Dynamic pricing with fairness constraints," SSRN, September 2021. Retrieved on October 1, 2023 from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3930622.

[23] Y. Wang, H. Liu, "De-biasing methods in neural networks: a survey," In Proceedings of the 2023 International Conference on Machine Learning and Cybernetics (ICMLC), 458-463, Adelaide, Australia, July 2023, doi: 10.1109/ICMLC58545.2023.10327985.

[24] H. Maheshwari, U. Chandra, D. Yadav, A. Gupta, R. Kaur, "Machine learning and blockchain: a promising future," In Proceedings of the 4th International Conference on Intelligent Engineering and Management (ICIEM), 1-6, London, United Kingdom, May 09-11, 2023, doi: 10.1109/ICIEM59379.2023.10166343.

[25] X. Chen, J. Ji, C. Luo, W. Liao, P. Li, "When machine learning meets blockchain: a decentralized, privacy-preserving and secure design," In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), 1178-1187, Seattle, WA, USA, December 10-13, 2018, doi:10.1109/BigData.2018.8622598.

[26] T. Wang, "A unified analytical framework for trustable machine learning and automation running with blockchain," In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), 4974-4983, Seattle, WA, USA, 2018, doi: 10.1109/BigData.2018.8622262.

[27] S. Badruddoja, R. Dantu, Y. He, A. Salau, K. Upadhyay, "Scalable smart contracts for linear regression algorithm," International Conference on Blockchain Technology and Emerging Applications (BlockTEA 2022), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, **498**, 19-31, Springer, Cham, April 2023, doi: 10.1007/978-3-031-31420-9_2.

[28] B. Gu, A. Singh, Y. Zhou, J. Fang, F. Nawab, "ML on chain: the case and taxonomy of machine learning on blockchain," In Proceedings of the 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 1-18, Dubai, United Arab Emirates, May 1-5, 2023, doi: 10.1109/ICBC56567.2023.10174908.

[29] M. Folk, G. Heber, Q. Koziol, E. Pourmal, D. Robinson, "An overview of the HDF5 technology suite and its applications," In Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, 36-47, Uppsala Sweden, March 25, 2011, doi: 10.1145/1966895.1966900.

[30] ConsenSys, "What is Ganache?" Overview - Truffle Suite, ConsenSys Software Inc., 2022. Retrieved on March 15, 2024 from https://archive.trufflesuite.com/docs/ganache/.

[31] al-Qerem, A. Hammarsheh, A. M. Ali, Y. Alslman, M. Alauthman, "Using consensus algorithm for blockchain application of roaming services for mobile network," International Journal of Advances in Soft Computing & its Applications, **15**(1), 99-112, March 2023, doi: 10.15849/IJASCA.230320.07.

[32] O'Hagan, T. Leonard, "Bayes estimation subject to uncertainty about parameter constraints," Biometrika, **63**(1), 201-203, 1976, doi: 10.1093/biomet/63.1.201.

[33] S. K. Ashour, M. A. Abdel-hameed, "Approximate skew normal distribution," Journal of Advanced Research, **1**(4), 341-350, October 2010, doi: 10.1016/j.jare.2010.06.004

[34] M. Felipe, H. Xu, "A scalable storage scheme for on-chain big data using historical blockchains," In 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security Companion (QRS-C), 54-61, IEEE BSC 2022, Guangzhou, China, December 5-9, 2022, doi: 10.1109/QRS-C57518.2022.00017.

[35] Ethereum, "Blocks," Ethereum Documents, Feburary 27, 2024. Retrieved on March 15, 2024 from https://ethereum.org/developers/docs/blocks.

[36] S. Ahn, T. Kim, Y. Kwon, S. Cho, "Packet aggregation scheme to mitigate the network congestion in blockchain networks," In Proceedings of the 2020 International Conference on Electronics, Information, and Communication (ICEIC), 1-3, Barcelona, Spain, January 19-22, 2020, doi: 10.1109/ ICEIC49074.2020.9051158.

[37] J. R. Beckstrom, "Auditing machine learning algorithms: a white paper for public auditors," International Journal of Government Auditing, **48**(1), 40-41, Winter Edition, 2021.

[38] R. Ming, H. Xu, "Timely publication of transaction records in a private blockchain," In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 116-123, IEEE BSC 2020, Macau, China, December 11-14, 2020, doi: 10.1109/QRS-C51114.2020.00030.