QATAR UNIVERSITY

COLLEGE OF ENGINEERING

NOVEL TECHNIQUES FOR BLOCKCHAIN-ENABLED IOT SYSTEMS LEVERAGING REINFORCEMENT LEARNING

BY

NARAM SULTAN MHAISEN

A Thesis Submitted to

the College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computing

June 2020

© 2020 Naram Sultan Mhaisen. All Rights Reserved.

COMMITTEE PAGE

The members of the Committee approve the Thesis of
Naram Sultan Mhaisen defended on 20/04/2020.

Prof. Mohsen Guizani
Thesis Supervisor

Dr. Noora Fetais
Thesis Co-Supervisor

Prof. Amr Mohamed
Committee Member

Dr. Aiman Erbad
Committee Member

Prof. Jalel Ben-Othman
Committee Member

Dr. Mohamed Arslan Ayari
Committee Member

Approved:

Khalid Kamal Naji, Dean, College of Engineering

ABSTRACT

Mhaisen, Naram, Sultan, Masters: June: 2020, Master in Computing

Title: Novel Techniques for Blockchain-enabled IoT Systems Leveraging Reinforcement Learning.

Supervisor of Thesis: Mohsen, Mokhtar, Guizani.

In the last decade, blockchain and Smart Contracts (SCs) have attracted unprecedented attention in academia and industry due to their technical innovation of providing an immutable distributed ledger with secure cryptographic consensus rules. However, to leverage the benefits of SCs in different domains, its adaptation should take into consideration the characteristics and requirements of specific applications and systems. In this thesis, we investigate the use of SCs in the Internet of Things (IoT) applications. Specifically, we identify and propose solutions to two potential issues that might arise from such integration. First, we demonstrate that because IoT monitoring requires replicated data sources that continuously submit data as transactions to the blockchain, naive integration with SCs is prohibitively expensive. Instead, the data submission should be optimized to minimize the cost while still meeting the use-case requirement of audibility and security. We propose a Reinforcement Learning (RL)-based approach to achieve such a tradeoff and show its superior performance compared to currently followed methods. On the other hand, we also demonstrate that using SCs for task-allocation in applications like service provisioning can lead to inefficient allocation decisions due to the static nature of SCs rules that aim to manage dynamic blockchain participants. We show that leveraging the ever-expanding blockchain data for online learning by means of RL provides

viable and adaptive task allocation that also outperforms currently deployed techniques in terms of cost-efficiency. Overall, the problem formulations presented here, as well as their proposed solutions, contribute to the establishment of secure and intelligent decentralized IoT applications.

DEDICATION

To my family, for their support and faith in me.

ACKNOWLEDGMENTS

I would like to sincerely thank my thesis supervisors, Prof. Mohsen Guizani and Dr. Noora Fetais, for encouraging me to deliver my best and believing in me. I would also like to thank Prof. Amr Mohamed and Dr. Aiman Erbad, for their valuable feedback throughout the thesis.

TABLE OF CONTENTS

DEDICATION v

ACKNOWLEDGMENTS vi

LIST OF TABLES ix

LIST OF FIGURES x

Chapter 1: Introduction 1

Motivation 1

Thesis Objectives and Contribution 5

Thesis Overview 6

Chapter 2: Background and Related Work 7

Background 7

Markov Decision Processes and Reinforcement Learning 7

Blockchain and Smart Contracts 8

Smart Contracts for IoT applications 9

Related Work 12

Smart Contracts/IoT Integration and Optimization 12

Reinforcement Learning for Blockchain Optimization 14

Reinforcement Learning for Service Provisioning Optimization 15

Chapter 3: IoT Transaction Optimization: An RL Approach 17

System Model 17

Smart Contract 18

Agent 19

Decentralized Oracles 20

Problem Formulation 20

State and Action Spaces 21

Environment Design 22

RL Agent Design 24

Performance Evaluation 31

Experimental Settings 31

Reward Convergence 33

Performance Comparison 35

Effect of Different Use Case Requirements 40

Experimental Case Study 42

Chapter 4: Task Allocation Optimization in SCs: An RL Approach 45

System model 45

MDP Formulation 45

Action Space 48

State Transition 48

Reward Structure 49

RL Agent Design 50

Performance Evaluation 53

Reward Convergence 53

Performance Comparison 54

Chapter 5: Conclusion 59

Publications 61

References 62

LIST OF TABLES

Table 2.1. SC/IoT Integration Approaches 14

Table 3.1. Oracle Management System Parameters 33

Table 3.2. Transactions number for policies.. 38

Table 4.1. Task Allocation System Parameters.. 54

LIST OF FIGURES

Figure 2.1. Reinforcement learning problem setup 7

Figure 3.1. The system model of the proposed SC-based IoT monitoring system 17

Figure 3.2. Reinforcement-based interaction between the smart oracle manager (DQN agent) and the blockchain network... 30

Figure 3.3. Ether pricing data 31

Figure 3.4. The training process, reward throughout episodes................................. 34

Figure 3.5. The learned policy... 34

Figure 3.6. Performance comparison between different policies 37

Figure 3.7. Distribution of update requests 38

Figure 3.8. Performance comparison with different period-based heuristic policies 40

Figure 3.9. Cost comparison of different security features evaluations.................... 40

Figure 3.10. Cost comparison for different expected business condition occurrence 41

Figure 3.11. Setup of the prototype for a blockchain network implementing the proposed framework... 42

Figure 3.12. Snapshots for a decision step in a monitoring episode. (a) The main code snippet of the agent using Torch and Web3 libraries to determine the action and set the contract's flag, respectively. (b) a Geth terminal showing the transaction submitted to set the contract's flag being mined into a block. (c) Ethereum Remix interface [66] showing the new value of the flag (d) Remix interface showing the temp value submitted to the contract by the IoT node 43

Figure 4.1. Task allocation for transactions (service requests) to service providers (indicated as "P") in the blockchain 46

Figure 4.2. The training process, reward throughout episodes 57

Figure 4.3. Comparison with (a) greedy (b) load-aware (c) planning methods 57

Figure 4.4. Performance of MPC and DRL during major changes. Red-shaded area indicates a score that is in the same range as heuristic methods. Green-shaded areas indicates a score that is not attained in previous experiments 57

NOMENCLATURE

$\mathcal{A}$ Action space

γ Discount factor

$\mathcal{P}$ Probability density function

π Policy

$\mathcal{R}$ Stochastic reward function

$\mathcal{S}$ State space

$\mathcal{T}$ State transition function

A Action random variable

a Sample action value

$A(s, a)$ The advantage function for a policy π

$Q^{\pi}(s, a'; \theta)$ The approximation of the state-action value function for a policy π using parameter vector θ

$Q^{\pi}(s, a)$ The state-action value function for a policy π

$Q^{\pi^*}(s, a), Q^*(s, a)$ The approximation of the optimal state-action value

$Q^{\pi^*}(s, a; \theta), Q^*(s, a; \theta)$ The approximation of the optimal state-action value function using parameter vector θ

R Cumulative reward

r Sample reward value

S State random variable

s Sample state value

$V(s)$ The state value function for a policy π

CHAPTER 1: INTRODUCTION

Motivation

Recently, Smart Contracts(SCs) have witnessed a surge in popularity in both academia as well as in industrial applications. Specifically, SCs/IoT integration is being increasingly utilized by several critical domains. For instance, in health applications, SC-enabled vital signs monitoring can enhance caregivers' informing/intervention measures [1] and interoperability between medical centers [2]. In energy applications, SCs are used to observe energy consumption behavior and detect manipulation or misuse in smart meters [3] and electric vehicles [4]. Also, surveillance systems use blockchain to record events and notify concerned parties [5]. The food industry heavily utilizes blockchain to track the origin and conditions of different goods [6], [7]. The same tracking application is used by shipping and logistics companies for insurance and related liabilities [8]. In general, the secure SC-based monitoring process is critical and widely applicable in multiple domains. However, less work was conducted on the study of SCs operation cost, which motivates this study. Such cost can be prohibitive when applied in a smart-city scale. Besides, it does not suit different users' requirements as they differ per use case.

Operating the IoT nodes for SC-based monitoring can be expensive and inefficient due to two main reasons. First, monitoring is a continuous process. In public blockchains, each measurement submission by a sensor incurs a cost known as a transaction cost. This means that transaction costs need to be continuously paid by the IoT device accounts throughout the required monitoring period (e.g., during the shipment period of an asset) [9]. Second, it is common in IoT/SC applications that multiple

independent IoT devices are used for the same purpose, which amplifies the cost issue even more. Such an array of redundant data sources is referred to as “decentralized oracles" and is essential for reliability reasons [10].

To minimize this cost, IoT interaction with the blockchain should be well controlled. Specifically, the transactions’ submission rate should be tuned so that a transaction is submitted only when necessary. Learning-assisted decision-making techniques help us address the dilemma of whether we should create a transaction in the blockchain for any value to be monitored, which is costly and not scalable, or create a transaction occasionally, which may not be acceptable from traceability and event tracking point of view. It is also essential to consider such a tradeoff in the context of the use case. For example, in the case of patient monitoring, the cost might be irrelevant during emergency cases. However, in other domains such as logistic shipping, it might be desirable to hold-off the monitoring for some period to save cost.

Optimizing the transaction rate from IoT devices (also referred to as Oracles in the context of blockchain) to SCs while being cost-efficient is challenging since this rate should be set intelligently based on multiple factors such as the current blockchain network usage cost and oracle system status. This information is available in real-time only, they are continually changing, and future values can not be known in advance. Further, use case requirements should be considered as well. To tackle this random environment, we adopt a Reinforcement Learning (RL) approach. RL is an artificial intelligence technique that studies the issue of complex decision making in a random environment, aiming to achieve maximum reward (or equivalently, minimum cost) over the long run [11].

On the other hand, even after the data is successfully and efficiently placed on-chain,

there still exist multiple operational issues pertaining to SCs. A critical one is the management of *service provisioning* contracts, which assign users to service providers according to some criteria.

Lately, Multiple domains started to leverage the autonomy, resilience, and transparency features of blockchain and SCs in their service provisioning. These include peer-to-peer (P2P) energy trading of household renewable energy or electric vehicles [12], with some projects already deployed in industry [13]. SCs were used to enhance P2P energy trading as there will be no need for a central authority (utility) to act as a mediator in the energy transfer. Cloud and edge resource allocation market also heavily utilize SCs to facilitate the resource auction mechanism and record requests and services for non-repudiation attack protection [14]. In the domain of the Internet of Things, SCs are being utilized for the provenance of data sources, as well as machine-to-machine (M2M) payments to enable the direct transaction between sensing/actuating nodes without relying on a centralized server [15]. In general, SCs provide appealing features for the service provisioning in any industry due to its secure and autonomous execution [16] and have been shown to support large deployments of devices (e.g., IoT) for automated service provisioning [17].

A crucial issue in service provisioning is the criteria of assigning users or tasks to service providers in such a way that maximizes global welfare (utility). Such allocation problems should consider multiple factors, including the *service cost* that users endure as well as the *operation cost* of service providers. In general, the service allocation should jointly optimize the service cost and operation cost. Obviously, service provisioning systems aim to assign users to the service provider with minimum fees to save service cost for the users. However, service providers have limited capacities, and overloading

them might lead to high operation costs over time. For example, if the service provider is an IoT node delivering remote information as a service, then energy consumption is a significant concern. Similar arguments can be made if the service provider is a smart battery/EV trading energy since more trading operations can cause a tear to the battery. In cellular systems providing spectrum access services, servicing more users will either force the station to rent extra spectra or degrade the service quality. In general, the operation cost of service providers, as deduced by their load, should also be considered by any service provisioning system.

The joint optimization that service provisioning and task allocation systems generally seek is especially challenging in blockchain environments due to multiple factors. First, the blockchain is a dynamic system where service providers of different capacities join and leave flexibly. Second, SCs, when deployed, are immutable and cannot be changed or require complex update process [18]. Hence, static task allocation logic implemented as SC cannot keep up with the dynamic nature of participants. In summary, the heterogeneity and flexibility of participants in blockchain systems impose a challenge to traditional optimization methods and calls for adaptive, real-time methods that are still required to allocate tasks optimally.

Most of the literature research on smart contracts for the job and task allocation implemented static rules. However, this approach misses the opportunity of leveraging the ever-expanding data of blockchain [19]. We envision SCs as *smart agents* that interact with their participants and make an online data-driven decision on task allocation using previous data recorded on-chain.

Overall, the wide-applicability of SCs and IoT in multiple domains while still suffering from the aforementioned integration issues motivates us to study these problems

with the aim of optimizing the SC/IoT paradigm toward optimal efficiency.

Thesis Objectives and Contribution

The objectives of this thesis are two folds. First, we aim to design efficient and scalable methods for IoT and SCs integration trough optimizing the submission rate of IoT nodes' transactions. This objective is tailored towards the problem of the expensive and continuous transaction illustrated earlier in the motivations sections. Second, we aim to design SCs with intelligent behavior for the service provisioning broad use cases. This objective tackles the problem of inefficient static service provisioning criteria illustrated earlier. In summary, the thesis addresses the issue of importing data to the blockchain, and using the data intelligently by SCs for service provisioning. Contributions are summarised as follows:

1. Optimizing IoT nodes' transaction submission to the blockchain. To that end, we:
   - Propose a cost optimization framework for blockchain-enabled monitoring applications that achieves a near-optimal tradeoff between blockchain security features and monetary cost required for leveraging those features.
   - Design a Deep Reinforcement Learning (DRL) agent to achieve the tradeoff in real-time and based on dynamic and flexible user requirements.
   - Evaluate the proposed system against multiple currently followed heuristics with results showing the superior performance of the smart oracle manager.
   - Develop a prototype to demonstrate the practical applicability of the proposed framework.
2. Optimizing SCc operation for service provisioning applications . To that end, we:

- Formulate the service provisioning as a multi-objective Markov Decision Process (MDP) whose solution achieves the optimal tradeoff between users' *service cost* and service providers' *operation cost*.
- Utilize a learning-assisted decision-making technique, Deep Reinforcement Learning (DRL), to model intelligent SCs that can leverage the chained data and tackle the problem in an online manner.
- Provide a comprehensive performance evaluation and analysis of the proposed method along with locally optimal heuristics as well as other planning techniques.

Thesis Overview

This chapter motivated the thesis topic and summarised its objectives and contributions. The remainder is organized as follows: In Chapter 2, we introduce the main concepts, terminologies, and frameworks. We also survey related work and contrast them to our work. Chapter 3 presents the formulation and proposed solution for the IoT transaction rate optimization problem and evaluates the performance of different approaches. We follow the same structure in Chapter 4 but for the issue of task allocation in SC-managed service provisioning. We then conclude the work, summarize main results, and state limitations, and potential improvements in Chapter 5.

CHAPTER 2: BACKGROUND AND RELATED WORK

Background

Markov Decision Processes and Reinforcement Learning

Reinforcement Learning (RL) is an artificial intelligence technique that studies the issue of complex decision making in a random environment, aiming to achieve maximum reward (or equivalently, minimum cost) over the long run [11]. Specifically, RL techniques aim to design agents that can learn optimal behavior in multi-stage decision problems through interaction with the environment (see figure 2.1). The problem of multi-stage decision making is mathematically formalized as a Markov Decision Processes (MDP).

A Markov Decision Process (MDP) is defined as tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. At every time step t, the agent receives a representation of the environment state $s \in \mathcal{S}$. The agent then executes and action $a \in \mathcal{A}$ using a policy $\pi(a|s)$, receives a reward $r_t \in \mathbb{R}$, and transition to the next state s' with probability $P(s'|s,a) = \mathcal{T}(s,a,s')$. The total feature discounted sum of rewards until some horizon H is denoted as $R_t = \sum_{t'=t}^{H} \gamma^{t'-t} r_t$, with the discount factor $\gamma \in [0,1)$. The state-action value function of a specific policy π is defined as $Q^{\pi}(s,a) = \mathbb{E}_{a \sim \pi, s' \sim \mathcal{T}}[R_t | s_t = s, a_t = a]$. It summarises the sum of rewards

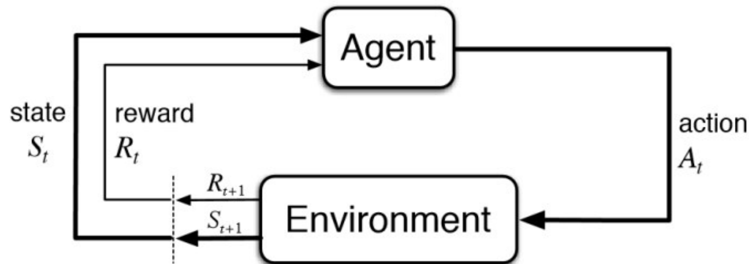

Figure 2.1: Reinforcement learning problem setup

resulting from taking the action a in state s, and thereafter following policy π. The state value function $V^{\pi}(s) = \mathbb{E}_{a \sim \pi}[Q^{\pi}(s, a)]$ assesses the quality of a state when following the policy π. The advantage function is then defined as $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$, which reflects the advantage of taking a in s.

The optimal policy maximizes the Q-function $Q^{\pi^*}(s, a) = max_{\pi} Q(s, a)$ (hereafter referred to as Q^*). The goal of a model-free reinforcement learning agent is to find such optimal policy online through direct interaction with the environment and without explicit or pre-encoded information about the environment, such as the transition probability $\mathcal{T}$. One way to find this function is through firstly finding Q^* and then acting greedily with respect to it $\pi^*(a|s) = argmax_a Q^*(s, a)$ [11].

Blockchain and Smart Contracts

Blockchain has recently emerged as one of the most secure distributed system architectures. It is based on P2P networking where all participant nodes exchange transactions and reach consensus on the general state of an asset. Each one of the blockchain's nodes preserves an append-only, cryptographically-linked list of all events of interests and transactions that occurred in the network. This record is also referred to as the distributed ledger. The peer-enforced consensus rules and absence of a centralized third party make data manipulation extremely hard [20]–[22].

The first application of blockchain was Bitcoin [23], which is a fully distributed P2P payment system. Transactions in Bitcoin's blockchain are in the form of currency transfers only, meaning that the distributed ledger consists of blocks of Bitcoins transfers. However, the ability of the underlying technology of Bitcoin (blockchain) to maintain the security of each participant's assets in a fully distributed, P2P, and open network without

any trusted centralized utility is what brought massive attention to this technology [24].

Blockchain platforms have evolved since then to form what is known as general-purpose blockchains, or programmable blockchains. General-purpose blockchains enable any programmed logic to be deployed on the distributed network. Therefore transactions can be calls to functions defined in that program as opposed to only currency transfers [25]. Depending on the platform, these programs are called Smart Contracts [26] or Chaincode [27]. Smart contracts can be defined as a set of instructions that are executed automatically once predefined conditions are met. The execution is guaranteed by blockchain architecture even when one or more nodes fail (crash failure) or exhibit malicious behavior (Byzantine failure), which is why smart contracts are considered as more secure and self-enforcing as opposed to conventional control programs [28].

Smart Contracts for IoT applications

Smart Contracts (SCs) are general-purpose software applications that are deployed on the blockchain distributed network. They reflect real-world contractual agreements in a cyber form that offers multiple appealing security features [18]. As a form of distributed applications, SCs provide high availability against potential node failures. Further, their code implementation is immutable since it is stored on every node in the network. Lastly, since blockchain is an always-on network whose data is modified only through cryptographic consensus, the execution of an SC is automatic, and its output is cryptographically verifiable by every node in the network [29]. Due to these security capabilities, which make agreements self-enforceable, SCs are used to manage, track, and monitor ownership and status of critical digital and physical assets [30]–[32].

IoT/SCs integration is a promising recent paradigm whereby SCs can implement

autonomous agreements related to physical assets or real-world phenomena, which are perceived by IoT perception layer devices [33], [34]. This enables a wide range of applications where IoT perception data can be tracked or monitored to enforce specific rules encoded in the SC. For instance, a supply-chain SC might specify the temperature condition of some goods in a process. If the temperature goes higher than a specified threshold, the goods are deemed unusable, and the shipper pays a penalty. Otherwise, the shipping fees are credited to the manufacturer from the customer's account. Such SC implementation can be generalized to provide neutral, verifiable, self-enforcing rules in different application areas without the need for a centralized third party [35].

There are two main security features that motivate the use of SCs for monitoring: Auditability and automated execution of business logic. Auditability, also known as traceability, refers to the fact that the history of a signal recording can be viewed and verified by any interested stakeholder. By design, the history of records on the blockchain is tamper-proof. Hence, blockchains are being widely used for data provenance (e.g., to validate the origin of different kinds of goods such as physical-assets, food, and medication) [15], [36]. Automated execution of business logic is also ensured by design. When an SC triggers a procedure, the execution of the business logic code in that method is self-enforceable since the blockchain's consensus requires all nodes to reach a consistent state of the SC. This ensures immediate and inevitable business logic execution, enabling use cases between parties that did not trust each other before [37], [38]. Hence, SC-enabled monitoring provides appealing features for stakeholders.

In order to preserve the security of the blockchain network, SCs are generally executed in isolated, self-contained environments (e.g., EVM in Ethereum SC platform and Docker container in IBM Hyperledger Fabric SC platform). Thus, SCs cannot

directly access external data (e.g., through APIs) [39]. However, In an IoT-enabled scenario, it is essential to communicate with the real world in order to capture events of interest and relay them to an SC, which then enforces the set of agreed-upon rules. To address this issue, trusted data feeds place external data of interest on the blockchain, and then SCs utilize this data. These data feeds are also known as “oracles". IoT sensing devices/systems can act as oracles for any SC that requires information about data in the real world [37].

Decentralized oracles are a necessity due to the fact that the execution of SCs is irreversible by design. Hence, their triggering should be done with certainty. In the supply chain example, it can be noted that the temperature sensor’s input plays an essential role in this contract. Thus, its input should be of high reliability. Otherwise, legal issues and waste of resources will occur, defeating the original promise of SCs, which aims to automate and accelerate procedures in the first place. In general, such certainty cannot always be guaranteed in IoT perception devices. Many sensors undergo noise or abnormal conditions that lead to fluctuating readings. Further, the IoT sensing layer is always susceptible to false data injection attacks [40], [41]. Triggering a smart contract based directly on a single source of input would be inaccurate, inefficient, and carries high risk. This challenge can be described as the “Trusted Data Feed” or “Oracle problem” [42] and is generally addressed by incorporating multiple input sources (sensing devices) and reliably aggregating their input (e.g., majority voting) [39]. Hence, the transactions’ cost can be prohibitive for multiple applications.

Related Work

Smart Contracts/IoT Integration and Optimization

The cost-efficiency issue that arises when integrating IoT devices with public blockchains has recently been identified in the literature [35]. Authors in [5] propose the use of “smart oracle" which gathers trust information about real-world physical resources (including IoT devices) and provide the smart contract with this information. The SC can then utilize only the most trusted entities. This approach can reduce cost by assuming the trust of the single entity as an intermediary system and hence avoiding sensors redundancy. However, it does not directly address the unreliability issues that might still arise from depending on a single IoT data source. Similarly, Authors in [43] proposed a general data carrier architecture (oracle) that is scalable for the IoT environment. The proposed system is designed to minimize contract deployment costs and monitor contract events. The cost-effectiveness and security of data sources were addressed through pushing the data fetching and computation to the client-side (i.e., client managed IoT central sensing device), which is not always a viable option, especially for some use cases like remote asset tracking.

Relaying systems such as Provable [44] and Town Crier [45] supply SCs with data from the web accompanied with an “authenticity proof". Authenticity proofs are cryptographic proofs based on Transport Layer Security (TLS). They can be used to show that the data received was, in fact, the same data that the server, or any other third party, gave back to Provable at a specific time (data integrity) [46]. Receiving SCs can verify this proof prior to using the data it accompanied. This approach proves the origin of the data while avoiding additional costs due to sensors’ redundancy. However,

intermediary costs are paid for the operators of these systems for fetching the data and proving its origin.

Chainlink [47] is a decentralized oracle solution that provides proof of authenticity of the relayed data and tackles the issue of the reliability of data sources themselves. This is done through aggregating the data from multiple independent sources. A similar decentralized oracle system that depends on voting from different participants about the validity of a piece of data is proposed in [48], where authors formulate a game theory-based voting mechanism and illustrate the existence of a Nash equilibrium in which the best action of all participants is to vote honestly. This game-theoretic line of work is extended by [49]; the principle remains as designing a game based on multiple independent oracles. The system is analyzed to show the existence of an honest Nash equilibrium and has properties like the simple interface and larger expected payoffs. Those decentralized oracles solutions favor security to cost efficiency since redundant data sources continuously monitor and submit readings to the blockchain.

The importance of optimized interaction with blockchain from IoT is highlighted in Danzi *et al.* [50]. The authors proposed a data aggregation technique that regulates IoT clients' transactions to the Ethereum blockchain, which is done through data aggregation and periodic transmission. The focus was on optimizing the energy and computational resources of IoT clients rather than the cost of monitoring use cases.

A comprehensive taxonomy of different methods of importing external data to SCs is provided in [10]. It is illustrated that for applications that question the reliability of the data sources, such as SCs that use the IoT perception layer, on-chain voting of aggregated independent data sources represents the better option, albeit the efficiency and viability of this option are not always guaranteed.Table 2.1 summarises key differences among the

Table 2.1: SC/IoT Integration Approaches

Ref.	Purpose	Cost efficiency	Txn. submission	Data reliability
[5]	Collect trust metrics for IoT nodes	Yes	Cont.	Risk of failure of smart oracle
[43]	Elastically connect multiple IoT nodes to the SC	Yes	Cont.	Requires client management
[44], [45]	Relay data from multiple IoT nodes	No	Cont.	Provided by authenticity proofs
[47], [48], [49]	Collect data from multiple nodes & vote	No	Cont.	Provided by decentralization of oracles
[50]	Aggregate and submit data	Yes	At specific rate	Risk of failure of data source
This work	Aggregate and submit data	Yes	Real-time adaptive rate	Provided by decentralization of oracles

different schemes. Cost efficiency is considered present if there is a single data source, or the transaction submission is not continuous. As illustrated earlier, data reliability is best achieved by decentralized oracles.

Reinforcement Learning for Blockchain Optimization

Research that integrates AI and learning-assisted decision-making techniques to enhance and optimize different aspects of blockchain-based applications is still limited [19]. Liu *et al.* [51] utilized reinforcement learning techniques to optimize block interval, block size, block producer, and consensus algorithm to maximize system throughput. Xiong *et al.* [52] used RL to better integrate IoT devices in the blockchain

network and control their transactions so as the probability of these transactions being successful (mined and added to the chain of blocks) is maximized given the size of the current pending transaction pool of the blockchain. Qiu *et al.* [53], focused on the issue of computational efficiency and proposed an adaptive offloading mechanism that allocates mining and data processing tasks to edge/cloud to achieve a computational tradeoff for blockchain's end devices. These studies achieved positive results for their targeted objectives. However, they did not explicitly model users' requirements and their implications on the monitoring application design and transaction submission rate by the IoT devices.

Reinforcement Learning for Service Provisioning Optimization

RL has been showing impressive results for optimization in dynamic domains. For example, [54] proves the ability of a single RL agent to adapt to different circumstances of wireless networks while still optimally allocating resources (transmission power) to different nodes in the network. Authors in [55] jointly optimize resource allocation and user association in Heterogeneous cellular networks for offloading mobile traffic and showed that it could achieve an optimal tradeoff between the network utility and users Quality of Service. Task allocation in heterogeneous cloud clusters is studied in [56]. A set of jobs are sequentially allocated to a set of heterogeneous machines in a way that minimizes job completion time. While these studies give insights about the potential of RL in dynamic environments for multi-objective optimization, they do not directly model our case of interest. We investigate the joint optimization of the service cost and operation cost of the heterogeneous service providers in a continuously changing environment, such as SCs in blockchain networks with non-fixed participants.

Most of the literature research on smart contracts for the job and task allocation implemented static rules. However, this approach misses the opportunity of leveraging the ever-expanding data of blockchain [19]. We aim to model SCs as *smart agents* that interact with their participants and make an online data-driven decision on task allocation using previous data recorded on-chain. To the best of our knowledge, such an ambitious framework has only been investigated in [57]. Authors address the current lack of intelligence in SCs systems and motivate the design of "rational" contracts that can make decisions based on the available data on-chain as well as their recent experience (i.e., transaction results), to maximize a given utility. This work is concerned with the design of such autonomous SCs in the context of service provisioning. Specifically, we aim to design SCs that can smartly allocate tasks (i.e., users) to certain service providers in a way that maximizes the overall social welfare.

CHAPTER 3: IOT TRANSACTION OPTIMIZATION: AN RL APPROACH

System Model

An SC-enabled IoT monitoring framework is shown in Fig. 3.1. In this figure, the signal of interest might originate from any application, and it is monitored by multiple independent IoT sensing devices (decentralized oracles). The decentralized oracles periodically check if a status update is required through reading a flag from the SC state. Note that the reading from the blockchain incurs no cost since each node stores a copy of the distributed ledger. If the flag is set, the oracles submit readings to the SC and pay the transaction fees. These fees are refunded to the oracles' accounts from the stakeholders' accounts. The refund is encoded in the SC. Stakeholders represent any entity that is interested in the status and history of records. Since the SC state is public and freely accessible, stakeholders can trace and verify the traced history. The agent is a software program that intelligently sets the flag when a status update is required. Stakeholders have access to this program and should agree on its strategy of setting the state update request frequency as they will eventually be responsible for the transaction fees. In the following subsections, we explain the role of each entity shown in the figure in detail.

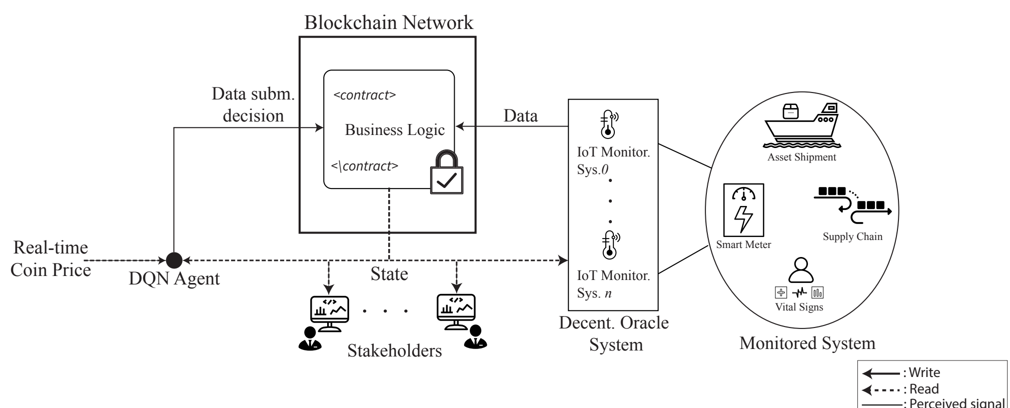

Figure 3.1: The system model of the proposed SC-based IoT monitoring system

Algorithm 1 Generic IoT-enabled Monitoring Smart Contract Template

Require: $\mathbf{d}$: Vector of data readings $\mathbf{d} =< d_0, d_1, d_2 ... d_n >$ from individual data feeds (oracles)

1: $on_chain_verification \leftarrow$ False (Initialization)
2: **procedure** SETFLAG(flag)
3: Verify agent's identity
4: $on_chain_verification \leftarrow$ flag
5: **end procedure**
6: **if** $on_chain_verification$ is True and $\mathbf{d}$ is set **then**
7: $aggregated_d \leftarrow$ AGGREGATE ($\mathbf{d}$)
8: APPLY BUSINESS LOGIC ($aggregated_d$)
9: **end if**
10: **procedure** AGGREGATE($\mathbf{d}$)
11: **aggregation logic**
12: $on_chain_verification \leftarrow$ False
13: **return** result of execution aggregation logic on $\mathbf{d}$
14: **end procedure**
15: **procedure** APPLY BUSINESS LOGIC ($aggregated_d$)
16: **if** $Condition_0$ on $aggregated_d$ **then**
17: **business logic here**
18: **else**
19: **business logic here**
20: **end if**
21: **end procedure**

Smart Contract

On a public blockchain network, the interested stakeholders deploy a smart contract that includes the business logic between them. The business logic can be represented by automatic funds transfer, alarm firing, or actuation triggering based on the monitored signal status and the encoded, agreed-upon rules (if / else conditions). A general outline of the contract is shown in Algorithm 1.

The contract has the flag variable *on_chain_verification*. When set to $True$, it indicates that a state update is required. Each oracle submits d_i, which is a full state

capture of the signal as recorded by oracle i. For example, consider the application of shipment monitoring, oracle 0 submits d_0 ={temperature: 25, humidity: 50, GPS cord.: $25, 50\}$. Then, An aggregation logic is triggered on $\mathbf{d}$, where $\mathbf{d} = \{d_i\}, i = \{0, 1, 2...n\}$. The aggregation logic can be majority voting among oracles, averaging, or any other outlier detection algorithm. The result of the aggregation is a single variable $aggregated_d$ that represents the current state of the monitored signal. $aggregated_d$ is recorded for verifiable auditability and checked against the agreed-upon rules to automatically execute business logic. The shown contract has one condition (term) that can include any business logic statements to be triggered. For instance, an SC might initiate specific actuators or inform stakeholders about some state updates of interest. The flag automatically resets to $False$ at the end of the aggregation function. The process repeats periodically every time step of the monitoring process; if on the following time step, the agent decided that a state update is required, the agent will set the flag again. Otherwise, it stays false until the time for another update is required. The agent is the only entity allowed to modify this variable.

Agent

The agent is an intelligent software program that decides on whether or not to set the *on_chain_verification* to *True*, at every time step, based on the current system state. Specifically, the flag is set based on current data submission monetary cost, the time-steps passed since the last update, and the use case monitoring requirements. This agent should balance the security requirements of the user with the cost. Meaning that the state should be recorded sufficiently frequently to meet the required auditability level and avoid any latency in business logic execution in case a condition in the SC is met.

However, such state updates should not be done continuously in order to avoid enduring a high cost. The agent communicates its decisions to the oracles through the SC to ensure the logging and enforceability of the agent's decisions.

Decentralized Oracles

The IoT sensing devices are used to monitor a signal of interest. The signal can originate from any monitored system. For example, the temperature/humidity/location of remote physical assets being shipped, items (such as medications) as they progress through a supply chain, vital signs for patients, or power consumption in smart grid applications.

There might exist up to n independent monitoring system to address the data source reliability issue. Each node on the decentralized oracle system has the ability to submit a state update to the chain. There are multiple ways of implementing such a requirement. For example, oracles submit their reading individually to the contract, which groups them based on timestamps or block times in order to prepare the vector $\mathbf{d}$.

Problem Formulation

At each time step of the monitoring process, the agent should make a decision on whether or not to request a state update. The decision is based on the current cost of such update as deduced by the current price of the blockchain's coin, and the number of time steps since the last update. The agent ultimately aims to meet pre-defined user requirements. This can be thought of as a sequential decision-making problem with interaction under uncertainty.

The MDP framework is an abstraction of the problem of goal-directed learning.

Formally, the MDP can be represented as a five-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \gamma)$. Where $\mathcal{S}$ is the system states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}(\cdot,\cdot)$ is the state transition probability, $\mathcal{R}(\cdot,\cdot)$ is the immediate reward, and γ is a discount factor. In the following sections, we define each of these elements in the context of our monitoring applications.

State and Action Spaces

The state space $\mathcal{S}$ consists of all possible states at each time step. We set $s_t =< p_t, T_t >$ for all $s_t \in \mathcal{S}$ where $p_t \in \mathbb{R}$ (in \$) is the current price of the chain's coin (cryptocurrency), $T_t \in \mathbb{N}$ is the number of time steps ago, starting at time step t, a state update was requested by the agent. Public blockchains have their own coin, which represents the monetary value exchanged by the nodes. Coins have a corresponding financial pricing index (similar to stocks). Based on this price, and the specific blockchain computation billing mechanism (e.g., “Gas"-based billing in Ethereum blockchain), the state update transaction cost can be calculated. Hence, p_t is required in the state for decision making. T_t is also required in a state point since it gives the agent insight about the frequency of state update transactions.

Since coin prices mostly follow hourly intervals, the time step, t, is set to one hour. Such a short time interval resolution ensures that the agent can closely follow the coin price and make decisions on an hourly basis. Taking higher values carries the risk of missing some considerable changes in the coin price (especially that it is unstable in most blockchains). While lower intervals are theoretically possible to follow. However, according to our experimentation, a one-hour decision interval is sufficient for most blockchains.

The agent's decision at every time step, a_t, is the binary value of the flag *on_chain_verification*.

Thus, for the action space, $a_t = \{0, 1\}$ for all $a_t \in \mathcal{A}$ where 1 indicates that state update is required, and 0 indicates the idle action. Note that when the action is 0, the agent does not need to submit this decision to the SC since the flag is automatically reset to 0 after every state update, as was explained in the SC subsection.

Environment Design

The environment of the MDP represents the blockchain network that the agents interact with. We design the environment so that it receives the agent's action a_t, transitions into a next state s_{t+1}, and emits a reward signal r_{t+1}. The environment should generate episodes of experiences for the agent to learn from. An episode is a finite sequence of states, actions, and rewards. In our context, the episode can represent a monitoring process (e.g., an asset shipment process during which the monitoring is done on hourly-bases until the delivery). During an episode, states are transitioned, and reward values are emitted according to the details in the following subsections. It is essential to mention that the agent does not know the environment design. It is supposed to learn an optimal policy (strategy) by solely interacting and observing rewards for different decisions taken in different states. A policy π is a mapping between states $s_t \in \mathcal{S}$ and (a distribution of) actions $a_t \in \mathcal{A}$. i.e., $\pi : \mathcal{S} \rightarrow \mathcal{A}$

State Transitions

Let S and A be the random variables of states and actions, respectively. Possible sampled state and action are denoted s, a. $P(\cdot|S_t, A_t)$ defines the probability distribution over the next states S_{t+1} conditioned on the current state and action. The state transition distribution cannot be analytically modeled since S_{t+1} cannot be determined from S_t

due to the uncertainty and randomness of the price data. A better alternative is to collect real-world hourly pricing data and utilize them to transition from p_t to p_{t+1}. This will represent real word transitions and provide the agent with samples $s_{t+1} \sim P(\cdot|S_t, A_t)$ without the need to assume or directly model the state transition distribution. For T_t, we check the last performed action, a_t, if set to 1 (fresh status update), T_{t+1} is reset to 0 . Otherwise, we set $T_{t+1} = T_t + 1$ to indicate that an additional time step has passed without state update.

Reward Function

We formulate a reward function that describes our high-level objective of minimizing total cost while meeting the security requirements of the monitoring process. Specifically, the security requirements include maximizing the auditability of the monitored signal/asset and simultaneously minimizing the delay in executing the business logic. The reward is a function of the current state s_t and current action a_t. The reward r_t at time step t is defined as

$$r_t = \begin{cases} -(N \times F_{txn}) - F_{agg} & \text{if } a_t = 1 \\ (-\alpha - \beta \times m) \times T_t & \text{if } a_t = 0 \end{cases} \tag{3.1}$$

Where N is the number of independent IoT sensing devices (oracles) submitting a state update transaction, F_{txn} is the fees per state update transaction, F_{agg} is the cost of executing the aggregation function on-chain on the array of states received by the redundant oracles. These values will be used if the agent decides to submit a state update request ($a_t = 1$).

α is a user-defined constant that represents the cost of not logging the state signal (measurement) at a time step t. Thus, it represents the monetary penalty of losing the audibility at this time step, β is a constant that represents the cost of the delay of executing

the business logic due to a potentially missed condition violation, m is the probability of an encoded condition occurrence in a time step. Thus, the quantity $-\beta \times m$ is the expected penalty for execution delays. These parameters can be determined per use case and will be used if the decision of the agent is not to submit state update on chain ($a_t = 0$).

The multiplication by T_t is to indicate higher penalties with more extended idle periods. For example, the decision of not requesting a state update when the last request was ten time steps ago $T_t = 10$ is worse than the same decision when $T_t = 1$. α and β have the unit of ($\$/timestep$). Thus, the unit of the reward is eventually is ($).

RL Agent Design

Objective Formulation

The agent's goal is to find an optimal policy, π_*, which is the policy that would result in maximum reward throughout the monitoring episodes. In order to learn an optimal policy, the agent should first build an accurate approximation of the optimal action-value function $Q^*(s, a)$, which describes the goodness of taking the action a in a given state s as a scalar that represents the future expected reward after taking that action and then following the optimal policy. When the Q^* function is learned, the optimal policy is obtained through acting greedily with respect to it [11]. Formally:

$$\pi_*(s) = \arg\max_a Q^*(s, a) \tag{3.2}$$

Thus, we wish to approximate $Q^*(s, a)$. The Q-value of a state-action pair is defined as the expected sum of discounted future rewards. The expectation is taken over a policy.

Formally:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{K}\gamma^{t} \times r_{t+1}\middle| s_0 = s, a_0 = a, \right.$$
$$\left. a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)\right]$$

where $Q^{\pi}(s,a)$ is the action value function, K is the episode length, r_t is the reward value at time step t,γ is the discount factor that balances the importance between the immediate reward and future rewards. For example, when set to 1, future rewards are as important as the immediate ones, and the policy is foresighted. When set to 0, only immediate reward is considered. $Q^{\pi}(s,a)$ can be written recursively in terms of the next the next state action pair (s',a') through the Bellman expectation equation:

$$Q^{\pi}(s,a) = \sum_{s',r} P(s', r|s,a)\left[r + \gamma \sum_{a'} Q^{\pi}(s',a')\right]$$

Q^{*} is then defined as the best action-value function across all possible policies.

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$Q^{*}(s,a)$ can also be written recursively through Bellman optimality equation:

$$Q^{*}(s,a) = \sum_{s',r} P(s', r|s,a)\left[r + \gamma \max_{a'} Q^{*}(s',a')\right]$$

Such recursive formulation is useful since it enables evaluating Q^{*} iteratively through dynamic programming as:

$$Q_{k+1}(s,a) = \sum_{s',r} P(s', r|s,a)\left[r + \gamma \max_{a'} Q_{k}(s',a')\right]$$

where Q_k is an estimate of Q at iteration k. As $k \to \infty$, Q_k converges to Q^{*} [11].

The dynamic programming method requires knowledge of the system's dynamics (the state transition distribution P). However, we only have samples of the states. Hence, we use Q-learning [58] which is a sample-based approximation of the Bellman optimality equation:

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha \times \left(\underbrace{r_t + \gamma \max_a Q_k(s_{t+1},a)}_{\text{TD target}} - Q_k(s_t,a) \right) \tag{3.3}$$

Where the temporal difference target (TD target) term represents an intermediate estimate of expected future rewards, which enables online learning from samples [59]. Note that Eq. (3.3) does not include P. Eq. (3.3) assumes tabular representation of each possible state-action pairs. However, this requires us to have a discrete and finite number of states. In our problem, the state space is continuous (due to the real-time pricing signal). Thus, function approximation is required. We choose to use Neural Network (NNs) to provide parametric function approximation of Q^* since they are universal function approximators [60] and have been successfully integrated into RL for continuous state problems [61].

Using parametric function approximation with parameters set θ, Q can be written as:

$$Q(s,a;\theta) = W_{L+1}\sigma(W_L\sigma(W_{L-1}...\sigma(W_1x + b1) + b_{L-1}) + b_L)$$

Where W_l and b_l are the weight matrix and bias vector of layer l, respectively, d_l is the depth of layer l. Hence, $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}$. θ is the set of the network parameters (weights and biases): $\{W_l, b_l\}_{l \in [L+1]}$, where L is the number of hidden layers, σ is the non-linear activation function. We use the Leaky ReLU as the non-linearity function:

$$\sigma(x) = max(0,x) - slope \times min(0,x)$$

where $slope < 0$.

The TD-target with function approximation is then written as

$$Y = r + \gamma \max_a Q(s,a;\theta) \tag{3.4}$$

The loss function is then defined and optimized through Stochastic Gradient Descent (SGD) methods :

$$L(\theta) = 1/n \sum_{i=1}^{n} [Y_i - Q(s_i, a_i; \theta)]^2$$

$$\nabla_\theta L(\theta) = \frac{2}{n} \sum_{i \in [n]} [Y_i - Q(s_i, a_i; \theta)] \cdot \nabla_\theta Q(s_i, a_i; \theta) \tag{3.5}$$

where n is the number of samples to learn from. (3.5) is iteratively executed (every time step) until convergence, Y_i is calculated based on the previous iteration's weights $\bar{\theta}$.

It has not yet been theoretically proven that (3.5) converges to the weights that best represent Q^*. However, there are multiple practical implementations that help in convergence to an estimate, Q_*. Namely, replay buffer, fixed targets, and soft updates. These are the main components of the Deep Q-Learning algorithm that uses NN as function approximators (hence the name Deep Q-Networks), which is discussed in the following section.

Learning the Optimal Policy

Deep Q-Network, or DQN, [62] is a family of algorithms that learn a deep parametric representation of Q^*. Also, These algorithms are model-free, which means that they do not require knowledge of the state transition model. Instead, they learn the state-action values by interaction and then take the best possible action given only the current state of the system. This learning approach enables the agent to act based on real-time signals (on per time step basis). However, there are no theoretical guarantees for the convergence of DQN [63]. Thus, multiple experimental techniques should be followed to help stabilize the training without divergence of the weights. The detailed steps are shown in Algorithm 2.

In Algorithm 2, we initialize two copies of parameters (neural networks weights).

The agent starts the training process by repeatedly generating episodes of experience (lines 3-6) and collecting experience tuples through interactions (lines 8-10). These tuples of state action transitions represent the experience data and are stored in the replay buffer $\mathcal{D}$ to be used in the learning process. In order to make sure that the agent explores the quality of actions in different states, an $\epsilon-$greedy is followed (line 8); The action taken at each time step is selected randomly with probability ϵ, or as the action corresponding to the highest Q-value (the best action learned so far) with probability $1 - \epsilon$. This is done to balance the exploration and exploitation behavior of the agent balance, where random actions represent the exploration behavior, and the best actions represent the exploitation behavior. We follow a decaying schedule of ϵ starting at 1, so that the agent can collect basic knowledge about the environment, and then it decays over time, allowing the agent to exploit accumulated knowledge and follow optimal behavior.

The agent then starts improving the estimate of Q^* values. In line 12, a random minibatch is sampled from the replay buffer $\mathcal{D}$. The TD target values for each tuple in the sample batch is then calculated. The TD targets $Y^{(i)}$ for experience tuple i represents the new estimate of $Q(s^{(i)}, a^{(i)})$ for the state action pair $(s^{(i)}, a^{(i)})$ and is calculated using the target network $\bar{\theta}$. This process is known as experience replay, which greatly helps the stabilization and convergence of the learning process [62]. We then fit θ to $Y^{(i)}$. This fitting can be done through Stochastic Gradient Desccent (SGD) optimizers such as Adam [64] (line 14). The target network parameters are held fixed for $target$ steps to stabilize learning. Then, we softly update the target network parameters towards the first one, so that the target always reflects the most recent knowledge gained (line 15). Soft updates towards the target network also help stabilize the learning. Eventually, $\theta \approx \bar{\theta}$ and convergence is achieved.

Algorithm 2 Oracles controller (agent) training

Input: Environment simulator,

Output: θ: The NN parameters for the approximation Q_*.

1: Initialize parameters of the first network θ randomly.
2: Initialize parameters of second (target) network $\bar{\theta} \leftarrow \theta$.
3: **for** episodes= 1:M **do**
4: Initialize random monitoring period $L \sim U(72, 168)$
5: Initialize state $s_0 = (p_0, T_0)$
6: **for** time step $t = 0 : L$ **do**
7: /**Interaction with the environment**/
8: Select state update action a_t based on ϵ-greedy policy
9: Execute a_t, observe s_{t+1} and r_{t+1}
10: Store the tuple of experience $(s_t, a_t, s_{t+1}, r_{t+1})$ in $\mathcal{D}$
11: /**Updating the estimates**/
12: Randomly sample a minibatch $\mathcal{F} = \{(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}, r_{t+1}^{(i)})\}_{i=1}^{|\mathcal{F}|}$ from $\mathcal{D}$
13: Calculate Q-targets using the target network $Y^{(i)} \leftarrow r_{t+1}^{(i)} + \gamma \max_a Q(s_{t+1}^{(i)}, a; \bar{\theta})$
14: Fit $Q(s^{(i)}, a^{(i)}; \theta)$ to the target $Y^{(i)}$: $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$
15: Every $target$ steps, update the target network $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$

Real time, Light-weight Oracle Management Agent

Algorithm 3 Smart Oracle Manager using Deep Q-Network (SOM-DQN)

Input: The trained NN parameters θ,

Output: Status update request decision,

1: **for** everty time step **do**
2: Obtain the current coin price p_t
3: Calculate the number of time steps since last update T_t
4: Calculate $Q_*(s_t, a_t)$ for both values of a_t (0 and 1) using θ
5: perform $a_t = \arg\max_a Q_*(s_t, a_t)$

After the training convergence of Algorithm 2, the trained parameters can be saved

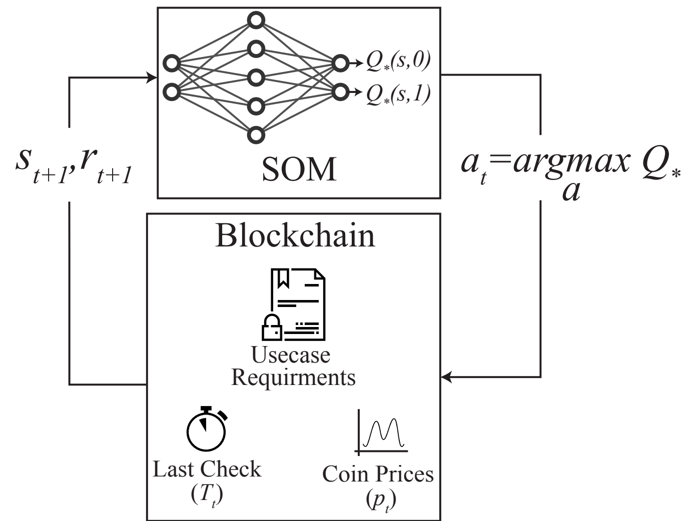

Figure 3.2: Reinforcement-based interaction between the smart oracle manager (DQN agent) and the blockchain network

and used for real-time decision making. The agent illustrated previously in Fig. 3.1 executes Algorithm 3 at every time step in order to set the state update request flag to 0 or 1. Note that the execution of Algorithm 3 is lightweight and efficient (single NN pass). Thus, it can be executed in real-time (every time step).

Fig. 3.2 summarizes the interaction between the proposed agent (smart oracle manager - SOM) and the blockchain network. The SOM receives the state from the blockchain network, where the signal of interest is being monitored and recorded. The maintained Q-network uses this state to predict the optimal action-value function, based on which a greedy action is selected. The reward is determined based on the use case requirements, represented by α and β parameters, the action taken, and the current state (p_t, T_t). This control loop is followed on a time-step basis; the agent responds to new states that arise throughout the monitoring process with optimal actions that are believed to maximize long term reward.

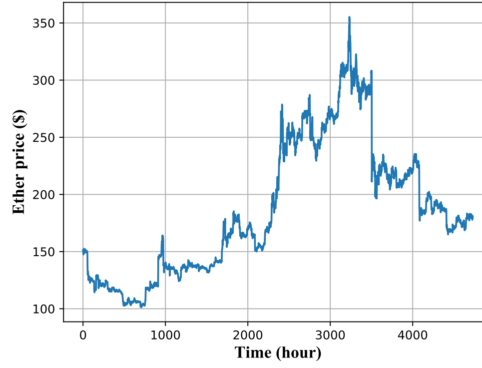

Figure 3.3: Ether pricing data

Performance Evaluation

Experimental Settings

The performance of the proposed method is evaluated using the Ethereum public blockchain settings [26]. The real-world Ethereum coin (Ether) prices starting from January 1st, 2019, and lasting nine months are used (shown in Fig. 3.3) [65]. In each month, a random week is extracted and added to the test set. The starting day, time, and duration of the monitoring process (e.g., shipping date and duration) are modeled as random variables with uniform distributions. Starting day $\sim U(0, 90)$, time $\sim U(0, 23)$, and monitoring period $\sim U(72, 168)$ hours, which corresponds to $3 - 7$ days.

For the reward function In (3.1), the value of N is usually set as an odd number since it simplifies the majority voting aggregation logic. We set $N = 5$. F_{txn} and F_{agg} are calculated based on Ethereum internal computational currency “Gas". Computational operations in Ethereum have Gas expenditure, and the execution cost is predetermined in terms of Gas units [42]. It can be seen from the Ethereum operations list in [26] that a transaction has a basic cost of $21,000$ units of Gas. Also, data storing is $5,000$ units

of Gas (on average, the zeroness of the stored bytes does not change). Sending 32 bytes of data with the transaction is $\sim 1,000$ units of Gas. Hence, F_{txn} requires $27,000$ Gas units. F_{agg} depends on the exact aggregation function used. For example, the majority voting aggregation code contains two main operations, array-max calculation and result storage, which correspond to approximately $5,000$ and $4,000$ Gas units, respectively.

The Gas values mostly depend on the number of non-zero bytes in each transaction/operation result. Thus, we used approximated values. These values can be either estimated using the Ethereum computation operation cost list or through coding the procedure and checking its cost in Ethereum IDE [66]. Users can set the price to pay per Gas unit. We set the price value to 25 Gweis/Gas unit (1Gwei = 10^{-9} Ether) which is an average value that guarantees fast transaction processing by miners [67]. Thus, the status update (on-chain verification) reward can be calculated as follows: $r_t = (-5 \times 27,000 \times 25 \times 10^{-9}) \times p_t - (9,000 \times 25 \times 10^{-9}) \times p_t$.

Regarding the reward of idle (off-chain) action, we set the values of α and β such that the reward of always taking the idle action is equivalent to the reward of always requesting state updates. This configuration indicates that both extreme policies are equally undesirable, and a tradeoff between them is required. One setup that approximately achieves this is $\alpha = \beta = 0.01$, for which the cost of always-on is ≈ 1.2 the cost of and always idle policy, on an average-length episode, with an average coin price. m is set to 0.5 to indicate the total uncertainty regarding the probability of a business logic condition occurrence. Thus, the idle action reward can be calculated as follows: $r_t = (-0.01 - 0.01 \times 0.5) \times T_t$. Since different users have biases towards their preference, we study the effect of multiple α, β, and m choices on the learned policy in dedicated sections.

Table 3.1: Oracle Management System Parameters

Parameter	Value
Learning episodes M	10^4
Discount factor γ	0.9
Exploration rate ϵ	1, with 1×10^{-3} Decay
Q-Network layers	3
Q-Network neurons/layer	$2, 64, 64, 2$
Q-Network learning rate	10^{-5}
Activation function $slope$	0.01
Replay buffer size $\|\mathcal{D}\|$	10^5
Batch size $\|\mathcal{F}\|$	64
Soft update factor τ	0.001
Soft update period $target$	4

As the reward function is part of the environment, the agent in the proposed method does not know nor depend on the parameters defined in this section. The agent can still and can adapt to any different cost structures for other blockchains and different user-based parameters.

Reward Convergence

We run the training Algorithm 2 according to the parameters shown in Table 3.1. These parameters are empirically set; we expect similar architectures to perform similarly. Then, we plot the reward vs. episode number in Fig. 3.4. The shown values are smoothed over a window of 50 episodes.

For the first 1000 episodes, the ϵ value is set to 1 (completely random behavior) in order to have an initial estimate of the random policy, which will form the basis for improved policies. Then, exponential decay of 0.999 is applied to ϵ at each episode in order to smoothly but quickly transition to better policies. The performance (reward) increases rapidly until episode 2000, where the progress starts to get slower as the

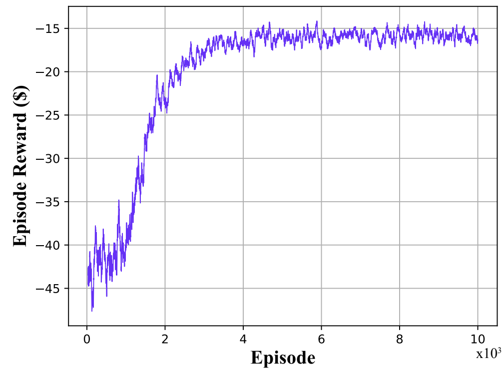

Figure 3.4: The training process, reward throughout episodes

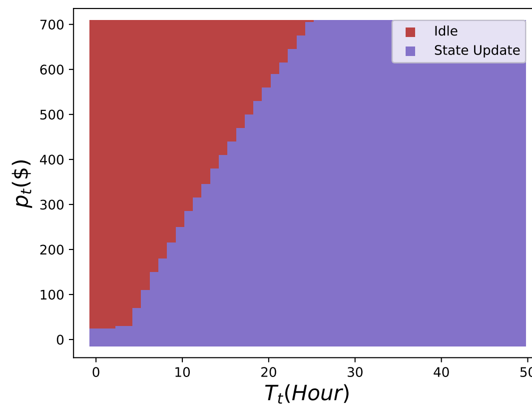

Figure 3.5: The learned policy

estimates get better. Eventually, we reach a convergence at a reward value of $\sim -16\$$.

The learned policy is visualized as a binary heat map in Fig. 3.5 where the state space is represented by the x and y axis. It can be seen that for a given price range, as the time from the last state update increases, the state update becomes more necessary and is required. Also, for a given distance from the last update, higher prices cause the agent to hold for more time steps before requesting a state update. However, after a certain idling period, which is 26 time steps, on-chain verification is required even when the current price is high.

Performance Comparison

To verify the performance of SOM-DQN, we compare it to a set of standard oracle management policies explained in the subsections below:

Always idle / Always on (baseline policies)

The always idle policy does not submit any transaction ($a_t = 0, \forall t$). Thus, it incurs the cost of audibility and latency at all time steps, on all episodes. On the contrary, The always-on policy continuously submits transaction ($a_t = 1, \forall t$), incurring the transaction fees at every time step, across all episodes.

Random Policy

The uniformly random policy is a heuristic that performs a status update randomly on every step. $a_t \sim \mathcal{U}[0, 1], \forall t$. This policy is simple and does provide a tradeoff. However, its "blindness" to the current blockchain state result in suboptimal performance.

Periodic Policy

Another class of followed heuristics is the periodic submission strategy. Such strategies create a transaction every pre-defined number of time steps, ω:

$$a_t = \begin{cases} 1 & \text{if } t \mod \omega = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The period is encoded beforehand, and then followed throughout the testing episodes

Ideal Policy

For the ideal policy, we assume perfect knowledge of future states and solve a deterministic optimization problem for finding the optimal frequency of state update submissions. This solution is not practical since future prices are unknown in advance. However, we provide it as a benchmark. First, we define F, which has the same formula of r but follows a submission rate of ρ.

$$F = \begin{cases} -(N \times F_{txn}) - F_{agg} & \text{if } t \mod \rho = 0 \\ \\ (-\alpha - \beta \times m) \times T_t & \text{otherwise} \end{cases} \tag{3.7}$$

The utility function is defined as $U(\rho) = \sum_{t=0}^{K} F(\rho)$, which is the sum of rewards in an episode. The optimization problem aims to find the optimal rates $\rho_0, \rho_1, \ldots, \rho_E$ that would result in the maximum reward (or equivalently, minimum cost) across all testing episodes:

$$\max_{\rho_0, \rho_1, \ldots, \rho_E} \sum_{e=0}^{E} U(\rho_e) \tag{3.8}$$

The problem in (3.8) is combinatorial and can be solved by means of enumeration (or more efficiently, branch and bound), which results in rates $\rho_0, \rho_1, \ldots, \rho_E$ that will be followed across the testing episodes. Note that the problem could be solved because we assumed prior knowledge of future states to be used in evaluating (3.7).

Results and Discussion

The evaluation method is running each of the policies for the eight testing weeks (eight episodes) previously extracted from the dataset. The criterion is the final accumulated cost.

The accumulated costs are plotted in Fig. 3.6. Each testing episode represents

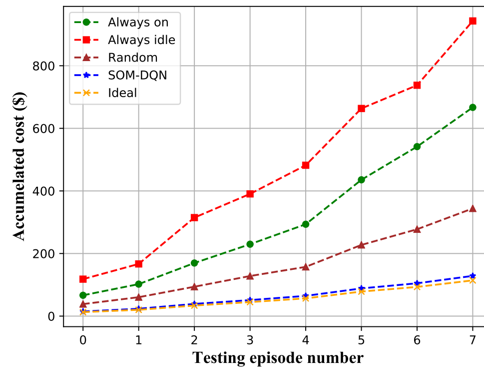

Figure 3.6: Performance comparison between different policies

a monitoring process (e.g., asset shipment, production line, supply chain, and others) with randomly generated lengths (3-7 days). At the end of the testing periods, the final accumulated costs (in $) are $943.1, 667.0, 344.0, 128.6, 114.2$ for the always idle, always-on, random, proposed, and ideal policies, respectively. The total number of status updates requested by each policy, per testing episode, is shown in Table 3.2.

The cost of both of the baseline policies overgrows throughout the episodes. This is expected as they are inefficient extreme solutions performing the same action all the time. We aim to strike a tradeoff between these two. The random policy does save considerable costs and achieves an appreciable tradeoff. However, it does not adapt to changing prices, nor does it take use case requirements into consideration. Specifically, the actions' probabilities stay the same whether the coin price is high or low, and regardless of the time since the last submission. Hence, further potential cost savings that could be achieved, for example, through reducing submission probability during peak hours or increasing it when the idle last submission was long ago.

On the other hand, the proposed policy changes the frequency of state updates to minimize cost. The submission decision is set in real-time according to the learned

Table 3.2: Transactions number for policies

Policy	W0	W1	W2	W3	W4	W5	W6	W7	Tot.
Always Idle	0	0	0	0	0	0	0	0	0
Always on	125	80	140	100	110	155	99	165	974
Random	70	48	65	55	47	74	46	84	489
Proposed	14	10	17	10	12	12	7	15	97
Ideal	13	8	17	12	13	12	7	16	98

policy, which specifies the best action for each price and state configuration by design. Thus, it is adaptive and does approach the ideal cost.

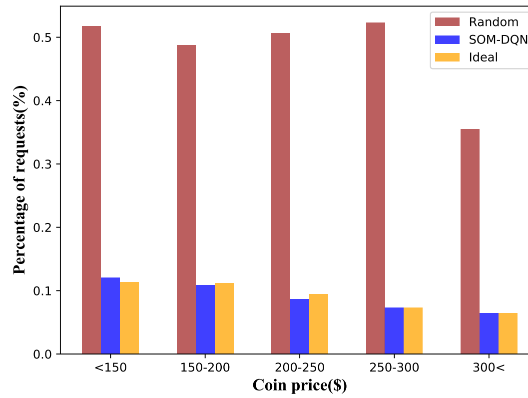

Figure 3.7: Distribution of update requests

Fig. 3.7 gives more insight into the timing requests made by the different policies. For each pricing group in the testing data set, we plot the ratio between the number of state updates requested while the price was in that group and the total number of time steps whose price is in that group. The always-on policy has all bars at 1.0. Thus, it is not depicted. The random policy requests updates around 50% of the time, regardless of the price groups. On the other hand, the proposed methods requests rates are $12.01\%, 10.83\%, 8.64\%, 7.34\%, 6.45\%)$, whereas the ideal case percentages are $11.31\%, , 11.19\%, 9.46\%, 7.33\%, 6.45\%$ for each price group, respectively. As expected,

the requests percentage decreases with higher prices. This pattern exists on both the ideal and proposed solutions. Also, it can be seen that the requests percentage is generally much less than the random heuristic because this is the optimal request rate for the use case requirements set earlier for α,β, and m parameters. In the following section, we study the effect of users' requirements on the performance of SOM-DQN.

Fig. 3.8 shows the cumulative cost for SOM-DQN and the periodic strategy with different submission rates, as well as the ideal case. The monitoring period is set to the maximum (i.e., seven days) in every testing episode to clarify the differences between these policies (the reward difference grows with extended testing periods). It can be seen that some rates offer close performance to the ideal one, which indicates that the simple periodic submission policy with tuned periods can achieve a good tradeoff.

However, it is usually hard to determine these periods prior to the monitoring process unless the prices are accurately predicted, which is rarely guaranteed in the case of cryptocurrencies. Even though a reasonable rate is calculated beforehand, it is still risky to statically follow such a period for extended periods of monitoring since the coin might undergo more significant fluctuations, deeming the original rate inefficient thereafter. In contrast, the real-time capability of SOM-DQN allows it to find near-optimal periods at run-time and to change such period as needed to suit different, even not previously seen, states due to the generalization capability of neural network. In general, SOM-DQN is the closest to the ideal costs.

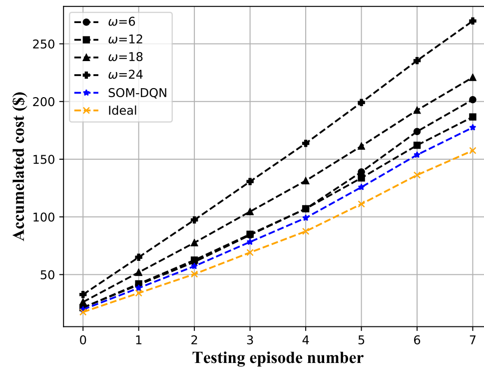

Figure 3.8: Performance comparison with different period-based heuristic policies

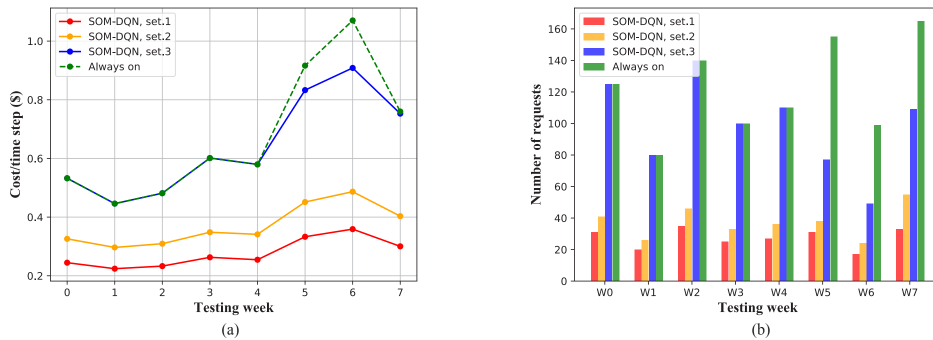

Figure 3.9: Cost comparison of different security features evaluations

Effect of Different Use Case Requirements

Auditability and Latency Preferences

As discussed earlier, the auditability-loss penalty, α, and the business logic latency penalty, β, can be set by users according to how much they evaluate these features. Fig. 3.9.a shows the normalized cost (i.e., divided by the episode length) for every testing week of data for different parameters settings; setting 1 corresponds to $\alpha = \beta = 0.05$, setting 2 corresponds to $\alpha = \beta = 0.1$, setting 3 corresponds to $\alpha = \beta = 0.5$. Note that α and β need not be equal, but were set so for simplicity. We also show the setting where the always-on policy is followed. It can be seen that in general, users incur more

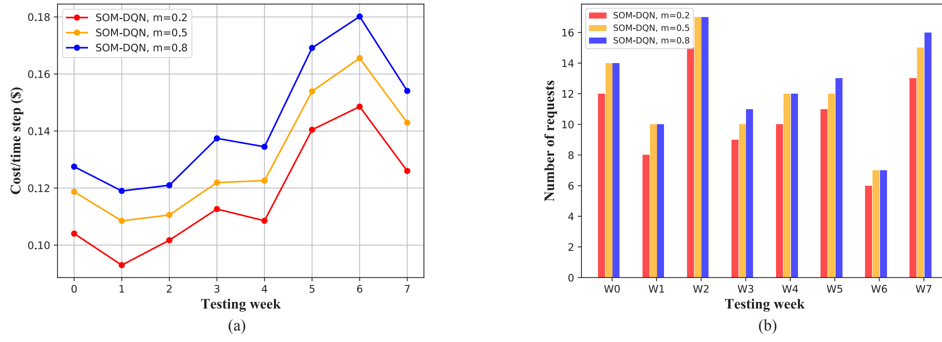

Figure 3.10: Cost comparison for different expected business condition occurrence

cost as they raise their evaluation parameters. When these parameters are large, the cost of the proposed method approaches the always-on policy. This is expected since the more penalties are assigned to being idle, the more requests are submitted by the agent. Eventually, as the parameters evaluation increases, The proposed policy will match the always-on policy, and it will do that earlier for lower prices. In Fig. 3.9.b, we can see that the number of requests of the proposed policy approaches the maximum (the episode length). For set.3, it does reach the always-on in episodes $0-4$, but still has less number of requests on $5, 6, 7$. This is because these last episodes have higher average coin prices. Hence, idling at some time steps is still a better option. The analysis in this section indicates that the always-on policy is actually a special case of the proposed policy that occurs when the user's evaluation for blockchain features exceeds the cost of continuous transaction submission.

Previous Knowledge on Conditions Occurrence

In the part describing the reward of the idle action in Section 3.3.1, we expressed the expected latency penalty as $\mathbb{E}_{Latency \sim p(Latency)}[Latency] = \beta \times m + 0 \times (1-m) = \beta \times m$ where $p(\beta) = m$. We assumed that $m = 0.5$ to indicate the fact that business conditions

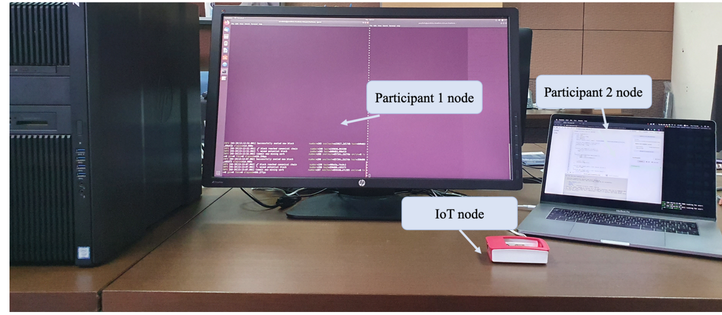

Figure 3.11: Setup of the prototype for a blockchain network implementing the proposed framework

occurrence is uncertain. However, if users estimated m at a different value, the cost will be different. Fig. 3.10 shows the cost at different m values (α and β are reset to 0.01). Higher m leads to higher idle penalties since the user believes that business logic conditions are more likely to occur, and hence idling is mostly not a good action. Thus, the proposed policy will also favor more frequent transaction submission. The analysis in this section shows the importance of precisely estimating m; this can be estimated according to users' previous experience or learned using machine learning techniques, especially that the history of all previous events in an SC is permanently recorded on the blockchain.

Experimental Case Study

To show how the proposed framework can be deployed in practical scenarios, we present a simple case study for a food supply monitoring SC, which is designed as per the template in Algorithm 1 with a single business logic condition that depends on the temperature of monitored goods. Consider the setup shown in Fig. 3.11 that shows three nodes in a P2P Ethereum blockchain network. These nodes operate through the Geth library [68] and utilize web3 library [69] to interact with the Ethereum network

```
22  address="0x1c7075eD0876E55e572f0b1ccF794C9B7180bC83"
23  w3.eth.defaultAccount = w3.eth.accounts[0]
24  contract=w3.eth.contract(address=address,abi=cabi)
25  s=torch.from_numpy(read_state()).float()
26  a=torch.max(agent.qnetwork_local(s),0)[1]
27  if a == 1 :
28      contract.functions.set_flag(True).transact()
```

(a)

```
Submitted transaction          fullhash=0x24b19ab8dc5acae1a7723bfad29dacfb60d864525e54da15a68766f4d4613b1d recipient=0x1c7075eD0876E55e572f0b1ccF794C9B7180bC83
Successfully sealed new block  number=43 sealhash=60f7a0…34a4c4 hash=af9f5a…f210a7 elapsed=14.999s
🔗 block reached canonical chain  number=36 hash=1e3eb2…d7b953
🔨 mined potential block         number=43 hash=af9f5a…f210a7
Commit new mining work         number=44 sealhash=413fe1…97febd uncles=0 txs=0 gas=0     fees=0         elapsed=417.905μs
```

(b)

```
call to FoodSupplier.get_flag

[call] from:0x622961a9f561b6984d9b1b5831b3e1c2be582091 to:FoodSupplier.get_flag() data:0x057...c3cf5
```

transaction hash	call0x622961a9f561b6984d9b1b5831b3e1c2be5820910x1c7075ed0876e55e572f0b1ccf794c9b7180bc830x057c3cf5
from	0x622961a9f561b6984d9b1b5831b3e1c2be582091
to	FoodSupplier.get_flag() 0x1c7075ed0876e55e572f0b1ccf794c9b7180bc83
hash	call0x622961a9f561b6984d9b1b5831b3e1c2be5820910x1c7075ed0876e55e572f0b1ccf794c9b7180bc830x057c3cf5
input	0x057...c3cf5
decoded input	{}
decoded output	{ "0": "bool: true" }
logs	[]

(c)

```
transact to FoodSupplier.set_temp pending ...

[block:345 txIndex:0] from:0xbc5...03534 to:FoodSupplier.set_temp(uint256) 0x1c7...0bc83 value:0 wei data:0xed9...00005 logs:0 hash:0xca9...f7608
```

status	0x1 Transaction mined and execution succeed
transaction hash	0xca9f58bb19d51b9c23206027a246359c3ee2e6419fd4448ade18faa36c2f7608
from	0xbc5b810bfc849ad8ddeceded66a34b11a7e03534
to	FoodSupplier.set_temp(uint256) 0x1c7075ed0876e55e572f0b1ccf794c9b7180bc83
gas	41539 gas
transaction cost	41539 gas
hash	0xca9f58bb19d51b9c23206027a246359c3ee2e6419fd4448ade18faa36c2f7608
input	0xed9...00005
decoded input	{ "uint256 x": "5" }
decoded output	-
logs	[]
value	0 wei

(d)

Figure 3.12: Snapshots for a decision step in a monitoring episode. (a) The main code snippet of the agent using Torch and Web3 libraries to determine the action and set the contract's flag, respectively. (b) a Geth terminal showing the transaction submitted to set the contract's flag being mined into a block. (c) Ethereum Remix interface [66] showing the new value of the flag (d) Remix interface showing the temp value submitted to the contract by the IoT node

(i.e., submit transactions). The owner node (Participant 1) contains the trained agent (a Pytorch model) that takes submission decisions and sets the contract's flag accordingly. The Raspberry Pi node (IoT) can then listen to these decisions and submit the sensory reading as necessary so that the SC can execute the business logic. The state and sensory inputs are emulated, and Participant 2 node is used for chain monitoring purposes. The core parts of a decision step are shown In Fig. 3.12.

System model

There are two types of participants in a service provisioning SC; Service providers that offer services and users that submit requests for services in the form of transactions to the concerned SC. The service providers can be a base station offering spectrum sharing services, IoT devices providing remote data sensing, smart meters offering energy in P2P energy trading, or simply individuals. The blockchain is by design agnostic to the type of participant. We denote the set of N service providers as $\mathcal{I} = \{0, 1, 2, \ldots, N\}$, and the set of transactions $\mathcal{O} = \{0, 1, 2, \ldots, O\}$.

Fig. 4.1 shows the envisioned blockchain-based model that leverages the chained data to continuously improve the policy; At a given point in time, a block would contain performance indicators of the allocation policy being followed by the SC (e.g., participants reviews), which can be cast in terms of an experience tuple of state s, action a, reward r, and next state s', as will be detailed in the following section. Based on these indicators, an optimization epoch is performed to improve the allocation policy and use it in the assignment to be done in the most recent block. The dynamically changing allocation policy, by means of learning, forms a promising paradigm that we investigate in this work.

MDP Formulation

The MDP is a modeling framework of the multi-stage sequential decision-making problem. It is the core framework that RL operates on and aims to solve through learning. A Markov Decision Process (MDP) is defined as tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. At

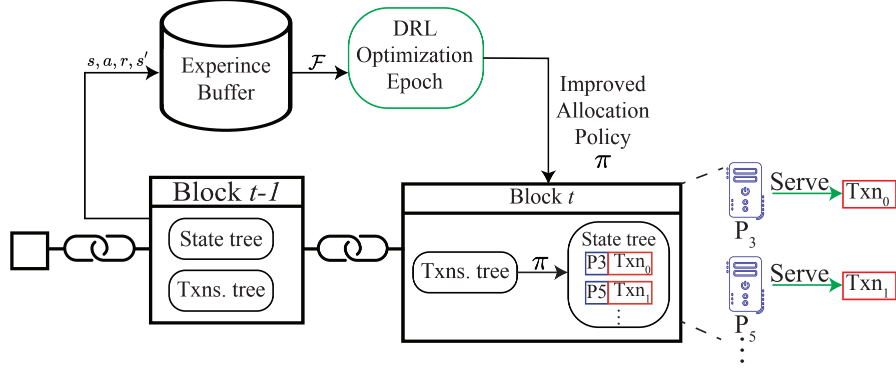

Figure 4.1: Task allocation for transactions (service requests) to service providers (indicated as "P") in the blockchain

every decision epoch t, the agent receives a representation of the environment state $s \in \mathcal{S}$. The agent then executes an action $a \in \mathcal{A}$ using a policy $\pi(a|s)$, receives a reward $r_t \in \mathbb{R}$, and transition to the next state s' with probability $P(s'|s, a) = \mathcal{T}(s, a, s')$. The total feature discounted sum of rewards from time step t until some horizon H is denoted as $R_t = \sum_{t'=t}^{H} \gamma^{t'-t} r_t$, with the discount factor $\gamma \in [0, 1)$. The state-action value function of a specific policy π is defined as $Q^\pi(s, a) = \mathbb{E}_{a \sim \pi, s' \sim \mathcal{T}}[R_t | s_t = s, a_t = a]$. It summarises the sum of rewards resulting from taking the action a in state s, and thereafter following policy π. The state value function $V^\pi(s) = \mathbb{E}_{a \sim \pi}[Q^\pi(s, a)]$ assesses the quality of a state when following the policy π. The advantage function is then defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, which reflects the advantage of taking action a in state s.

The optimal policy maximizes the Q-function $Q^{\pi^*}(s, a) = max_\pi Q(s, a)$ (hereafter referred to as Q^*). The goal of an RL agent is to find such optimal policy through direct interaction with the environment and without explicit or pre-encoded information about it, such as the transition probability $\mathcal{T}$. Q-learning finds such policy through firstly finding Q^* and then acting greedily with respect to it $\pi^*(a|s) = argmax_a Q^*(s, a)$ [11].

We define the MDP in the context of our system model.

State

At a given decision step t, a service request transaction $o \in \mathcal{O}$ should be allocated to a service provider $i \in \mathcal{I}$. The learning agent (i.e., SC) should decide on the allocation based on the state $s_t = \langle \mathbf{l_t}, \mathbf{c_t}, \mathbf{p_t}, d_t, U_t \rangle$ defined as follows:

- $\boldsymbol{l_t} = \{l_t^{(i)}\}$ for $i \in \mathcal{I}$, $l_t^{(i)} \in [0-1]$ is a vector representing the current load for each participant i. A load of 1 indicates a fully loaded participant that cannot serve users.

- $\boldsymbol{c_t} = \{c_t^{(i)}\}$ for $i \in \mathcal{I}$, $c_t^{(i)} \in [0-1]$ is a vector representing the normalized cost of serving the transaction o by each participant i. $c_t^{(i)} = 0$ indicates that the user cannot be served by participant i. 1 represents the highest cost.

- $\boldsymbol{p_t} = \{p_t^{(i)}\}$ for $i \in \mathcal{I}$, $p_t^{(i)} \in [0-1]$ is a vector of the normalized “reputation score" for each participant i. It can be used as a measure for the Quality of Experience (QoE) that can be provided by this participant. Such a score is popular and widely used in blockchain-based systems due to the previously explained features of provenance and immutability, which made the blockchain one of the most reliable platforms for QoE data.

- $d_t \in [0-1]$ The normalized demand of transaction o. A demand of 1 is the maximum serviceable demand by the participants.

- $U_t = \begin{pmatrix} u_{0,1} & u_{0,2} & \dots & u_{0,N} \\ u_{1,1} & u_{1,2} & \dots & u_{1,N} \end{pmatrix}$ Where $u_{0,i} \in \mathbb{N}$ for $i \in \mathcal{I}$ is the smallest number of time steps until a task is released (user service is ended) from participant i.

$u_{1,i} \in [0-1]$ is the the amount of load that will be released (set to free), after $u_{0,i}$ steps, for a participant i.

Such normalization of state elements is standard in service provisioning system in order to focus on the system performance and abstract away units and unit conversions that might be specific to the application. Nonetheless, the normalized elements can always be interpreted or converted back to represent units of interest [56].

Action Space

Based on the state information, the action $\boldsymbol{a_t} = \{a_t^{(i)}\}, a_i \in \{0,1\}$, for $i \in \mathcal{I}$ is taken by the SC. $a_t^{(i)}$ represent whether the participant i is serving the user in time slot t. Note that while it is possible to serve the same user (transaction o) by multiple service providers, we study the case of individual assignments.

State Transition

The state transition of the MDP defines the next state s_{t+1} based on the current state action pairs (s_t, a_t), we model the transition of the state elements as follows:

$$l_{t+1}^{(i)} = \begin{cases} l_t^{(i)} & \text{when } a_t^{(i)} \neq 1, u_{0,i} \neq 0 \\ l_t^{(i)} - u_{1,i} & \text{when } a_t^{(i)} \neq 1, u_{0,i} = 0 \\ l_t^{(i)} + X^{(i)} \times d_t & \text{when } a_t^{(i)} = 1, u_{0,i} \neq 0 \\ l_t^{(i)} + X^{(i)} \times d_t - u_{1,i} & \text{when } a_t^{(i)} = 1, u_{0,i} = 0 \end{cases} \tag{4.1}$$

The set of calculations in (4.1) describe how the load increases or decreases for a participant based on whether it was assigned a new task to serve, or a task is released (i.e., its service time has ended). $X^{(i)} \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is the load increase factor, which is a characteristic of each participant as it depends on its processing power. In other words,

the service request is reflected differently on the participant's capacity. For example, the same task can be insignificant to a workstation participant but causes considerable load on an embedded system participant.

The other components of a given s_t are transitioned according to the following: $c_{t+1}^{(i)} \sim \mathcal{U}(0,1)$ and $d_{t+1} \sim \mathcal{U}(0,1)$. For U_{t+1}, $u_{0,i} = f_0(B)$ where f_0 evaluates the smallest number of time steps until a task is released from participant i. $u_{1,i} = f_1(B)$ where f_1 evaluates the amount that would be released after the u^t steps, which is equal to $d_{t^-} \times X^{(i)}$ (i.e., the load due to assigning the user, at some previous time $t^- < t$, to participant i).

Reward Structure

Since each service provider might have a different cost to service, it is desirable to assign users to low fees providers to save *service costs*. On the other hand, service providers have limited service capacities. In general, we assume that the more loaded the service provider is, the more *operation costs* it endures. Hence, the assignment should aim to also minimize the load across service providers.

In order to have viable assignments, the following constraints should hold:

$$l_t^{(i)} <= 1, \forall i \in \{0, 1, \ldots, N\} \tag{4.2}$$

$$a_t^{(i)} = 1 \implies c_t^{(i)} > 0, \forall i \in \{0, 1, \ldots, N\} \tag{4.3}$$

The reward at times step t, r_t, can then be calculated as a function of the state and action

pair (s_t, a_t) as:

$$r_t = \begin{cases} \underbrace{p_t^{(j)} \times (1 - c_t^{(j)})}_{\text{service cost saving}} + \underbrace{\frac{1}{N}\sum_{i=0}^{N}(1 - l_t^{(i)})}_{\text{operation cost saving}} & (4.2), (4.3) \text{ hold} \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

where $j = i : a_t^{(i)} = 1$. The "service cost saving" term indicates the preference of lower service fees, scaled by the QoE, whereas the operation cost saving term indicates the preference to lower loads across service providers. Note that $r_t \in [0 - 2]$ as it is the sum of two normalized terms.

As the MDP elements are now defined, we explore solution methods that lead to the optimal policy π^*.

RL Agent Design

The optimal Q-function can be written recursively through the Bellman Optimally Equation [11]:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{T}}[r_t + \gamma \max_{a'} Q^*(s', a') | s_t = s, a_t = a] \quad (4.5)$$

Q^* can be iteratively calculated through interaction with the environment using dynamic programming, where at each update iteration k (which is typically a time step t), the following Bellman update is calculated:

$$Q^*_{k+1}(s, a) = \mathbb{E}_{s' \sim \mathcal{T}}[r_t + \gamma \max_{a'} Q^*_k(s', a') | s_t = s, a_t = a]$$

where Q_k is an estimate of Q^* at iteration k. As $k \to \infty$, Q_k converges to Q^* [11].

The Q-function (including the optimal one) can be of very high dimensionality or, as in our case, continuous. Thus, they should be approximated. Neural networks are general function approximators that proved successful in RL domains. Hence, we use a deep $Q-$network $Q(s, a; \theta)$ whose parameters are θ. We optimize those parameters using the

Temporal Difference error (TD-error) loss function, which pushes the parameters in the direction that adequately approximate Q^* [11]. At each iteration k, the loss L is:

$$L_k(\theta_k) = (y_k - Q(s, a; \theta_k))^2 \tag{4.6}$$

where y_k is the TD-target defined as:

$$y_k = r_k + \gamma \max_{a'} Q(s', a'; \theta_k) \tag{4.7}$$

However, optimizing the above objective is likely to diverge or result in poor performance [62]. We utilize collective improvements from the RL community to stabilize learning. Namely, replay buffer, fixed targets, double estimation, and dueling network architecture. As illustrated in [62], keeping a replay buffer of previous experience (i.e., transition tuples s, a, r, s') and then optimizing (4.7) through stochastic gradient descent greatly helps stability. In addition, the parameters used in the TD-target evaluation are frozen to some previous values $\bar{\theta}$ (fixed targets). The loss is then defined as:

$$L_k(\theta_k) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{U}(\mathcal{D})} \left[(y_k - Q(s, a; \theta_k))^2 \right] \tag{4.8}$$

$$y_k = r + \gamma \max_{a'} Q(s', a', \bar{\theta}) \tag{4.9}$$

Note that using the same network in (4.9) to choose the best action a' and to evaluate it can lead to over estimation bias. Thus, it is suggested that the freezed network is used for the evaluation of the action, wheras the online network is used for choosing that action [70], making the TD targets as:

$$y_k = r + \gamma Q(s', \arg\max_{a'}(Q(s', a'; \theta_k)); \bar{\theta}) \tag{4.10}$$

Of specific interest to this paper is the dueling network architecture introduced in [71], which is especially useful when multiple actions are approximately similar, which is the case in this paper. Estimating Q-function for every state-action pair might be impractical and slows-down learning. This is because in many states, the value of many actions might

be either irrelevant or similar. For example, when two service providers have relatively similar states, then assigning one of them should provide some information about the value of assigning the other. We use the dueling network architecture, which employs multi-stream neural network design with two streams, one to estimate the state value function $V(s;\theta,\alpha)$ regardless of the action, and another stream to estimate the advantage function $A(s,a,\theta,\beta)$ where α and β are the parameters of the two streams, respectively. The two streams are then aggregated to provide the $Q-$values of a given state with every possible action, and the action with the high Q value is taken. As illustrated in [71], the simple sum aggregation may suffer from the identifiability issue. Hence, we use the aggregation in (4.11), which provides the best performance empirically.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\alpha) + (A(s,a;\theta,\beta) - b) \tag{4.11}$$

where $b = \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\beta)$. The final algorithm used for training is provided in Algorithm 4. At every decision step (line 7), an assignment a_t is chosen to be either random (with probability ϵ, or the best action known so far (i.e., as determined by the network, with probability $1-\epsilon$). This is known as $\epsilon-$greedy policy, and it allows for balancing the exploration-exploitation in RL agents. Then, the agent observes the next state and the reward. These values constitute a tuple of experience that is stored in the experience replay buffer $\mathcal{D}$ (line 11) and used for optimization (lines 11-13). Finally, the target network is softly updated towards the online network. The parameters are listed in Table 4.1.

Algorithm 4 Service Provisioning Agent

Input: System's parameters,

Output: θ: The NN parameters for the approximation Q^*.

1: Initialize parameters of the online network θ randomly.
2: Initialize parameters of second (target) network $\bar{\theta} \leftarrow \theta$.
3: **for** episodes= 1:M **do**
4: Initialize state $s_t = \langle \mathbf{l_0}, \mathbf{c_0}, \mathbf{p_0}, d_0, U_0 \rangle$
5: **for** time step $t = 0 : L$ **do**
6: /**Interaction with the environment**/
7: Assign a service provider through selecting a_t based on ϵ-greedy policy
8: Execute a_t, observe s_{t+1} and r_{t+1}
9: Store the experience tuple $(s_t, a_t, s_{t+1}, r_{t+1})$ in $\mathcal{D}$
10: /**Updating the estimates**/
11: Randomly sample a minibatch $\mathcal{F} = \big\{(s_t^{(m)}, a_t^{(m)}, s_{t+1}^{(m)}, r_{t+1}^{(m)})\big\}_{m=1}^{|\mathcal{F}|}$ from $\mathcal{D}$
12: Calculate Q-targets using (4.10): $Y^{(m)} \leftarrow \{y^{(m)}\}_{m=1}^{|\mathcal{F}|}$
13: Fit $Q(s^{(m)}, a^{(m)}; \theta; \alpha; \beta)$ to the targets $Y^{(m)}$: $\theta \leftarrow \theta - \delta \nabla_\theta L(\theta)$
14: Every $target$ steps, update the target network $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$

Performance Evaluation

Reward Convergence

Algorithm 1 is executed using the system parameters illustrated in Table 4.1. Fig. 4.2 shows the reward over time, smoothed over a window of 500 steps. It can be seen that the agent learns more intelligent behavior with time, learning to achieve the desired tradeoff that results in the highest normalized reward units. Convergence occurs approximately after 2.0×10^4 episodes at a value of ~ 1.5 units of rewards.

Table 4.1: Task Allocation System Parameters

Parameter	Value
Learning episodes M	3×10^4
Episode length L	50
Service duration h	6
Service providers N	10
Discount factor γ	0.9
Exploration rate ϵ	1, with 5×10^{-4} Decay
Q-Network arch. & layers	1 common, 2 streams, 2 layers/stream
Q-Network neurons/layer	$51, 256$, streams :$(128, 10)$
Q-Network learning rate	10^{-4}
Activation function	Leaky ReLU, 0.01 -ve slope
Optimizer	ADAM
Replay buffer size $\|\mathcal{D}\|$	2.5×10^5
Batch size $\|\mathcal{F}\|$	64
Soft update factor τ	10^{-4}
Soft update period $target$	4

Performance Comparison

Greedy Heuristics

To verify the performance of the proposed method, we compare it to a set of task assignment algorithms currently followed. These include the greedy heuristics family with its three variants: Greedy with respect to *service cost* (GS), which always assigns the user to the service provider with the minimum cost, Greedy with respect to *operation cost* (GO), which always assigns the user to the service provider with the least current load, and Greedy with respect to the reward function in (4.4), denoted as (GR). We plot the performance of these greedy heuristics, along with the random policy baseline, which performs random actions regardless of the state, throughout a 1000 testing episodes in Fig. 4.3.a. The reward value is averaged over every 50 episodes.

The performance of the greedy heuristic is much better than the baseline. Specifi-

cally, The GR variant performs the best across the heuristics. This is because the reward function defined earlier captures the desired tradeoff between service cost and operation cost, albeit in the current time step only, without lookahead or consideration to the assignment in the future states (i.e., how would the load of the assigned participant change and the potential effects on its ability to service future transactions). Acting greedily with respect to the service cost performs slightly better compared to the operation costs. This might be explained by the fact that the operation cost is calculated by averaging the loads across participants. Thus, even if an allocation strategy did not directly optimize for it (as GC does), and caused some high loading of a participant, the effect is alleviated through the averaging process. Nevertheless, The proposed DRL-based allocation scheme considerably outperforms the greedy heuristics. This is mainly because of lookahead characteristic of the RL algorithms that do not necessarily always choose locally optimal solutions, but rather plan to maximize the reward over the long run.

Load-aware Heuristics

Another family of heuristics in service provisioning is load-aware methods, LA(ω), that assign the user to the service provider with the lowest service cost whose load is less than some threshold ω. These are compared with the baseline and the DRL solution in Fig. 4.3.b. In general, the performance is similar to that of the greedy heuristics; a notable pattern is that with lower ω the reward tends to increase slightly. This is because overloading a participant, and hence disallowing it from serving a user, causes a reward of zero. Lower values of ω are less likely to cause such overload. The DRL-based method leverage the learned knowledge about the system dynamics (specifically, $X^{(i)}$ state element) and assign the user to the best service provider even though its load is

high so long as it is not expected to be overloaded.

Planning Methods

We also test Model Predictive Control-based (MPC-based) planning methods [72]. The used MPC variants perform exhaustive search over a horizon η among the combinatorial options $(\boldsymbol{a_0}, \boldsymbol{a_1}, \ldots, \boldsymbol{a_{\eta-1}})$. Then, the first action of the best option is taken. We test with horizons 2 and 3 only as the computation time becomes prohibitive with increasing values. Besides, resetting the environment simulator is also time-consuming. For example, using $\eta = 2$ consumes an order of magnitude more in time compared to DRL at inference time. The results are plotted in Fig. 4.3.c

The performance of the MPC-based method is, while still slightly inferior, comparable to that of DRL. Further, the effect of the horizon value is not evident in the testing cases. This is not unusual as selecting the horizon value is a tunable decision that can be set according to several techniques in the area optimal control [72]. However, a significant drawback of MPC-based methods is their dependence on a planning model. Hence, if a change happens in the environment, the results would be suboptimal due to biased planning, whereas learning-assisted methods adapt to such changes.

To illustrate this, we conduct another experiment shown in Fig. 4.4. For the first 500 episodes, the performance is similar. Then, we introduce a change in the environment that is obscure from both methods. Two service providers undergo an outage; They cannot serve users, and an assignment to them leads to a zero reward. While both methods initially suffer from a hit in performance, the interaction-driven nature of the RL method allows it to modify the network parameters based on recently obtained rewards, and thus adjusts the policy to avoid those participants, regardless of whether

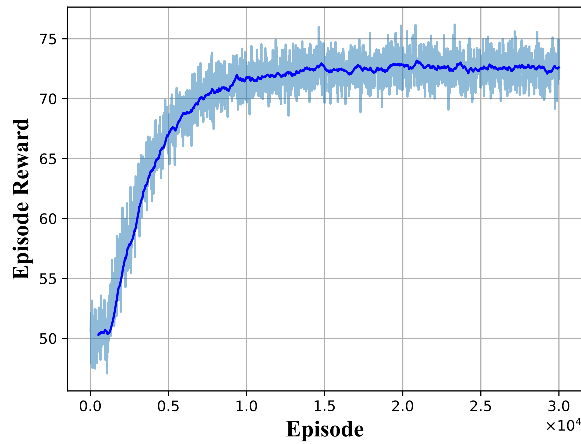

Figure 4.2: The training process, reward throughout episodes

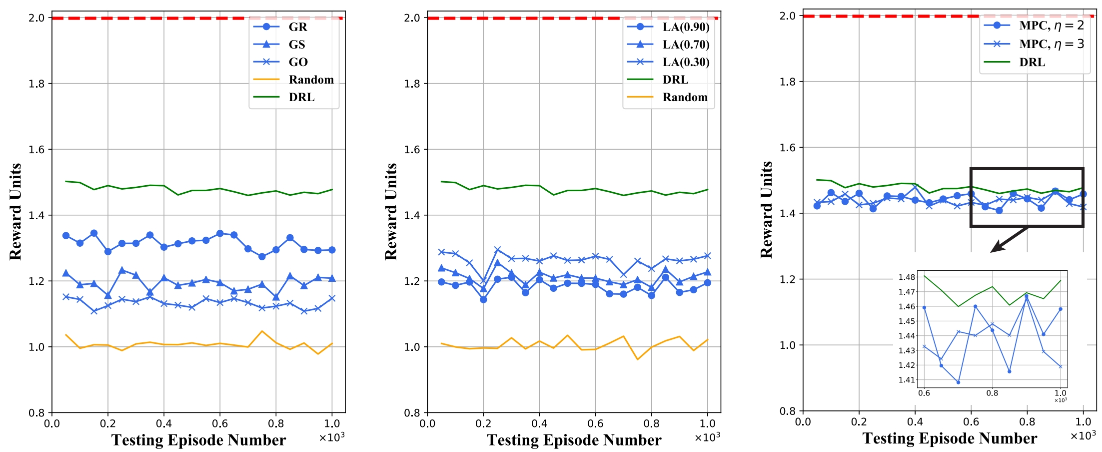

Figure 4.3: Comparison with (a) greedy (b) load-aware (c) planning methods

Figure 4.4: Performance of MPC and DRL during major changes. Red-shaded area indicates a score that is in the same range as heuristic methods. Green-shaded areas indicates a score that is not attained in previous experiments

or not they advertise their services. At episode 1000, another change is introduced; The two participants are replaced by more powerful and affordable ones whose service cost is 10% of the previous ones, and operation cost is negligible. While this change reflects on all methods through higher rewards, it is much more pronounced in the DRL methods, which leverages these two participants more effectively. Hence, online learning is of utmost importance in a dynamic environment.

CHAPTER 5: CONCLUSION

In this thesis, we investigated the integration of blockchain and IoT (The blockchain-enabled IoT paradigm). While the benefits of such integration were highlighted with multiple use cases and examples, several issues have been identified. The first issue is the cost of submitting IoT sensing data to the chain, and the second issue is the inability of static service provisioning SCs to cope with heterogeneous and dynamic blockchain participants/ IoT nodes.

First, we studied the transaction submission cost issue and proposed the use of an RL agent to set this rate in real-time. We then showed that the agent was able to achieve near-optimal transaction submission rate that can be deduced in real-time based on the use case requirements and cryptocurrency prices. Our proposed method was able to achieve the desired tradeoff between low cost and the required security features, outperforming the currently adopted heuristics.

Second, we studied the task allocation issue in service provisioning SCs and proposed modeling the SCs as RL agents that leverage the chained data for online-learning. We established the importance of such a design approach to SCs as opposed to static task allocation and showed that the RL-based approach delivers better performance and adaptability as compared to conventional heuristic and planning methods.

In general, we conveyed that online-learning and data-driven optimization by means of RL provide promising solutions to the blockchain-enabled IoT challenges and constitute a viable path towards the realization of intelligent and secure cyber-physical systems.

Future work can extend the line of important, but monetarily and computationally expensive, IoT/Blockchain integration. The focus can be on embedding the intelligence

in the IoT devices, which might create new potentials such as sensed data-driven submission decisions/task allocation. This, however, requires further investigation and analysis of the emerging multi-agent behavior. In general, applying optimal (distributed) control approaches to optimize different blockchain metrics is a critical direction to be explored.

PUBLICATIONS

- N. Mhaisen, N. Fetais, A. Erbad, A. Mohamed, M. Guizani "To chain or not to chain: a reinforcement learning approach for blockchain-enabled monitoring applications", Future Generation Computer Systems Journal.
- N. Mhaisen, N. Fetais, and A. Massoud, "Secure smart contract-enabled control of battery energy storage systems against cyber-attacks," Alexandria Engineering Journal, Nov. 2019.
- N. Mhaisen, N. Fetais, and A. Massoud, "Real-Time Scheduling for Electric Vehicles Charging/Discharging Using Reinforcement Learning" in IEEE International Conference for Informatic, IoT, and Enabling Technologies (ICIoT), Doha, 2020.
- N. Mhaisen, N. Fetais, A. Erbad, A. Mohamed, M. Guizani "Deep Reinforcement Learning for Optimal Service Provisioning in Blockchain-powered Systems", (Submitted).

REFERENCES

[1] J. Xu, K. Xue, S. Li, H. Tian, J. Hong, P. Hong, and N. Yu, "Healthchain: A Blockchain-Based Privacy Preserving Scheme for Large-Scale Health Data," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8770–8781, Oct. 2019, ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2019.2923525.

[2] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeD-Share: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain," *IEEE Access*, vol. 5, pp. 14 757–14 767, 2017, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2730843.

[3] J. Gao, K. O. Asamoah, E. B. Sifah, A. Smahi, Q. Xia, H. Xia, X. Zhang, and G. Dong, "GridMonitoring: Secured Sovereign Blockchain Based Monitoring on Smart Grid," *IEEE Access*, vol. 6, pp. 9917–9925, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2806303.

[4] Z. Su, Y. Wang, Q. Xu, M. Fei, Y.-C. Tian, and N. Zhang, "A Secure Charging Scheme for Electric Vehicles With Smart Communities in Energy Blockchain," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4601–4613, Jun. 2019, ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2018.2869297.

[5] P. Kochovski, S. Gec, V. Stankovski, M. Bajec, and P. D. Drobintsev, "Trust management in a blockchain based fog computing platform with trustless smart oracles," en, *Future Generation Computer Systems*, vol. 101, pp. 747–759, Dec. 2019, ISSN: 0167739X. DOI: 10.1016/j.future.2019.07.030. (visited on 08/29/2019).

[6] R. V. George, H. O. Harsh, P. Ray, and A. K. Babu, "Food quality traceability prototype for restaurants using blockchain and food quality data index," en, *Journal of Cleaner Production*, vol. 240, p. 118 021, Dec. 2019, ISSN: 09596526. DOI: 10.1016/j.jclepro.2019.118021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0959652619328914 (visited on 11/19/2019).

[7] Y. P. Tsang, K. L. Choy, C. H. Wu, G. T. S. Ho, and H. Y. Lam, "Blockchain-Driven IoT for Food Traceability With an Integrated Consensus Mechanism," *IEEE Access*, vol. 7, pp. 129 000–129 017, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2940227.

[8] H. Hasan, E. AlHadhrami, A. AlDhaheri, K. Salah, and R. Jayaraman, "Smart contract-based approach for efficient shipment management," en, *Computers & Industrial Engineering*, vol. 136, pp. 149–159, Oct. 2019, ISSN: 03608352. DOI: 10.1016/j.cie.2019.07.022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360835219304140 (visited on 11/04/2019).

[9] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," en, *Future Generation Computer Systems*, vol. 88, pp. 173–190, Nov. 2018, ISSN: 0167739X. DOI: 10.1016/j.future.2018.05.046. (visited on 08/29/2019).

[10] J. Heiss, J. Eberhardt, and S. Tai, "From oracles to trustworthy data on-chaining systems," in *Proc. IEEE International Conference on Blockchain (ICBC2019)*, 2019.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," en, *Renewable and Sustainable Energy Reviews*, vol. 100, pp. 143–174, Feb. 2019, ISSN: 13640321. DOI: 10.1016/j.rser.2018.10.014.

[13] E. Mengelkamp, J. Gärttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, "Designing microgrid energy markets: A case study: The brooklyn microgrid," *Applied Energy*, vol. 210, pp. 870–880, 2018.

[14] H. Yao, T. Mai, J. Wang, Z. Ji, C. Jiang, and Y. Qian, "Resource Trading in Blockchain-based Industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019, ISSN: 1551-3203. DOI: 10.1109/TII.2019.2902563.

[15] J. Abou Jaoude and R. George Saade, "Blockchain Applications – Usage in Different Domains," en, *IEEE Access*, vol. 7, pp. 45 360–45 381, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2902501. (visited on 06/10/2019).

[16] Y. Xu, G. Wang, J. Yang, J. Ren, Y. Zhang, and C. Zhang, "Towards secure network computing services for lightweight clients using blockchain," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[17] Y. Xu, J. Ren, G. Wang, C. Zhang, J. Yang, and Y. Zhang, "A blockchain-based nonrepudiation network computing service scheme for industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3632–3641, 2019.

[18] S. Rouhani and R. Deters, “Security, Performance, and Applications of Smart Contracts: A Systematic Survey,” *IEEE Access*, vol. 7, pp. 50 759–50 779, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2911031.

[19] M. Wu, K. Wang, X. Cai, S. Guo, M. Guo, and C. Rong, “A Comprehensive Survey of Blockchain: From Theory to IoT Applications and Beyond,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8114–8154, Oct. 2019, ISSN: 2372-2541. DOI: 10.1109/JIOT.2019.2922538.

[20] C. Shen and F. Pena-Mora, “Blockchain for cities - a systematic literature review,” *IEEE Access*, pp. 1–1, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2880744.

[21] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2566339.

[22] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” in *2017 IEEE International Congress on Big Data (BigData Congress)*, Jun. 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85.

[23] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008, http://www.bitcoin.org/bitcoin.pdf.

[24] F. Tschorsch and B. Scheuermann, “Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies,” *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016, ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2535718.

[25] D. Macrinici, C. Cartofeanu, and S. Gao, "Smart contract applications within blockchain technology: A systematic mapping study," en, *Telematics and Informatics*, vol. 35, no. 8, pp. 2337–2354, Dec. 2018, ISSN: 07365853. DOI: 10.1016/j.tele.2018.10.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0736585318308013 (visited on 12/24/2018).

[26] D. G. Wood, "ETHEREUM: A Secure Decentralized Generalised Transaction Ledger," en, *Ethereum project yellow paper*, p. 32, 2014.

[27] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," *Proceedings of the Thirteenth EuroSys Conference on - EuroSys '18*, pp. 1–15, 2018, arXiv: 1801.10228. DOI: 10.1145/3190508.3190538. [Online]. Available: http://arxiv.org/abs/1801.10228 (visited on 12/24/2018).

[28] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling Blockchain: A Data Processing View of Blockchain Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018, ISSN: 1041-4347. DOI: 10.1109/TKDE.2017.2781227.

[29] ——, "Untangling Blockchain: A Data Processing View of Blockchain Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018, ISSN: 1041-4347. DOI: 10.1109/TKDE.2017.2781227.

[30] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," en, *Technological Forecasting and Social Change*, vol. 144, pp. 1–11, Jul. 2019, ISSN: 00401625. DOI: 10.1016/j.techfore.2019.03.015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0040162518305547 (visited on 11/11/2019).

[31] D. Bumblauskas, A. Mann, B. Dugan, and J. Rittmer, "A blockchain use case in food distribution: Do you know where your food has been?" en, *International Journal of Information Management*, S026840121930461X, Oct. 2019, ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2019.09.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S026840121930461X (visited on 11/11/2019).

[32] P. Helo and Y. Hao, "Blockchains in operations and supply chains: A model and reference implementation," en, *Computers & Industrial Engineering*, vol. 136, pp. 242–251, Oct. 2019, ISSN: 03608352. DOI: 10.1016/j.cie.2019.07.023.

[33] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2566339.

[34] H. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A Survey," *IEEE Internet of Things Journal*, pp. 1–1, 2019, ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2920987.

[35] S. K. Lo, Y. Liu, S. Y. Chia, X. Xu, Q. Lu, L. Zhu, and H. Ning, "Analysis of Blockchain Solutions for IoT: A Systematic Literature Review," *IEEE Access*,

vol. 7, pp. 58 822–58 835, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2914675.

[36] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, "Security services using blockchains: A state of the art survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 858–880, 2019, ISSN: 2373-745X. DOI: 10.1109/COMST.2018.2863956.

[37] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2019, ISSN: 2168-2216. DOI: 10.1109/TSMC.2019.2895123.

[38] S. Moin, A. Karim, Z. Safdar, K. Safdar, E. Ahmed, and M. Imran, "Securing IoTs in distributed blockchain: Analysis, requirements and open issues," en, *Future Generation Computer Systems*, vol. 100, pp. 325–343, Nov. 2019, ISSN: 0167739X. DOI: 10.1016/j.future.2019.05.023.

[39] I. Makhdoom, M. Abolhasan, H. Abbas, and W. Ni, "Blockchain's adoption in IoT: The challenges, and a way forward," en, *Journal of Network and Computer Applications*, vol. 125, pp. 251–279, Jan. 2019, ISSN: 10848045. DOI: 10.1016/j.jnca.2018.10.019. (visited on 08/29/2019).

[40] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures," *IEEE Access*, vol. 7, pp. 82 721–82 743, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2924045.

[41] F. Li, A. Shinde, Y. Shi, J. Ye, X. Li, and W. Song, "System Statistics Learning-Based IoT Security: Feasibility and Suitability," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6396–6403, Aug. 2019, ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2897063.

[42] G. Wood and A. M. Antonopoulos, *Mastering Ethereum: Building Smart Contracts and DApps*, en, 1st. O'Reilly Media, 2018, ISBN: 978-1-4919-7194-9.

[43] X. Liu, K. Muhammad, J. Lloret, Y.-W. Chen, and S.-M. Yuan, "Elastic and cost-effective data carrier architecture for smart contract in blockchain," en, *Future Generation Computer Systems*, vol. 100, pp. 590–599, Nov. 2019, ISSN: 0167739X. DOI: 10.1016/j.future.2019.05.042. (visited on 08/31/2019).

[44] *Provable Documentation*, http://docs.provable.xyz/, 2019. [Online]. Available: http://docs.provable.xyz/ (visited on 08/30/2019).

[45] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, *Town crier: An authenticated data feed for smart contracts*, Cryptology ePrint Archive, Report 2016/168, 2016.

[46] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Capkun, "Tls-n: Non-repudiation over tls enabling ubiquitous content signing," Jan. 2018. DOI: 10.14722/ndss.2018.23282.

[47] S. Ellis, A. Juels, and S. Nazarov, "Chainlink: A decentralized oracle network," *white paper*, 2017.

[48] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A Decentralized Blockchain Oracle," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communica-*

tions (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Jul. 2018, pp. 1145–1152. DOI: 10.1109/Cybermatics_2018.2018.00207.

[49] M. Merlini, N. Veira, R. Berryhill, and A. Veneris, "On Public Decentralized Ledger Oracles via a Paired-Question Protocol," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, pp. 337–344. DOI: 10.1109/BLOC.2019.8751484.

[50] P. Danzi, A. E. Kalor, C. Stefanovic, and P. Popovski, "Delay and Communication Tradeoffs for Blockchain Systems With Lightweight IoT Clients," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2354–2365, Apr. 2019, ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2019.2906615.

[51] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song, "Performance Optimization for Blockchain-Enabled Industrial Internet of Things (IIoT) Systems: A Deep Reinforcement Learning Approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3559–3570, Jun. 2019, ISSN: 1551-3203. DOI: 10.1109/TII.2019.2897805.

[52] Z. Xiong, Y. Zhang, C. Nguyen, D. Niyato, P. Wang, and N. Guizani, "The best of both worlds: A general architecture for data management in blockchain-enabled internet-of-things," *IEEE Network*, Jan. 2020.

[53] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online Deep Reinforcement Learning for Computation Offloading in Blockchain-Empowered Mobile Edge Computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8,

pp. 8050–8062, Aug. 2019, ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2019.2924015.

[54] H.-S. Lee, J.-Y. Kim, and J.-W. Lee, “Resource Allocation in Wireless Networks With Deep Reinforcement Learning: A Circumstance-Independent Approach,” *IEEE Systems Journal*, pp. 1–04, 2019, ISSN: 2373-7816. DOI: 10.1109/JSYST.2019.2933536.

[55] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, “Deep Reinforcement Learning for User Association and Resource Allocation in Heterogeneous Cellular Networks,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, Nov. 2019, ISSN: 1558-2248. DOI: 10.1109/TWC.2019.2933417.

[56] M. Cheong, H. Lee, I. Yeom, and H. Woo, “SCARL: Attentive Reinforcement Learning-Based Scheduling in a Multi-Resource Heterogeneous Cluster,” *IEEE Access*, vol. 7, pp. 153 432–153 444, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2948150.

[57] G. Ciatto, A. Maffi, S. Mariani, and A. Omicini, “Towards Agent-Oriented Blockchains: Autonomous Smart Contracts,” in *Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection*, Y. Demazeau, E. Matson, J. M. Corchado, and F. De la Prieta, Eds., Cham: Springer International Publishing, 2019, pp. 29–41, ISBN: 978-3-030-24209-1.

[58] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[59] H. Yu, A. R. Mahmood, and R. S. Sutton, “On generalized bellman equations and temporal-difference learning,” *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 1864–1912, 2018.

[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[61] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, “Reinforcement learning for control: Performance, stability, and deep approximators,” *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[62] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, p. 529, Feb. 2015. [Online]. Available: `https://doi.org/10.1038/nature14236`.

[63] Z. Yang, Y. Xie, and Z. Wang, “A Theoretical Analysis of Deep Q-Learning,” *arXiv:1901.00137 [cs, math, stat]*, May 2019, arXiv: 1901.00137. [Online]. Available: `http://arxiv.org/abs/1901.00137` (visited on 11/06/2019).

[64] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: `1412.6980 [cs.LG]`.

[65] *Ether price dataset*, `https://www.cryptodatadownload.com/data`, 2019. [Online]. Available: `https://www.cryptodatadownload.com/data/`.

[66] *Remix - Ethereum IDE*, `https://remix.ethereum.org/`, 2019. [Online]. Available: `https://remix.ethereum.org/`.

[67] *Ethereum Gas Price Tracker*, https://etherscan.io/gastracker, 2019. [Online]. Available: https://etherscan.io/gastracker (visited on 11/12/2019).

[68] *Geth Documentation*, https://geth.ethereum.org/docs/, 2020. [Online]. Available: https://geth.ethereum.org/docs/.

[69] *Web3.py documentation*, https://web3py.readthedocs.io/en, 2020. [Online]. Available: https://web3py.readthedocs.io/en.

[70] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[71] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1995–2003.

[72] D. P. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.