Received 17 March 2023, accepted 11 April 2023, date of publication 19 April 2023, date of current version 24 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3268283

METHODS

A New Efficient Method for Refining the Reinforcement Learning Algorithm to Train Deep Q Agents for Reaching a Consensus in P2P Networks

ARAFAT ABU MALLOUH[1], ZAKARIYA QAWAQNEH[2], OMAR ABUZAGHLEH[3], AND AHMAD AL-RABABA'A[4]

[1]Computer Science Department, Manhattan College, Riverdale, NY 10471, USA
[2]Department of Computing Sciences, State University of New York Brockport, Brockport, NY 14420, USA
[3]Department of Computer Information Science, Higher Colleges of Technology, Dubai, United Arab Emirates
[4]Computer Science Department, The World Islamic Science and Education University, Amman 11947, Jordan

Corresponding author: Arafat Abu Mallouh (aabumallouh01@manhattan.edu)

ABSTRACT The usage of distributed Peer-to-Peer (P2P) networks has been growing steadily for a reasonable period. Various applications rely on the infrastructure of P2P networks, where nodes communicate to accomplish a task without the need for a central authority. One of the significant challenges in P2P networks is the ability of the network nodes to reach a consensus on a shared item; the challenge increases as time passes. Thus, this work proposes a new effective method for tweaking the Deep Reinforcement Learning (DRL) algorithm to train Deep Q Network (DQN) learning agents to reach a consensus among the P2P nodes. We propose various hierarchies of deep agents to address this crucial challenge in P2P networks. DRL is utilized to build and train agents; more precisely, DQN learning agents are constructed and trained. Two scenarios are proposed and evaluated. In the first scenario, one DQN agent is trained to find the consensus between the network nodes. In the second scenario, three hierarchies with different numbers of layers of agents are proposed and evaluated. In both scenarios, the P2P network used is a blockchain network. The best result was obtained using the third hierarchy of the second scenario; the overall accuracy of the model is 87.8%.

INDEX TERMS Blockchain, consensus, deep reinforcement learning, DQN, P2P.

I. INTRODUCTION

Recently, the need for an efficient mechanism for reaching a consensus in P2P networks is mightily back to the lights because of the rapid growth in blockchain technologies, which has attracted great interest in research and industry. In a P2P network environment, various nodes are randomly connected in an equivalent manner. For a large-scale P2P network, which is the current norm, communication between nodes is performed asynchronously. The most well-known technique used for asynchronous communication among large-scale P2P nodes is gossip Stochastic Gradient Descent (SGD) [1], [2]. This is a simple technique in which each node averages only with its neighbors to reach consensus. It is commonly used with well-known methods, such as consensus-based distributed SGD [3] and decentralized parallel SGD [4]. However, different types of consensus protocols are used in different blockchain networks. For example, consensus in permissioned blockchain networks might be implemented with conventional Byzantine Fault Tolerant (BFT) protocols [5], [6] to enforce strict synchronization between consensus nodes. In permissionless networks where the synchronization among the consensus nodes could be poor, incentive-based consensus protocols are implemented, such as the Proof of Work (PoW) protocol [7], [8]. PoW is a common consensus algorithm used by Bitcoin, the most

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Guidi.

popular cryptocurrency network. The PoW is a decentralized consensus algorithm that requires the participants of the network to spend effort solving an arbitrary mathematical puzzle to prevent anybody from playing the system. Proof of work is widely used in cryptocurrency mining. Bitcoin was the first cryptocurrency project to use PoW for validating transactions and mining new blocks. In the case of Bitcoin, the miners mine a new block roughly every 10 min, the new block is added to the blockchain, and then propagates to all other nodes to keep the blockchain synchronized among all nodes at all times. The blockchain is a constantly growing ledger that keeps a permanent record of all transactions that have taken place in a secure, chronological, and immutable way. Once a block is added to a blockchain, it cannot be altered. However, 51% of attacks on a blockchain can occur only if a group of miners can control over 50% of a network's mining hash rate, which allows them to exploit the mining of new blocks and take over the blockchain and its reward.

In this work, we propose a framework for DQN agents to reach a consensus between distributed P2P blockchain nodes. A DQN approximates a state-value function in a Q-Learning framework using a neural network [9], [10]. DRL combines artificial neural networks with a Reinforcement Learning (RL) framework that helps software agents learn how to achieve their goals [11]. This discipline of study has been capable of resolving an extensive variety of complicated decision-making tasks. Thus, DRL opens many new applications in a wide range of domains, such as robotics, finance, healthcare, and smart grids. RL is a ML training approach primarily based on rewarding desired behaviors and punishing undesired behaviors. In general, a RL agent is capable of understanding and interpreting its environment, taking action, and learning through trial and error [12], [13].

Recently, DQN agents have shown remarkable success in training agents in a variety of applications, such as Atari games, which have almost achieved human-level control. Motivated by this success, we utilized DQN RL agents to reach consensus in P2P networks. A DQN is a combination of Q-learning (reinforcement technique), DNN, and experience replay. Q-learning is performed by finding the variables of the Q-function. The Q-function of policy π, Q^{π}(s, a), measures the expected return sum of rewards gained from state s by taking action a followed by policy π. In reality, for most applications, it is unfeasible to represent the Q-function as a table with values for each s and a. Instead, the values are estimated by a training model, such as a Deep Neural Network (DNN). In this paper, we make the following contributions:

- We Propose an effective method for modifying the reinforcement learning algorithm using deep Q-Learning to achieve consensus within a blockchain P2P network.
- We successfully deploy and train DQN agents within the blockchain P2P network.
- We propose and evaluate various DQN agent architectures to collect informative data from distributed miners, as well as examination of different agent hierarchies to determine the most effective approach for achieving consensus.
- We propose a new communication mechanism between miners and DQN agents (update message), whereby miners only communicate with their sector agent.
- We reduce the network bandwidth utilization required for achieving consensus, and we also reduce consensus delay times by minimizing the number and size of sent messages.

The remainder of this paper is organized as follows: The literature review is introduced in Section II, and the methodology of the proposed work is introduced in Section III. In Section IV, the experiments and discussion are presented, and finally, the conclusion of this work is presented in Section V.

II. RELATED WORK

Lucarelli et al. [14] proposed a deep Q-learning portfolio management framework for maximizing return and minimizing risk by allowing agents to reallocate funds into different financial assets. The framework in their work is composed of two types of agents: a set of local agents that learns asset behaviors and a global agent that describes the global reward function. Additionally, the local agent was based on three different deep RL approaches: deep Q-learning, double deep Q-learning, and dueling double deep Q-learning. To evaluate their work, the authors tested their framework on a crypto-portfolio composed of four cryptocurrencies: Bitcoin (BTC), Litecoin (LTC), Ethereum (ETH), and Ripple (XRP). Their results showed that, in 80% of the cases, the proposed frameworks had higher daily returns with respect to cryptocurrencies. Based on their results, the authors showed that the DR portfolio management framework is a promising approach for dynamic portfolio optimization.

Zeng et al. [15] studied the protection of the relay system in a distribution network, where distributed generation access increases the uncertainty and affects distributed network security. The authors proposed solving the problem by following three main phases: the first phase, which analyzes the relay protection characteristics of the distribution network under distributed generation access; the second phase, which transforms the distribution network relay protection problem into a multi-agent RL problem; and the third phase, which proposes a distribution network distributed protection method based on a multi-agent deep deterministic policy gradient. The authors explained the main advantage of the proposed work because there is no need to build a distribution network security model in advance, which will overcome the impact of uncertainty caused by distributed generation access on distributed network security.

Kim et al. [16] proposed an on-chip training method for DQN that is applicable to hardware-based Spiking Neural Networks (SNNs). The authors showed how their training method could minimize memory usage and reduce power consumption and area occupation levels. For simple problems, the proposed method can significantly reduce memory dependency while achieving a high performance without using replay memory. In addition, the authors studied the effect of nonlinearity characteristics and variations in non-ideal synaptic devices and found that their work was strongly immune to such variations because of the proposed on-chip training scheme. The performance of the proposed training method was evaluated in two games: a fruit-catching game and a Rush Hour game. The results showed that the network was trained well without significant performance differences compared with the software-based training method. In the fruit-catching game, high performance with a catching rate of approximately 98% was achieved, although replay memory was not used.

Chen et al. [17] presented a DRL algorithm without external noise called self-guided deep deterministic policy gradient with multi-actors, which is a combination of a deep deterministic policy gradient and Generative Adversarial Networks (GANs). The proposed algorithm employs the generator of GANs to guide the learning of the agent, and makes the discriminator constitute a subjective reward. In addition, to stabilize the learning process, the proposed algorithm applies a multi-actor mechanism based on the temporal phase of an episode. The results of this study are encouraging.

Jang et al. [18] proposed a DQN-based multi-criteria decision-making framework for virtual agents in real time for automatic goal selection based on motivations in virtual simulation environments, as well as for planning the required behaviors to achieve those goals. The motivations in the proposed work are classified according to Maslow's five-level hierarchy of needs, and the virtual agents train a double DQN using big social data, select optimal goals depending on motivations, and perform behaviors relying on a predefined Hierarchical Task Network (HTNs). The authors reported the performance of the proposed method as efficient by increasing the accuracy from 63.24 to 80.15%. In addition, for behavioral performance using predefined HTNs, the number of methods was increased from 35 in the Q-network to 1511 in the proposed framework.

Hu et al. [19] addressed the bipartite consensus problem for a class of double-integrator multiagent systems with antagonistic interactions. The authors reported two cases: if the communication time delays are not considered, the bipartite consensus of the studied multi-agent systems with directed signed graphs is possible based on their proposed distributed controller. In the second case, if non-uniform communication time delays are considered, the bipartite consensus of multi-agent systems with undirected signed graphs is possible if the time delays are less than a limit.

Wang et al. [20] investigated the problem of free-fault-tolerant consensus tracking for multi-agent systems subjected to actuator faults. The authors proposed a new variable based model free algorithm to detect the faults, the algorithms can detect the faults which occur on different agents through any agent. Once a fault is detected, a radial basis function neural network is applied to estimate actuator faults. Then, fault estimations are utilized to reconstruct the faulty agent controller. Moreover, the authors introduced a distributed model-free adaptive fault-tolerant consensus control method to ensure that all agents can track the expected trajectory.

Zhao et al. [21] analyzed the typical application architecture of blockchain technology and the security risks of blockchain application architecture. The authors proposed combining the DOC mechanism with Internet of Things (IoT) application scenarios to introduce a blockchain-based confidential IoT service model that supports first-order homomorphic multiplication and a blockchain-based confidential IoT service model that supports high-order homomorphic multiplication. The authors found that Beekeeper 2.0 significantly improved the Beekeeper 1.0 in server capabilities, verification efficiency, verification key length, and device work diversity. The authors reported that the chain code call delay increases with an increase in the transaction-sending rate, and the average blockchain access delay increases with an increase in the transaction-sending rate. In addition, the blockchain access delay is affected by lower throughput, and the transaction success rate is stable at 98%.

Oikonomidis et al. [22] proposed deep learning-based models to evaluate the performance of the underlying algorithms with respect to different performance criteria. The authors evaluated the following algorithms in their study: the XGBoost Machine Learning (ML) algorithm, Convolutional Neural Network (CNN), DNN, CNN-XGBoost, CNN-Recurrent Neural Networks (RNN), and CNN-Long Short-Term Memory (LSTM). The authors performed experiments on a public soybean dataset consisting of 395 features, including weather and soil parameters, and 25,345 samples. The case study results showed that the hybrid CNN-DNN model outperformed the other models, with an RMSE of 0.266, an MSE of 0.071, and an MAE of 0.199. The predictions of the model fit, with an R2 of 0.87. The second-best result was achieved by the XGBoost model, which required less time to execute than the other DL-based models.

Videgaín et al. [23] developed a series of intelligent agents with different reasoning and decision capacities based on different artificial intelligence techniques applied to game theory, such as Minimax or RL. Their capabilities have been tested not only by playing games with each other, but also against human players, obtaining remarkable results. The experimental results ratify conclusions already known at a theoretical level but also provide a new contribution that could be the basis for future research.

Mohamed-Amine et al. [24] proposed an Epidemiological Model (EM) that is inherently suitable for analyzing different control policies. The Authors validated the potential of the developed EM in modeling the evolution of COVID-19 infections with a mean Pearson correlation of 0.609 CI 0.525–0.690 and P-value < 0.001. To automate the process of analyzing control policies and finding the optimal one, the authors adapted the developed EM to a RL setting and conducted several experiments. The case study results of this work showed that the problem of optimal epidemic control can be significantly difficult for governments and policymakers, especially if faced with several constraints simultaneously; hence, there is a need for such ML based decision support tools. In addition, it demonstrates the potential of deep RL in addressing real-world problems.

Nicolas et al. [25] proposed a Multistage Secure Pool (MSP) framework to address the vulnerabilities of blockchain. The proposed framework was designed to handle both discrete and general issues affecting the overall security of the blockchain. Additionally, the authors present the ML capabilities of the system to enable a progressive aspect of the design. Providing their application with the ability to analyze data to recognize and classify distinct actions will enable greater comprehension. An application that learns, updates, and configures to meet specified defensive standards presents key design features that enable greater understanding and future analysis of the overall blockchain network. The authors' findings using this application showed that there was a decrease in the number of attacks propagating through the system based on the system's robustness and capabilities.

Qiu et al. [26] studied a DRL empowered adaptive approach for future blockchain networks. Instead of using one consensus protocol as the best fit, and to improve the scalability and meet the requirements of different users, blockchain networks launch different consensus protocols based on users' Quality of Service (QoS) requirements. The authors quantified four consensus protocols. Additionally, the authors dynamically allocated the computation and bandwidth resources to the blockchain networks. They then formulated these three items, that is, the selection of consensus protocols, computation resources, and network bandwidth resources, as a joint optimization problem. A DRL approach was used to solve this problem. The simulation results obtained by the authors demonstrate the effectiveness of the proposed scheme.

Rabieinejad et al. [27] proposed a conceptual model to improve blockchain throughput in IoT-based devices with limited power through DRL. This model benefits from a recommender agent based on DRL in the mobile edge computing layer to improve throughput and select the right mining method.

Yang et al. [28] presented a clustering-based sharded-blockchain strategy for collaborative computing in the IoT, where the sharing of the blockchain system was implemented in two steps: K-means clustering-based user grouping and the assignment of consensus nodes. The authors described data transactions among IoT devices using a data transaction flow graph based on a dynamic stochastic block model. Subsequently, formed as a Markov decision process, the optimization of the cluster number and the adjustment of consensus parameters are jointly trained by DRL. The simulation results show that the proposed scheme improves the scalability of the sharded-blockchain in IoT applications.

Gong et al. [29] proposed a blockchain network-slicing broker that can support the network slice resource agency process in an end-to-end manner. The network slicing broker provides a resource brokering solution between network slice tenants, which are regarded as blockchain nodes, and the Complex Network theory is used to obtain the topological information and define the value of blockchain nodes according to the importance of the nodes. The framework consists of three parts: the network slice tenants request resources, the network slicing broker receives the request and allocates the resource using the DRL algorithm, and the Infrastructure Provider allocates the entity resource according to the Smart Contract. The authors described the computing resource allocation process as an example and the optimization problem modeled by the DRL framework. The results reported by the authors indicate that the proposed solution is effective

Ma et al. [30] introduced an intelligent incentive-based DCR management framework to determine a specific reward for each prosumer providing power support to the grid and proposed a framework that employs a Deep Deterministic Policy Gradient algorithm with a designated reward function to find the optimal incentives for each prosumer on an hourly basis. The proposed algorithms were implemented on a Hyperledger Fabric blockchain network. Laboratory prototypes were built using an actual fabric network. Case studies have shown that the developed framework can handle a large number of prosumers with a promising performance. The proposed work provides a proof of concept for the effectiveness of the integration of AI and blockchain in distributed resource management in modern smart grids.

In [31], the framework introduces different hierarchies of DR agents distributed across multiple layers. The layers were divided into sectors; each sector contained distributed nodes, and each sector had one agent that was trained using DRL. Three models were developed and examined in this study. Model 1 achieved 71.6%, Model 2 achieved 73.3%, and Model 3 achieved 82.2%.

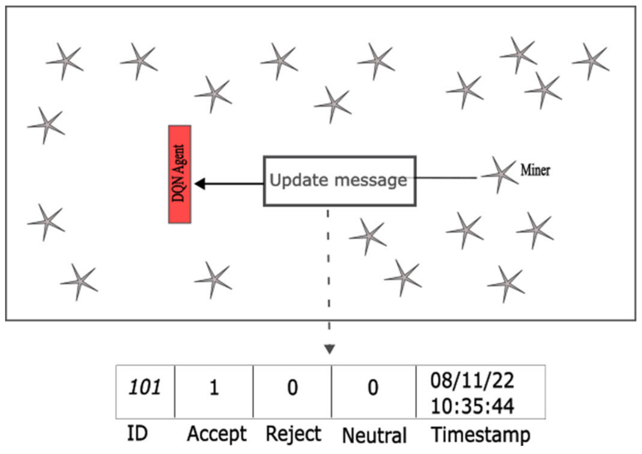
III. METHODOLOGY

In this study, we propose the utilization of DRL agents to reach a consensus among distributed P2P nodes. More specifically, the DQN agents will be used. In part A, a description of the environment is introduced. Part B introduces and explains the DQN learning process.

A. THE ENVIRONMENT

The main objective of this work is to train agents that make decisions in a P2P network environment to reach a consensus; therefore, we assume that the local node decision is made and sent to the agents. The local decisions of the network nodes are randomly simulated, and the decision can be: Accept, Reject, or Neutral.

Once a node receives an update about the current version of the chain, it must make its local decision and send it to a specific agent in the network. In this work, we introduce a new type of message called an "update message" that nodes and agents use to communicate their decisions. This message allows for more efficient and effective communication between the nodes and agents in the network, enabling them to work together towards achieving consensus



101	1	0	0	08/11/22 10:35:44
ID	Accept	Reject	Neutral	Timestamp

FIGURE 1. The update message that nodes and agents use to communicate their decisions.

in a more streamlined and effective manner. The message consists of three main fields: the ID of the current version of the blockchain, the node local decision, and a timestamp. The local decision field also consists of three subfields that represent each node's decisions regarding the new blockchain as shown in Fig 1. One of these subfields was set to one, while the rest were set to zero. The first subfield is set to one if the node accepts the new version of the blockchain, the second subfield is set to one if the node rejects the new version, and the last subfield is set to one if the node is a neutral decision. The network uses the timestamp field to set a time interval to receive the local decisions of the nodes. The message is considered legitimate if it is delivered within a specified time interval; otherwise, the message is discarded.

B. DQN LEARNING

In RL, the agent's goal is to find the optimal series of actions that should be taken each time the agent faces a new observation in its environment. When moving from the start state to the target state, the agent should select a series of actions that hit the highest possible optimal action value for the Q(s_i, a_i) function, where a_i is the action taken by the agent at state s_i. There are many methods for calculating the Q value. In our proposed approach, we utilize Bellman equations in conjunction a DNN to estimate the Q value. By leveraging these techniques, we are able to predict the optimal action that an agent should take given a particular state.

Based on the Bellman equations, there is a Q value for action a_i at state s_i, defined as Q(s_i, a_i). Value function Q(s_i, a_i) is a nonlinear function, and a neural network can be used to approximate such a complex function. The neural network is trained using supervised learning, where the targets for this network are calculated using Bellman (1).

$$Q_\pi\ (s, a) = \sum\nolimits_{s'} P\left(r\ (s, a) + \gamma . \sum\nolimits_{a}^{\prime} \pi(a'|s').Q_\pi(s', a')\right) \tag{1}$$

where Q_π (s, a) is the value of taking action ***a*** in state ***s*** by following policy π, P is the probability of selecting action ***a*** in state ***s*** and ending at state ***s'***, s' is the next state and a' is the action to be taken in s', ***r*** is the reward for selecting action ***a*** in state ***s,*** and gamma (γ) is the discount factor. It is noteworthy that the Q table is a matrix listing all possible states down the side to define the rows and all possible actions for every state defining the columns. The contents of the table are the values of a_i while the agent is in s_i. As a matter of fact, in our proposed approach, the use of Q tables to represent the Q values for agents in a blockchain P2P network would result in a very large table, and creating such tables is not feasible because of the required space and computation power. Therefore, in reality, the best option for calculating the entries for such a large table is to use approximation techniques. Hence, in this study, we used DRL to approximate the values of the Q function in a timely manner. In RL, the agent is forced to find the best actions that maximize its reward within an episode through experience. Strictly speaking, the episode is all things that happen between the initial and final states within the environment. In this work, in the blockchain P2P network, the episode is a sequence of states that starts when a new block is added to the blockchain, the local decisions of the nodes that are sent to the DQN agent(s), and the final decision of the DQN agent(s) as shown in Fig. 2.

As discussed earlier, the DQN agent can make three different decisions to label the update message: Accept, Reject, and Neutral. Nodes must send their decisions to the nearest agent. After receiving these messages, the agents calculate the reinforcement signal. The reinforcement signal is calculated as the difference between the predicted agent's decision and the actual target of actions with higher Q values. Initially, the weights of the deep network used to train the DQN agents were randomly initialized, which resulted in the poor performance of the DQN agent in the early stages. However, while agents advance in the training, they will learn how to enhance their performance and take the right decisions as a result of exploring different environmental states with appropriate actions. The DQN agent can calculate the Q-value for every possible action for every state in the environment in a single forward pass as depicted in Fig 3. This resulted in a vector containing all possible values for each action, which can be used as a base by the agents to select those actions with maximum Q values.

1) THE DEEP NETWORK ARCHITECTURE

It has been noted that the learning process becomes unstable if a deep network is combined with RL, particularly when a neural network is used to represent the state-action values. The network weights are liable for oscillation and divergence because of the high correlation between actions and states. Training such networks successfully requires a huge amount of data, and even in the presence of a huge dataset, we cannot guarantee that the network converges to find the optimal values. Many studies have suggested different types of efficient techniques to reduce the effect of instability while training

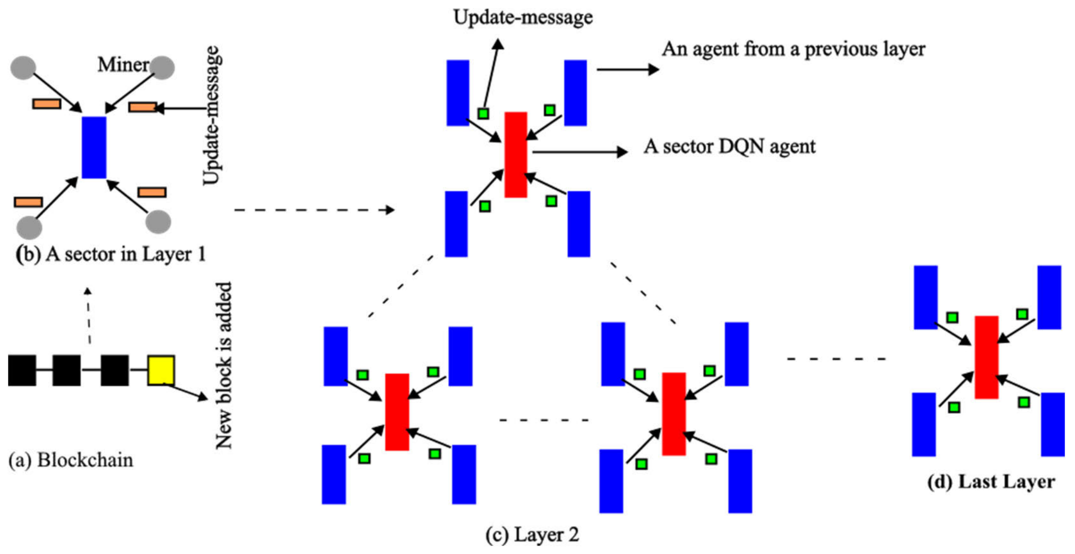


FIGURE 2. The episode is a sequence of states that starts when a new block is added, the local decisions of the nodes that are sent to the DQN agent(s), and the final decision of the DQN agent(s).

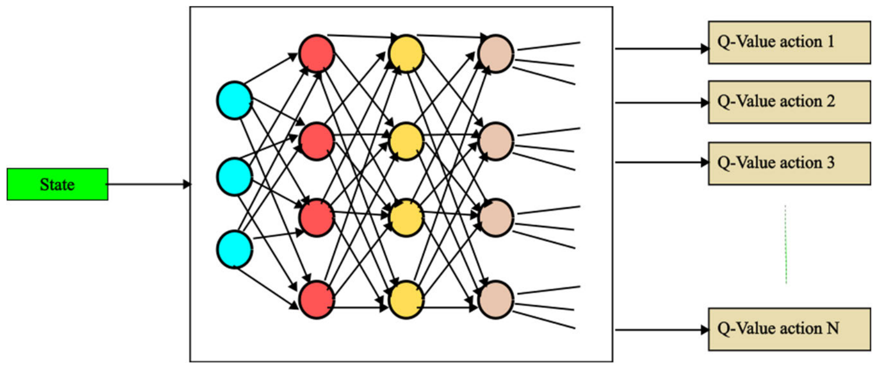


FIGURE 3. Agent Q values calculations for each state.

such networks. In this study, two techniques were used: the experience buffer and the target network.

2) THE EXPERIENCE BUFFER

Efficient model training requires the dataset observations to be independent and normally distributed. However, when an agent interacts with the environment, the generated sequence of experience tuples (messages of local decisions) can be highly correlated. Training agents using such highly correlated data mostly results in high fluctuations in the results. Thus, to reduce the effect of such oscillations and diverging state-action values, the experience buffer was used to sample the training dataset. The experience buffer consists of experience tuples collected from previous experiences. It is also worth noting that new experiences are regularly added to the buffer while the agents interact with the environment.

Each tuple (S, A, R, S') consists of observation (S), action (A), reward (R), and the next state (S'). Technically, the buffer is implemented as a queue with a fixed number of tuples, and whenever new experience is added while the buffer is full, the oldest experience in the buffer is pushed out. The experience buffer is a queue in which small batches are sampled from the buffer observations for training the agents. This helps reduce the high correlation between the experience tuples and increases the chance of reusing every individual tuple several times during training. Sequential samples are highly correlated, which can negatively impact the performance of the proposed approach. To mitigate this issue, samples in the experience buffer are randomly selected in small batches. This approach helps to minimize the correlation between sequential samples, resulting in improved performance. In this way, the experience buffer plays a vital role in reducing correlation and enhancing the effectiveness of the proposed method. It is also worth noting that the observations in the buffer might change when the agent interacts with the environment. Whenever the agent receives a new observation, it creates a new tuple and pushes it to the buffer as shown in Fig. 4.

After a careful study of the relationship between the messages generated by the nodes, we chose to set 50 stacked

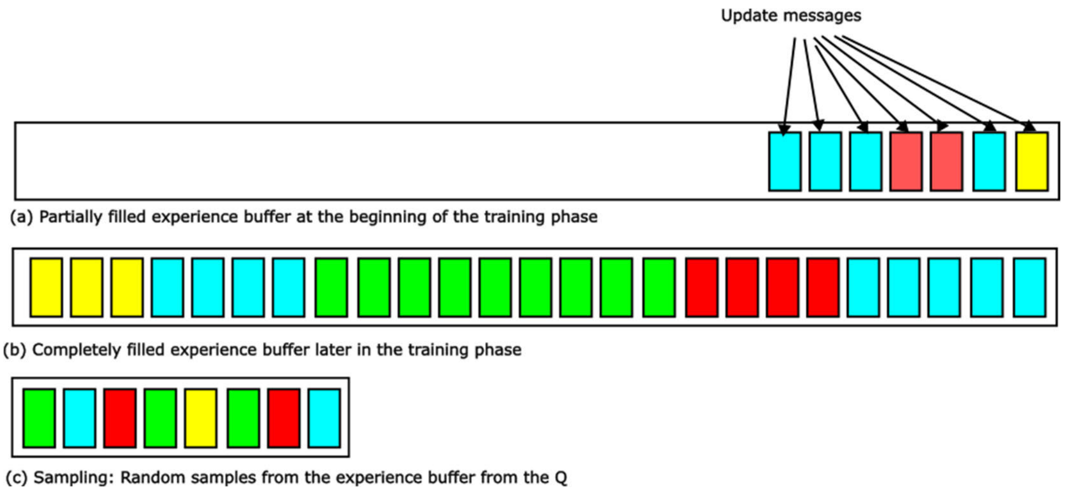

Update messages
(a) Partially filled experience buffer at the beginning of the training phase
(b) Completely filled experience buffer later in the training phase
(c) Sampling: Random samples from the experience buffer from the Q

FIGURE 4. The random sampling in the experience buffer queue.

messages as the input for the network; to speed up the training process and exploit spatial relationships and temporal properties across messages. We noticed that for the majority of the messages, especially the intermediate messages, the nodes were most likely to take the same action for almost every 50 messages. This resulted in an insignificant change in the subsequent messages.

3) TARGET NETWORK

In Q-learning, the value of each Q(s, a) is initialized randomly and the values are changed during training with the most optimal values. The Bellman equation (1) states that the current value of Q(s, a) relies on the next value of Q(s', a'), however, there is only one step between s and s'; therefore, they are likely very similar, resulting in difficulty in finding differences between them during training. It is worth noting that while training the DQN agent to find the best Q(s, a) values, Q(s', a') and the next close states are modified and changed, which results in disorder and instability. In such cases, to reduce the effect of instability in the network, an additional network is used, called the target network. The target network is a copy of the main Q-network used to regress the expected Q values for each (s, a), which are, in turn, used to calculate the loss in the main Q-network. The target network weights did not change as frequently as the main Q-network weights. They were changed every 500 messages and copied from the main Q-network. The main Q-network and the target network are shown in Fig 5. 500 messages are chosen After trying different numbers of messages to synch the target network with the main Q-network weights.

4) THE DQN ALGORITHM

The first step in the DQN algorithm [10] training is to initialize the main Q-network, target network, empty experience buffer D, and agents.

DQN Learning is conducted in two interleaved phases: the sampling phase and learning phase. In the sampling phase, observations were collected by the agents to be used in the training phase; the agent interacts with the environment by performing actions and storing observed experience tuples in the experience buffer. In our proposed approach, each sector in the blockchain is associated with a corresponding DQN agent. The agent receives update messages from the nodes within the sector, which are then stored in the agent's experience buffer for further processing. Notably, the nodes within the sector do not directly communicate with each other, but instead interact solely with the sector's DQN agent. This approach helps to streamline communication within the network, enabling more efficient and effective consensus formation in the context of blockchain P2P networks. On the other hand, in the learning phase the main Q-network is trained to find the best Q(s, a) values. As a matter of fact, both steps are performed alternately, however, initially we start with the sampling phase until the experience buffer is filled.

5) ACTION POLICY

In RL, the agent is trained to maximize the value of Q(s, a) by following policy π. There are different types of policies for choosing the most candidate action such as the random policy, the greedy policy and the ϵ greedy policy. In a random policy, the action is chosen randomly, resulting in an equal probability for each action to be chosen. In the greedy policy, the agent selects the action with the maximum Q-value if taken in state s. In the ϵ greedy policy, the agent is set to select most of the actions with the maximum Q value (greedy) and select a few actions randomly. The percentage of the actions selected greedily and randomly is determined by ϵ, $\epsilon = 0$ resulted in using the greedy policy while $\epsilon = 1$ resulted in using the random policy. It is worth to note that, the ϵ greedy policy helps to balance between the need for exploiting the

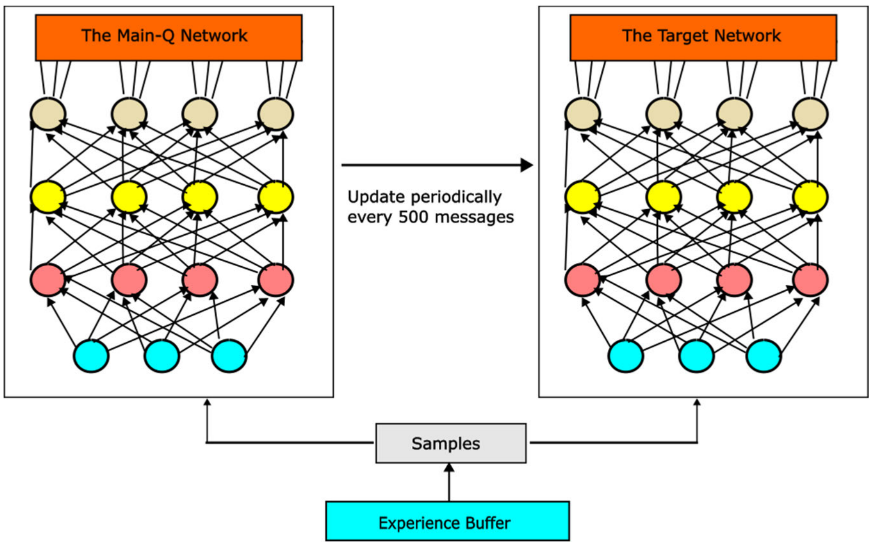


FIGURE 5. Training DQN agents using main Q-network and the target network.

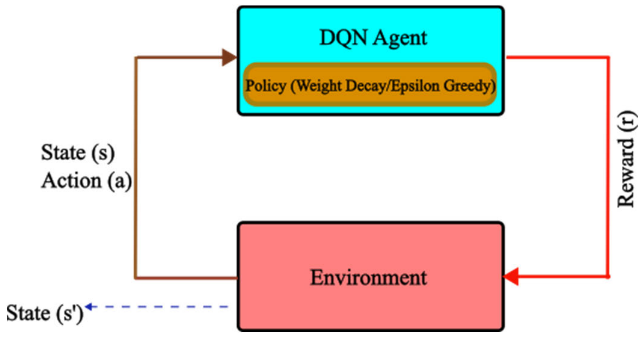


FIGURE 6. Observation cycle: 1) Action a is chosen while in state s given policy. 2) Action a is taken, reward is granted. 3) The next state s' is generated.

agent knowledge by selecting the action with maximum Q value, and the need to explore other actions that may lead to better rewards in the future.

6) DQN-AGENT

In the sampling phase, the experience buffer is filled by enforcing agents to collect observations while interacting with their environments. As shown in Fig. 6 each observation is collected by performing three steps: 1) Action a is chosen while in state s using ϵ greedy(Q). 2) Action a is taken, and its reward r(Q(a)) is calculated. 3) The next state s' is retrieved. Each observation is then stored in the buffer as (s, a, r, s'). We kept accumulating the actions' rewards in the episode until we hit the last action.

As previously noted, there are three potential actions an agent can take in our proposed approach: Accept, Reject, and Neutral. The reward granted to an agent is dependent on the action taken, with observations being inferred from 50 update messages sent by local nodes. To calculate the observation reward, we consider the majority decision of the update messages. Accept and Reject actions are weighted equally in the blockchain consensus process, and therefore, are assigned the same value of reward. In contrast, Neutral actions are not desired in the network, as they do not contribute to consensus formation. As a result, agents receive a reward of +1 for observations resulting in an Accept or Reject action, while Neutral observations are given a reward of 0. For example, if 27 update messages result in an Accept action, 18 in a Reject action, and 5 in a Neutral action, the initial observation reward is calculated as +1 based on the majority decision.

For training purposes, the expected value for each observation must be provided. Unfortunately, the target value is not present; therefore, the Bellman equation was used to calculate an approximation value for each observation. However, if the observation is not the latest in the episode, its target value y_i is calculated as shown in (2).

$$y_i = r_i + \gamma \times max_{a_i \in} Q'(s_i', a_i') \tag{2}$$

where $\boldsymbol{\gamma}$ is the discount factor and ***Q'*** is the target network that is used to find the maximum value for being in the next state s_i' and taking action a_i' in that state. If the observation is the latest in the episode, then $y_i = r_i$, because there is no next state.

After that, the mean squared error loss (L) is calculated between the state-action values and the target (expected) values, L is calculated as shown in (3).

$$L = \frac{1}{N} \times \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2 \tag{3}$$

The main Q-network weights are updated using the Stochastic Gradient Descent (SGD) algorithm to minimize loss. As mentioned earlier, the target DQN network weights were also synchronized with the main DQN network for every 500 messages.

TABLE 1. First scenario architectures.

Architecture#	No. Of Nodes	Input size	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Training Accuracy	Testing Accuracy
1	1,000,000	250	512	512	-	83.6%	78.2%
2	1,000,000	250	1024	1024	1024	85.1%	81.4%
3	1,000,000	250	1024	1024	512	87.8%	83.1%

TABLE 2. DQN agent configurations for the three hierarchies.

		Input size	Hidden 1	Hidden 2	Hidden 3
Hierarchy 1	Layer 1	250	1024	1024	512
	Layer 2	50	128	128	64
	Layer 3	100	256	-	-
Hierarchy 2	Layer 1	250	1024	1024	512
	Layer 2	50	64	64	32
	Layer 3	25	32	32	16
	Layer 4	50	64	-	-
Hierarchy 3	Layer 1	250	1024	1024	512
	Layer 2	50	64	64	32
	Layer 3	50	64	64	32
	Layer 4	25	32	32	16
	Layer 5	10	16	-	-

IV. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, the proposed method is evaluated in two scenarios to reach a consensus between the nodes. The first scenario consisted of one DQN trained using three different deep architectures, as presented in Table 1. In the second scenario, three DQN agent hierarchies were proposed, as shown in Table 2. The data used to evaluate the two scenarios were randomly generated to simulate real working environments.

A. TRAINING PARAMETERS

In this work, the training converges when the agent almost chooses the correct decision 40 times for every update message in the last 50 episodes (update messages). The discount factor (γ) was 0.85, the minibatch size was 100 (each sample contained 50 messages), the learning rate was 1e-4, the experience buffer size was 5000 samples, and the main DQN network was synchronized with the target DQN network every 500 messages. The input for the architectures comprises samples of the update messages.

B. FIRST SCENARIO: SETTINGS AND RESULTS

In the first scenario, a DQN agent, which is the only agent in the network, is trained to reach consensus. All nodes in the network send their local decisions about a particular chain version (update message) to the main DQN agent. The DQN agent acts greedily to select actions that maximize the Q-value. Three deep network architectures are introduced to train the agent in this scenario. The input size and settings for each architecture with the training and testing accuracy results are presented in Table 1. A dataset of 1,600,000 observations was collected to evaluate the proposed method for this scenario. The data were divided into two sets: training and testing sets. A total of 1,200,000 observations were chosen randomly for training, while the rest were used for testing.

As indicated in Table 1, the number of nodes used in the experiment for all the architectures was one million. It is also obvious from the table that Architecture 3 performs slightly better than Architectures 1 and 2. The training accuracies are 83.6%, 85.1%, and 87.8% for Architecture 1, 2 and 3, respectively. The test accuracy is78.2%, 81.4%, and 83.1% for Architecture 1, 2, and 3, respectively.

To study the effect of increasing the number of nodes that reject the update message on the result, Architecture 3 was evaluated. Fig. 7 shows that the number of nodes that reject the message initially is chosen to represent 20% of the total nodes, and then gradually increases to 80%. It is obvious from the figure that the model is able to discover whether there is a consensus between the nodes when less than 40% or when approximately 63% or more of the nodes reject the update message. However, the figure also suggests that if the rejection percentage among the nodes is between 40% and 62%, the DQN agent accuracy decreases (68%-79%).

As discussed earlier in the methodology section, the ϵ-greedy(A) policy algorithm is used to train DQN agents to select the action with the maximum Q value. Hence, Fig. 8 is introduced to study the effect of changing the ϵ value on agent performance. That might be due to the fact of as a matter of fact, when ϵ value is close to 1 the agent is always trying to select as much random actions for exploring more options. On the other hand, when ϵ value is getting close to 0 the agent

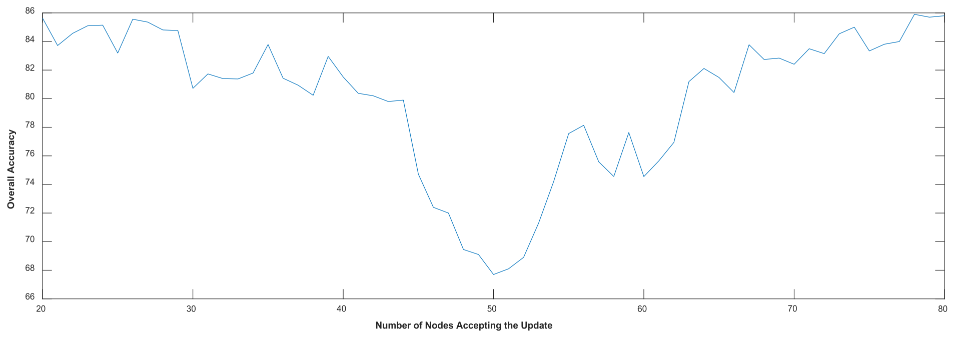


FIGURE 7. Overall accuracy vs number of nodes that accepts the update message.

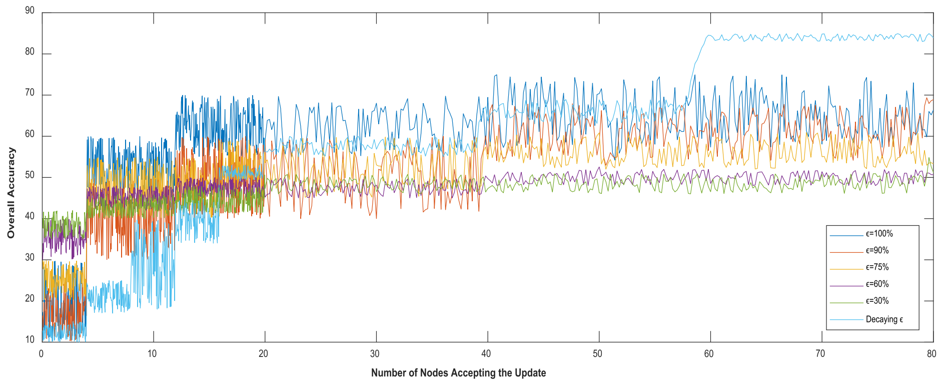


FIGURE 8. The performance of the DQN agent with different ϵ values.

starts acting greedily and exploiting and choosing the action with the higher Q value. The results in Fig. 8 show that the agent performance is getting enhanced if ϵ value is initialized with a high value that is decreased while learning over the time. The overall accuracy increased from 83.1% to 84.6%. That might be due to the fact that when ϵ value is close to 1 the agent is always trying to select as much random actions for exploring more options to find actions with the higher Q values. On the other hand, when ϵ value is getting close to 0 the agent starts acting greedily and exploiting and choosing the action with the higher Q value.

It can also be observed from Fig. 8 that the agent's performance sharply decreases (approximately 50%) if 70% of the actions are greedily chosen, while 30% are randomly chosen by the agent ($\epsilon = 30\%$).

C. SECOND SCENARIO: SETTINGS AND RESULTS

The second proposed scenario was motivated by our previous study [31]. In [31], we proposed the utilization of a hierarchy of DRL agents. The distributed nodes are divided into multiple layers, with each layer consisting of sectors of varying sizes. The size of each sector is proportional to the density of nodes in the targeted area. Specifically, an area with a high density of nodes will be divided into relatively small sectors, whereas an area with a low density of nodes will have larger sectors. The primary objective of this division is to ensure that the number of nodes in each sector is approximately equal. For every sector, a DQN agent is added to the sector, and the nodes within the sector communicate with the agent as depicted in Fig. 9. The subsequent layer will cover the area of the distributed agents from the previous layers, and the same sector-based division will be applied, with one agent selected for each sector. The agents within a given sector in the second layer communicate with their respective sector agent. This pattern is repeated for each subsequent layer, as illustrated in Fig. 10.

The primary objective of communication that takes place in each layer between the nodes in each sector and their agent, as well as between agents in different sectors, is to achieve a collective consensus within a short timeframe. All agents will be trained to find the optimal policy for consensus based on the available data received from local nodes and agents from other sectors, utilizing DQN reinforcement learning. The optimal number of layers, sectors within each layer, and nodes within each sector will be studied in section IV.

The agents in [31] were trained using a deep network along with a cross-entropy algorithm to determine the optimal policy π. Policy π is the route followed by the agent to opt for the next action (Accept, Reject, or Neutral) based on the current state of the environment observations. In the second Scenario, we propose to use three hierarchies of DQN agents. Table 2 lists the deep network settings for the three hierarchies of DQN agents. The table shows the number of layers used in

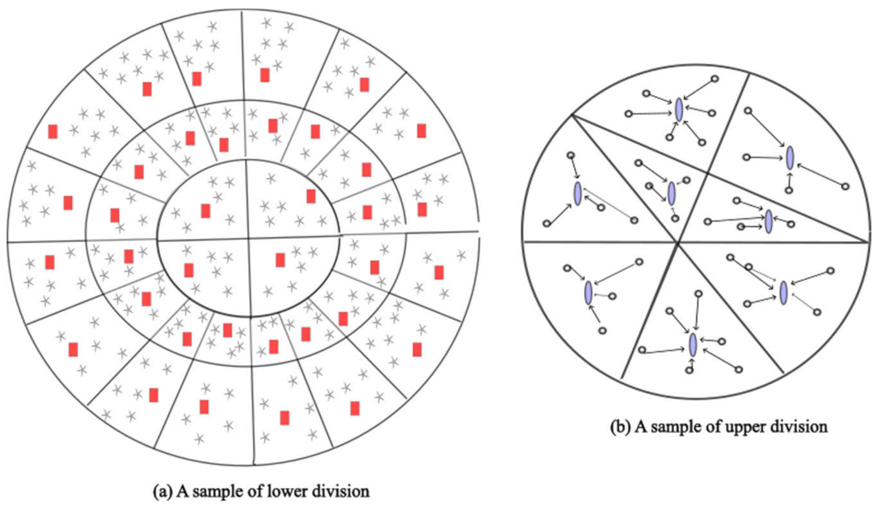

(a) A sample of lower division
(b) A sample of upper division

FIGURE 9. Integrating DQN agents into the blockchain network.

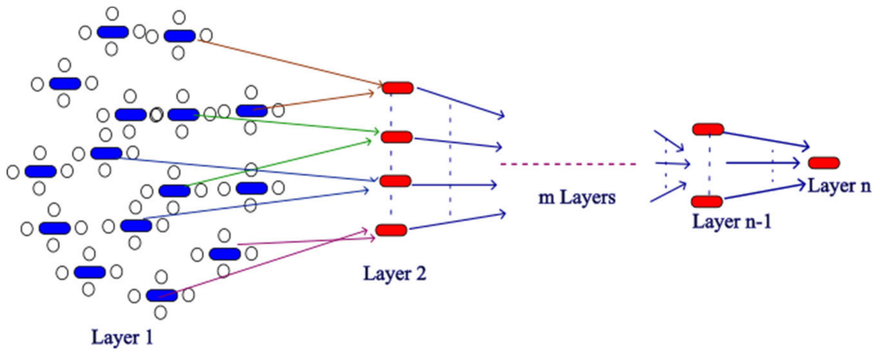


FIGURE 10. Communication within the sectors and among DQN agents across different layers.

every hierarchy and the number of hidden layers required for each DQN agent level.

A dataset of 1,600,000 observations was collected to evaluate the proposed method for this scenario. The data were divided into two sets: training and testing sets. A total of 1,200,000 observations were chosen randomly for training, while the rest were used for testing. The numbers of training observations used per agent for the first, second, and third hierarchies were 6000, 2400, and 1200, respectively. The number of testing observations used per agent for the first, second, and third hierarchies was 1000, 800, and 400, respectively. It should be noted that different sets of observations were used to test each agent. Furthermore, different values for the propagation time (delay time) for each hierarchy were chosen, as listed in Tables 3, 4, and 5. In each hierarchy initially ϵ is set to a high value, that decays while the agent is being learned. At the beginning of the learning process, the agent attempts to explore the environment to update its settings. However, as the agent moves forward in the learning process, exploration decreases.

The settings and accuracy results for the three hierarchies are presented in Tables 3, 4, and 5. As shown in Table 3, the first hierarchy consists of 3 layers of agents. The number of nodes was 1,000,00, 200, and 20 in layers 1, 2, and 3, respectively. The number of agents was 200, 20, and 1 in the same layers. Table 4 shows that, in the second hierarchy, there were four layers. The number of nodes was 1,000,000, 500, 50, and 10 for layers 1, 2, 3, and 4, respectively. The number of agents was 500, 50, 10, and 1 for layers 1, 2, 3, and 4, respectively. Table 5 indicates that the third hierarchy is divided into five layers. The number of nodes is 1,000,000, 1000, 100, 10, and 2 for layers 1, 2, 3, 4, and 5, respectively. The number of agents was 1,000, 100, 10, 2, and 1 for layers 1, 2, 3, 4, and 5, respectively. It can be observed from Tables 3, 4, and 5 that the propagation time is increased as we move forward from the lower layers toward the upper layers in every hierarchy. This is due to the fact that the agents in the upper layers need to wait until the communication is finished in the lower layers.

The results in Tables 3, 4, and 5 show the overall accuracy for each hierarchy. The overall accuracy of the training set was 88.1%, 88.4%, and 90.1% for hierarchies 1, 2, and 3, respectively. The accuracy results for the testing set were 85.2, 86.3, and 87.8% for hierarchies 1, 2, and 3, respectively.

TABLE 3. The first hierarchy configuration.

	Layers			
	1	2	3	Average
Nodes	1,000,000	200	20	-
Sectors	200	20	1	-
Nodes/Sector	5000	10	20	-
Delay Time	14	15	15.6	14.87
Train Accuracy	87.7%	87.9%	88.7%	88.1%
Test Accuracy	84.7%	85.3%	85.7%	85.2%

TABLE 4. The second hierarchy configuration.

	Layers				
	1	2	3	4	Average
Nodes	1,000,000	500	50	10	-
Sectors	500	50	10	1	-
Nodes/Sector	2,000	10	5	10	-
Delay Time	13	14	14.7	14.8	14.13
Train Accuracy	87.9%	88.4%	88.5%	88.7%	88.4%
Test Accuracy	85.8%	86.1%	86.2%	87.1%	86.3%

TABLE 5. The third hierarchy configuration.

	Layers					
	1	2	3	4	5	Average
Nodes	1,000,000	1,000	100	10	2	-
Sectors	1,000	100	10	2	1	-
Nodes/Sector	1,000	10	10	5	2	-
Delay Time	11	12	12	12.5	12.1	11.92
Train Accuracy	89.3%	89.7%	89.9%	90.7%	90.8%	90.1%
Test Accuracy	87.6%	87.7%	87.8%	87.9%	88.1%	87.8%

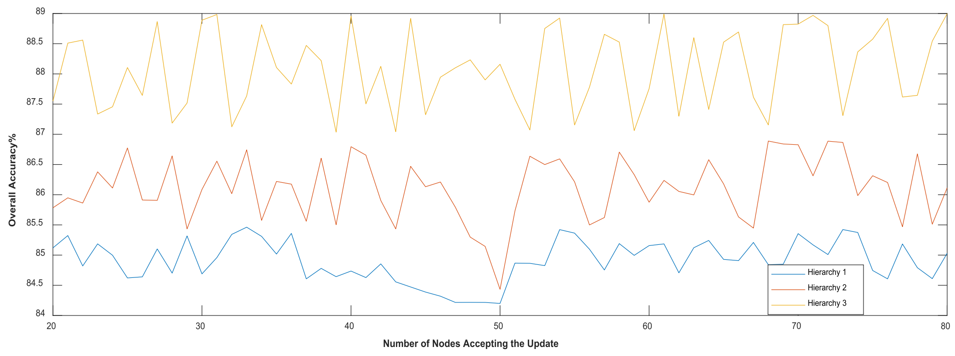


FIGURE 11. Accuracy comparison between Hierarchy 1, Hierarchy 2, and Hierarchy 3.

It is obvious that accuracy is enhanced as we move from the first to the third hierarchy. This might be because of two main reasons: a) the number of DQN agents in every layer for each hierarchy; when comparing the number of DQN agents in every equivalent layer in the three hierarchies, the number is always increased while we are moving toward hierarchy 3. For instance, the number of DQN agents in the first layer is 200, 500, and 1000 for hierarchies 1, 2, and 3, respectively, whereas the number of DQN agents in the second layer is 20, 50, and 100 for hierarchies 1, 2, and 3, respectively. Thus, it is clear that increasing the sector numbers in every layer resulted in an increase in the

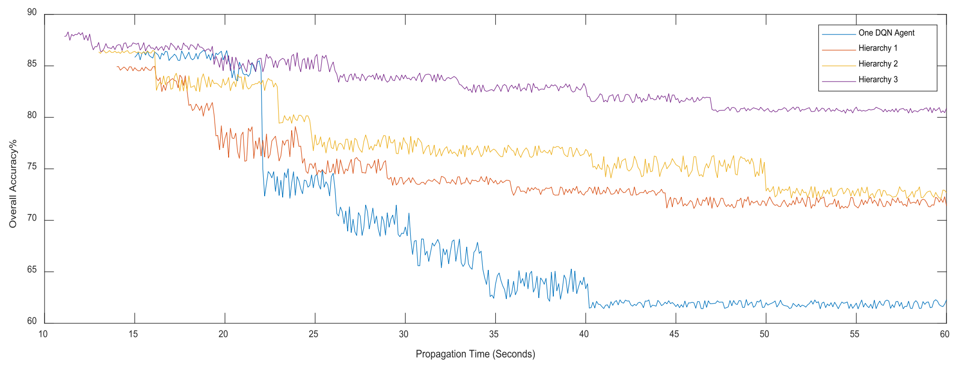


FIGURE 12. The overall accuracy vs. the propagation time for all architectures in both scenarios.

number of DQN agents, which can help the hierarchy reach a better consensus. b) The number of layers in each hierarchy: The number of layers is also increased as we move from the first to the third hierarchy, resulting in an increase in the number of DQN agents. Moreover, it appears that having more layers of DQN agents reduces the error caused by the lower agents with the help of the upper agents while they are learning.

Even though the result is enhanced by having more agents and more layers in the hierarchy, it negatively impacts the total propagation time. It is clear from the results in Tables 3, 4, and 5 that the propagation time is slightly increased by increasing the number of agents and layers. This is a reasonable result because the number of messages exchanged within each sector and between agents across the layers increases as the number of agents increases in the hierarchy. However, the proposed approach can contribute to reducing bandwidth utilization. Communication occurs between nodes and their agents within each layer, as well as between agents in different layers, as opposed to the standard method utilized in other consensus protocols, which involves the exchange of update messages between neighboring nodes. This can lead to nodes sending millions of handshake packets and receiving the same message multiple times from their neighbors, causing significant bandwidth issues, startup latency, and other network-related problems. Additionally, the proposed approach is relatively immune to packet loss. Lost or delayed messages can be handled as in the case of messages with neutral votes. Neutral votes are discarded, and decisions are made based on the majority vote. It is worth noting that other blockchain P2P consensus protocols, such as PoW and Proof of Stake (PoS), also suffer from lost and delayed packets, as it is an inherent problem within P2P networks.

Fig. 11 studies the effect of nodes number that rejects the update message on the performance of the three hierarchies. Number of nodes that reject the message starts by 20% and then gradually is increased to 80%. Even though the three hierarchies reach to a consensus among the nodes, however, the performance of the first and the second hierarchies was slightly affected when number of nodes was around 50%. On the other hand, the third hierarchy retained its performance and was not affected.

Fig. 12 shows the effect of increasing the propagation time on the four DQN agent proposed in both scenarios. Initially the propagation time is set to 11 seconds, and it gets increased gradually to 60 seconds. The result shows a slightly better performance for hierarchy 3 over the other proposed DQN agents in both scenarios. The degradation in the overall accuracy of hierarchy 3 was the smallest compared with the other DQN agents in both scenarios. And this might be as per the fact that number of agents is the largest in hierarchy 3 among all other hierarchies in both scenarios, which leads to an efficient distribution for the agents among the layers in this hierarchy. And also leads to assign an efficient number of nodes per agent in every layer. Furthermore, hierarchy 3 is barley affected by the propagation time as the other hierarchies in both scenarios because the number of nodes that communicate with each agent is smaller than those are in the other hierarchies in both scenarios.

D. LIMITATION AND COMPARISON WITH POPULAR CONSENSUS ALGORITHMS

This section provides an overview of the limitations of commonly used consensus protocols in blockchain, including PoW, PoS, Delegated Proof of Stake (DPoS), Practical Byzantine Fault Tolerance (PBFT), and Ripple, and illustrates how the proposed approach addresses most of these shortcomings.

One of the key challenges with PoW is its computational intensity, which necessitates a significant amount of computing power to solve complex mathematical problems, resulting in high energy consumption [8]. In contrast, the proposed DQN agent approach does not need to solve complex mathematical problems, it involves limited small messages that are sent within the sector nodes and between the agents across sectors, leading to a dramatic reduction in power consumption across the network. While PoS and DPoS algorithms have effectively reduced the amount of power required compared to PoW, they still limit participation in the consensus process to only those miners who have significant financial resources (stakeholders) [32], [33]. Conversely, the proposed method leverages agents trained using efficient reinforcement DQN learning techniques, which are able to incorporate insights from network nodes to arrive at a final decision. This approach offers a more democratic and inclusive consensus process that is not limited by financial resources alone.

PoW, PoS, and DPoS can be applied to public blockchain since any node can participate in the consensus process. Conversely [8], [32], [33], PBFT and Ripple are suitable only for private blockchain and consortium blockchain since only permitted nodes can participate in the consensus process [6], [34]. DQN agents can be applied to both public and private blockchain because initially, all miners can participate in the consensus process by sending small messages to the sector agent inside the sector. At the second level, specific nodes (agents) decide and lead the consensus process.

Most of these protocols have good scalability and can be used for a large-scale network, except for PBFT, which is a good fit for high-performance networks with limited nodes. DQN agent is highly scalable, and a new sector with a new agent can be added anytime to the network. Additionally, a new node is easily added to the network by communicating with the agent sector.

V. CONCLUSION

In this study, RL was utilized to enable agents to learn how to reach consensus between distributed P2P nodes. More precisely, DQN agents were deployed to achieve the necessary agreement on a single state of the network (the update message) among the nodes in the network.

Two scenarios are introduced and evaluated. In the first scenario, we introduced one main DQN agent trained using three different deep network architectures. Using the greedy policy ($\epsilon = 0$), the overall test accuracies of the three architectures are 78.2%, 81.4%, and 83.1%, respectively. However, the overall accuracy is getting enhanced to 84.6% when the agent trained with a policy that follows decaying values of ϵ. The decaying policy enables the agent to combine two impressive policies: random and greedy; using both the agent learns to explore different actions at the early stages of the train and then learns to exploit the actions with higher Q values at the later stages. In the second scenario, a set of DQN agents with different hierarchies was deployed and evaluated. We proposed three hierarchies (hierarchies 1, 2, and 3), where each hierarchy has a different number of layers with different numbers of DQN agents. The overall test accuracies for hierarchies 1, 2, and 3 were 85.2%, 86.3%, and 87.8% respectively.

To conclude, a relatively high accuracy for consensus in P2P nodes was achieved using DQN agents. Thus, it seems that utilizing deep agents to reach consensus in P2P networks has promising results. Therefore, we plan to improve our findings by 1) utilizing different type of deep agents. 2) Study the effect of combining different types of agents in different layers of the proposed hierarchies in the second scenario. 3) Utilizing different sets of actions.

ACKNOWLEDGMENT

(Arafat Abu Mallouh and Zakariya Qawaqneh contributed equally to this work.)

REFERENCES

[1] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006.

[2] R. Ormándi, I. Hegedus, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency Comput., Pract. Exper.*, vol. 25, no. 4, pp. 556–571, May 2012.

[3] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *Proc. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 5904–5914.

[4] X. Lian, C. Zhang, H. Zhang, C. J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 5330–5340.

[5] F. Sun and P. Duan, "Solving Byzantine problems in synchronized systems using bitcoin," Tech. Rep., 2014. [Online]. Available: https://allquantor.at/blockchainbib/pdf/sun2014solving.pdf

[6] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.

[7] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018.

[8] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2084–2123, 3rd Quart., 2016.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, M. G. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[11] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2018, vol. 32, no. 1, pp. 1–8.

[12] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[13] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, and A. Matyasko, "Technical report on the CleverHans v2. 1.0 adversarial examples library," 2016, *arXiv.1610.00768*.

[14] G. Lucarelli and M. Borrotti, "A deep Q-learning portfolio management framework for the cryptocurrency market," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17229–17244, Dec. 2020.

[15] P. Zeng, S. Cui, C. Song, Z. Wang, and G. Li, "A multiagent deep deterministic policy gradient-based distributed protection method for distribution network," *Neural Comput. Appl.*, vol. 35, no. 3, pp. 2267–2278, Feb. 2022.

[16] J. Kim, D. Kwon, S. Y. Woo, W.-M. Kang, S. Lee, S. Oh, C.-H. Kim, J.-H. Bae, B.-G. Park, and J.-H. Lee, "On-chip trainable hardware-based deep Q-networks approximating a backpropagation algorithm," *Neural Comput. Appl.*, vol. 33, no. 15, pp. 9391–9402, Feb. 2021.

[17] H. Chen, Q. Liu, and S. Zhong, "Self-guided deep deterministic policy gradient with multi-actor," *Neural Comput. Appl.*, vol. 33, no. 15, pp. 9723–9732, Mar. 2021.

[18] H. Jang, S. Hao, P. M. Chu, P. K. Sharma, Y. Sung, and K. Cho, "Deep Q-network-based multi-criteria decision-making framework for virtual simulation environment," *Neural Comput. Appl.*, vol. 33, pp. 10657–10671, Sep. 2021.

[19] W. Hu, Y. Yang, G. Chen, and M. Meng, "Bipartite consensus of double-integrator multi-agent systems with nonuniform communication time delays," *Neural Comput. Appl.*, vol. 33, no. 7, pp. 2285–2295, Apr. 2021.

[20] Y. Wang and Z. Wang, "Model free adaptive fault-tolerant consensus tracking control for multiagent systems," *Neural Comput. Appl.*, vol. 34, no. 12, pp. 10065–10079, 2022.

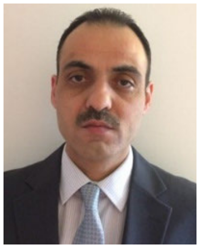
[21] H. Zhao, M. Zhang, S. Wang, E. Li, Z. Guo, and D. Sun, "Security risk and response analysis of typical application architecture of information and communication blockchain," *Neural Comput. Appl.*, vol. 33, no. 13, pp. 7661–7671, Jul. 2021.

[22] A. Oikonomidis, C. Catal, and A. Kassahun, "Hybrid deep learning-based models for crop yield prediction," *Appl. Artif. Intell.*, vol. 36, no. 1, pp. 1–18, Jan. 2022.
[23] S. Videgaín and P. G. Sánchez, "Performance study of minimax and reinforcement learning agents playing the turn-based game iwoki," *Appl. Artif. Intell.*, vol. 35, no. 10, pp. 717–744, Jun. 2021.
[24] M.-A. Chadi and H. Mousannif, "A reinforcement learning based decision support tool for epidemic control: Validation study for COVID-19," *Appl. Artif. Intell.*, vol. 36, no. 1, pp. 1–33, Feb. 2022.
[25] K. Nicolas and Y. Wang, "A novel double spending attack countermeasure in blockchain," in *Proc. IEEE 10th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*. New York, NY, USA: Columbia University, Oct. 2019, pp. 0383–0388.
[26] C. Qiu, X. Ren, Y. Cao, and T. Mai, "Deep reinforcement learning empowered adaptivity for future blockchain networks," *IEEE Open J. Comput. Soc.*, vol. 2, pp. 99–105, 2021.
[27] E. Rabieinejad, S. Mohammadi, and M. Yadegari, "Provision of a recommender model for blockchain-based IoT with deep reinforcement learning," in *Proc. 5th Int. Conf. Internet Things Appl. (IoT)*. Isfahan, Iran: University of Isfahan, May 2021, pp. 1–8.
[28] Z. Yang, R. Yang, F. R. Yu, M. Li, Y. Zhang, and Y. Teng, "Sharded blockchain for collaborative computing in the Internet of Things: Combined of dynamic clustering and deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16494–16509, Sep. 2022.
[29] Y. Gong, S. Sun, Y. Wei, and M. Song, "Deep reinforcement learning for edge computing resource allocation in blockchain network slicing broker framework," in *Proc. IEEE 93rd Veh. Technol. Conf. (VTC-Spring)*, Helsinki, Finland, Apr. 2021, pp. 1–6.
[30] R. Ma, Z. Yi, Y. Xiang, D. Shi, C. Xu, and H. Wu, "A blockchain-enabled demand management and control framework driven by deep reinforcement learning," *IEEE Trans. Ind. Electron.*, vol. 70, no. 1, pp. 430–440, Jan. 2023.
[31] A. A. Mallouh, O. Abuzaghleh, and Z. Qawaqneh, "A hierarchy of deep reinforcement learning agents for decision making in blockchain nodes," in *Proc. IEEE EUROCON 19th Int. Conf. Smart Technol.*, Lviv, Ukraine, Jul. 2021, pp. 197–202.
[32] S. King and S. Nadal, "PPCoin: Peer-to-peer crypto-currency with proof-of-stake," *Self-Published Paper*, vol. 19, no. 1, pp. 1–6, Aug. 2012.
[33] D. Larimer, "Delegated proof-of-stake (DPoS)," Bitshare, White Paper, 2014.
[34] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OsDI*, New Orleans, LA, USA, 1999, pp. 173–186.

ARAFAT ABU MALLOUH received the Ph.D. degree in computer science and engineering from the University of Bridgeport, in 2017. He is currently an Assistant Professor with the Computer Science Department, Manhattan College, NY, USA. His research interests include artificial intelligence, machine learning, deep learning, reinforcement learning, blockchain and cryptocurrencies, and cybersecurity.

ZAKARIYA QAWAQNEH received the Ph.D. degree in computer science and engineering. He is currently an Assistant Professor with the Department of Computing Sciences, State University of New York Brockport, USA. His research interests include artificial intelligence, machine learning, deep learning, reinforcement learning, data science, data analytics, blockchain and cryptocurrencies, and cybersecurity.

OMAR ABUZAGHLEH received the Ph.D. degree in computer science and engineering. He is currently an Assistant Professor with the Department of Computer Information Science, Higher Colleges of Technology, Dubai, United Arab Emirates. His research interests include signal and image processing, machine learning, artificial intelligence, network security, blockchain technology, and reinforcement learning.



AHMAD AL-RABABA'A received the Ph.D. degree in computer science from Laval University, in 2016. He is currently an Assistant Professor with the Computer Science Department, The World Islamic Science and Education University. His research interests include data compression and coding, information theory, the applications of signal processing, natural language processing, and AI.

• • •