Expert Systems With Applications 214 (2023) 119160

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A reinforcement learning model for the reliability of blockchain oracles

Mona Taghavi [a,*], Jamal Bentahar [a,b], Hadi Otrok [b,a], Kaveh Bakhtiyari [c,d]

[a] Concordia Institute for Information System Engineering, Concordia University, Montreal, Canada
[b] Department of EECS, Khalifa University, Abu Dhabi, United Arab Emirates
[c] Interactive Systems, Duisburg-Essen University, Duisburg, Germany
[d] Department of Electrical Engineering, The National University of Malaysia, Bangi, Malaysia

ARTICLE INFO

Keywords:
Smart contract
Blockchain oracle
Multi-armed bandit
Reinforcement learning
Reputation model

ABSTRACT

Smart contracts struggle with the major limitation of operating on data that is solely residing on the blockchain network. The need of recruiting third parties, known as oracles, to assist smart contracts has been recognized with the emergence of blockchain technology. Oracles could be deviant and commit ill-intentioned behaviors, or be selfish and hide their actual available resources to gain optimal profit. Current research proposals employ oracles as trusted entities with no robust assessment mechanism, which entails a risk of turning them into centralized points of failure. The need for an effective method to select the most economical and rewarding oracles that are self-interested and act independently is somehow neglected. Thus, this paper proposes a Bayesian Bandit Learning Oracles Reliability (BLOR) mechanism to identify trustless and cost-efficient oracles. Within BLOR, we learn the behavior of oracles by formulating a Bayesian cost-dependent reputation model and utilize reinforcement learning (knowledge gradient algorithm) to guide the learning process. BLOR enables all the blockchain validators to verify the obtained results while running the algorithm at the same time by dealing with the randomness issue within the limited blockchain structure. We implement and experiment with BLOR using Python and the Solidity language on Ethereum. BLOR is benchmarked against several models where it proved to be highly efficient in selecting the most reliable and economical oracles with a fair balance.

1. Introduction

Blockchain technology has the ability to cut the role of middlemen by enabling self-enforcing digital contracts (called smart contracts), whose execution does not require any human involvement in a safe, secure, and immutable way. The emergence of the blockchain as a revolutionary technology has been compared to the Internet, and it has predicted that it will erode power from centralized authorities. With its deployment as a service (Lu, Xu, Liu, Weber, Zhu, & Zhang, 2019) and its integration with IoT (Baygin, Yaman, Baygin, & Karakose, 2022; Ho, Tang, Tsang, Tang, & Chau, 2021), blockchain has a promising approach in supporting business collaborations by ensuring transparency to all the stakeholders if conflicts arise (Hull et al., 2016). However, the integration of blockchain with external data is one of the major obstacles preventing widespread adoption. Imagine that two persons place a bet on who wins a football match and deposit their funds in a smart contract. Based on the results of the game, the smart contract should release the funds to the winner. However, a smart contract does not have access to the data out of its network and should ask a trusted party to learn who won the match.

In blockchain, the term oracle refers to an entity that can access external data without compromising the integrity of the blockchain. Oracles are assumed to be third-party agents that are trustworthy and can communicate with the outside world, and fetch the data into the blockchain Xu et al. (2016). Oracles are also able to connect the blockchain to external databases. This way, costly computations can be carried out outside of the blockchain. Oracles ensure the integrity of the retrieved data by providing some evidences (Kochovski, Gec, Stankovski, Bajec, & Drobintsev, 2019). Thus, cryptographic-based evidences such as the ones used by Oraclize,[1] or trusted hardware-based evidences such as the ones used by the Town Crier system that leverages Intel SGX (Zhang, Cecchetti, Croman, Juels, & Shi, 2016) are used as part of a number of oracle-based systems. These evidences are not only insufficient to ensure that the data is tamper-proof, they are impractical in many real-world applications where the digital data is not available or human involvement is required.

Oracles could display ill-intentioned behaviors, or unable to perform their tasks due to lack of capacity and being selfish by failing to

* Corresponding author.
E-mail addresses: m_tag@encs.concordia.ca (M. Taghavi), bentahar@ciise.concordia.ca (J. Bentahar), hadi.otrok@ku.ac.ae (H. Otrok), academic@bakhtiyari.com (K. Bakhtiyari).

[1] https://provable.xyz/

https://doi.org/10.1016/j.eswa.2022.119160
Received 9 February 2022; Received in revised form 25 October 2022; Accepted 25 October 2022
Available online 29 October 2022

report their real available resources (Lo, Xu, Staples, & Yao, 2020). Thus, placing a reliable mechanism to select the right oracles plays a significant role in a blockchain network's success. There are several proposals for organizing one or more oracles as a group with trustworthy mechanisms, specifically designed for computer hardware and software (Berryhill & Veneris, 2019; Goel, van Schreven, Filos-Ratsikas, & Faltings, 2020). However, these methods are not applicable when human intervention is involved or when the original data source is malicious. Moreover, these proposals sought to organize one or more oracles with enhanced security features or incentive mechanisms (Khosravifar, Bentahar, Moazin, & Thiran, 2010). To the best of our knowledge, there is no smart mechanism to promote how to select the most rewarding oracles among the existing ones in a market of oracles that might act selfishly to gain optimal profit.

In this paper, we utilize a Bayesian multi-armed bandit to learn the most rewarding oracles from the two perspectives of reliability and cost efficiency, to perform specific tasks within a blockchain. Multi-armed bandit is a reinforcement learning method that assumes the player does not know how much it will earn each time playing a particular slot machine, but the player has a distribution of belief, which could be wrong. The only way the player learns who has the highest expected reward is to try all machines, even those that do not appear to be the best. While trying these machines, the player may be earning lower rewards. The ultimate goal is to balance what we earn against what we learn (to improve future decisions) to maximize the expected sum of rewards. In our case, oracles are considered to be slot machines and blockchain beneficiaries are players who try to recruit the best oracles. Reinforcement learning methods have been applied in many real-world applications (Alagha, Singh, Mizouni, Bentahar, & Otrok, 2022; Rjoub, Bentahar, Wahab, & Bataineh, 2021; Rjoub, Wahab, Bentahar, Cohen & Bataineh, 2022; Sami, Bentahar, Mourad, Otrok, & Damiani, 2022; Sami, Mourad, Otrok, & Bentahar, 2021; Sami, Otrok, Bentahar, & Mourad, 2021) and their employment within blockchain has great advantages including high accuracy, ability to learn with few or no historical record, and low computational resources consumption (Sutton & Barto, 2018). To the best of our knowledge, these methods have not been applied in the field of blockchain yet, and even though it would be very interesting and novel, serious challenges in design and implementation within current platforms arise.

Theoretical and practical challenges: The issue of selecting the most rewarding oracle is a decision-making problem that should capture the tensity between exploration of new oracles and exploitation of the good and well-known ones. For simple and low number of choices, dynamic programming can compute the optimal solution. However, it is very computationally inefficient in the blockchain environment with the growing number of oracles working for blockchains. There is a need for an algorithm that runs quickly with a very minimal computation surcharge. The reason is that this algorithm has to be running by all blockchain validators (i.e., miners) acting within the network. Furthermore, current solutions of multi-armed bandit assume that the player retains little information about the past, or switch between exploration and exploitation either randomly or after a fixed number of trials. These solutions are not practical for our problem, since oracles could be run and managed by intelligent agents that can change their behavior anytime. Another challenge of utilizing current solutions is that our decision-making procedure should be based not only on the oracles' performance, but also on their cost of performing the task considering applications' limited budgets. There could be some reliable and high performance oracles that are expensive, but current solutions would always select them based on their past performance records. We assume a fixed cost for each oracle, and consider the oracles reputation and cost of other oracles in the market could change the behavior of each individual oracle.

To overcome the aforementioned challenges, we formulate a Bayesian cost-dependent reputation model to learn the behavior of oracles and utilize knowledge gradient algorithm which guides the learning process based on the marginal value of information. Using a Bayesian model for blockchain is complex, since the algorithm has to produce the same results in every course of experiment. This is because all the validators should verify the results and it only happens if all of them come up with the same results while running the algorithm. This adds further complexity since all the Bayesian reinforcement learning methods include randomness and use random variables. At last, the current platforms of blockchains and smart contracts are very limited, for example no floating number can be defined within blockchain, or limited number of variables can be defined for Ethereum. This paper discusses how the proposed model and mechanism tackles and solves these issues by formalizing the oracles' performance optimization as a Bayesian bandit problem. Our algorithmic model defines a distribution over oracles with different reputations (representing their reliability and costs) to be used by blockchain participants to choose best performing oracles on future requests.

Contributions: This paper contributes as follows:

1. Formulating a new model using a Bayesian cost-dependent reputation model (BCRM) and knowledge gradient (KG) to find the most rewarding oracles. BCRM captures the behavior of the oracles elegantly, and KG unfolds the exploration/ exploitation dilemma in multi-armed bandit with very low computational cost and high accuracy.
2. Proposing a framework to show how to employ the model within a blockchain where all the validators need to achieve a consensus. This framework incentivizes oracles to continuously act honestly and provide a fair balance of quality and price with minimal possibility of acting maliciously.
3. Adapting a reinforcement learning algorithm for blockchain environment with limited computational resources and capabilities (e.g., there is no floating number in Ethereum). Designing and implementing a reinforcement learning solution for the oracle selection problem is an objective yet to be achieved.

We simulated and implemented our proposed model using Python on Google Colab and Solidity on Ethereum. The implementation of BLOR deals with many challenges raised by the complexity of machine learning and limitations of blockchain and Ethereum, such as floating numbers, randomness and advanced mathematical numbers that are not supported in blockchain. Since there is no real-world data on oracles working for blockchains, we had to simulate the behavior of 100 oracles during 1000 observations to assess the performance of our model and compare it with other comparative algorithms.

The reminder of this paper contains the following sections: Section 2 explains the trust paradox of oracles and blockchains to motivate the problem statement. Section 3 discusses the related work. Section 4 presents BLOR as our proposed model and framework and provides an illustrative example to show how the model works. Section 5 provides a case study in which BLOR is applied. Experimental details and results are covered in Section 6. Lastly, the conclusion is drawn in Section 7.

2. Motivational scenario: Trust paradox of oracles and blockchains

Many blockchain platforms have been experiencing the oracle idea since the beginning of Ethereum, but the oracle dilemma continues unsolved at a large scale. The most challenging part is that majority of oracles require a level of trust, which directly opposes the trustless blockchains' nature. The main complication of using oracles is trusting them as outside sources of information. The trust issue connected with oracles is referred to as the oracle problem.

Fig. 1 presents the motivating scenario of this paper. Let us assume a smart contract running an insurance marketplace platform in a trustless environment. Imagine that an insured costumer has a car crash and makes a claim to its insurance company. According to the agreed policies signed in the smart contract, the insurance company requires the crash sensors' data and some evidences to process the claim, but

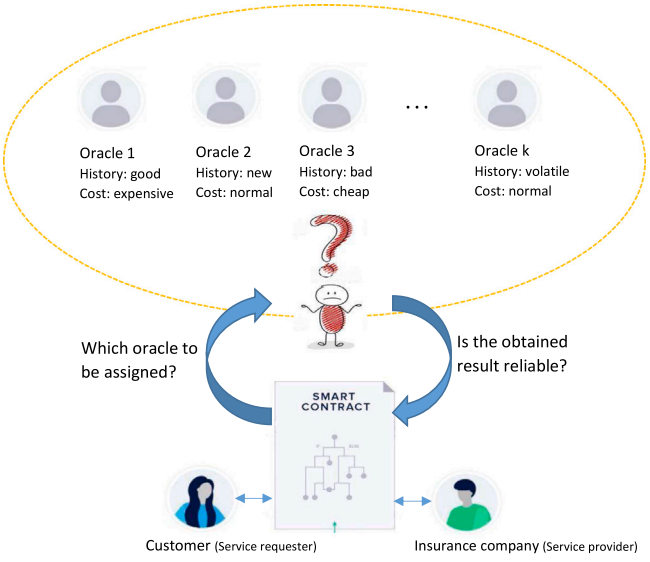


Fig. 1. Motivational scenario.

the blockchain network cannot provide such data. To transmit the data and estimate the situation, the smart contract needs to employ an oracle. Since the oracle determines what the smart contract sees, it is crucial to employ not only an economical oracle, but also a reliable and trustworthy one. If the oracle is malicious, it may report in favor of the insurance company and the smart contract accepts the result blindly. Besides, the smart contract cannot rely solely on the historic behavior and reliability of the oracle, since it might decide to deviate for any reason. Furthermore, usually the allocated budget for these tasks are limited. It gets more complicated when some oracles are new with little or no history offering low prices. The smart contract has to find an optimal choice with a balance between reliability and price that all the blockchain validators agree on the obtained result.

Oracles, like human subjects and computer applications, are susceptible to bad behavior that can manifest in gamified attacks in blockchains. In its most basic form, a centralized oracle can supply misleading data, which can impact the actions of blockchain nodes in ways that make them susceptible to attacks. In some situations, the incentives to submit non-truthful data may surpass the gains of acting truthfully. It is argued that a high decentralization of the oracle model would lead to less vulnerability to the "Oracle Paradox" (Ellis, Juels, & Nazarov, 2017). No matter how centrally centralized it is, an oracle will always come with a price. The most profitable strategy should be always acting honestly, which is why strong incentives must be placed. This raises the need to investigate an incentive mechanism that can motivate oracles to behave honestly. In this paper, we argue that assigning a reputation value to these oracles and make them subject to tests against other oracles make the option of reporting misleading data not profitable.

3. Background and related work

The literature review is summarized from three different areas: blockchain, multi-armed bandit, and crowdsensing. As the blockchain oracle selection is somehow neglected in the literature, we were not able to find a proper related work and compare different methods of a third-party selection in a blockchain environment. Therefore, the most similar approach, that is "worker selection in a blockchain-based crowdsensing" is reviewed in this section.

3.1. Background on blockchain

Blockchain is a distributed database system built upon a timestamped list of transaction records. Its main innovation lies in allowing parties to transact with untrusted parties using a computer network (Mendling et al., 2018). The blockchain data structure is a hierarchy of blocks that aggregates transactions. Each block is uniquely identifiable and linked to its predecessor in the chain, and integrity is ensured using cryptography-based techniques.

Nodes in a blockchain network might perform arbitrary or malicious behaviors, or possessing misinformation. So, consensus mechanisms are the core of blockchain networks to ensure that all participants agree on the state of the network in such trustless environments (Bouraga, 2021). The most important consensus algorithms for blockchains are Proof of Work (PoW) and Proof of Stake (PoS). In blockchains using PoW (e.g. Bitcoin), the algorithm rewards participants for solving cryptography-based puzzles to validate transactions and build new blocks. In PoS-based blockchains (e.g. Ethereum's upcoming Casper implementation), validators take turns to propose and vote for the next block. The weight of validators vote depends on their stake or deposit on the network. PoS provides enhanced scalability, fast transactions, low computation and energy consumption, and high security.

Blockchain was originally designed to operate as a trustless peer-to-peer network for financial transactions. Since then, the technology has grown to include many other applications including smart contracts. A smart contract lives on the blockchain and has its own unique address. Moreover, smart contract technology allows users to create autonomous applications that operate independently without any intervention from a system entity. While BLOR can theoretically support any smart contract with low computation surcharge, this paper focuses on its use in Ethereum for implementation due to its publicly accessible platforms. Ethereum initially developed its platform based on PoW, but recently is performing a significant upgrade to presents Ethereum 2.0, using the Casper protocol (Buterin & Griffith, 2017). The Casper protocol eases the transition from the current PoW to a pure PoS protocol. Ethereum's cryptocurrency is called Ether. In the current version, Ethereum functions through gas which is an Ether-based purchase of the consumed resources. This will help Ethereum prevent DoS attacks, infinite loops within contracts, and in general control network resource expenditure. Every function, such as sending and retrieving data, executing computation, and storing data, has a gas cost.

Smart contracts have two types: deterministic and non-deterministic (Morabito, 2017). The deterministic smart contract code is implemented on a blockchain with complete isolation of external environments, and the decisions and the contract states are maintained by participants within the blockchain. By contrast, the nondeterministic smart contract code needs external information to make decisions, making it dependent on actors outside the blockchain network. For example, the external actor could be a weather information provider or a sensor data provider, who are known as oracles in blockchain.

Blockchains have revolutionized corporate governance since they allow higher transparency among all participants and easy administration, along with the provision of innovative infrastructures for sharing business transactions in real-time. Although blockchain has a great potential to help businesses share data and collaborate in a secure way, very little research has been done on real-world applications (Yermack, 2017). Tractica, a market research firm, estimates that enterprise applications of blockchain will reach $19.9 billion in revenue by 2025 (Jiao, Wang, Niyato, & Xiong, 2017). Most of research about blockchain's application have centered on economics (Underwood, 2016), energy (Münsing, Mather, & Moura, 2017) and IoT applications (Zhang & Wen, 2017).

3.2. Background on multi-armed bandit

Reinforcement learning is one of the most popular machine learning techniques that is inspired by behavioral psychology of a biological agent. The idea is that an intelligent agent learns the outcome of its actions by interacting with the environment in which it optimizes its actions based on the accumulated rewards it receives (Liu, Zhu, Jiang, Ye, & Zhao, 2022). Multi-armed bandit is a classic reinforcement learning problem. Its name comes from slot machines in casinos, where a gambler is in front of a row of slot machines and he should decide which machine, how many times, and in which order to play the slot machines in order to maximize his potential prize. In this context, each machine gives a stochastic reward from a probability distribution. The gambler's objective is to maximize the total of rewards earned by pulling the sequence of levers at slot machines. Multi-armed bandit problem is exploited in many fields such as medical (Villar, Bowden, & Wason, 2015), recommender systems (Li, 2010), and crowdsourcing (Jain, Narayanaswamy, & Narahari, 2014). But, its application in blockchain along with its specific challenges has not been explored in any research yet to the best of our knowledge.

Only a few initiatives utilizing reinforcement learning for blockchain and smart contracts have been undertaken in academia. Among them, one focused its attention on energy-efficient resource management problem in cloud data centers and developed a robust blockchain-based resource management framework to boost energy efficiency (Xu, Wang, & Guo, 2017). This research uses a reinforcement learning model embedded in a smart contract to mitigate the cost of energy production. The authors demonstrated their method based on Google cluster tracing and electricity prices, which allowed them to cut the data centers' cost greatly.

Multi-armed bandit is very accurate with minimal complexity and required computing resources comparing to other reinforcement learning methods. These specifications made it a very optimal solution to be employed on blockchain and smart contract platforms where computation and storage are very precious.

3.3. Related work: Blockchain oracles

The way an oracle retrieves its data depends on whether it relies on human involvement or functions completely automated. Automated oracles operates solely through software and hardware by accessing a data source and retrieving the required data. This means that the oracle itself is fetching the data and is not the original source of the data. Automated oracles only provide deterministic inquiry results as they retrieve existing information from a data source. However, this is not the case with autonomous oracles or oracles involving human intervention. These oracles are not only able to transmit deterministic data, but also to respond to arbitrary inquiries which could be hard to be deducted by machine.

Oracle systems can be centralized or decentralized. Oraclize (now is called Provable)[2] is a centralized oracle service based on Amazon Web Service that provides data feedback for smart contracts and blockchain applications. The main attention of Oraclize is given on proving that the obtained data from its original source is genuine and untampered. Town Crier (Zhang et al., 2016) is also a centralized authenticated data feed that operates as a trusted bridge between existing HTTPS-enabled websites and Ethereum. In fact, it uses trusted hardware and software to be able to prove that the tasks are performed with no tamper and results are reliable. However, similar to any other centralized solution, its validity relies on a central authority and there is no guarantee if the task is performed correctly. It also pays attention to bring data to smart contracts in a trustworthy way, but the data resource is questioned. Chainlink (Ellis et al., 2017) is a decentralized oracle network on the Ethereum platform. It originally aims to provide tamper-proof data for smart contracts through accessing key data resources using designated APIs. Chainlink operates through incentives and aggregation models, however, it has cost and scalability issues. In another attempt, the authors in Ma, Kaneko, Sharma, and Sakurai (2019) proposed a decentralized oracle system equipped with verification and disputation mechanisms. ASTRAEA (Berryhill & Veneris, 2019), is an interesting decentralized oracle working based on a voting game to decide about the truthfulness of propositions. All voters place some amount of stake to have the opportunity to vote on a selected randomly proposition. The authors analyzed the game-theoretical incentive structure to prove the existence of Nash equilibrium under the assumption of honesty.

In a nutshell, the literature on blockchain oracles is limited to only automated oracle solutions and the trust issues surrounding the transmission of information to smart contracts and neglects the trust and quality issues concerning the data source itself. Autonomous oracles and human intervention-based oracles cannot be distinctly separated from the data source, and to the best of our knowledge, there is no proposal addressing the issue of trustworthiness for these types of oracles that is being investigated by this paper. Besides, optimization of the cost of individual oracles, which is one of our objectives in this paper, has not been tackled in the literature at the time of writing this article.

3.4. Related work: Worker selection in a blockchain-based crowdsensing

Worker selection is the crucial component for crowdsensing to control the quality of sensing services. The literature concentrates on selecting highly reliable workers to maximize the sensing data quality in two methods centralized and decentralized. SenseChain (Kadadha, Otrok, Mizouni, Singh, & Ouali, 2020) is a decentralized crowdsensing framework for multiple requesters with multiple workers that is deployed on Ethereum. In SenseChain, the quality information of the tasks is evaluated by integrating three independent metrics of reputation value, time, and distance. Chatzopoulos et al. concentrated on the influence of the opinions of different requesters on worker selection (Chatzopoulos, Ahmadi, Kosta, & Hui, 2017). A beta distribution is used to assess the probability distribution over the ability of workers to return results in time. Furthermore, it allows different requesters to exchange their opinions about the workers to improve their knowledge about workers as much as possible to make the most appropriate selection, and then update the reputation of workers based on the task completion results.

Zhao et al. developed a blockchain-based mobile crowdsensing system and then presented a privacy-preserving reputation management scheme in order to defend against malicious workers (Zhao, Tang, Zhao, & Wu, 2019). It can ensure data privacy during reputation computation based on additive secret sharing. A blockchain-based crowdsourcing platform with a robust reputation management scheme is proposed in Bhatia, Gupta, Dubey, and Kumaraguru (2020). To have a reliable worker selection by the task requester, the worker reputation values are calculated based on the task data evaluation results from other equivalent participants on the platform. It placed an attractive incentive mechanism to encourage workers who are required to have the corresponding computing skills and conditions to actively participate in the task evaluation.

Ding et al. paid major attention to improving the sensing services quality and ensuring the fairness of task data evaluation (Ding, Chen, Lin, & Tang, 2019). They proposed a reputation mechanism and an arbitration mechanism for worker selection and data evaluator selection respectively. Chatzopoulos et al. identified Internet service providers to complete location-based task recommendation (Chatzopoulos, Gujar, Faltings, & Hui, 2018). It designed a cost-optimal auction to select workers, thereby minimizing the cost of crowdsensing providers. TrustWorker (Gao, Chen, Zhu, Dong, & Ma, 2021) is another attempt to form a trustworthy and privacy-preserving worker selection scheme

[2] https://provable.xyz/

for blockchain-based crowdsensing. TrustWorker dynamically updates workers' reputations by evaluating data quality and adopts these values to evaluate the reliability of workers.

In the domain of mobile edge computing, Xu et al. proposed a trustless crowd-intelligence ecosystem based on the common decentralization feature of mobile edge computing and blockchain technology (Xu, Wang, Bhargava, & Yang, 2019). It uses a reward penalty model to align the incentives of stakeholders and the predefined rules through smart contracts running on many edge servers of the mobile edge computing network. Machine learning was practiced off-chain to improve the performance of mobile offloading in Xiao et al. (2020). The authors proposed a blockchain-based mobile edge computing trust mechanism to improve the computational performance of the mobile devices and suppress the attack motivation of selfish edge devices. A reputation assignment method is developed to choose the miner and a reinforcement learning approach is utilized to develop an algorithm for edge CPU allocation to optimize the number of CPUs for the offloading task.

To summarize, throughout the literature on worker selection in crowdsensing, cost and reputation management have been fully addressed as workers are mainly known as autonomous or human agents. However, to the best of our knowledge, it has not considered on-chain machine learning algorithms to optimize the selection of the workers considering the factors of trust, quality, and cost. This paper, which is inspired by the reputation management methods presented in this section, is going to address these issues using reinforcement learning.

4. BLOR: A Markovian multi-armed bandit-based solution

The main concern of a blockchain-based system, which requires obtaining data from the outside world, is how to maximize total rewards from various oracles in an uncertain setting through trial and observation. BLOR provides an optimal solution using Bayesian theorem and reinforcement learning techniques. In the process of BLOR's sequential decision to choose a proper, reliable, and cost-efficient oracle, two components have to be considered:

1. Learning: BLOR utilizes observations to update its understanding and knowledge regarding the reliability of oracles.
2. Choice: BLOR selects an action that has a proper balance between the immediate reward of oracles (short-term objective) and the increased knowledge of oracles (long-term objective).

For the learning component, BLOR uses a novel Bayesian cost-dependent reputation model (BCRM), and for the control component it uses Knowledge Gradient (KG) (Frazier, Powell, & Dayanik, 2009). BLOR is responsible to manage trust establishment among the blockchain participants and oracles by assigning tasks to the best performing oracles. A digitally assigned reputation value is an effective factor that can be used by BLOR to recognize the premier oracles. To assess and model reputation, we require information and evidence about the history of the evaluated oracle. However, solely employing oracles with good history implies loosing significant opportunities of stranger oracles that BLOR has never encountered before. Furthermore, these oracles tend to be more costly, which can lead to an exceeding budget. Thus, it is crucial to combine the cost factor with the reputation value while assigning a task to an oracle. BCRM assumes that reward rates can change during the experiment depending on the reputation state of the oracle. There are just two possible actions for the choice component: 0 (freeze) which produces no reward nor state change, and 1 (continue) with reward θ_k^t and reputation state changes, according to Markov dynamics.

KG prefers the actions that inspects the choices with little information. Exploration is an endeavor to gain knowledge with a minimal use of precious time and computing resources. Pure exploration can waste time and computing resources if it searches irrelevant areas of the environment. This also means that the agent's learning efficiency may be poor, since it is wasting time on actions that do not contribute to its goals. Finding a good balance between exploration and exploitation is highly beneficial. The agent may be able to discover the most worthy areas to explore by exploiting its current knowledge of the environment. Furthermore, optimizing the cost of learning (i.e., making the agent's performance during learning as high as possible) cannot be achieved without some level of exploration of the environment, which is important in identifying efficient behaviors.

4.1. BLOR framework

Fig. 2 illustrates the workflow of BLOR that aims to select the most optimal oracles in terms of reliability and cost. Majority of this process happens on-chain so that the blockchain's validators can verify and achieve a consensus to avoid any bias. Let us consider the insurance company as a service provider and the customer as a service requester, trading through a smart contract. When a claim request is triggered by the service requester through the smart contract, BLOR will decide which oracles will perform the task. At first, once the smart contract receives the request, it automatically creates a new contract with the network of oracles who are the other beneficiary party. This contract contains a new set of rules and conditions such as the payments and compensations policies.

In order to understand the oracles behavior and learn their rewards (in terms of reliability and cost), BLOR creates a BCRM. However, since BCRM is a Bayesian model and probability is involved, we need to generate a random number from the prior distribution in each trial. Generating a random number by each node of the blockchain is a challenge as each node could come up with a different number, around which making a consensus is impossible. Therefore, we propose a Random Number Generator (RNG) with participation of all the validators' nodes. Thus, in step 2, a RNG contract will be created to issue an agreeable random number which will be elaborated in Section 4.3. To select the most rewarding oracle, the KG algorithm is used thanks to its high performance and fast computation that makes it suitable for the blockchain environment. In step 3, KG uses the generated reputation state of each oracle to calculate its degree (D).

The KG algorithm selects the most rewarding oracles over a period of time, so at each request, there could be some chosen oracles with unknown or low rewards. For this reason, the chosen oracle has to be checked if additional oracles have to be hired. Depending on the strategies and objectives of the blockchain and its validators, as well as the sensitivity of the task being outsourced to the oracle, the smart contract has to decide if the probability of truthfully and successfully performing the task is high enough for the chosen oracle in step 4. If this probability is low, the next two oracles with highest degree of knowledge (obtained from KG) shall be selected to ensure a reliable result (step 5: No). The intuition behind selecting three in total is to evaluate the trustworthiness of the results based on the majority vote and then rate each oracle accordingly. However, more than three oracles is not economically justified since each of them charges the smart contract to perform its task. If only one oracle is chosen, we can have some random tests by using other oracles from time to time to use for training our model (step 5: Yes). These random tests ensure that even trusted oracles do not behave maliciously and the learning is processed without deviation. The nominated oracles will be called by smart contract triggers within step 6 and the results will be reported by sending a signed transaction to BLOR in step 7. BLOR verifies the results, updates the reputation values of the participated oracles, then pays these oracles according to the defined rules and conditions in step 8, and informs the requester about the result (step 9). In the last step (10), the updated posterior reputation including the success or failure of the oracle will be sent to BCRM.

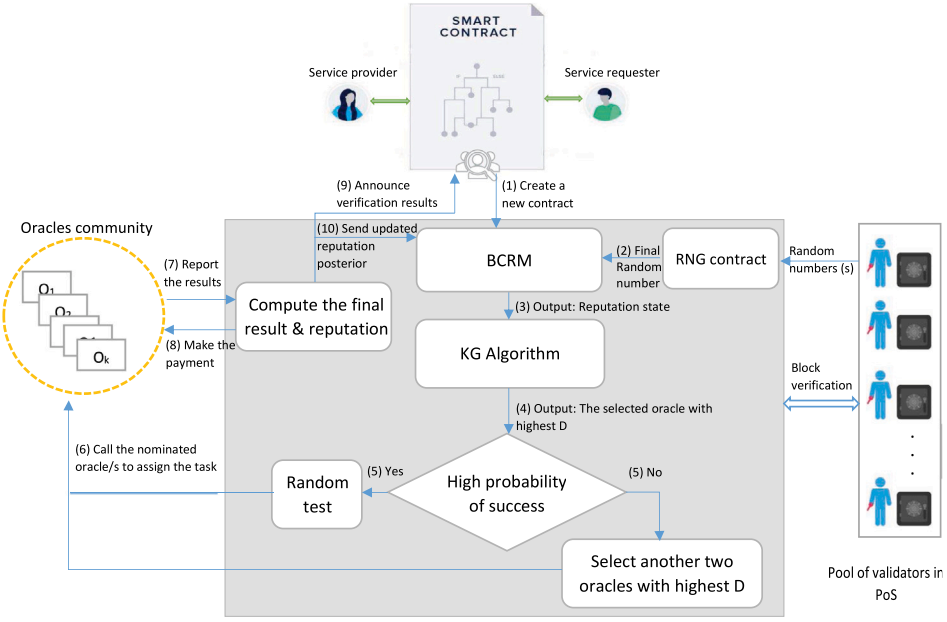


Fig. 2. BLOR framework.

4.2. Formulating oracles problem in a bandit setting

Suppose on each data request, we have K oracles known with reward rates, θ_k, $k = 1, \ldots, K$. At first, we assume θ_k to be the true reward mean if oracle k is to be chosen. We do not know θ_k, but we assume that it is normally distributed with prior mean θ_k^0, variance $(\sigma_k^0)^2$ and precision $\beta_k^0 = 1/(\sigma_k^0)^2$. Let $R^t = (\theta_k^t, \beta_k^t)$ be the vector of reputation states with the means and precisions for all the choices of oracles after t trials.

Let k^t be the oracle that we choose after t trials, meaning that our first choice is k^0 made based on the prior, purely. These trials are made based on a policy π to be run by smart contract which depends on the history of trials. Policy is a decision rule that BLOR adopts on behalf of all the blockchain participant to assign tasks to oracles. Let us assume that K_k^π is the random variable representing the total number of selecting oracle k, given the policy π. This number is random since the results depend on the observed rewards. Our objective is to choose a policy π that solves the following supremum objective function $supV$ where R_k is the reputation state of the oracle k:

$$supV^\pi = \mathbb{E}^\pi \sum_{k=1}^{K} K_k^\pi R_k \theta_k \tag{1}$$

$\mathbb{E}^\pi$ stands for the expected value depending on π to reflect the underlying probability space that we are going to construct. Learning problems can be easily formulated in a Bayesian framework, where we are able to capture the uncertainty in our belief about a system. In our oracle bandit problem, θ_k is the true rewards value of oracle k, but we do not know this value. Instead, we assign a probability distribution from the Beta distribution that describes what we think θ_k is for each oracle. Since each oracle can have two outcomes of success or failure, we employ the Beta distribution where trials are generated independently and identically from an unknown Bernoulli distribution for each oracle. The following section explains how we construct our Bayesian model for oracles.

4.3. Bayesian cost-dependent reputation model

We formulate two components of Bayesian learning as follow: (1) Bayesian Inference: to update the reputation representing the belief about the probability of a successful and truthful evaluation (reward) based on new information; and (2) Bayesian Learning: to compute the posterior probability distribution of the target features.

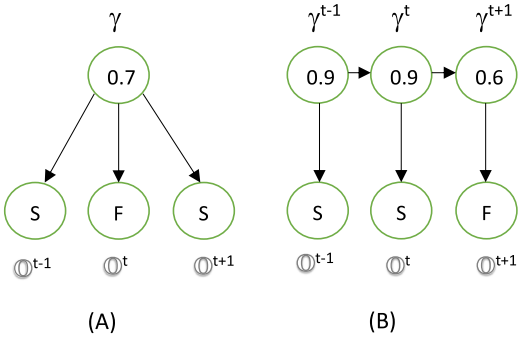


Fig. 3. Graphical model of (A) Fixed model vs. (B) Dynamic model. The circled numbers present example variables' values. S denotes the oracle success and F denotes the oracle failure to deliver reliable results with a fixed probability of γ or dynamic probability of γ^t.

Usually bandit solutions using Bayesian learning assume that there is a fixed probability of $\gamma \in [0, 1]$ for the experiment repeated on any given trial t. Then, the appropriate value of γ shall be learned over the time period of experiment. This approach is naive to solve the problem of oracles, since oracles might change their behavior and deviate in a certain point of time for certain cases. For this reason, we adopt a dynamic model for reputation state in which γ^t has a Markovian dependency on γ^{t-1}. A graphical illustration of these two models is presented in Fig. 3.

Under the dynamic reputation model, the reward probabilities might change at times during the experimental session, as each oracle is an autonomous agent in our problem and might change its behavior. Thus, during any trial, the prior reputation of each oracle combines a generic prior and the posterior reputation from the previous trial. The main purpose of BLOR is to monitor the reward probability of each oracle during the trial period. The prior distributions that produce the Bernoulli rates are assumed to be Beta distributions, Beta (a, b), whose two hyper-parameters, a and b, identify the pseudo-counts referred to by the prior. The beta distribution we use in our model requires a pseudo-random number to operate on different cycles. Smart contracts and blockchains cannot generate a pseudo-random number, so it

should be resolved by either developing a deterministic pseudo-random number generator or using a trusted oracle.

4.3.1. Random number generator

Let n be the number of validators. In BLOR, the random number is generated by all the n validators to enable the final verification of results. First, we need to create a Random Number Generator (RNG) contract in BLOR, which defines the participation rules and computes the final random number. The basic process of generating a random number can be divided into two phases:

1. Any validator who wants to participate in the random number generation and the final candidate verification needs to send a secret number ($s_i \in \text{Beta}(a,b), 1 \le i \le n$) encrypted by Keccak-256 hash algorithm through a transaction to the RNG contract in a specified time period (e.g, 3 blocks period). The RNG contract will check if s is valid by running Keccak-256 against s. Valid s is kept to calculate the final random number.
2. After collecting all the secret numbers, the RNG contract calculates the random number from the function $f(s)$ and the final random number will be sent to BLOR and all the validators, where s_{min} and s_{max} are the minimum and maximum numbers respectively:

$$f(s) = \frac{\sum_{i=1}^{n} s_i - s_{min}}{s_{max} - s_{min}} \tag{2}$$

4.3.2. Oracles reputation model

Let S_k^t and F_k^t stand for the success and failure rates of the kth oracle after t trials, and let θ_k^t be the approximate payoff probability of oracle k at trial t. θ_k^t has a Markovian dependency on θ_k^{t-1}, so that with probability γ, $\theta_k^t = \theta_k^{t-1}$. Also with probability $1-\gamma$, θ_k^t is pulled from the prior distribution Beta(a,b). To infer the new posterior distributions, BLOR combines the sequentially generated prior reputations with incoming observations (successes and failures on each oracle). The observation $\mathbb{O}_k^t$ is assumed to be Bernoulli(θ_k^t).

In order to maximize the utility of the blockchain's participants, BLOR needs to select the oracles that perform their tasks correctly at a lower price. Let c_t be the normalized cost that oracle k charges the blockchain with a weight of w to adjust the value of cost to the chain participants. We denote $R(\theta_k^t)$ as reputation state which is the posterior distribution of θ_k^t given the observed sequence. At each trial, the updated cost-based reputation state can be computed using Bayes' rule as follows:

$$R(\theta_k^t) \sim P(\mathbb{O}_k^t \mid \theta_k^t) P(\theta_k^t \mid S_k^{t-1}, F_k^{t-1}) / w c_t \tag{3}$$

The prior probability of reward state is the sum of the posterior of the last trial with a weigh of γ, plus the generic prior $R^0 \equiv f(s)$, as defined below:

$$P(\theta_k^t = \theta \mid S_k^{t-1}, F_k^{t-1}) = \gamma R_k^{t-1}(\theta) + (1-\gamma) R^0(\theta) \tag{4}$$

4.4. BLOR's final decision based on knowledge gradient

In reinforcement learning methods, the entire reward is generally received after the final measurement, which is impractical for our problem in blockchain. Thus, we need to receive the reward given in pieces over time. This will not only decrease the complexity of the solution and computational resources, but also will shorten the results time to identify the best oracles from the beginning of the process in an online manner. The KG policy can achieve this by maximizing the single period reward.

The objective given by Eq. (1), Eq. (3) has a terminal reward, $V^{T,\pi}(R^T) := \max \theta_k^t$. However, we require to restructure it in order to provide single period reward $V^{T,\pi}(R^t)$ at trial t, and $V^{T,\pi}(R^{t+1}) - V^{T,\pi}(R^t)$ at times $t+1, \dots, T$, meaning that:

$$\max \theta_k^t = [V^{T,\pi}(R^T) - V^{T,\pi}(R^{T-1})] + \cdots + [V^{T,\pi}(R^{t+1}) - V^{T,\pi}(R^t)] + V^{T,\pi}(R^t)$$

KG is defined as a single-step and look-ahead policy, which selects the next instance with the largest expected reward, greedily. Its algorithm is close to the optimal policy. It pretends only one more exploration is allowed and assumes that after the next measurement, all of the remaining options will exploit what is already known. It calculates the anticipated change in each oracle's estimated reward rate, according to the current reputation state R_k^t. The value function of selecting oracle k, $D^t = k$, on trial t is:

$$V_k^t = \mathbb{E}[\max \theta_k^{t+1} \mid D^t = k, R^t] - \max \theta_k^t \tag{5}$$

The first term indicates the projected highest reward rate on the following step if the kth oracle were elected, assuming all possible outcomes of making that choice were taken into account. The second term presents the expected greatest reward in the absence of extra exploitative choices. Their difference is the "knowledge gradient" of discovering an additional exploratory sample.

Let us imagine we have T trials of which $(t-1)$ measurements were already made. For the tth measurement, the KG decision rule is defined as follows:

$$D^t = \arg\max \theta_k^t + (T-t-1) V_k^t \tag{6}$$

Other than very minimal computational resources that validators require to compute KG, it accommodates the issue of cold start for oracles who join the network later on. It means that the KG policy measures those alternatives that has less knowledge about. The predictive distributions of these alternative oracles (k') have large variance $(\sigma_{k'}^t)^2 > (\sigma_k^t)^2$, or equivalently, small precision $\beta_{k'}^t < \beta_k^t$.

4.5. An illustrative example

In this section, we provide an illustrative example to show how BLOR works in details. At first, BLOR shall construct BCRM. Assume that we have 5 oracles, $K = 5$, $O1, \dots, O5$. For each one of them, we shall calculate the reputation state. Fig. 4 illustrates how we form that for each oracle. Consider $O1$ is measured for the first trial and returns a success. In the beginning, we have no knowledge of its performance, therefore its prior for the next step $(S1)$ is equally distributed in the Beta setting. Since computation of prior includes randomness, according to Fig. 2, BLOR calls the node in charge of generating random numbers to have the same random number for all the validators. The summation of the prior with the random number is multiplied by the Bernoulli trial to obtain the posterior. This posterior will be used as the prior for the next step $(S2)$. During the next trial $(S3)$, $O1$ returns a failure which negatively affects the posterior distribution.

After creating BCRM, BLOR seeks to find out the best oracles for the task by computing and comparing their KG degrees. Table 1 presents these calculations. Let us assume that the total number of trials is 500 and BLOR is exploring its 65th trial, $(T = 500, t = 65)$. We further assume that at $t = 64$, $O1$ is selected and its number of successes became 20. BLOR must compute the expected reward rate, value function and KG decision using Eq. (3), Eq. (5), and Eq. (6). Consider the two oracles $O1$ and $O4$, which have the highest numbers of successes according to Table 1, with a reputation state of 0.66 obtained from the previous trial.

1- Choosing $O1$:

$$\begin{cases} O1\ wins \xrightarrow{(P=0.67)} & R_1^{65} = 0.68/0.8 = 0.85 \\ O1\ loses \xrightarrow{(P=0.33)} & R_1^{65} = 0.65/0.8 = 0.81 \end{cases}$$

Therefore, we have:

$$V_1^{65} = (0.67 * 0.85 + 0.33 * 0.81) - 0.66 = 0.16$$
$$D^{65} = 0.85 + (436 * 0.16) = 70.61$$

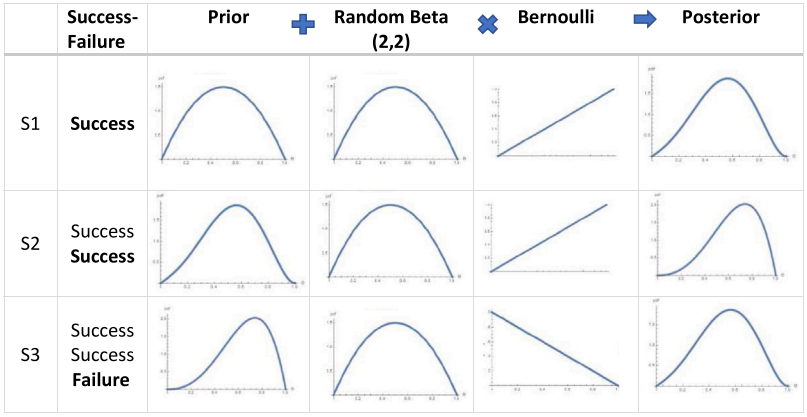


Fig. 4. Illustration of partial rewards state of $O1$ within the constructed BCRM.

Table 1
BLOR in the illustrative example ($t = 65$).

$T = 500$	$O1$	$O2$	$O3$	$O4$	$O5$
Success	20	5	0	10	5
Failure	5	10	5	0	5
Cost	0.8	0.5	0.5	0.7	0.6
$R(\theta_k^t)$	0.85	0.82	0.7	0.97	0.83
V_k^t	0.16	0	0	0.95	0

1- Choosing $O4$:

$$\begin{cases} O4\ wins \xrightarrow{(P=0.65)} & R_4^{65} = 0.68/0.7 = 0.97 \\ O4\ loses \xrightarrow{(P=0.35)} & R_4^{65} = 0.66/0.7 = 0.94 \end{cases}$$

Therefore, we have:

$V_4^{65} = (0.65 * 0.97 + 0.35 * 0.94) - 0.66 = 0.95$

$D^{65} = 0.97 + (436 * 0.95) = 415.17$

Thus, among $O1$ and $O4$, $O4$ with the highest degree of KG will be selected. BLOR chooses $O4$ rather than $O1$ based on its lower cost, higher reputation and lower number of experience. Please note that to save computation time, BLOR does not calculate value function and KG for the oracles with very low chance of being selected.

The algorithm of BLOR will be explained in the next section in a case study where we construct smart contracts and the relationship among them.

4.6. Integration of BLOR in an Ethereum-based dapp

When it comes to the long-term success of utilizing oracles in Dapps, efficient integration and reliable data flow are essential. In the following steps, we explain an architectural perspective on how BLOR can fit within a generic Ethereum-based Dapp.

1. Dapp would call BLOR to register its request for oracles on-chain.
2. In order for Ethereum Dapp users to interact with BLOR, a minimum of one smart contract, such as a user contract (UC), is necessary. UC holds a list of all oracles and their past history. Each oracle announces its cost to perform the task c_k.
3. UC implements BLOR to find qualified oracles to monitor the SLA. UC runs the designed models and formulas through a set of defined functions. BLOR selects at last three oracles and assigns them tasks by registering events on-chain.
4. Selected oracles will be triggered to start the tasks off-chain. At this stage, the proper money shall be deposited into UC.
5. Once the oracle's job is done, it reports the outcome back to be registered on the blockchain.
6. Once all oracles reported their results, BLOR would make a decision on the outcome and rank the oracles based on the consensus. The final outcome would be used to train the defined models in BLOR.
7. BLOR passes the final result on-chain and it is accessible by Dapp.

To develop secure and robust blockchain-based Dapps, the oracle should support data availability and data integrity. BLOR can efficiently achieve data availability by having multiple oracle servers preventing a single point of failure problem. Moreover, using the past performance and reputation system, BLOR tries to ensure that the external data is safely received and is not tampered with before transferring to the blockchain.

Updating the model of an already deployed smart contract is very difficult as Ethereum smart contracts are, by design, immutable once deployed. This could be a major challenge in integrating BLOR with an existing smart contract model. The path to be approached for the integration of BLOR is as follows. The deployed smart contract (contract A) can be a pointer to another smart contract (contract B) that implements the actual functionality. If the functionality needs changing, A's reference to contract B gets updated to point to a replacement (contract C that contains BLOR).

Depending on the platform in which BLOR is employed, the transaction throughput could differ which leads to a scalability challenge. For example, Ethereum supports roughly 30 transactions per second. Smart contracts are usually designed in a minimalist manner to have less computational cost. However, since BLOR is implementing a machine learning approach, it costs more than the usual cost of smart contracts to be executed.

5. A case study of cloudchain (cloud services trading over blockchain)

The aim of the Cloudchain case study is to present how BLOR can offer a unique smart model for employment of oracles and transform the way cloud services are delivered. Cloudchain (Taghavi, Bentahar, Otrok, & Bakhtiyari, 2018) is a blockchain-based platform designed to allow cloud providers to interact, co-operate and compete through outsourcing their pending or unmet computing demands.

With the help of smart contracts, Cloudchain is able to provide higher transparency, visibility, and reliance in its decentralized arrangements deployed over Ethereum. Cloudchain fails to enforce the SLA's requirements, which require accessing the outside world of the blockchain network. There may be disagreements about SLA compliance between cloud providers. However, the blockchain's self-contained execution environment makes it impossible to investigate the behavior exhibited by the digital codes embedded in smart contracts. Oracle is

thus tasked with performing the valuable verification task and will confirm if the SLA is met. Many researchers assume oracle to be a fully-trusted third-party application that has access to external data and feeds it into the blockchain to be accessible by its applications (Taghavi, Bentahar, Otrok, & Bakhtiyari, 2020). Furthermore, oracle is assumed to be single or act as a member of a group, while in real world, each oracle could be a selfish agent trying to maximize its own gain. We explain how BLOR can contribute to this situation.

5.1. Background: Smart contracts of cloudchain

Three kinds of smart contracts are incorporated into Cloudchain that include executable functions and state variables. Taghavi et al. (2018).

Cloudchain Registry (CCR), *Contract 1*, is a global contract mapping Ethereum addresses (equivalent to public keys) to cloud providers identification values (including *Name*, *Reputation Value*, *Computing Capacity* and *Storage Capacity*). A contract can include policies governing the registration of new providers or changes to the mappings of existing ones. Only certified cloud providers can register for the cloud provider program. In addition, CCR maps identities to Cloudchain Contract (CCC) addresses, where an exclusive contract concerning the profile of each provider and the list of services is saved.

Cloudchain Profile (CCP), *Contract 2*, contains a list of references to CCC, which depict all the participants' past and present interactions with each other. CCP uses a feature to generate provider notifications. Ethereum enables the creation of events to indicate that certain actions have been performed (e.g., a change to the data of profile). Providers propagate and raise their requests to other nodes by transmitting them to the CCP contract first. Transactions are tracked by status variables. The status of a transaction indicates if it is new, waiting for an update, or if it is completed. This contract is crucial because it stores addresses of new CCC contracts, and without it Cloudchain could lose track of all those contracts.

Cloudchain Contract (CCC), *Contract 3*, is generated among two nodes when one agrees to provide the requested service to the other. Similarly, the contract can be completed or canceled by its beneficiaries. The remaining balance will be transferred once the contract is completed or canceled, and its status will also be updated. Cloudchain members are able to join and depart from the system at any time. These flexible memberships allow members to provide or demand services as many times as required.

5.2. BLOR in cloudchain

Fig. 5 illustrates interactions among the contracts and cloud providers. In step 1, The Cloud Provider as a requester (CP_r) and the Cloud Provider as a supplier (CP_s) should register in CCR. Public key pairs are assigned to each registered user in CCR.

In the case of a computational resource deficiency, CP_r can create a new CCC in step 2 by requesting a service using CCP. CCP ties identities to the CCC's address on the blockchain and keeps a history of providers past and present engagements with other nodes as well as any SLA violations.

CCC allows the interaction between two nodes in the network in which one node responds to the other's request in step 3. In order to complete a contract, CP_r has to pay a deposit in advance. Beneficiaries can choose to end the contract or cancel it. Yet, CP_r should confirm termination of the contract and delivery of the requested service. Once the contract is complete or canceled, CCC will calculate and charge fines if any exist. The balance remaining on the contract will be transferred to CP_r or CP_s accordingly. The contract status will be updated as well.

In step 4, CP_r can initiate quality monitoring at any time to verify whether the provider meets the SLA conditions during the runtime. To do so, the request shall be initiated through CCC in which a new contract of CloudChain Oracle (CCO) will be created to perform the verification in step 5. CCO holds a list of all oracles with their past history and their costs. CCO implements BLOR to find qualified oracle/s to monitor the SLA. CCO runs the designed models to nominate one or multiple oracles. Then the verification task is assigned and the proper money is deposited into CCO in step 6. In the last step (6), the obtained result is extracted to be push into CCC (step 7) and be used to train the defined models in BLOR.

Function calls within contracts are transactions, and those that change the contract storage have to be verified by blockchain validators. When a block containing the newly linked contract is mined, it will be broadcast to all the nodes, and the first node who accepts the request shall update the contract consequently.

All the explained procedure and interactions among smart contracts are elaborated in Algorithm 1 and 2. Algorithm 1 illustrates the process of requesting a service and triggering a quality monitoring request and Algorithm 2 presents the process of selecting the best oracle/s by BLOR. The complexity of Algorithm 2 increases linearly with the number n of oracles, so it has a time complexity of $O(n)$. Algorithm 1 depends on the number of requests m, and it also grows linearly with the number of oracles. Therefore, the computational complexity of Algorithm 1 is $O(n \times m)$.

Algorithm 1 Cloud providers service agreements within Cloudchain

Input: Ether deposit; Cloud requester's Ethereum address (CP_r); Cloud supplier's Ethereum address (CP_s); c_k; CCO Ethereum address.

procedure ServiceAgreement
  CP_r makes a service request in CCP
  CCP creates a CCC
  CP_r.SendTo(CCC, Ether deposit)
  CCC.Availability = True
  EventLog.Create("New request is available")
  while CP_r requests a quality verification **do**
    CCC calls BLOR in ***Algorithm 2*** ▷ Outsource the task to oracle/s to obtain the verification result
    CCC.SendTo(CCO, c_k) ▷ Pay the cost of the oracle c_k to perform the task
  end while
  if CCC.Completed **then**
    EventLog.Create("CCC is completed")
    ContractDeposit = CCC.TotalAmount
    CCC.SendTo(CP_s, ContractDeposit)
    EventLog.Create("Fund is transferred to the Cloud supplier")
  end if
end procedure

6. Experimental results

Because there is no available dataset about blockchains' oracles, in order to evaluate the performance of BLOR, we simulated 100 oracles operating within a blockchain in 1000 observations. We implemented and experimented with BLOR using Python on Google Colab and the Solidity language on Ethereum, the code is publicly available on Github.[3] Because a bandit is an online learner, it needs a record of the oracles history prior to the current time step we are simulating in order for it to act like in a production setting. Each oracle is assumed to have a different historical performance drawn from a beta distribution. The normalized cost of each oracle is assumed to be fixed and normally distributed with a mean of 0.54 and standard deviation of 0.17. We first discuss the challenges that we dealt with, and then provide the obtained results.

[3] https://github.com/kavehbc/Cloudchain

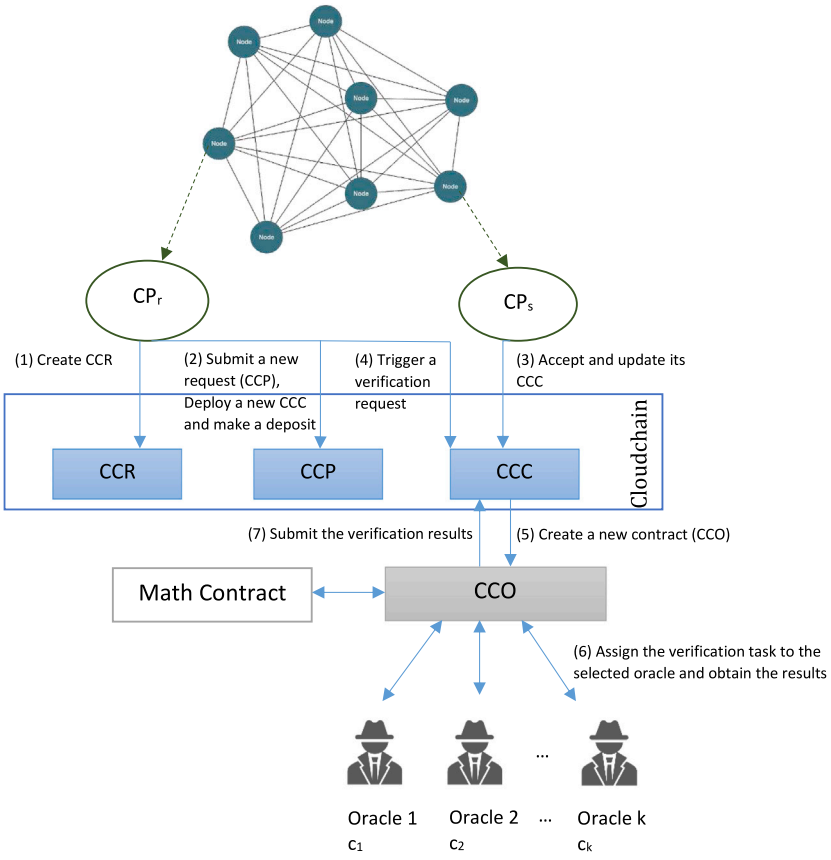


Fig. 5. Application of BLOR within Cloudchain.

Algorithm 2 BLOR process within CCO

Input: CCC Ethereum address; Oracles' Ethereum address; c_k; Ether deposit.
Output: Verification.Result ▷ Boolean

procedure OracleSelection
  CCC.Deposit(CCO, Ether deposit)
  Retrieve CCC's terms and conditions to monitor the service
  RNGContract.getRandom()
  MathContract.calculate($R(\theta_k^t)$) ▷ refer to Eqs. (3), (4)
  BLOR calculates D^t and selects the oracle with the highest D^t ▷ refer to Eqs. (5), (6)
  if probability of success is low **then**
    Select another two oracles with highest D^t
    if EachVerification.Result = True for majority of oracles **then**
      Verification.Result = True
    else if EachVerification.Result = False for majority of oracles **then**
      Verification.Result = False
    end if
  else
    Test randomly against oracles with high reputations
  end if
  CCO.SendTo(selected oracle/s, ContractDeposit)
  CCO.Update(Oracle/s reputation/s, Results)
  CCO.ReportTo(CCC contract, Verification.Results)
  Math contract receives the results and updates the posterior reputation of the selected oracle/s in BCRM
end procedure

6.1. Smart contract development limitations

The current version of Cloudchain is using Ethereum and it is not easy to run a machine learning algorithm on Ethereum that is a public blockchain with various limitations. Besides, at the time of writing this paper, Ethereum is still using PoW consensus and has not been upgraded to PoS yet. We came up with some solutions to test BLOR on the current platform of Ethereum.

- **Random number:** In blockchain, there is no pure random generator mechanism, because when the code is being run by other nodes, they all should reach the same result to achieve a consensus. There are some possible scenarios to generate a random number such as using a centralized system using an oracle, publicly verifiable secret sharing, or even hash-block. For the purpose of our simulation, we used a simple, yet efficient solution, which is using the block number to generate a hash number to be employed as a random number. This solution is practical and efficient since the block number is not known before being generated.
- **The limited number of variables:** Within Solidity EVM, there is no hard limit on the number of variables, but there is a variable limit of gas which is currently around 10 million gas. When there is a read operation from a single point variable (not a list or dictionary), it consumes 200 gas, and if there is a write operation, it consumes 20,000 gas. Based on the code complexity, we should limit the usage of variables since the gas limit can be exhausted quickly and it can become very expensive. Machine learning, artificial intelligence, and complex business logic algorithms can easily end up with high gas if the code is not properly monitored.
- **Smart Contract Size:** There is a limit of 24 KB for the Ethereum smart contracts or they will be out of gas. Smart Contract codes are very minimalistic in order to limit the gas required to make sure it can run faster, cheaper, and possibly with fewer logical bugs. This limitation can easily block the development of complex logic such as machine learning and reinforcement learning algorithms. In order to cope with this issue, we can make sure to use as fewer read/write operations as possible; separate logic to

Table 2
Probability of selection for the sample oracles.

Oracle ID	0	2	7	27	85	95
Cost	0.3	0.7	0.9	0.6	0.5	0.5
Success	4	5	9	6	3	1
Failure	5	0	1	3	4	3

read-only and write functions so that read-only functions can be called via Web 3 for free or at a small cost; and possibly spread the logic in multiple contracts if it is necessary.

- **Float number:** Blockchain does not support any float/decimal number with floating points. The reason is because all CPUs work based on a binary mechanism, and there is no exact representation of fractions in binary mode, so they are round to the nearest match. For this very important reason, blockchains do not support any number with floating point. Even for financial transactions, they have introduced smaller units such as wei, gwei, etc. instead of using float numbers. Basically, the only supported numerical data type in Ethereum is integer (either signed or unsigned). So, we require to scale up all the variables in integer level. For instance, if we want to take a number between 0 and 1, we have to change the scale to 0 and 100 to replicate the behavior of $0--1$ with one or two floating points precision. This would also impact parts of the algorithm, since the mathematical behavior of $0--1$ is different from $1--100$. So, the formulas need slight adjustments by scaling the values. For instance, if there are two float numbers, we need to scale them by multiplying them by 100. Also, we need to make sure that the scaling does not affect the outcome.
- **Advanced mathematical functions:** Blockchain languages do not support complex mathematical functions by default due to various issues such as the ones discussed earlier. In the BLOR algorithm, we used complex mathematical functions. Thus, in order to run BLOR on blockchain, we built a new contract called Math Contract. Math contract implements our required functions using four primitive operations only. This contract supports Sin, Cos, Log, exponential, Gamma function, square root (SQRT), Beta Distribution etc. All these functions are developed in Solidity solely based on integer numbers (with no floating point) and primitive operations.

It is worthy to mention that all these limitations only affect the implementation of the algorithms and had no impact on the evaluation phase. The evaluations performed a simulation of the oracles and the process of oracle selection to assess the performance of our developed reinforcement learning model (BLOR), which is mainly based on our computed formulas. During the implementation of those calculations and formulas, we had these limitations, therefore we had to come up with proper solutions to make it possible to be executed over the blockchain.

6.2. Simulation results

In order to assess the effect of performance and cost in BLOR decision making, we compared six oracles containing half faulty, during first 100 observations. Table 2 represents their detailed histories and costs. Fig. 6 presents the variation of value function for each oracle. At the beginning, oracles with shorter history, such as Oracle 95 and Oracle 2, provide more value in the learning process of BLOR. Consequently, they have a higher chance to be selected by BLOR, as can be seen in Fig. 7. After few observations, more value is earned with the oracles with cheaper price (such as Oracle 0 and Oracle 85). However, the chance to be selected is more among the oracles with a balance of price and performance (Oracle 27) and those that are very cheap (Oracle 0). The value function of all the oracles tend to zero after a while, when BLOR learned their behavior and there is no more value

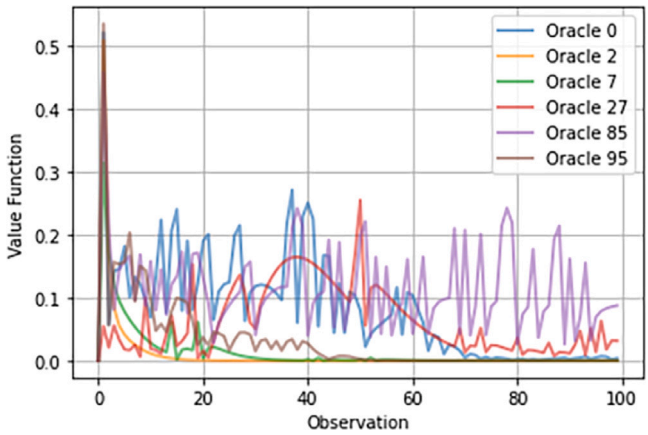


Fig. 6. Cost and history of the sample oracles.

Table 3
Algorithms comparison.

	Performance	Cost	Time (mean; STD)
BLOR	90	52	(0.04; 0.005)
Monte Carlo	74	58	(10.3; 1.07)
Random	62	54	(0.008; 0.0003)
ϵ-greedy	82	66	(0.03; 0.003)

in exploring them. Meanwhile, Oracle 85 generates an unsteady value, which means BLOR is willing to measure the change into the future expected reward of this oracle. This could be because of gaining more successful history combined with its good price.

6.2.1. Benchmarking

Our multi-armed bandit-based solution for the oracle selection problem can vary based on how we perform exploration and exploitation. We compare the performance of BLOR against other algorithms as follows:

- No exploitation: this is the most naive approach where the system selects randomly.
- Exploitation with exploration at random: ϵ-greedy is among the most popular and efficient methods of this group. The ϵ-greedy is a heuristic model that assumes decision-making is determined by a parameter ϵ to control the balance between random exploration and exploitation. With probability of ϵ, the oracle is chosen randomly, and $(1-\epsilon)$ the oracle with the greatest estimated reward rate will be selected.
- Exploration smartly with preference to uncertainty: BLOR is in this category.

We further investigate the performance of Markov chain Monte Carlo in our problem. Markov chain Monte Carlo is a probabilistic machine learning method that creates samples from a continuous random variable. The experiments are conducted using Python in Google Colab with Intel(R) Xeon(R) CPU @ 2.00 GHz and 13 GB RAM. Table 3 presents the average performance, cost and elapsed time of each algorithm with mean and standard deviation (STD). Here, cost is the total money that has to be paid to the selected oracles within all the observations. In general, BLOR had the highest performance and Random the least. From the economic perspective, BLOR was the most economical solution and ϵ-greedy was the costliest one. However, in terms of computation time, Random runs very fast, followed by ϵ-greedy with a comparable time against BLOR. As expected, Monte Carlo lasted the longest.

Fig. 8 presents the performance of each method in noisy observations. By noise, we mean that the oracle did not behave as expected. It is obvious that the performance of all the methods decline when the noise

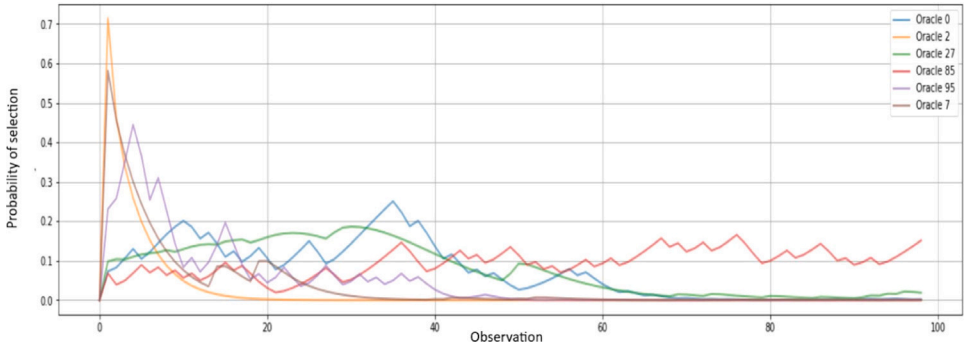


Fig. 7. Comparison of value function for the sample oracles.

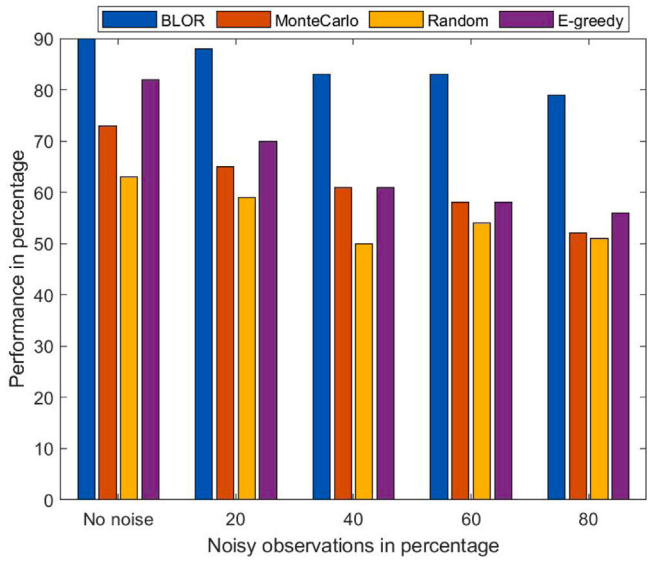


Fig. 8. Performance comparison against noisy observations.

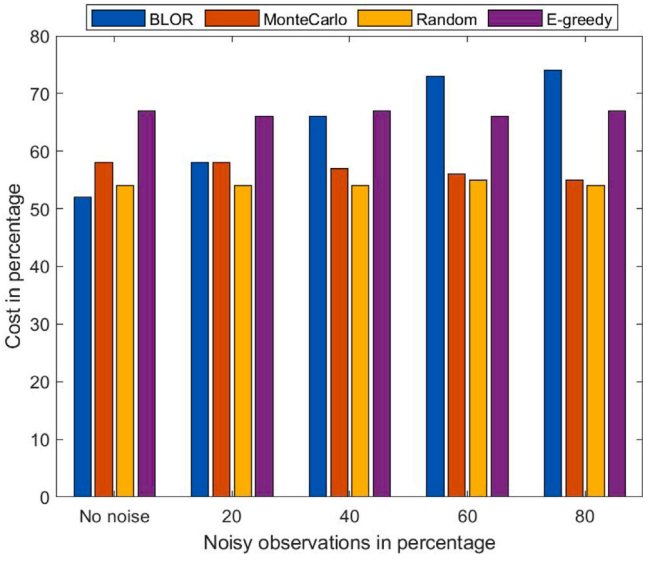


Fig. 9. Total cost comparison against noisy observations.

increases, however, BLOR had the most steady accuracy. Even in a very noisy situation, BLOR could maintain its performance by almost 80%. After BLOR, even though ϵ-greedy had higher accuracy, it was the most influenced by the noise. Since this heuristic algorithm mostly picks the oracle with the highest reward, it is unable to recognize its change of behavior and is not suitable for noisy subjects. Consequently, ϵ-greedy picked the most expensive oracles in non-noisy or less noisy situations and followed the same expense as the noise increased. As expected, random selection did not show a significant change in its performance, which was mostly around 50%. The moderate performance of Markov chain Monte Carlo was mainly because of two reasons: (1) this method requires several observations to build its model; and (2) Markov chain Monte Carlo needs equal historical records of the oracles, while we assumed that each oracle joined in different time and their age of operation is not the same.

We further measured the total cost of the selected oracles by all the considered methods as reported in Fig. 9. In a non-noisy or less noisy environment, BLOR performed very well. However, as the noise increases, BLOR sacrifices the cost to maintain the high performance. In a very noisy situation (more than 50%), BLOR is the costliest method. ϵ-greedy picked the most expensive oracles in non-noisy or less noisy situations and followed the same expense as the noise increased. Random and then Markov Chain Monte Carlo were economical in all the situations.

In order to investigate the effect of number of faulty oracles on the performance of these methods, we run several experiments with

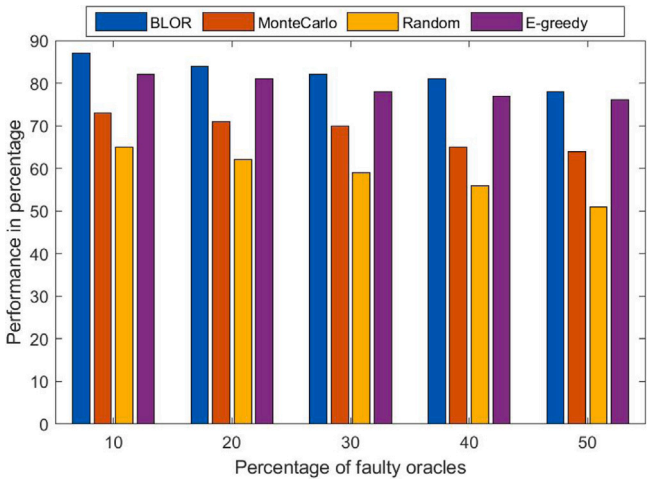


Fig. 10. Performance comparison against the number of faulty oracles.

different percentages of faulty oracles. A faulty oracle is an oracle with a history of more than 50% failures. As Fig. 10 shows, performance or accuracy of BLOR was the highest among these methods, followed by

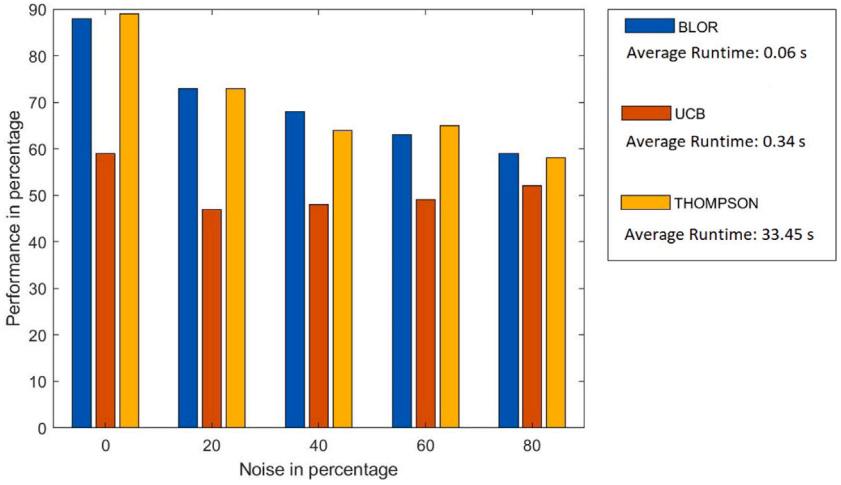
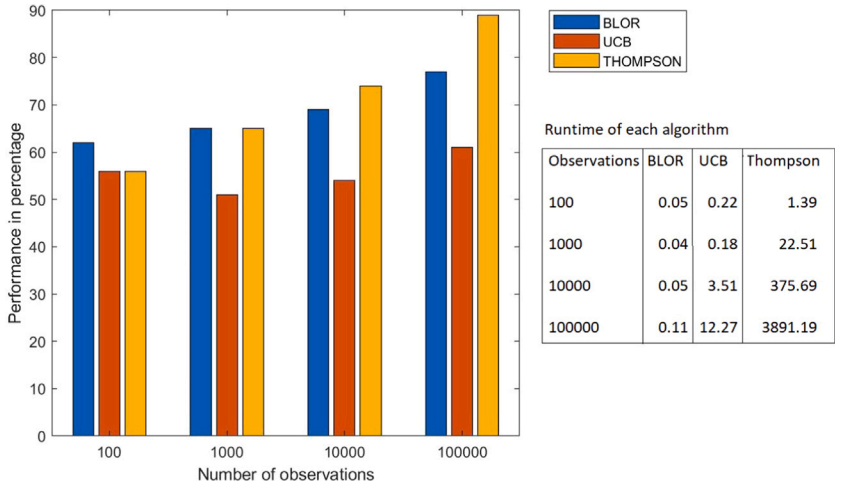


Fig. 11. Performance comparison against noisy observations along with the average run-time of each algorithm.



Runtime of each algorithm

Observations	BLOR	UCB	Thompson
100	0.05	0.22	1.39
1000	0.04	0.18	22.51
10000	0.05	3.51	375.69
100000	0.11	12.27	3891.19

Fig. 12. Performance comparison against number of observations along with the run-time of each algorithm (in sec.).

ϵ-greedy. However, as the percentage of faulty oracles decreases, the performance of all the methods increases.

6.2.2. *Bandit algorithms under uncertainty*

Random exploration is easy and fast, but not very efficient. It would be preferred to use the policies that operate under high uncertainty and choose those having high potential for being optimal in each round of selection. Two widely known algorithms for uncertain environments are UCB1 (referred as UCB throughout this paper) and Thompson sampling.

The UCB algorithm has been proposed in Auer, Cesa-Bianchi, and Fischer (2002) as a simple yet elegant solution of optimism in the face of uncertainty. This algorithm keeps the number of times that each arm has been selected in addition to their rewards. Each arm will be selected once at initial rounds, and then the algorithm picks greedily the most rewarding ones. The UCB method strictly prefers slightly higher UCB valued treatment arms.

Thompson sampling, known as probability matching, was first proposed in Thompson (1933) for two-armed bandit problems in clinical trials. The main idea is to assume a simple prior distribution on the parameters of the reward distribution of each arm, and at any time step, play an arm according to its posterior probability of being the best arm. Even though Thompson sampling is mathematically well defined, it is computationally inefficient.

We compared BLOR's performance against UCB and Thompson sampling under two different conditions. The first condition was observing the behavior of algorithms in 1000 observations when we increase the noise, which is shown in Fig. 11. In the second condition that is presented in Fig. 12, the noise was steady and set to be 50%, but the number of observations were increased from 100 to 100000. In all cases, runtime (in sec.) of each algorithm is presented since it is the most important element to consider while being executed in blockchain.

Overall, BLOR outperformed UCB in all cases and had a comparable performance with Thompson sampling in our simulations. The UCB algorithm explores the oracles based on the number of times each oracle has been selected and explored before. The more each oracle is chosen, the more confident it becomes about receiving the future rewards. Since UCB assumes that the behavior probability of the oracles will be maintained in the future as they become more confident, facing the noise in the beginning has a more negative effect on the performance. Since, in our scenario, it has been simulated for the oracles to change their behavior, it becomes challenging for UCB to adapt very quickly. A longer history of UCB observations could help in picking the winning oracle.

Thompson sampling uses a sample over a probability distribution of oracles trustworthiness, thus, it tends to adapt faster with any changes. Moreover, this algorithm always considers the possibility of change of behavior for the oracles, so it has a more stable performance. However,

Thompson sampling is a much more computationally complex algorithm, and it is not a suitable solution to be deployed over blockchain. Not only it consumes very high computational resources, but it also can cost a lot to be executed.

BLOR has adapted the concept of sampling over a probability distribution similar to Thompson sampling, so that it could outperform UCB and follow closely the performance of Thompson sampling. Even though Thompson sampling could beat BLOR in very high trials (e.g. 100,000 trials), BLOR is extremely less complex and faster, which makes it overall a suitable solution to be used and executed on blockchain.

7. Conclusion

Oracles gather information from the real world and transport it onto the blockchain for further use. Hence, the use of oracles is imperative to promote a widespread adoption of smart contracts. Yet, research about oracles and their practical application is very immature. This paper tried to shed some light by addressing two major challenges in this area. The first challenge is about employing a smart mechanism in place to identify the trustless and cost-efficient oracles. This challenge was addressed by developing a Bayesian cost-dependent reputation model in a multi-armed bandit setting, named BLOR. The second challenge of actual application of a smart mechanism using reinforcement learning was dealt with by implementing BLOR on Ethereum. To the best of our knowledge, this paper is the first to implement a machine learning algorithm for smart contracts in general, and for oracles' recruitment in particular. We showed how to solve various challenges that could raise while implementing complex algorithms like machine learning in Ethereum. To prove the efficiency of BLOR, we simulated 100 oracles operating within a blockchain in 1000 observations and benchmarked it against several algorithms that vary by their degree of exploration and exploitation. It was found that BLOR prioritizes the newer oracles which hold less history and those with a fair balance of performance and price. Through experiments using various benchmarks, BLOR proved a steady performance in a low to high noisy environment ranging from 80 to 90 percent, whereas other algorithms performance was ranged from 80 to 50 percent. BLOR had the most cost saving selection when the noise was lower. Overall, BLOR performed competitively better than the other algorithms. As future work, we plan to integrate advanced trust concepts such as group trust, distributed trust and propagated trust (Bentahar, Drawel, & Sadiki, 2022; Drawel, Bentahar, Laarej, & Rjoub, 2020, 2022; Drawel, Qu, Bentahar, & Shakshuki, 2020) into our oracle recruitment mechanism and investigate the potential of federated learning for a federated selection mechanism of oracles (Rjoub, Wahab, Bentahar & Bataineh, 2022; Wahab, Rjoub, Bentahar, & Cohen, 2022).

CRediT authorship contribution statement

Mona Taghavi: Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft. **Jamal Bentahar:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Funding acquisition, Writing – review & editing. **Hadi Otrok:** Conceptualization, Investigation, Methodology, Validation, Writing – review & editing. **Kaveh Bakhtiyari:** Conceptualization, Investigation, Formal analysis, Software, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

M. Taghavi was supported by NSERC Vanier, and J. Bentahar is supported by NSERC, FRQNT, and MITACS.

References

Alagha, A., Singh, S., Mizouni, R., Bentahar, J., & Otrok, H. (2022). Target localization using multi-agent deep reinforcement learning with proximal policy optimization. *Future Gener. Comput. Syst.*, *136*, 342–357.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2), 235–256.

Baygin, M., Yaman, O., Baygin, N., & Karakose, M. (2022). A blockchain-based approach to smart cargo transportation using UHF RFID. *Expert Systems with Applications*, *188*, Article 116030. http://dx.doi.org/10.1016/j.eswa.2021.116030.

Bentahar, J., Drawel, N., & Sadiki, A. (2022). Quantitative group trust: A two-stage verification approach. In P. Faliszewski, V. Mascardi, C. Pelachaud, & M. E. Taylor (Eds.), *21st international conference on autonomous agents and multiagent systems* (pp. 100–108). International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Berryhill, R., & Veneris, A. (2019). ASTRAEA: A decentralized blockchain oracle. IEEE Blockchain Technical Briefs.

Bhatia, G. K., Gupta, S., Dubey, A., & Kumaraguru, P. (2020). WorkerRep: Immutable reputation system for crowdsourcing platform based on blockchain. arXiv preprint arXiv:2006.14782.

Bouraga, S. (2021). A taxonomy of blockchain consensus protocols: A survey and classification framework. *Expert Systems with Applications*, *168*, Article 114384.

Buterin, V., & Griffith, V. (2017). Casper the friendly finality gadget. CoRR abs/1710.09437, arXiv:1710.09437.

Chatzopoulos, D., Ahmadi, M., Kosta, S., & Hui, P. (2017). Flopcoin: A cryptocurrency for computation offloading. *IEEE Transactions on Mobile Computing*, *17*(5), 1062–1075.

Chatzopoulos, D., Gujar, S., Faltings, B., & Hui, P. (2018). Privacy preserving and cost optimal mobile crowdsensing using smart contracts on blockchain. In *2018 IEEE 15th international conference on mobile Ad Hoc and sensor systems* (pp. 442–450). IEEE.

Ding, Y., Chen, Z., Lin, F., & Tang, C. (2019). Blockchain-based credit and arbitration mechanisms in crowdsourcing. In *2019 3rd international symposium on autonomous systems* (pp. 490–495). IEEE.

Drawel, N., Bentahar, J., Laarej, A., & Rjoub, G. (2020). Formalizing group and propagated trust in multi-agent systems. In C. Bessiere (Ed.), *Proceedings of the twenty-ninth international joint conference on artificial intelligence* (pp. 60–66). ijcai.org.

Drawel, N., Bentahar, J., Laarej, A., & Rjoub, G. (2022). Formal verification of group and propagated trust in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, *36*(1), 19.

Drawel, N., Qu, H., Bentahar, J., & Shakshuki, E. M. (2020). Specification and automatic verification of trust-based multi-agent systems. *Future Generation Computer Systems*, *107*, 1047–1060.

Ellis, S., Juels, A., & Nazarov, S. (2017). Chainlink a decentralized oracle network. *White paper*, *11*.

Frazier, P., Powell, W., & Dayanik, S. (2009). The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*.

Gao, S., Chen, X., Zhu, J., Dong, X., & Ma, J. (2021). TrustWorker: A trustworthy and privacy-preserving worker selection scheme for blockchain-based crowdsensing. *IEEE Transactions on Services Computing*.

Goel, N., van Schreven, C., Filos-Ratsikas, A., & Faltings, B. (2020). Infochain: A decentralized, trustless and transparent oracle on blockchain. In C. Bessiere (Ed.), *Proceedings of the twenty-ninth international joint conference on artificial intelligence* (pp. 4604–4610).

Ho, G., Tang, Y. M., Tsang, K. Y., Tang, V., & Chau, K. Y. (2021). A blockchain-based system to enhance aircraft parts traceability and trackability for inventory management. *Expert Systems with Applications*, *179*, Article 115101. http://dx.doi.org/10.1016/j.eswa.2021.115101.

Hull, R., Batra, V. S., Chen, Y.-M., Deutsch, A., Heath III, F. F. T., & Vianu, V. (2016). Towards a shared ledger business collaboration language based on data-aware processes. In *ICSOC* (pp. 18–36).

Jain, S., Narayanaswamy, B., & Narahari, Y. (2014). A multiarmed bandit incentive mechanism for crowdsourcing demand response in smart grids. In C. E. Brodley, & P. Stone (Eds.), *Proceedings of the twenty-eighth AAAI conference on artificial intelligence* (pp. 721–727). AAAI Press.

Jiao, Y., Wang, P., Niyato, D., & Xiong, Z. (2017). Social welfare maximization auction in edge computing resource allocation for mobile blockchain. arXiv preprint arXiv:1710.10595.

Kadadha, M., Otrok, H., Mizouni, R., Singh, S., & Ouali, A. (2020). SenseChain: A blockchain-based crowdsensing framework for multiple requesters and multiple workers. *Future Generation Computer Systems*, *105*, 650–664.

Khosravifar, B., Bentahar, J., Moazin, A., & Thiran, P. (2010). Analyzing communities of web services using incentives. *International Journal of Web Services Research*, *7*(3), 30–51.

Kochovski, P., Gec, S., Stankovski, V., Bajec, M., & Drobintsev, P. D. (2019). Trust management in a blockchain based fog computing platform with trustless smart oracles. *Future Generation Computer Systems*, *101*, 747–759.

Li, H. (2010). Customer reviews in spectrum: Recommendation system in cognitive radio networks. In *2010 IEEE symposium on new frontiers in dynamic spectrum* (pp. 1–9).

Liu, X., Zhu, T., Jiang, C., Ye, D., & Zhao, F. (2022). Prioritized experience replay based on multi-armed bandit. *Expert Systems with Applications*, *189*, Article 116023. http://dx.doi.org/10.1016/j.eswa.2021.116023.

Lo, S. K., Xu, X., Staples, M., & Yao, L. (2020). Reliability analysis for blockchain oracles. *Computers and Electrical Engineering*, *83*, Article 106582.

Lu, Q., Xu, X., Liu, Y., Weber, I., Zhu, L., & Zhang, W. (2019). uBaaS: A unified blockchain as a service platform. *Future Generation Computer Systems*, *101*, 564–575.

Ma, L., Kaneko, K., Sharma, S., & Sakurai, K. (2019). Reliable decentralized oracle with mechanisms for verification and disputation. In *2019 seventh international symposium on computing and networking workshops* (pp. 346–352).

Mendling, J., Weber, I., Aalst, W. V. D., Brocke, J. V., Cabanillas, C., Daniel, F., et al. (2018). Blockchains for business process management-challenges and opportunities. *ACM Transactions on Management Information Systems*, *9*(1), 4.

Morabito, V. (2017). *Business innovation through blockchain*. Cham: Springer International Publishing.

Münsing, E., Mather, J., & Moura, S. (2017). Blockchains for decentralized optimization of energy resources in microgrid networks. In *CCTA* (pp. 2164–2171).

Rjoub, G., Bentahar, J., Wahab, O. A., & Bataineh, A. S. (2021). Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. *Concurrency and Computation: Practice and Experience*, *33*(23).

Rjoub, G., Wahab, O. A., Bentahar, J., & Bataineh, A. (2022). Trust-driven reinforcement selection strategy for federated learning on IoT devices. *Computing*, http://dx.doi.org/10.1007/s00607-022-01078-1.

Rjoub, G., Wahab, O. A., Bentahar, J., Cohen, R., & Bataineh, A. (2022). Trust-augmented deep reinforcement learning for federated learning client selection. *Information Systems Frontiers*, in press. http://dx.doi.org/10.1007/s10796-022-10307-z.

Sami, H., Bentahar, J., Mourad, A., Otrok, H., & Damiani, E. (2022). Graph convolutional recurrent networks for reward shaping in reinforcement learning. *Inf. Sci.*, *608*, 63–80.

Sami, H., Mourad, A., Otrok, H., & Bentahar, J. (2021). Demand-driven deep reinforcement learning for scalable fog and service placement. *IEEE Transactions on Services Computing*, http://dx.doi.org/10.1109/TSC.2021.3075988.

Sami, H., Otrok, H., Bentahar, J., & Mourad, A. (2021). AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach. *IEEE Transactions on Network and Service Management*, *18*(3), 3527–3540.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.

Taghavi, M., Bentahar, J., Otrok, H., & Bakhtiyari, K. (2018). Cloudchain: A blockchain-based coopetition differential game model for cloud computing. In *ICSOC* (pp. 146–161). Springer.

Taghavi, M., Bentahar, J., Otrok, H., & Bakhtiyari, K. (2020). A blockchain-based model for cloud service quality monitoring. *IEEE Transactions on Services Computing*, *13*(2), 276–288.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, *25*(3/4), 285–294.

Underwood, S. (2016). Blockchain beyond bitcoin. *Communications of the ACM*, *59*(11), 15–17.

Villar, S. S., Bowden, J., & Wason, J. (2015). Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges. *Statistical Science: A Review Journal of the Institute of Mathematical Statistics*, *30*(2), 199.

Wahab, O. A., Rjoub, G., Bentahar, J., & Cohen, R. (2022). Federated against the cold: A trust-based federated learning approach to counter the cold start problem in recommendation systems. *Information Sciences*, *601*, 189–206.

Xiao, L., Ding, Y., Jiang, D., Huang, J., Wang, D., Li, J., et al. (2020). A reinforcement learning and blockchain-based trust mechanism for edge networks. *IEEE Transactions on Communications*, *68*(9), 5460–5470.

Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., et al. (2016). The blockchain as a software connector. In *2016 13th working IEEE/IFIP conference on software architecture* (pp. 182–191). IEEE.

Xu, J., Wang, S., Bhargava, B. K., & Yang, F. (2019). A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing. *IEEE Transactions on Industrial Informatics*, *15*(6), 3538–3547.

Xu, C., Wang, K., & Guo, M. (2017). Intelligent resource management in blockchain-based cloud datacenters. *IEEE Cloud Computing*, *4*(6), 50–59. http://dx.doi.org/10.1109/MCC.2018.1081060.

Yermack, D. (2017). Corporate governance and blockchains. *Review of Finance*, *21*(1), 7–31.

Zhang, F., Cecchetti, E., Croman, K., Juels, A., & Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 270–282).

Zhang, Y., & Wen, J. (2017). The IoT electric business model: Using blockchain technology for the internet of things. *Peer-To-Peer Networking and Applications*, *10*(4), 983–994.

Zhao, K., Tang, S., Zhao, B., & Wu, Y. (2019). Dynamic and privacy-preserving reputation management for blockchain-based mobile crowdsensing. *IEEE Access*, *7*, 74694–74710.