2021 International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-7281-9127-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/COMSNETS51098.2021.9352838

# Design and Validation of *BlockEval*, A Blockchain Simulator

Deepak Kumar Gouda*, Shashwat Jolly† and Kalpesh Kapoor‡
*Department of Mathematics*
*Indian Institute of Technology Guwahati*
Guwahati, India
Email: *deepak.gouda@alumni.iitg.ac.in, †shashwatjolly@alumni.iitg.ac.in, ‡kalpesh@iitg.ac.in

***Abstract*—Blockchain technology is finding its application in a wide range of areas. Due to the highly decentralized and distributed nature of blockchain technology, it is essential to understand the behaviour of a system before its actual deployment. In this paper, we introduce the design and architecture of our blockchain simulator, BlockEval, which mimics the behaviour of concurrent operations that occur in a real-life blockchain system. We have established the correctness of our simulator by comparing it with an independent model that is built from the real Bitcoin-transactions data using deep-learning techniques. Unlike existing validation procedures which do not scale to large topologies, our method is scalable and efficient in terms of time and compute-infrastructure requirements. The observations made using our simulator are found to match with the results obtained from the model trained on real Bitcoin-transactions data.**

***Index Terms*—Blockchain, Simulation, Simulator validation, Deep Learning**

## I. Introduction

In recent years, blockchain technology is finding its application in a range of diverse domains such as online transactions [1], authenticating artworks [2] and governance [3]. The main characteristic that makes blockchain different from a traditional system is that it does not depend on any central authority. However, all the participants in a system still share a common logical state which is typically arrived at using a decentralized consensus algorithm.

While decentralization alleviates the dependency on a single entity to maintain the state of the entire system, it introduces several other challenges, such as detection and correction of bugs [4], execution delays, performance [5], and security [6]. A large blockchain system can consume a high amount of power. For example, the energy required to run the Bitcoin network for four seconds is sufficient to fulfill one household's total electricity requirement for one-year [7]. Thus, it has become more critical than earlier to understand the behaviour of a decentralized application before its deployment.

The development of decentralized apps for a large user base is a difficult task. One of the ways to understand a system is to do a test deployment and observe its behaviour. For a user or an organization, emulating a blockchain framework is expensive. Such an approach has been used earlier, for instance, in BlockBench [8] and the Bitcoin P2P network [9] nodes were deployed on actual machines or cloud. Obviously, this method is not only expensive but also cannot possibly give a good idea about real implementation due to the significant difference in the size of the test setup and actual deployment. Besides, a significant amount of time and effort is also needed to set up and maintain the experimental network. Apart from this, it is also challenging to foresee the implication of upgradation of a protocol in use. In a real network, solutions such as soft forks and hard forks are possible; these unforeseen circumstances might lead to permanent splits in the network [10]. Similar difficulties arise in the development and testing of new consensus algorithms and blockchain architectures. Performing tests on a massive scale on a real test-bed is therefore not a viable option.

An alternative to real test deployment scenario is to use a simulator tool to study the behaviour of a system. The availability of a simulation tool solves many of the problems that are mentioned above. The availability of simulator can also support test-driven development and extreme programming. However, it introduces a new challenge of ascertaining the correctness of a simulator itself. In this paper, we illustrate the design of a new simulator, BlockEval, and also overcome the challenge of validation. In particular, our main contributions are as follows.

(i) A modular blockchain simulation framework that can simulate a Proof-of-Work (PoW) blockchain network up to transaction-level operations, and
(ii) A deep-learning based technique to build a model from the real blockchain network to evaluate the correctness of any blockchain simulator framework. We validate our simulator using the same technique.

We also draw several relational observations from the simulated data.

The remainder of the paper is organised as follows. In Section II, we review the existing simulators and their approach to the problem. In Section III, we introduce our simulator, BlockEval, and present its architecture. In Section IV, we present a technique to check the correctness of blockchain simulators and use it to verify the observations that are drawn from BlockEval simulations. The insights gained from this data are presented in Section V. Finally, in Section VI, we conclude and mention some directions for future improvements.

## II. Background and Related Work

We first give a brief overview of the Blockchain terminology used in the paper along with a high-level description on how Blockchain works.

*Transactions* are the atomic units of any Blockchain. A transaction represents the transfer or creation of an asset or entity. Many such transactions are combined into a bigger unit, called as a *block*. A Blockchain consists of multiple blocks linked together in a list using cryptographic hash values. A *fork* occurs when a Blockchain diverges into two potential paths due to a change in protocol or when two miners propose a block at the same time. A fork results in multiple chains diverging from the forking point until one chain becomes longer than the other. The network is said to reach a *consensus* when it decides which branch to accept as the truth. A Blockchain network consists of different types of *nodes* such as *miners* and *full nodes*. Miners are responsible for generating blocks in the network, and full nodes simply store a copy of the entire blockchain.

Transactions are generated and multi-cast from multiple sources who wish to transfer assets to a different address. All miners receiving a transaction store it into their local memory pool (or *mempool*). As soon as they are idle, they extract a subset of transactions from their mempool, and generate a block consisting of those transactions, hence proposing a block. There are various block proposal algorithms like Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Activity (PoA) and Proof-of-Burn (PoB). Whenever a node receives a block, it adds the block to it's local Blockchain instance.

We now discuss three recently proposed simulators, namely Bitcoin-Simulator [9], VIBES [11] and BlockSim [12].

Bitcoin-Simulator [9] has been developed to study the impact of network-layer parameters on the security of Proof-of-Work blockchains. The simulator is built on ns-3, a discrete event network simulator and is implemented in C++. It keeps track of the events scheduled to execute at a particular simulation time and executes them in sequential order. The ns-3 simulator is not scalable to large topologies (e.g. 1000 nodes) [13]. Bitcoin-Simulator does not model the propagation of transactions. All parameters captured are at the block-level, and transaction-level statistics cannot be obtained from the simulations.

VIBES [11] has been developed to simulate large-scale peer-to-peer networks. It is built using Scala, and it leverages multi-threading to scale the network to tens of thousands of nodes. It uses Actor Models [14] of the Akka [15] framework to simulate nodes. The nodes use asynchronous message passing for communication. The network consists of a Master node or Coordinating Actor that controls the execution of events in the network. It is the only entity that can issue a request to a node to do work, that is, mine a block of transactions. The Master actor randomly asks the nodes to create a transaction, and the nodes push the request to their mailing list, which is a priority queue sorted according to the time of execution. All the nodes ask for permission from the Master Node to mine the block. The Master node collects all the requests from the nodes, sorts them in increasing order of execution timestamp and issues work requests to the nodes in order. A request for work is executed in a separate thread, and the process repeats recursively after mining of the blocks.

The main limitation of VIBES is that it does not implement any validation method to check the correctness of the simulated data. The focus is on the consistency of the simulator results, rather than similarity with real-world metrics. Further, the presence of a central controlling entity, the Master actor, deviates from the distributed nature of a blockchain network.

BlockSim is another simulation framework that aims to assist in the design, implementation, and evaluation of existing or new blockchains. BlockSim is a discrete event simulator, implemented in Python. It consists of a *Simulation World* component, which is responsible for handling the configuration parameters of the simulation. *Transaction* and *Node Factory* are responsible for creating transactions and nodes used during the simulation. The *Monitor* captures metrics during the simulation and stores them in the *Report* component.

BlockSim's validation method involves a comparison of the simulated results with statistics drawn from a two-node private Ethereum network deployed on AWS. Several crucial blockchain phenomena, such as forking, manifest only in larger topologies, and no validation tests have been conducted on such scale.

An important observation arising from these reviews is that none of the earlier blockchain simulators has attempted to validate their results for a large number of nodes.

## III. Architecture of BlockEval

We now present the architecture of our simulator and its components. We also give a brief workflow of BlockEval along with its input and output parameters. BlockEval can be extended to simulate any blockchain framework. In the following section, we discuss the Proof-of-Work implementation of BlockEval. A similar procedure may be followed to extend BlockEval to other protocols like Proof-of-Stake. Using the current implementation, we can modify the input parameters to extend BlockEval to any Proof-of-Work blockchain network. The source code of BlockEval has been made available in a public repository on GitHub [16]. Since in Proof-of-Work blockchains, blocks are proposed by Miner nodes, we use the terms *Miner* and *Proposer* interchangeably.

We also provide the pseudo-code for the Miner class, which is a central component of our simulator.

### A. Framework

It should be easy for end-users to customize a simulation framework for specific use cases. Python was chosen as the language base for our simulation framework because of its broad user base, exhaustive open-source library support and omnipresent applications. *SimPy* [17] is a popular process-based discrete-event simulation framework developed in standard Python.

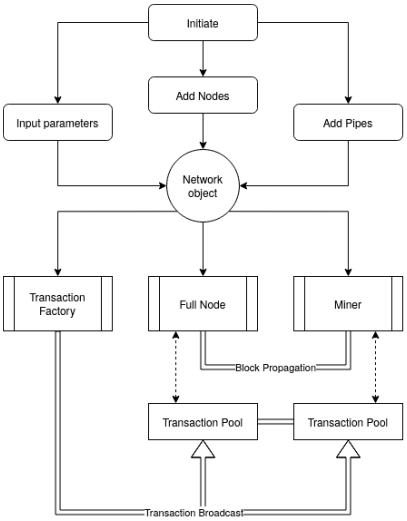


Fig. 1. BlockEval architecture

It provides various tools and abstractions that make it easy to develop a simulation tool. The simulated time scale can be set to either match the real time scale or to simulate as fast as possible. The user can also manually step through the simulated atomic events. The behaviour of the components of our simulation framework is modelled using *Processes*. The processes are part of a SimPy environment, and inter-process interaction happens through the environment using various events such as interrupts or SimPy resources, such as *Store*.

These processes are defined as Python generator functions that create events, yield them and suspend themselves for a *Timeout* period. After the Timeout period, the SimPy environment resumes the suspended processes. The SimPy environment executes the events and then fast-forwards or jumps onto the next event.

### B. Network Components

The entire network is modelled as a *Network* object which consists of Transaction Factory, Full Node and Miner objects, each of which simulates the real-world entities corresponding to their name.

*Transaction factory:* The Transaction factory is implemented as a generator function, which gets called when the network is initialized. It generates a transaction and broadcasts it to a random number of full nodes and miners. The degree of connectivity of these nodes is specified as an input parameter. After transaction generation, it suspends itself for a random period of time, which is drawn from a probabilistic distribution with the distribution parameters defined by the user.

*Pipes:* Pipes are instances of the SimPy *Store* resource, which models the production and consumption of objects. They are customised to act as the propagation channels among nodes. Each Pipe object consists of a source and a destination location which represent the connecting nodes. The propagation delay of each pipe is drawn from real-world data, based on the locations of source and destination nodes. Hence, whenever an object, that is, a block or transaction is added to a pipe, the destination node receives the object after a timeout value simulating the propagation delay. The implementation of propagation delay leads to observations of crucial blockchain phenomena such as forking and stale blocks.

*Full nodes:* Full nodes keep a copy of the complete blockchain and are responsible for verification and broadcast of the transactions as well as blocks. These have been implemented as objects of the *Full Node* class. The nodes spawn at random geographic locations and are connected to other full nodes and miners. The degree of connectivity is defined as an input parameter. The nodes encompass a transaction pool or mempool and a blockchain. When a node receives a block, its content is verified. If the block has not been received before, it is appended to the local blockchain instance. The block is then flooded across the network. If a block received by a node has an invalid hash, the node fetches the local blockchain instances from its neighbours and updates its instance. During the update, if forking is observed, then the longest chain prevails, and the rejected blocks are classified as Stale or Orphan blocks.

*Miners: Miners* are a subset of full nodes. They function as full nodes along with the added responsibility of block generation. Hence, the *Miner* class inherits the Full Node class and implements a block generator process as a Python generator. It generates a block and suspends itself for a random timeout value, which simulates the time required to brute-force and find a solution to the cryptographic hash problem in real Proof-of-Work blockchains.

Once a miner generates a block with some selected transactions from its transaction pool, it broadcasts the block to its neighbours. The transactions are selected on the basis of higher miner reward. Once a block is received by a miner, its timeout is interrupted and it starts its attempt to generate a new block. The complete implementation is given in Algorithm 1.

*Transaction pool:* Each node owns an instance of the *TransactionPool* class, which simulates mempools in real networks. They hold transactions received by the node in a custom priority queue in order of decreasing miner reward. Transaction pools perform the broadcast function of the nodes, that is, they broadcast the transactions on behalf of the node to its neighbours.

### C. Workflow

The simulation begins by instantiating an object of the Network class, with input parameters specified in a JSON configuration file (Table I). Node and Pipe instances are initialized and added to the Network object, and the transaction factory begins the generation and propagation of transactions. All the propagation happens through the Pipe instances. All transactions received by a node are added to its local transaction pool. These are maintained in the order of decreasing miner reward. Further, received transactions are broadcast to its neighbouring nodes.

The primary function of a Miner is its block generation function, which commences from the moment of instance

**Algorithm 1** Miner

```
t ← 0
GENERATEBLOCKS()

function GENERATEBLOCKS()
    sleep_time ← GETSLEEPTIME()
    while t < sleep_time do
        if interrupted then
            RECEIVEBLOCK()
            break
        end if
        t ← t + 1
    end while
    block ← GETBLOCK()
    transactions ← GETTRANSACTIONS()
    block.add(transactions)
    network.broadcast(block)
end function

function RECEIVEBLOCK(block)
    INTERRUPTBLOCKGENERATION()
    if ISVALID(block) then
        local_blockchain.append(block)
    else
        UPDATEBLOCKCHAIN()
    end if
end function

function UPDATEBLOCKCHAIN()
    all_blockchains ← RECEIVENEIGHBOURCHAINS()
    for b ← all_blockchains do
        if length(b) > length(local_blockchain) then
            local_blockchain ← b
        end if
    end for
end function
```

initialization. In this function, a Miner gathers transactions from its local pool and attempts to generate a block. To generate a block, a Miner has to suspend for a timeout value, which is drawn from a random distribution. The distribution parameters are estimated from real block generation data. This timeout can be interrupted if the Miner receives a block from its neighbour. If the received block is valid, the Miner adds it to the local blockchain instance, broadcasts it, and restarts its attempt to generate a block. Generated blocks are broadcast to neighbouring nodes which are further propagated across the network. An overview of the simulator workflow has been depicted in Figure 1.

The interplay of network connectivity, propagation delays and block generation time lead to forking in the network. For instance, consider two miners $M_1$ and $M_2$ which are initialized at time 0. Let us assume that miner $M_1$ generates a block in time $t_1$ and miner $M_2$ in time $t_2$ such that $t_1 < t_2$ and the propagation delay between $M_1$ and $M_2$ is $t_d$ such that $t_d > t_2 - t_1$. In this case, $M_1$ generates the block at $t_1$ (before $M_2$) and by the time the interrupt sent from $M_1$ is received by $M_2$, $M_2$ has already mined its block and propagated it to its neighbours. Hence, we have two different blockchains in the network, both of which are valid. Thus, we observe a fork in the network.

TABLE I
SIMULATOR PARAMETERS

| **Input parameter** | **Description** |
|---|---|
| Block generation interval | Time interval between block proposals |
| Transaction generation interval | Time interval between transactions |
| Miner count | Number of miners |
| Full node count | Number of full nodes |
| Block capacity | Number of transactions in a block |
| Transaction value | Amount of asset being transferred |
| Miner reward percentage | Fractional fee for mining transactions |
| Propagation delay | Probability distribution parameters |
| Possible node locations | Set of geographical locations |
| **Output parameter** | **Description** |
| Geographical node distribution | Percentage of nodes in each location |
| Block propagation | Mean and quantile values |
| Block | Includes block count, mean block size |
| Transaction waiting time | Mean and quantile values |
| Mining reward | Mean and quantile values |
| Number of stale blocks | Number of orphaned blocks |
| Number of forks | Number of forking instances |

## IV. VALIDATION

We now discuss the need for validation, the motivation for our novel validation method, and the method itself. We also present a comparative analysis of the validation and simulator results for BlockEval.

A validation criterion is necessary for any simulator to prove its reliability and correctness. While significant research has been done on validating traditional network simulations, no methods exist for verifying the correctness of large-scale blockchain simulations. Attempts have been made to validate simulated blockchain data by comparing it with emulated results. However, such methods cannot be implemented to validate large-scale networks, owing to physical and economic constraints. Critical blockchain phenomena such as forking are only observed in large-scale networks.

Hence, we put forth a new performance evaluation method by leveraging the power of machine learning, which can be used for validating large-scale simulations as well. We use real-life Bitcoin transaction data [18] to train the machine learning model. The model is then used to get system behaviour and is checked against the observed behaviour obtained from our simulator (see Figure 2).

### A. Method

Our validation method utilizes the data collected in [18], which has been drawn from the Bitcoin network directly using *blockchain.info* and the open-source client *bitcoind*. Any statistical results drawn from this data will be independent of any assumptions or bias developed when drawing data to validate a simulator. Some parameters in these sources are difficult to model in a simulator (for instance, the USD to BTC

TABLE II
MODEL PERFORMANCE - TRAINING DATA

| | | medianTxValue | medianFee | blockSize | blockCount |
|---|---|---|---|---|---|
| **XGBoost** | Relative Error | 5.01e-2 | 2.01e-2 | 1.83e-2 | 3.67e-2 |
| | RMSE | 20.09 | 6.97e-5 | 8.63e-3 | 5.77 |
| **ANN** | Relative Error | 41.32 | 34.24 | 26.58 | 6.65 |
| | RMSE | 205.40 | 2.55e-4 | 0.17 | 24.40 |

TABLE III
MODEL PERFORMANCE - TEST DATA

| | | medianTxValue | medianFee | blockSize | blockCount |
|---|---|---|---|---|---|
| **XGBoost** | Relative Error | 7.05e-2 | 5.73e-2 | 3.42e-2 | 7.65e-2 |
| | RMSE | 22.31 | 2.06e-3 | 1.67e-2 | 12.07 |
| **ANN** | Relative Error | 22.13 | 17.13 | 13.02 | 3.28 |
| | RMSE | 183.61 | 2.66e-4 | 0.18 | 24.18 |

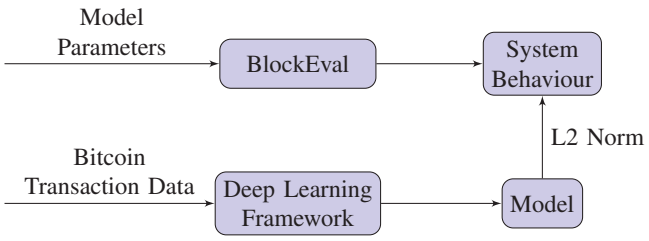


Fig. 2. BlockEval Validation Approach

conversion rate). Exploratory data analysis was performed to select relevant features which were used to train our validation model, treating independent and dependent features as input and output features, respectively. The refined data used for training consists of 10 features including Mempool Size ($S_m$), Transactions per Second ($\nu$), Median Confirmation Time ($\widetilde{t_c}$), Transactions per Block ($\eta$), Block Generation Frequency ($\omega$) and Transaction Fee ($\phi$). The existing simulators discussed in Section II differ in many of these input and output features, which limits the application of our trained model for their validation. This was the primary motivation to develop BlockEval, which has similar input and output features as the available data. This does not, however, limit the applicability of the validation procedure to other simulators.

A predictive model $\mathcal{M}$ is developed, which is trained on the input vector $X$ and the output vector $Y{:=}\langle\phi, \eta, \widetilde{t_c}\rangle$. Vector $\widehat{X}$ is constructed using parameters of the simulator network, the results of which are to be validated. $\widehat{X}$ is provided as input to $\mathcal{M}$ and its prediction is denoted as $\widehat{Y}$. $\widehat{X}$ is also provided as input to BlockEval, with resultant vector $\widetilde{Y}$. Similarity between $\widehat{Y}$ and $\widetilde{Y}$, which can be measured by a low value of $\left\|\widehat{Y} - \widetilde{Y}\right\|^2$ is a good indicator of the correctness of the simulator (Figure 3). Clearly, with this validation method, there is no bound on the number of nodes in the network.

### B. Deep Learning in Network Parameter estimation

Deep-learning techniques have been used in the recent past to study computer networks and its peripheries. In [19], the relationship between traffic volume and network flows has been studied through the use of Hidden Markov Models based on Kernel Bayes Rule. Further, predicting future network traffic using Recurrent Neural Networks (RNN) with Long Short Term Memory (LSTM) units has also been attempted. In [20], attempts have been made to detect intrusion attempts in incoming network traffic. The proposed system was built using Deep Belief Networks composed of Restricted Boltzmann Machines and trained on NSL-KDD dataset. In [21], Random Neural Networks have been used to allocate optimal radio parameters to users of a Radio Resource Management framework and ultimately improve the performance of an LTE uplink system.

While deep-learning methods have been utilized for various purposes earlier, to best of our knowledge no earlier work has used deep-learning techniques to analyse and validate the parameters of a network simulator.

### C. Model Architecture

Two methods were explored for developing the predictive model : 1) Artificial Neural Networks (ANNs) and 2) Decision tree based ensemble using XGBoost. We also perform a comparative analysis of the two and list our findings in Table II and Table III. The ANN and XGBoost model hyperparameters were found using a GridSearch [22] algorithm which minimizes the training loss of the models.

*ANN:* ANN is a very powerful non-linear function approximator given sufficient number of hidden neurons. It consists of several interconnected neurons arranged in layers. Each neuron takes a value as input and performs a non-linear activation on it. The result is then propagated to the next layer. In the training phase, the weights of connections between two adjacent layers of neurons are optimised to minimize the prediction errors. Backpropagation using gradient descent or more recently Adaptive Moment Estimation (Adam) [23] is commonly used to train a neural network.

The deep neural network developed for validation consists of fully connected neurons, with seven hidden layers. *Leaky ReLU* [24] activation function was used for each layer . As the response variable $\widehat{Y}$ is continuous, Root Mean Squared Error was used as the loss function. The training of the model is performed using Adam optimizer. To stabilize the

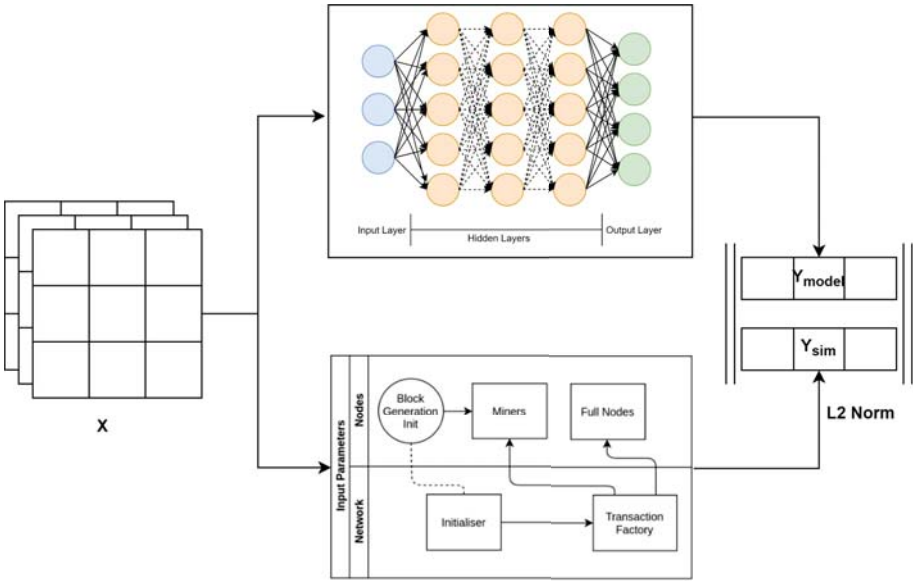


Fig. 3. Validation Flow

training procedure, *Batch Normalization* [25] technique was used. Dropout layers, as well as early stopping callbacks, were used to prevent over-fitting. The model consisting of 3,27,951 parameters was trained up to 70 epochs.

*XGBoost:* Decision Tree is a non-parametric supervised learning method used for classification and regression. The target value is predicted by learning simple decision rules inferred from the data features. A tree ensemble consists of a finite set of decision trees, which are used in association for better predictive performance. The final prediction for a given data sample is the sum of predictions from each tree. In tree ensemble models, for a given dataset of $m$ features and $n$ data samples, $K$ additive functions are used to predict the output. In the training phase, a greedy approach is used to optimize the decision functions at each level of a tree so as to minimize the loss. XGBoost [26] is a scalable gradient boosting framework to train tree-based ensemble models. Gradient tree boosting has been successfully used in classification [27], learning to rank [28], structured prediction [29] as well as other fields.

The XGBoost model developed for validation consists of 50 estimators along with L1 and L2 regularization terms of value 10. The objective function was set to minimize the squared error.

### D. Observations

The fitness of the model was evaluated using validation and test datasets. Tables II and III consist of the results of performance evaluation of the model and Table IV compares the BlockEval results to the predictions of the model.

It can be observed from Tables II and III that XGBoost consistently outperformed ANN for all parameters. This indicates that RMSE and relative error values can be further improved with better model architectures and machine learning methods.

TABLE IV
COMPARATIVE RESULTS

| | XGBoost | BlockEval |
|---|---|---|
| **medianTxValue (USD)** | 397.87 | 403.02 |
| **normalised blocksize** | 0.402 | 0.459 |
| **medianFee** | 2.89e-3 | 2.32e-3 |

The observation that a neural network can predict results comparable to those of a simulator raises a compelling question: can we replace a simulator with a neural network? In the current stage, we cannot sufficiently rely on a neural network as it has to be built for a highly specific objective function. Any deviation in input data distribution can produce undefined variation in results, which cannot be explained given the fact that the behaviour of hidden layers of deep neural networks is still under research and has not been fully understood. Further, building such a model requires a considerable amount of data that is not readily available, especially when we are trying to simulate an environment that highly prizes privacy and anonymity. Furthermore, simulators have the advantage of being able to produce data at atomic levels, while a prediction model produces a singular output value. Hence, building a generalized version of a deep learning model seems distant as of now and a simulator has many advantages over a deep learning model.

### E. Generalisation

The procedure described here can be applied to any blockchain framework. Changing the simulator parameters, the model architecture and the input data will enable validation of virtually any blockchain simulator. For instance, to validate Proof-of-Stake blockchain simulators, an appropriate model architecture needs to be developed, and trained on Proof-of-Stake blockchain data.

## V. Observations

Simulations were run for varying input parameters on an Ubuntu 18.04 machine running on an Intel i7-7700HQ processor with 16 GB RAM. The insights drawn from the simulated data are shown in Tables V and VI.

BlockEval generates a log of all events in the simulated environment from which several parameters can be drawn. The parameters of interest may vary depending on the use case. A system aiming to handle higher volume of transactions would consider the transaction confirmation time as a useful metric. Further, stale block generation rate can be used as an indicator of system efficiency. Similarly, the degree of network graph can be varied to examine the partition tolerance of a system architecture.

### A. Simulation time vs Number of nodes

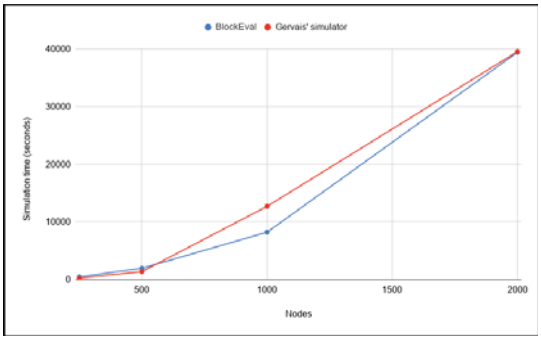


Fig. 4. Simulation time vs Number of nodes

Simulation time refers to the actual time taken to simulate a fixed number of virtual seconds. It can be observed from Figure 4 that the simulation time increases with an increase in the number of nodes. This can be explained by the fact that more nodes imply a greater number of blocks and transactions being propagated in the network, which results in an increase in the total propagation time. For a network of $n$ nodes, where each node has a degree $\frac{n}{p}$, the total number of connections is $\mathcal{O}(n * \frac{n}{p})$. This quadratic trend can also be observed in the simulated data. BlockEval can realistically be used to simulate a private blockchain network deployment of hundreds of nodes.

Further, we observe that our simulator performed similar to Bitcoin-Simulator. It is noteworthy that Bitcoin-Simulator is developed in C++ using the ns-3 framework, while BlockEval is based on Python. So, the similarity in simulation times is an indication of the efficiency of our simulator.

### B. Number of forks vs Degree of connectivity

Forking refers to the presence of two or more valid blockchain instances in the network. The fork is resolved when subsequent blocks are added, and one of the chains becomes longer than the alternatives. It can be observed from Figure 5 that the number of forks decreases with an increase in the degree of connectivity. In a fully-connected network, a block proposed by a node would reach all other nodes before they

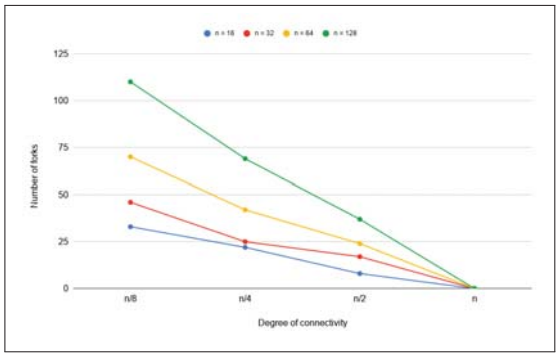


Fig. 5. Number of forks vs Degree of connectivity

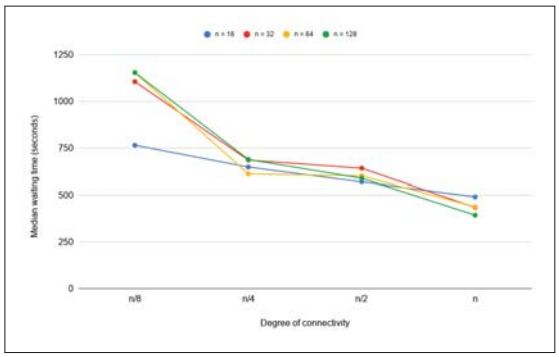


Fig. 6. Mean waiting time vs Degree of connectivity

can propose a block. This decreases the probability of a fork in the network. However, as the degree of connectivity decreases, nodes would receive blocks after a considerable delay, leading to fewer interruptions in block generation. Hence, many nodes would propose blocks at the same time, leading to more forks in the system.

In a private blockchain deployment that demands high consistency, limiting the number of forks in the system requires a higher degree of connectivity, which is expensive. Hence, a trade-off exists between ensuring consistency and limiting the degree of connectivity.

### C. Median waiting time vs Degree of connectivity

The waiting time of a transaction is the time it takes for the transaction to be included in a proposed block. It can be observed from Figure 6 that the median waiting time decreases with increasing degree of connectivity. This can be explained by the fact that decreasing the connectivity implies that transactions would not reach a larger number of nodes in a short time. So, fewer miners work on any particular transaction, leading to an increase in the waiting time.

A lower median waiting time implies a lower expected confirmation time for a transaction. Imposing an upper limit on the transaction confirmation time requires a higher degree of connectivity. Hence, a trade-off exists between achieving faster transaction confirmations and limiting the degree of connectivity.

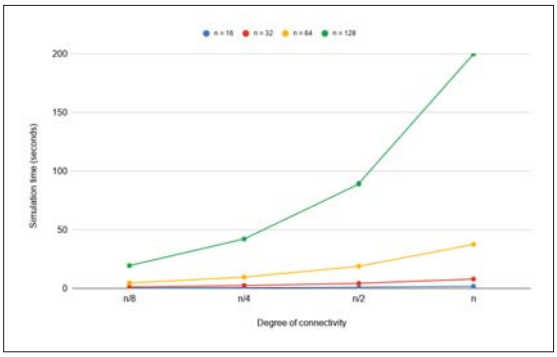


Fig. 7. Simulation time vs Degree of connectivity

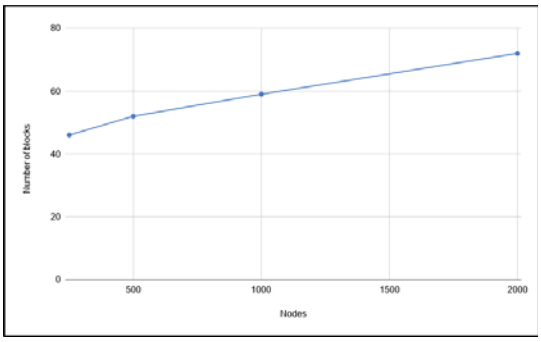


Fig. 8. Number of blocks vs Number of nodes

### D. Simulation time vs Degree of connectivity

It can be observed from Figure 7 that the simulation time increases with increasing degree of node connectivity. This can be explained by the higher number of transaction and block propagations in the system due to a greater number of neighbours.

TABLE V
VARYING DEGREE OF CONNECTIVITY (N = 128 NODES)

| Degree of connectivity | Number of forks | Median waiting time (in seconds) | Simulation time (in seconds) |
|---|---|---|---|
| **n/8** | 110 | 1153.60 | 19.58 |
| **n/4** | 69 | 688.77 | 42.36 |
| **n/2** | 37 | 588.85 | 89.20 |
| **n** | 0 | 393.54 | 199.79 |

TABLE VI
VARYING NUMBER OF NODES (FULLY CONNECTED NETWORK)

| Number of nodes | Simulation time (in seconds) | Number of blocks | Number of forks |
|---|---|---|---|
| **250** | 458.70 | 46 | 39 |
| **500** | 1950.76 | 52 | 45 |
| **1000** | 8218.03 | 59 | 52 |
| **2000** | 39412.92 | 72 | 65 |

### E. Number of blocks vs Number of nodes

Increasing the number of nodes leads to an increase in the number of blocks proposed (Figure 8). This trend can be attributed to the increase in miners, leading to a higher frequency of block generation. To counter this effect, the difficulty of the cryptographic problem is increased periodically in Bitcoin. This maintains the block proposal rate at 1 block per 10 minutes.

While scaling a private blockchain deployment to a higher number of nodes, maintaining a constant block proposal rate requires the difficulty to be adjusted accordingly. In such a scenario, the linear relationship between the number of nodes and the number of blocks generated, as demonstrated by the simulated data, can help in determining how the difficulty needs to be adjusted.

### F. Number of forks vs Number of nodes

With an increase in the number of nodes, we also observe an increase in the number of forking instances in the system. A higher number of nodes implies a greater probability of simultaneous block proposals, hence increasing the number of forks.

## VI. CONCLUSION

We have introduced an extensive, modular simulator for blockchains and an evaluation technique for blockchain simulators based on deep learning. Our simulator can be used to gather insights and metrics and test the performance of private blockchain networks before actual deployment. Improving scalability of blockchains has been an open research problem and side chains, in which transactions are settled between parties in a quick manner outside the main chain have been proposed to accelerate the performance of blockchains. Further, the tradeoff between security and efficiency has been a major research topic in permissioned and permissionless blockchains. BlockEval can be used to simulate, analyze and study such open problems.

The correctness of the simulator has been shown by our deep learning model trained on real-world data. Further, specific relational observations have been drawn from the simulator results. The novel validation method proposed can be used to evaluate the correctness of any blockchain simulator. To the best of our knowledge, BlockEval is the first simulator to have been tested against actual Bitcoin statistics. In comparison to existing validation procedures such as emulating a blockchain network, which does not scale to large topologies, our method is scalable and efficient in terms of time and capital.

Simulation results have been drawn up to 2000 nodes, which have been validated against actual Bitcoin data. However, there is a scope of improvement to both the simulator and the validation architecture. For instance, the addition of propagation latency data with sufficient variance will improve the accuracy of simulation results. Further, our deep learning model is based on Bitcoin data, a similar procedure can be followed for validating the simulation framework for other blockchain architectures.

## REFERENCES


[1] R. Beck, J. Stenum Czepluch, N. Lollike, and S. Malone, "Blockchain – the gateway to trust-free cryptographic transactions," in *European Conference on Information Systems (ECIS)*, no. 153, 2016.

[2] M. Zeilinger, "Digital art as 'monetised graphics': Enforcing intellectual property on the blockchain," *Philosophy & Technology*, vol. 31, pp. 15–41, 2018.

[3] W. Reijers, F. O'Brolcháin, and P. Haynes, "Governance in blockchain technologies & social contract theories," *Ledger*, pp. 134–151, Dec. 2016.

[4] Z. Wan, D. Lo, X. Xia, and L. Cai, "Bug characteristics in blockchain systems: A large-scale empirical study," in *IEEE/ACM* $14^{th}$ *International Conference on Mining Software Repositories (MSR)*, 2017, pp. 413–424.

[5] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *IEEE* $26^{th}$ *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 264–276.

[6] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, vol. 107, pp. 841–853, 2020.

[7] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 05 2018, pp. 1545–1550.

[8] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *SIGMOD Conference*, 2017.

[9] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. ACM, 2016, pp. 3–16.

[10] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. X. Song, and R. Wattenhofer, "On scaling decentralized blockchains - (a position paper)," in *Financial Cryptography Workshops*, 2016.

[11] L. Stoykov, K. Zhang, and H.-A. Jacobsen, "Vibes: Fast blockchain simulations for large-scale peer-to-peer networks: Demo," in *ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, ser. Middleware '17. New York, NY, USA: ACM, 2017, pp. 19–20.

[12] C. Faria and M. Correia, "Blocksim: Blockchain simulator," in *IEEE International Conference on Blockchain*, 2019, pp. 439–446.

[13] N. Documentation, "Routing Overview ns3 documentation," 2019. [Online]. Available: https://www.nsnam.org/docs/release/3.14/models/html/routing-overview.html#global-centralized-routing

[14] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in $3^{rd}$ *International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[15] "Akka documentation: Introduction to akka," 2019. [Online]. Available: https://doc.akka.io/docs/akka/current/guide/introduction.html?language=scala

[16] D. K. Gouda and S. Jolly, "Blockeval source code repository," 2020. [Online]. Available: https://github.com/deepakgouda/BlockEval

[17] SimPy, "Simpy documentation," 2020. [Online]. Available: https://simpy.readthedocs.io/en/latest/

[18] D. Kondor, "Bitcoin network dataset." [Online]. Available: https://senseable2015-6.mit.edu/bitcoin/

[19] Z. Chen, J. Wen, and Y. Geng, "Predicting future traffic using hidden markov models," in *IEEE* $24^{th}$ *International Conference on Network Protocols (ICNP)*, 2016, pp. 1–6.

[20] Z. Alom, V. R. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief network and extreme learning machine," *International Journal of Monitoring and Surveillance Technologies Research*, vol. 3, no. 2, pp. 35–56, Apr. 2015.

[21] A. Adeel, H. Larijani, and A. Ahmadinia, "Random neural network based novel decision making framework for optimized and autonomous power control in lte uplink system," *Physical Communication*, vol. 19, pp. 106–117, 2016.

[22] "Sklearn : Gridsearch," 2020. [Online]. Available: http://bit.do/scikit-learn

[23] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[24] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[26] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[27] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 04 2000. [Online]. Available: https://doi.org/10.1214/aos/1016218223

[28] C. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, 01 2010.

[29] T. Chen, S. Singh, B. Taskar, and C. Guestrin, "Efficient second-order gradient boosting for conditional random fields," in *AISTATS*, 2015.