# Scaling Blockchain using Codes and DRL based Approach for Blockchain and UAV

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Doctor of Philosophy*
*in*
*Electronics and Communication Engineering*

by

Divija Swetha Gadiraju
2018802001
divija.swetha@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
September 2023

International Institute of Information Technology
Hyderabad, India

# CERTIFICATE

It is certified that the work contained in this thesis, titled "Scaling Blockchain using Codes and DRL based Approach for Blockchain and UAV" by Divija Swetha Gadiraju, has been carried out under my supervision and is not submitted elsewhere for a degree.

| | |
|---|---|
| Date | Adviser: Dr. Lalitha Vadlamani |

*Dedicated to all my well wishers*

# Acknowledgments

The successful outcome of this thesis required a lot of support and encouragement from many people. I am fortunate to have the guidance and assistance till the completion of my PhD thesis work. I extend my gratitude to all of them.

First and the foremost I want to thank my advisor Dr Lalitha for accepting me as a student under her guidance and for considering my research interests. I thank Prof Vaneet Aggrawal for his guidance throughout the years. Without his valuable guidance and support this work would not have been possible. I am very grateful to Prof Vaneet for teaching me many valuable things during my stay at Purdue. I also thank Dr Vijay Shah for his guidance and encouragement.

I would like to thank Qualcomm for awarding me Qualcomm Innovation Fellowship (QIF), 2019 India. I thank Dr. Tanumay Datta and Mr. Gowrisankar Somichetty for their mentorship throughout the duration of the fellowship. QIF has provided me the opportunity to interact and learn from researchers in industry and in academia. It has been a remarkable experience.

I thank Science and Research Board (SERB), India for providing the funding necessary for the Overseas Doctoral Fellowship (OVDF), 2020 to visit Purdue University.

I extend my deepest gratitude to all my friends and well wishers for their constant encouragement and their helpfulness. I am fortunate to get the support from my friends during this journey. Also, I would like to extend my sincere regards to all my colleagues and lab members.

# Abstract

Blockchain and Reinforcement Learning (RL) are two game-changing research areas that have received a lot of attention recently. In recent years, significant advances in RL have resulted in tremendous success in solving various sequential decision-making problems in machine learning. The two most successful RL applications are discussed in this work, unmanned aerial vehicles, and blockchain. Blockchain is a distributed ledger technology with its first application in Bitcoin. The main challenge in blockchain-based cryptocurrencies is to provide a distributed trust environment with high security like in a centralized financial system. The current throughput of Bitcoin is around 4 to 7 transactions per second and confirmation latency is about one hour. If Bitcoin has to go mainstream, the throughput has to be in the order of thousands of transactions per second with very low latency in the order of a few seconds. Recent advances in blockchain research proposed consensus algorithms that scale bitcoin, such as sharding and Prism-based blockchain. However, the security of Bitcoin is very high that it can tolerate up to 50% adversarial nodes and avoids double spending attacks. The current blockchain size is over 260 GB and is growing at an astonishing rate imposing a huge storage requirement on the nodes. Recent developments improving the Bitcoin consensus have shown that there is a trade-off between decentralization, scaling, and security. In order to scale blockchain, we leverage coding theory and RL in this thesis.

Due to the increasing storage requirement for blockchains, the computation can be afforded by only a few miners. Sharding has been proposed to scale blockchains so that the storage and transaction efficiency of the blockchain improves at the cost of a security guarantee. Incorporating coding theory into existing consensus algorithms has demonstrated improvements in terms of storage efficiency and low latency. A Secure-Repair-Blockchain (SRB) is proposed which aims to decrease the storage cost at the miners. In addition, SRB also decreases the bootstrapping cost, which allows for new miners to easily join a sharded blockchain. In order to reduce storage, coding-theoretic techniques are used in SRB. In order to decrease the amount of data that is transferred to the new node joining a shard, the concept of exact repair secure regenerating codes is used. The proposed blockchain protocol achieves lower storage than those that do not use coding and achieves lower bootstrapping costs as compared to the different baselines.

Prism is a recent blockchain algorithm that achieves the physical limit on throughput and latency without compromising security. However, like the traditional blockchain systems, Prism

also has a trade-off between security, latency, and cost. In recent days, reinforcement learning approaches are investigated in traditional blockchains, to improve performance. In this work, we apply Deep Reinforcement Learning (DRL) to one of the promising blockchain protocols, Prism, to optimize its performance. We propose a Deep Reinforcement Learning-based Prism Blockchain (DRLPB) scheme which dynamically optimizes the parameters of Prism blockchain and helps in achieving a better performance. In DRLPB, we apply two widely used DRL algorithms, Dueling Deep Q Networks (DDQN) and Proximal Policy Optimization (PPO). This work presents a novel approach to applying DDQN and PPO to a blockchain protocol and comparing the performance. The analysis of Prism in terms of latency, and security level considering other blockchain parameters is provided. Using the analysis, the DRLPB scheme adapts the Prism blockchain parameters to enhance the security upto 84% more than Prism, while still preserving the performance guarantees of Prism.

The recent advancements in the field of Internet of Things (IoT) motivate the development of a secure infrastructure for storing and sharing vast amounts of data. Blockchain, a distributed and immutable ledger, is best known as a potential solution to data security and privacy for IoT. The scalability of blockchain, which should optimize the throughput and handle the dynamics of the IoT environment, becomes a challenge due to the enormous amount of IoT data. The critical challenge in scaling blockchain is to guarantee decentralization, latency, and security of the system while optimizing the transaction throughput. this paper presents a deep reinforcement learning (DRL)-based performance optimization for blockchain-enabled IoT. We consider one of the recent promising blockchains, Prism as the underlying blockchain system because of its good performance guarantees. We integrate the IoT data to Prism Blockchain and optimize the performance of the system by leveraging Proximal Policy Optimization (PPO) method. The DRL method helps to optimize the blockchain parameters like mining rate and mined blocks to adapt to the environment dynamics of the IoT system. Our results show that the proposed method can improve the throughput of Prism blockchain based IoT systems while preserving Prism performance guarantees. Our scheme can achieve 1.5 times more system rewards than IoT integrated Prism and improve the average throughput of the system by about 6,000 transactions per sec.

Unmanned aerial vehicles (UAVs) are widely used for missions in dynamic environments. DRL can find effective strategies for multiple agents that need to cooperate to complete the task. The challenge of controlling the movement of a group of UAVs is addressed by Multi-Agent Deep Reinforcement Learning (MARL). The collaborative movement of the UAV fleet can be controlled centrally and also in a decentralized fashion, which is studied in this work. We consider a dynamic military environment with a group of UAVs, whose task is to destroy the targets while avoiding obstacles like mines. The UAVs inherently come with a limited fuel capacity directing our research to focus on the minimum task completion time. GLIDE, a continuous-time based PPO algorithm is leveraged in which the UAVs coordinate among

themselves and communicate with the central base to choose the best possible action. The simulator called UAV SIM is developed for our experimentation in which the mines are placed at random locations unknown to the UAVs at the beginning of each episode. The performance of the proposed scheme is evaluated through extensive simulations and a comparison of the centralized action control and the decentralized action control is presented.

# Contents

# List of Figures

# List of Tables

# Chapter *1*

# Introduction

## 1.1 Introduction

The future of various emerging technologies rely on the advancement in blockchain and reinforcement learning research. Blockchain has various applications in IoT, health care and privacy, in addition to cryptocurrencies [2]. The main challenge in cryptocurrencies is to provide a distributed trust environment with high security like in a centralized financial systems. Bitcoin is a decentralized cryptocurrency which involves a peer-to-peer (P2P) network. The current throughput of Bitcoin is around 4-7 transactions per second and confirmation latency about one hour [3]. In order to make the system reliable, each user has to give a cryptographic proof for a valid transaction. This proof involves digital signature verification and the amount of bitcoins involved in the transaction. Bitcoin also preserves the anonymity of users by identifying each user by a public key address [4].

If Bitcoin has to go mainstream, the throughput has to be in the order of thousands of transactions per second with very low latency in the order of a few seconds. However, the security of Bitcoin is very high that it can tolerate upto 50% adversarial nodes and avoids double spending attack [3]. The current blockchain size is over 260 GB and is growing at an astonishing rate imposing a huge storage requirement on the nodes. Recent developments

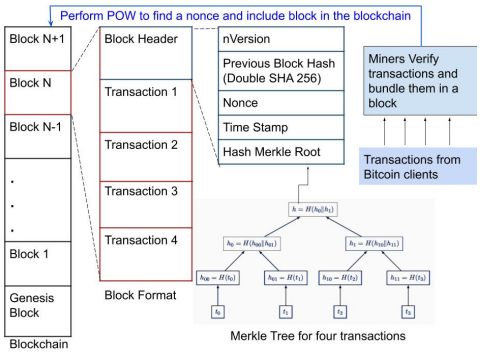



Figure 1.1: An illustration of the blockchain and mining process

improving Bitcoin consensus have shown that there is a trade-off between decentralization, scaling and security [5]. Incorporating coding theory into existing consensus algorithms have demonstrated improvements in terms of storage efficiency and low latency [6, 7]. The consensus protocols can be improved by leveraging Reinforcement Learning methods [8, 9].

In order to prevent double-spending by detecting it whenever it occurs, a database consisting of the history of transactions is used, called the blockchain. Every node in the network stores the blockchain and hence called a distributed ledger. Any node in the network can extend the blockchain by adding a block to the existing blockchain by solving a computationally hard problem (cryptographic puzzle). This process is called mining and the nodes solving the puzzle are called miners. An illustration of adding a block to blockchain is shown in Fig 1.1. The hash of the previous block is stored in the block header (hash pointer) along with the Merkle root of the transactions. A miner finds a nonce and appends the block with a timestamp to the blockchain. Blockchain has numerous applications such as smart contracts, e-governance, healthcare and IoT [4]. In Bitcoin, every node runs a consensus protocol to validate a transaction. Currently, Bitcoin processes about 4-7 transactions per second (TPS) with a block size of 1 Mb, while Visa can manage about 1669 TPS [4]. The delay is because of the time taken to process a transaction and the time taken to reach consensus [10]. Throughput is defined as increase in TPS with increase in the number of nodes and scale-out is a linear increase in throughput with increasing network nodes. Consensus is slowed with an increase in the number of nodes as every node has to validate every transaction. Also, the computation burden to solve the proof-of-work (PoW) puzzles gets harder with time, which drops the throughput. Proof-of-Stake (PoS) has many advantages over PoW and could make the protocol faster.

Nakamoto consensus [4] used in bitcoin happens in a linear manner i.e. throughput decreases or is constant with increase in the number of nodes. The current blockchain has grown to around 260 GB as of January, 2020 and is growing at an astonishing rate [11]. This imposes high storage capacity requirements on miners which, only few miners can afford and this could lead to centralization [10].

### 1.1.1 Need for Scaling Blockchain and Role of Coding Theory

Scaling blockchain can be done in two ways namely, on-chain scaling and off-chain scaling [10, 12, 13]. On-chain scaling solutions aim at increasing the capacity by handling more transactions in a given time. Sharding and block size increase in consensus are on-chain scaling solutions [10]. On the other hand, off-chain solutions process the micro-transactions outside the blockchain network and include only the final and important transactions on the ledger. We focus only on on-chain scaling improvements in blockchain protocols.

#### 1.1.1.1 Block Size Increase

A method to improve their throughput is by increasing the block size so that more transactions can be included in a block. This comes with a disadvantage of increasing the propagation delay which has a huge impact on block chain forking [10]. But, the propagation delay can be significantly reduced using coding techniques. In [14], the authors propose using rateless erasure codes to reduce the propagation delay and scale the blockchain. Although we gain in terms of propagation delay, the orphan rate increases with increasing block sizes. Another challenge with block size increase is the amount of computations performed by a node per block becomes very high.

#### 1.1.1.2 Sharding

Sharding is a technique employed to scale the throughput. Sharding is similar to replication coding which splits the overhead of processing transactions among multiple groups of nodes (committees/shards) which operate in parallel speeding up the consensus process [8, 10, 15, 16]. Each committee proposes a state block which has a Merkle root of the transactions and is also verified by all the committee members [17]. This final state block is appended to the blockchain by the leader. The leader is the node who solves the PoW puzzle and is elected for an epoch. The Byzantine consensus protocol used can tolerate up to 1/3 of the total nodes to be malicious in an authenticated setting (with digital signature verification) with asynchronous communication channels [1]. The sharding protocols require a linear amount of communication. In [1], Information Dispersal Algorithm (IDA) is used for message gossiping. They use error correcting codes for message propagation in a sharding regime and achieve good gossiping guarantees for large messages. The message is divided equal sized chunks by the sender and then Reed-Solomon erasure codes are applied to create an additional parity chunk. If the sender is honest, the original message can be reconstructed from any subset of these chunks. The authors of [7] proposed an architecture, called Secure Fountain (SeF), which is based on fountain codes and enables any full node to encode validated blocks into a small number of coded blocks, thereby reducing its storage costs by orders of magnitude. The authors in [7], propose using rateless codes for blockchain storage and for bootstrapping a new node in the network. The work mainly focuses on making the protocol secure against adversarial nodes during the bootstrapping phase of a new node in the network. Moreover, this approach helps in pruning the blockchain network in such a way that even if the network consists of all pruned nodes and light clients (no archival nodes), we can still rebuild the entire blockchain in a decentralized way. A sub-chain of the blockchain is encoded into coded blocks and each node stores the header chain. A new node joining the network collects coded blocks from any arbitrary node and by using an error-resilient peeling decoder, recovers the entire blockchain. During the decoding phase, the decoder discards the coded block which is malicious by comparing the header chain stored with the decoded header.

#### 1.1.1.3 Coded Sharding

The information-theoretic upper bounds on security, throughput and storage efficiency can be achieved by coded sharding [6]. A coded shard is an encoded linear combination of uncoded sub-chains and the Lagrange polynomial coefficients ensure that the verification function is oblivious. The work mainly focuses on intra-shard protocol.

#### 1.1.1.4 Coded Merkle Tree

Light nodes are the nodes which download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid [4]. Light nodes also have limited storage and computational resources. A fraud and data availability proof scheme is presented in [18] which enables security guarantees of light clients almost at the level of a full node. They use two dimensional Reed-Solomon code to provide the incorrect coding proofs. In [19], the authors propose a Coded Merkle Tree (CMT) structure which leverages erasure codes. Based on the CMT structure, the authors propose a new data availability verification system named SPAR for light clients. Moreover, SPAR uses a hash aware peeling decoder which achieves linear decoding complexity.

#### 1.1.1.5 Other Methods to Scale Blockchain

In Nakamoto consensus the longest blockchain rule is followed in which, only the blocks on the longest blockchain will eventually be adopted by honest miners, and other honest blocks are orphaned. In order to avoid forking, the mining rate for new blocks creation is set longer than the latency for propagating a block to most miners in the network. The Prism protocol [20] is a structured-DAG blockchain with a proposer block and many voter chains. The proposer blocks reference transactions to include in the ledger. The voter blocks vote on these proposer blocks. By running many parallel voter blockchains, the network bandwidth is fully utilized. A miner does not know if the new block will be a proposer or voter. A random sortition procedure is followed using hash function which determines the post-mining role of each block. The authors in Prism show that the latency lower bound (speed of light propagation delay) and the throughput lower bound (network capacity) are achievable. Prism achieves high throughput, low latency and maintains the security like in bitcoin protocol.

#### 1.1.1.6 Prism Blockchain

In Prism [20], an approach to decouple the functionality of the blockchain is presented and it achieves a much higher performance than the existing schemes. Prism has the throughput of 70,000 transactions per sec and latency is the network communication delay, reaching the physical limits of the system [21]. Prism follows the longest chain protocol and achieves good

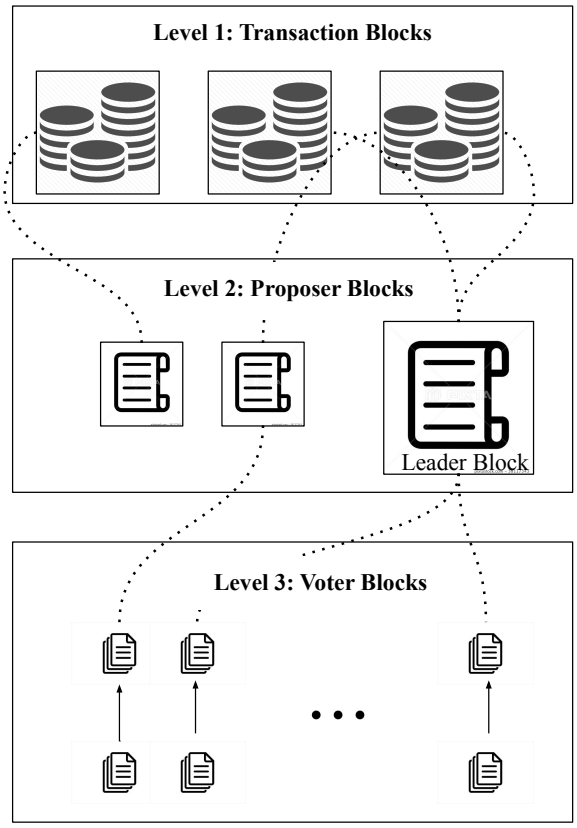



Figure 1.2: An illustration of Prism Blockchain

security guarantees. The performance of Prism [20], is limited by the attributes of the underlying network. Latency is limited by the propagation delay and by the confirmation reliability. In Prism, the blocks are categorized into three types of blocks. For ease of understanding, we represent them as levels of blocks, which are the proposer blocks, the transaction blocks, and the voter blocks as shown in Fig. 5.2. A miner's block is randomly sortitioned using cryptographic sortition into one of the three types of blocks, and if it is a voter block, it will be further sortitioned into one of the $v$ voter trees.

**Level 1: Transaction Blocks**: To decouple security from throughput, separate transaction blocks are employed in Prism. They consists of an ordered list of transactions, drawn from a memory pool, are independent (without a parent block), and are mined at a faster rate. The number of transactions confirmed per sec in Prism depends on transaction blocks.

**Level 2: Proposer Block**: Proposer blocks are the elected leader blocks. Proposer tree is built based on the longest-chain protocol. Proposer blocks choose as their parent block and contain a list of reference links to transaction blocks (which contains transactions), as well as a single reference to a parent proposer block. The final confirmed sequence of proposer blocks

is the leader sequence and is determined by the voter trees. The reliability of Prism depends on Proposer blocks.

**Level 3: Voter blocks**: Voter trees are built using the longest-chain protocol. It contentsa list of references to proposer blocks called votes. The number of voter chains, $v$ considered in Prism experiments is 1000 [21]. The leader voter block at each level is the one that has the highest number of votes among all the proposer blocks at the same level. The security of Prism depends on voter blocks and the number of votes received by them.

The analysis of [21] has an adversary hash power less than 50%,($\beta \leq 0.5$), and the mining rate in each of the voter trees is kept small to minimize forking. The security of Prism depends on the mining rate of voter blocks, the number of voter chains and their votes. There is a tradeoff between the latency, mining rate, and the number of voter chains. This also implies that latency is proportional to the propagation delay and independent of the reversal probability of each voter block.

## 1.2 UAV Action Control

Unmanned aerial vehicles (UAVs), are employed extensively in missions involving navigating through unknown environments, such as wildfire monitoring [22], target tracking [23], and search and rescue [24], as they can host a variety of sensors to measure the environment with relatively low operating costs and high flexibility. Most research on UAVs depends on the target model's accuracy or prior knowledge of the environment [25]. A DRL agent autonomously learns an optimal policy to maximize its rewards through its interactions with the environment. The flight environment for a UAV is usually local or completely unknown during online path planning. Hence, the agent has to react to the dynamic environment using incomplete information, which is the key challenge in UAV path planning [26]. The agent has to react to the dynamic environment using incomplete information, which is the key challenge in UAV path planning [27]. In [28], DRL was used with independence on environment model, and prior knowledge, solving the online path planning problem by trial and error interactions with its environment. Model-free RL methods [29] and the Q learning methods [30] have gained recent popularity. We focus on one of the most important applications of UAV, which is target tracking and obstacle avoidance.

The application of UAVs in military or civil fields such as reconnaissance, strike, rescue, and early warning [22] involves path planning in dynamic environments and is challenging as the UAV has to avoid obstacles that are not known to it beforehand [31]. A single agent in such an environment might lack the fuel capacity to accomplish the tasks efficiently. Hence multiple UAVs are employed to coordinate and complete the task. Interestingly, the majority of successful applications in DRL literature involve the participation of multiple agents, called multi-agent reinforcement learning (MARL). MARL specifically addresses the sequential decision-making

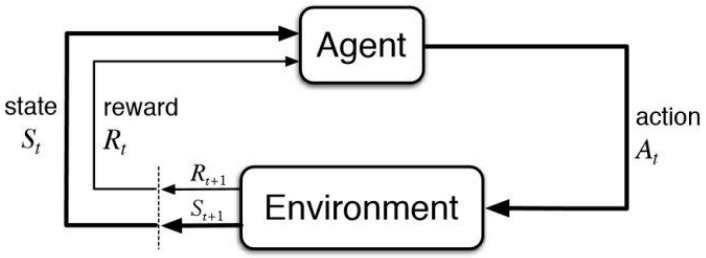



Figure 1.3: An illustration of DRL agent interaction with environment

problem of multiple autonomous agents operating in a shared environment, which together seek to maximize their own long-term return by interacting with the environment as well as other agents [32]. MARL algorithms can be classified into different categories based on the types of situations they handle as fully cooperative, fully competitive, and a mix of the two. This allows a variety of new solutions that build on concepts such as cooperation or competition [32]. However, multi-agent settings also introduce challenges, including inadequate communication, difficulties in reward assignment to agents, and environment non-stationarity. We implement a continuous-time based Proximal Policy Optimization (PPO) algorithm in which the UAVs coordinate among themselves and communicate with the base to choose the best possible action.

## 1.3 Deep Reinforcement Learning Approaches

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal [33]. Deep Reinforcement Learning (DRL) has made incredible advances in recent years in many well-known sequential decision-making tasks , including playing the game of Go, playing real-time strategy games, controlling robots, and autonomous driving, especially in conjunction with the development of deep neural networks (DNNs) for function approximation [34]. A DRL agent autonomously learns an optimal policy to maximize its rewards through its interactions with the environment. The optimization problem is formulated as an Markov Decision Process (MDP). The MDP environment represented by the tuple $< S, A, R, \lambda >$, where, $S$, is the state space, $A$, is the action space, $R$, represents the reward function and $\lambda$ is the discount factor ranging from 0 to 1. At every time step, $t$, the agent receives a state representation, $S_t$ of the environment as shown in Fig, 1.3. The agent takes an action $A_t$ in the environment based on a policy $\pi(A_t|S_t)$. Then, a transition to the next state, $S_{t+1}$ occurs with a probability $P(S_{t+1}|(S_t, A_t))$, and a reward, $R_t$ is received by the agent, which is explained in detail in the following subsections. A DRL method is then used to solve the formulated MDP. In this thesis, we leverage two popular DRL algorithms, Dueling Deep Q Networks (DDQN) and Proximal Policy Optimization (PPO). These methods are discussed in depth in the sections where they are used.

### 1.3.1 DDQN

DDQN is a value-iteration based method [35]. DDQN is similar to DQN, but with a modification to the target network [36]. In DQN, the evaluate and target networks use the same maximum operator. In large scale networks, DDQN is favored. The target network in DDQN uses the expected Q values for the next state by considering the action that gives the maximum Q-value of the next state and updates the Q values of the evaluate network based on the estimated Q value from the target network.

$$Q^*(S_t, A_t) = R_t + \gamma Q_\theta(S_{t+1}, \text{argmax}_\pi Q_{\theta'}(S_{t+1}, A_{t+1})) \tag{1.1}$$

where, $\theta$ are the weights of Q network and $\theta'$ are the weights of the old Q network.

### 1.3.2 PPO

Unlike DDQN, PPO is a policy gradient method. PPO is an advantage-based actor-critic algorithm that tries to be conservative with policy updates [37, 38]. Using KL divergence and clipped surrogate function it can update multiple steps of update in one trajectory [39]. In PPO, the primary network consists of two deep neural networks, called the actor-network and the critic network. The actor-network is used to explore the policy, and the critic network estimates the performance. The critic-estimate value helps the actor to learn the gradient of the policy. The target network consists of an actor-network and a critic network.

## 1.4 Objectives and Scope of the thesis

In this section, the objectives of the problems addressed as a part of this thesis are discussed and a summary of the scope of this thesis is presented. This section first discusses coding theory based scaling blockchain, DRL based approach for blockchain, then DRL based IoT enabled blockchain and lastly DRL based UAV action control. We use public blockchain for our implementation.

Blockchain is a distributed ledger with wide applications. Due to the increasing storage requirement for blockchains, the computation can be afforded by only a few miners. Sharding has been proposed to scale blockchains so that storage and transaction efficiency of the blockchain improves at the cost of security guarantee. A new protocol, Secure-Repair-Blockchain (SRB) is presented in this thesis, which aims to decrease the storage cost at the miners. In addition, SRB also decreases the bootstrapping cost, which allows for new miners to easily join a sharded blockchain. In order to reduce storage, coding-theoretic techniques are used in SRB. To decrease the amount of data that is transferred to the new node joining a shard, the concept of exact repair secure regenerating codes is used. We consider sharded blockchains and apply coding theoretic techniques to reduce the storage cost as well as the bootstrap cost. Bootstrap cost is the cost required to recover the coded blocks which are stored at the node. We present a

sharding scheme based on secure regenerating codes which are not only storage efficient but also bandwidth efficient while repairing single node failures. We draw the equivalence between the process of bootstrapping a node and repairing a failed node. The proposed blockchain protocol achieves lower storage than those that do not use coding, and achieves lower bootstrapping cost as compared to the different baselines. Hence, our bootstrap cost is low as compared to two schemes in literature, one of which is RapidChain [1] and the other is SeF [7] based sharding.

DRL is incorporated into blockchain systems to better understand the dynamics of the blockchain system and optimize its performance. According to the blockchain trilemma, any blockchain system can only have at most two of the three features, decentralization, security and scalability [5]. These three features have a tradeoff relationship with each other, where maximizing one feature can drastically degrade the other feature [40]. The practical solutions that focus on scalability may need to consider the performance degradation in the security level or decentralization. Therefore, it is essential to find the optimal scalability point suitable for the current network situation without compromising other performance values like security. We leverage two reinforcement learning approaches, DDQN and PPO are adopted to optimize the Prism blockchain [21] performance.

An IoT enabled blockchain would require the blockchain nodes to process the high volume of transactions from the IoT devices [5]. This will require scaling of blockchain algorithms for high throughput and low latency without compromising on other aspects like security and decentralization of the blockchain [13]. In Prism [20], the performance guarantees on throughput and latency are physical limits of the network. Prism achieves higher performance than the existing schemes by decoupling the functionality of blockchain [21]. We implement a DRL approach for IoT enabled Prism blockchain leveraging Proximal Policy Optimization (PPO) algorithm.

We consider a military environment application for multiple UAVs, where each UAV can act independently of the other agents' actions. The environment contains a set of targets that the UAVs need to coordinate and destroy. There are several mines in the field that are capable of destroying the UAVs. These mines are placed at any random location which is unknown to the UAVs. We assume that the mines have a sensing radius in which they can detect the UAV and destroy it. In this scenario, the UAV must learn to detect and avoid mines within a minimum distance from the sensing radius of the mine. The goal here for each UAV is to destroy the enemy target, by avoiding the mines and reaching the base safely. The challenge for a UAV is that it has no prior knowledge of the location of the mines and has to find a suitable policy to adapt to the dynamic environment. The UAV action control problem is formulated as a Markov Decision Process (MDP) with the action space involving acceleration of UAV, which has not been presented in the literature so far. We use a MARL [32] approach to control the action of a set of UAVs. Our MARL based approach aims to find optimal strategies for agents in settings where multiple agents interact in the same environment. We use Proximal Policy

Optimization (PPO) to train our agents, which is a very well-known state of art continuous control algorithm. We propose two PPO-based approaches for UAV action control namely centralized PPO (C-PPO) and Decentralized PPO (D-PPO) with a continuous action space. We are the first to present two PPO-based approaches for UAV action control. The performance of the proposed algorithms is observed through extensive simulations. We build a simulator for our experimentation called UAV SIM. The simulator is a military environment with random locations of targets and mines. The results show that the UAVs can accomplish minimum task completion time with the proposed PPO approach.

### 1.4.1 Organization of the thesis

In Chapter 2, the literature is reviewed which is relevant to this thesis work. In the next chapter, the proposed research aims to develop coding techniques to improve the system throughput and latency. Followed by Chapter 4 which explains a deep reinforcement learning based approach to Prism Blockchain. In Chapter 5, a deep reinforcement based approach for IoT enabled Prism Blockchain is discussed. In Chapter 6, Multi agent reinforcement learning is used for UAV Action Control. Chapter 7 summarizes the thesis and provides some key concluding remarks and future research directions.

# Chapter *2*

# Review of Literature

## 2.1 Scaling Blockchain

Blockchain has numerous applications in e-governance, smart contracts, healthcare and IoT. Reinforcement learning(RL) is mostly applied in IoT networks for deploying smart contracts [41]. In [42], a comprehensive survey of the existing blockchain protocols for the Internet of Things (IoT) networks is presented. They discuss the scenarios to deciding when to use a blockchain-based solution in an IoT application. In most RL in blockchain papers, a system model is considered and then a learning algorithm like Q learning is applied for further actions to be performed. Then, they try to optimize the performance mostly by making use of exploration and exploitation trade-off. For Example: In computation offloading approach for blockchain empowered mobile Edge Computing is considered, in which both mining tasks and data processing tasks are performed. To achieve long-term offloading performance, a model-free deep reinforcement learning to adapt to highly dynamic environments and maximize the long-term reward. [43]

## 2.2 Blockchain protocols

### 2.2.1 PBFT Protocol

PBFT has three phases Pre-prepare, Prepare, and Commit [1]. During the pre-prepare phase, the primary node collects transactions from the transaction pool and creates a block. Later broadcasts the block to the other nodes. These nodes are called replicas. In the prepare phase, all the replicas (except the primary node) compare if the blocks received from the primary node are same. This prevents selective forwarding of blocks from the primary node. If the replicas receive the same block from the primary, the commit phase validates the block. In each phase of PBFT, all steps are successfully processed if more than two-thirds of the votes from the participating nodes approve it. Therefore, the ratio of malicious nodes acceptable in a PBFT agreement should not exceed one third of the total number of nodes participating in the consensus process, which is the consensus bound of PBFT. PBFT exchanges a large number

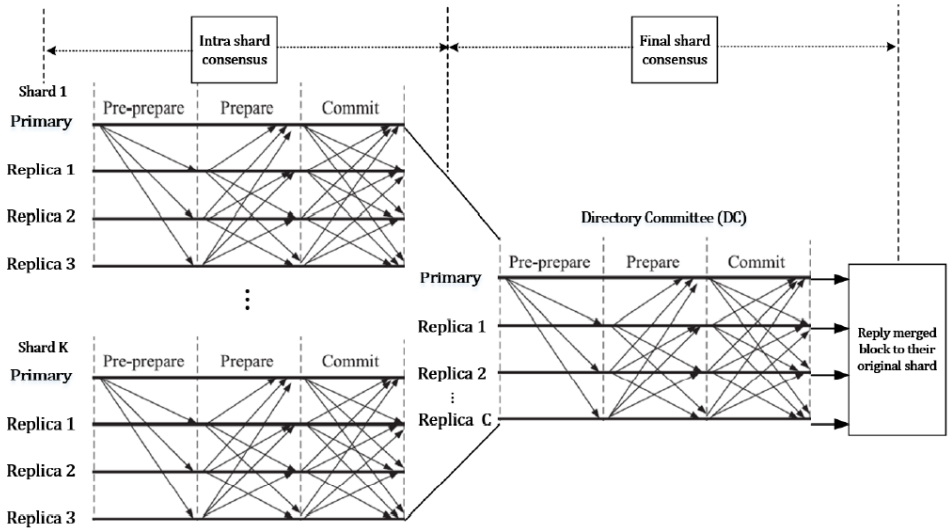



Figure 2.1: PBFT based Consensus in sharding protocols [1]

of messages to reach a consensus taking a long time. Also the time consumption increases exponentially with the number of participating nodes increase.

### 2.2.2 Sharding

Sharding is a technique employed to scale the throughput. Sharding is similar to replication coding which splits the overhead of processing transactions among multiple groups of nodes (committees/shards) which operate in parallel speeding up the consensus process [8, 10, 15, 16]. Each committee proposes a state block which has a Merkle root of the transactions and is also verified by all the committee members [17]. This final state block is appended to the blockchain by the leader. The leader is the node who solves the PoW puzzle and is elected for an epoch. The Byzantine consensus protocol used can tolerate up to 1/3 of the total nodes to be malicious in an authenticated setting (with digital signature verification) with asynchronous communication channels [1]. Fig 2.1 illustrates PBFT based consensus in sharding protocols. The sharding protocols require a linear amount of communication. The authors of [7] proposed an architecture, called Secure Fountain (SeF), which is based on fountain codes and enables any full node to encode validated blocks into a small number of coded blocks, thereby reducing its storage costs by orders of magnitude.

### 2.2.3 Prism Blockchain

The longest chain protocol followed in Nakamoto consensus [3] has tolerance up to 50% malicious nodes in the network. The blocks on the longest blockchain will eventually be adopted by honest miners, and other honest blocks are orphaned. To avoid forking, the mining rate for

new blocks creation is set longer than the latency for propagating a block to most miners in the network. The Prism protocol [20] is a structured-DAG blockchain with a proposer block and many voter chains. The proposer blocks reference transactions to include in the ledger. The voter blocks vote on these proposer blocks. Many parallel voter blockchains are run which utilize the network bandwidth fully. A miner does not know if the new mined block will be a proposer or voter. A random sortition procedure is followed using a hash function which determines the role of each block after mining. The authors of [20], show that the latency upper bound (speed of light propagation delay) and the throughput lower bound (network capacity) are achievable. The implementation of Prism protocol is discussed in [21], in which Prism achieved achieves a throughput of 70,000 transactions per second with a confirmation latency of tens of seconds using 1000 EC2 Virtual Machines. In [44], the bitcoin and Prism backbone protocols are analyzed and the probabilistic guarantees are provided. Their results show that it achieves security against up to 50% adversarial hashing power, optimal throughput up to the capacity of the network, and fast confirmation latency for honest transactions. The analysis of Prism in a continuous-time realistic data is presented in synchronous networks in [45]. The application of Prism to smart contracts and its usefulness is discussed in [46].

### 2.2.4 Deep Reinforcement Learning based Blockchain

Reinforcement learning techniques can help optimize many parameters and improve the performance to a great extent. The integration of DRL and Blockchain was used in serving different fields such as vehicular Ad Hoc networks, wireless networks, and the Internet of Things (IoT) [5,16,47,48]. In [47], the authors used Dueling Double Deep Q-Network (D3QN) to manage the selection of the optimal block transaction fees considering different constraints. The authors in [5] used the Double Deep Q-Learning prioritized experience replay approach to find the optimal number of consensus nodes. DRL was utilized in [8] to increase the throughput, consensus algorithm and block producer, block size, and block interval. To maximize the reward to the miners, reinforcement learning algorithms are implemented. The DRL and blockchain models operate independently [49]. To efficiently handle significantly large amounts of data from the IoT network, a sharded blockchain network is used to process numerous transactions in a parallel manner. The performance optimization framework of scalable blockchains for IoT systems can be categorized as IoT blockchains, sharding, and Deep reinforcement learning (DRL) technology applied to blockchains [8]. The state in [8] is the number of nodes participating in the network along with the probability of a node being malicious. The action is the number of nodes assigned to each shard and the reward is the throughput of the system. In [50], the DRL-based blockchain is applied to health care systems. In [9], the authors use Deep Deterministic Policy Gradient (DDPG) to traditional blockchain for demand management. The survey in [51] provides a deeper understanding on the security and privacy risks of the key components of a blockchain from the perspective of machine learning, which is useful in the

design of practical blockchain solutions for IIoT. In [52], the IoT data is grouped into clusters using K means algorithm and the initial training parameter of cluster number is trained by using double deep Q Network.

In [49], the authors highlight that the RL techniques learn through interaction wheras in supervised learning, knowing the correct target values for the training data can determine exactly whether a decision is right or wrong (classification) or how far a prediction is from the correct value (regression). The agent in RL does not usually learn whether a decision is the best. Since the experience of the agent only grows with its interaction, the agent often only knows for a part of the possible actions that have led to a greater or smaller reward in the past. Based on this it estimates the future reward and tries to take an action in the present time that maximizes the reward.

1. **Q-learning:** In Q-learning, an agent takes action based on the Q-values in the Q-table. There are four major components of a Q-network, i.e., a stage set, an action set, a reward, and transition probabilities. For each state, the agent executes some action under its pre-defined policy. Subsequently, the agent adopts the policy such that it has maximum Q-value. After each sequence, the agent revises the Q-table for a more accurate estimation of the Q-values and updating the policy of the agent. Thus, in due course of time and after many steps, the policy converges to the optimal policy of the agent.

2. **Multi-armed Bandit Learning:** The agent in the multi-armed bandit approach selects the action without having the state information of the environment. Relevant to the action performed by the action time step, the agent receives a reward that is maximized in the next iterations. Since the agent is unaware of the environment and the associated reward, there always exists a tradeoff between exploitation and exploration. Due to this reason, the multi-armed bandit learning, although lower in complexity, is not efficient for highly dynamic environments.

In large scale networks, blockchain needs to have high scalability. In applications like IoT, millions of devices are connected. So, the rate at which transactions are recorded on blockchain needs to be very high. In order to maximize the reward to the miners, reinforcement learning algorithms are implemented. The RL and blockchain models operate independently [49]. To efficiently handle significantly large amounts of data from the IoT network, a shard blockchain network is used to process the numerous transactions in a parallel manner. The performance optimization framework of scalable blockchains for IoT systems can be catogorized as: IoT blockchains, sharding, and Deep reinforcement learning (DRL) technology applied to blockchains [8]. A model based Q-learning has four major components, a state set, an action set, reward and transition probability. A state set is a set of all the possible states, action is a set of possible actions, reward is a scalar and transition probability is the probability of transition to the next state from the current state by taking a particular action. In [5],

authors used DRL for providing a methodology for evaluating the system from the aspects of scalability, decentralization, latency and security.

### 2.2.5 Minimizing Forking Events

Q-learning technique is widely used to minimize forking events. Forking occurs in the blockchain when an extending blockchain diverges into two potential chains [49] . The miners participating in the blockchain need to use common consensus algorithm to maintain the ledger of blockchain. Sometimes this may result in creating a fork. Forking can also be caused when the nodes after completing the proof-of-work, do not convey the results to the other computing nodes. To avoid the forking in blockchain, the transmission delay at the miner can be reduced with the help of RL. The agents can be trained for an IIoT environment that develops an optimal policy for minimizing such delays.

### 2.2.6 Time to Finality Minimization

Finality is inclusion of tamper proof blocks on to the final blockchain ledger. In [49], an RL system can be used to detect selfish miners that try to consume the resources unnecessarily. For instance, the Q-learning technique is be used to minimize the probabilistic time to the finality of proof-of-work for blockchain-enabled IIoT networks [5]. The agents can update themselves by learning from the received rewards and probabilistically improving the time to finality. Time to Finality is also referred to as latency.

### 2.2.7 Enhancing Transaction Throughput

Transaction throughput in IIoT-blockchain networks has challenges of scalability. One simple solution can be to increase the number of transactions per block. Another solution can be to increase the frequency with which the blocks are added into the network. Thus, recording more transactions may have an adverse effect on the decentralization of blockchain-enabled IIoT networks. This could also increase the mining time since a miner needs to check the validity of all the digital signatures on the transactions before mining a block. Moreover, the optimal policies and the tradeoffs between the transaction throughput and decentralization can also be identified with the help of RL techniques [49]. The agent can provide the specific set of actions needed to increase the throughput while not compromising the decentralization of blockchain networks.

### 2.2.8 Average Block Time Reduction

Block time is sometimes referred to as the block interval. It is the total amount of it takes to mine a block. The high variability in time for mining a block causes the average block time to be an important factor in large-scale networks. Therefore, the average block time of

a blockchain network is directly related to the complexity of the proof-of-work algorithm. RL techniques are used for optimizing the long-term utility of a blockchain network instead of relying on instantaneous gains in the consensus. For example, a multi-armed bandit learning RL network can be used to identify the complexity of the algorithms based on the characteristics of the blockchain network. Thus, in order to develop a long-term optimal policy, if the average block time is less than the expected block time, then the level of difficulty can be increased. In contrast, if the expected time is less than the average block time, then the level of difficulty can be reduced for mining. Therefore, RL techniques can help optimize the performance based on different characteristics of the end-to-end blockchain network.

### 2.2.9 Security

Due to the broadcast nature of messages exchanged between blockchain devices in applications, it is crucial to secure the links through physical layer security techniques. The RL based approaches can be extensively used for the establishing of link security for blockchain networks. Multi-armed bandit techniques are used to identify the nearby eavesdroppers in the blockchain network [5]. Deep Q-learning can be applied to introduce the artificial noise in the network, without damaging the quality of the legitimate link [5].

### 2.2.10 DQN based Blockchain Systems

The research in [8] focuses on applying DQN to a sharded blockchain system for IoT applications. They propose a Deep Q Network Shard based Blockchain (DQNSB) scheme that dynamically finds the optimal throughput configuration.The sharded blockchain's latency analysis and security level characterization is provided. Using this analysis, they estimate the level of maliciousness and adapts the blockchain parameters to enhance the security level considering the amount of malicious attacks on the consensus process. Any blockchain system can only have at most two of the following three features: decentralization, security, and scalability. These three features have a trade-off relationship with each other, where maximizing one feature can drastically degrade the other feature. This is explained in detail in Omniledger [17]. The DQNSB model [8] is based on Elastico to perform decentralization of shard clustering. Sharding results in a throughput and security trade-off based on the number of shards. The security level can be actively changed by adjusting the number of shards. Their scheme adopts the DRL approach to estimates the network's malicious probability by monitoring the consensus result of each epoch. Based on the network trust, the number of safe shards are computed and adaptive control is used to maintain optimal throughput conditions.

In this work, we introduce an adaptive, secure, and intelligent DRL based Prism blockchain system. The algorithm of DDQN is a value-based method that focuses on learning a function approximator to predict Q-values that satisfy the recursive Bellman Equation [35]. The optimal actions are then derived from the action which maximizes the Q-value. PPO is a policy-gradient

Table 2.1: A table summarizing the related works

| Paper | Number of UAVs | Objective | Solution Approach | Environment | Performance |
|---|---|---|---|---|---|
| [26] | Single | Path Planning | Centralized DDPG | 3D continuous environment with aerial obstacles | Reward Convergence |
| [37] | Multiple | Jointly control multiple agents | Centralized PPO | military environment | reward convergence |
| [39] | Multiple | Drone racing competition | Decentralized PPO | environment was created using AirSim | task completion |
| [54] | Multiple | Path planning | Centralized D3QN combined with greedy heuristic search | Military environment developed using STAGE Scenario | task completion |
| [55] | Multiple | Data harvesting | Centralized DQN | urban city like structure map | Successful landing and collection ratio |
| **This work, GLIDE** | Multiple | Coordinated action control | Centalized and decentralized PPO | Military environment created with our simulator, UAV SIM | task completion time and reward convergence |

method that optimizes the expected reward by estimating the gradient of the policy from the trajectories taken by the agent [53]. To the best of our knowledge, this is the first work leveraging deep reinforcement learning to optimize Prism using two state-of-art DRL algorithms, DDQN, and PPO.

## 2.3 UAV Action Control

UAVs have recently become popular in commercial and many other fields for target identification and detection. Many studies have been conducted concentrating on the applications of UAVs or the use of UAVs to complete specific scientific research tasks. In this section, we present a review of the literature related to our research.

### 2.3.1 Path Planning and Action Control

There are many works in path planning using DRL with applications in drone fleets for delivery, traffic flow control, automated driving, and particularly in UAVs [32]. The authors of [54] used a dueling double deep Q-networks (D3QN) approach for path planning UAVs in dynamic contexts. They assume the availability of global situational data for the UAV which is used for its decision-making and path planning. In [55], the path planning for a cooperative, non-communicating, and homogeneous team of UAVs is formulated as a decentralized partially observable Markov decision process (Dec-POMDP). DDQN with combined experience replay is used to solve the problem. In [56], a study on UAV ground target tracking under obstacle environments using deterministic policy gradient (DDPG) algorithm. The authors attempt to improve the DDPG algorithm for UAV target tracking. In [26], a 3D continuous environment with static obstacles is built, and the agent is trained using the DDPG algorithm. In [57], an approach to exploit global-local map information that allows the trajectory planning. The work tries to scale to large and realistic scenario environments efficiently with an order of magnitude more grid cells compared to previous works in a similar direction. In [58], the UAV executes computational tasks offloaded from mobile terminal users (TUs) and the motion of each TU follows a Gauss-Markov random model, and Deep Q Learning method is used. The authors of [25] proposed an algorithm in which each UAV makes autonomous decisions to find a flight path to a predetermined mission area. Each UAV's target destination is not predetermined, and the authors discuss how their algorithm can be used to deploy a team of autonomous drones to cover an evolving forest wildfire and provide virtual reality to firefighters. In this work, GLIDE leverages a continuous-time based algorithm for coordinated UAV control in dynamic military environment. Here, we not only consider the global situational data but also the local situation data of each UAV and control the actions of a group of UAVs.

### 2.3.2 Multi Agent Approach for UAV Control

In [59] the UAV control policy that generalizes over changing scenario parameters is learnt. In [60], a path planning for data harvesting from distributed Internet of Things (IoT) devices using multiple cooperative, non-communicating and homogeneous team of UAVs is discussed. In [61], the communication limitation in UAV is discussed and addressed using DRL-based energy-efficient control for coverage and connectivity called DRL-EC 3 which has better coverage, fairness, and energy consumption. In [38], dynamic environments with potential threats is considered for path planning for UAVs based on the global situation information using D3QN. In [62], an online distributed algorithm for tracking and searching is proposed. The authors in [63] propose Geometric Reinforcement Learning (GRL) algorithm for path planning of UAVs. In this work, we focus on action control of a group of UAVs in a centralized and decentralized implantation of GLIDE. Our aim is to scan the entire field in the minimum possible time and complete the task.

### 2.3.3 PPO based MARL

PPO is an advantage-based actor-critic algorithm that tries to be conservative with policy updates [37]. Using KL divergence and clipped surrogate function it can update multiple steps of update in one trajectory [39]. A review of MARL and its application to autonomous mobility is discussed in [32], where the state-of-the-art methods used in MARL are presented along with the implementation details. In [64], contemporary autopilot systems for UAVs are studied with a DRL controller to handle the nonlinear attitude control problem. A PPO algorithm is used and is observed to converge in unseen disturbances in the form of wind and turbulence. In [39], a long-term planning scenario that is based on drone racing competitions held in real life is discussed. The racing environment was created using AirSim Drone Racing Lab and the DRL agent was trained using PPO algorithm. Their result shows that a fully trained agent could develop a long-term planning scenario within a simulated racing track.

A summary of related works is presented in Table 2.1. In this work, we introduce a novel formulation of the UAV action control problem. Following that, we propose GLIDE, which leverages continuous-time PPO-based approach for the action control of a group of UAVs in a dynamic environment. A centralized action control algorithm C-GLIDE is compared with a decentralized action control algorithm D-GLIDE. We implement the simulation environment called UAV SIM for our experimentation.

# Chapter *3*

# Codes for Sharded Blockchain

## 3.1 Introduction

Blockchain technology's unique ability to provide an open ledger for recording transactions while simultaneously ensuring security and verifiability lends itself to a variety of uses. Blockchain experts envision a huge amount of possible applications, with everything from supply chain management to online personal identification. Its applications includes Bitcoin, which is a decentralized cryptocurrency which processes online payments though a peer-to-peer (P2P) system without traversing through a financial institution. The size of the Bitcoin blockchain has experienced consistently high levels of growth since its creation, reaching approximately 269.82 gigabytes in size as of the end of March 2020 [65]. The increasing size of blockchain requires high storage capacity at miners which can be afforded only by a few miners [10]. Further, in order to allow efficient entry for new miners, the amount of data that must be sent to the the new miner (denoted as the bootstrap cost) must be low. In this chapter, coding theoretic techniques are considered to reduce the storage cost as well as the bootstrap cost that would help more miners to enter the market.

In order to reduce the computation burden to solve the proof-of-work (PoW) puzzles, an approach called sharding has been proposed [15]. This work considers sharded blockchains and apply coding theoretic techniques to reduce the storage cost as well as the bootstrap cost. We present a sharding protocol, called SRB, which is based on secure regenerating codes that are not only storage efficient but also bandwidth efficient while repairing single node failures. We draw the equivalence between the process of bootstrapping a node and repairing a failed node. Hence, our bootstrap cost is low as compared to uncoded sharding [1] and Secure Fountain Architecture (SeF) based sharding [66].

In the previous approaches based on codes, the bootstrapping process has two steps: (i) all the blocks are recoverd at the new node (ii) the coded blocks to be stored are again computed as linear combinations of the recovered blocks. In the proposed approach, we bypass the first step by directly trying to recover the coded blocks which are to be stored in the new node. Hence, the bootstrap cost is less as compared to previous method of SeF based sharding. In

| Parameter | RapidChain | SeF | SRB |
|---|---|---|---|
| Storage Overhead | $n_S$ | $(1+\delta)$ | $\frac{n_S\alpha}{L}$ |
| Bootstrap Cost | L | $L + O(\sqrt{L}\log^2(L/\delta))$ | $\alpha < L$ |
| Security Guarantee | $\frac{n_S}{2}$ | $n_S - \frac{L+O(\sqrt{L}\log^2(L/\delta))}{\rho}$ | $\frac{n_S-\alpha}{2}$ |

Table 3.1: Performance comparison of RapidChain, SeF based and SRB protocols for sharding.

the proposed approach, $L$ blocks in the shard are encoded to obtain $n_S\alpha$ blocks and stored on $n_S$ nodes in the shard, with $\alpha < L$ blocks on each node. The code construction uses a product matrix code construction. The bootstrapping problem has connections to the secure repair problem in regenerating codes, and these connections are exploited to provide an efficient bootstrapping transfer. In addition to the encoding and decoding mechanisms, the entire protocol efficiently integrates the mechanisms of sharding, reference committee reconfiguration and intra-shard consensus. The main result can be seen in Table 3.1. As compared to RapidChain, the proposed framework allows for significantly lower storage overhead ($n_S$ vs $\frac{n_S\alpha}{L}$) and bootstrapping cost ($L$ vs $\alpha$). As compared to SeF based sharding, the proposed framework allows for significantly lower bootstrapping cost ($L + O(\sqrt{L}\log^2(L/\delta))$ vs $\alpha$) with somewhat increase in storage overhead. Further, the performance of the proposed approach for security, encoding complexity are obtained.

**Sharding:** Sharding in blockchains was discussed first in Elastico [15]. They proposed to partition the network into shards, each of which processes a disjoint set of transactions. The number of shards grow linearly with the total computational power of the network. Each shard runs a classical byzantine consensus protocol to decide their agreed set of transactions in parallel. In every epoch, a proof-of-work (PoW) puzzle is solved based on an epoch randomness obtained from the last state of the blockchain. Elastico improved the throughput and latency of Bitcoin. Omniledger [17] is an improvement over Elastico which preserves long term security in blockchains. It introduced atomic cross-shard commit protocol for transactions affecting multiple shards. It ensures security by using a bias-resistant public-randomness protocol.

**Coding Techniques in Blockchains:** Coding theory is applied used to gain storage efficiency in blockchain systems. Recently, rateless codes have been actively applied in various aspects of storage and computing. The authors in [66], propose using rateless codes for blockchain storage and for bootstrapping a new node in the network. The fountain codes approach used in [66], has about 1000x storage savings. In [6], each node stores the coded version of the entire blockchain. So, they achieve up to 30% storage savings compared to full replication of the blockchain on each node. In [67], the authors propose a novel distributed storage code for blockchains as a combination of secret key sharing, private key encryption. Interestingly, they focus on using dynamic zone allocation, which uses a combination of cryptographic and

information-theoretic security. The authors in [18], discuss the possibility of using erasure codes for low storage, especially for light clients. Here, blockchain is stored as only some coded fragments of each block. In [68], the authors focus on tackling storage bloating problem with network coding with such a framework does not lose any information. Moreover, with their storage scheme they can save storage room for each node. The problem of designing fault tolerant distributed storage based on blockchains in the context of industrial network environments has been discussed in [69]. Here, the authors employ codes with local regeneration in order to repair from single and multiple node failures.

## 3.2 Preliminaries of Sharding and Coded Blockchains

### 3.2.1 RapidChain

In this section, we will describe the RapidChain protocol which allows scaling of blockchain by sharding. Our approach of using secure regenerating codes in the framework of sharding will be based partly on the ideas discussed here.

The RapidChain protocol for sharding consists of the following elements:

- **Reference Committee and Epoch Randomness:** The protocol is divided into epochs. A reference committee is elected at the start of the protocol. Reference committee is a set of nodes which are responsible for generating epoch randomness and verifying the PoW puzzles which the new nodes joining the network solve. During the start of each epoch, the reference committee comes to consensus on a reference block which contains the list of all nodes and their assigned shard (We will use the term shard instead of committee in this chapter). The reference block is sent to all the nodes.

- **Committee Reconfiguration and Cuckoo Rule:** Malicious nodes could strategically leave and rejoin the network and they can take over one of the shard to break the security guarantees of the protocol. Moreover, we assume a byzantine adversary that can actively corrupt a constant number of honest nodes. In order to prevent this attack, shards have to be periodically reconfigured faster than the adversary's ability to generate churn. The communication overhead is very high if all the committees need to re-elected. RapidChain follows the Cuckoo rule to shuffle only a subset of nodes during the reconfiguration at the beginning of each epoch. RapidChain ensures that shards are balanced with respect to their sizes as nodes join or leave the network. To map the nodes to shards, each node is mapped to a random position in $(0, 1]$. Then, the range is partitioned into $m$ regions. In the Cuckoo rule, when a node wants to join the network, it is placed at a random position, while all nodes in a constant-sized interval surrounding the new node's position are moved to new random positions in the interval $(0, 1]$.

- **Ledger Pruning:** Ideally, a new node joining the committee has to download all the blocks in the shard. However, since the throughput of the system is high, the amount of download is quite high. To avoid this, the nodes which join the network prune the ledger and store. The disadvantage of pruning is that there are only a few archival nodes which need to be contacted if the complete ledger needs to be downloaded.
- **Intra-committee Consensus and Cross-shard Transactions:** The intra-committee consensus is accomplished by a byzantine consensus protocol based on fast gossip algorithm with application of erasure codes. Cross-shard transactions are performed based on a certain inter-committee routing scheme.

### 3.2.2 Secure Fountain Codes for Sharded Blockchains

RapidChain has high storage overhead since every node in the shard stores a full copy of the blocks in the shard. To save on the storage costs, one of the existing solutions is based on Secure Fountain codes (SeF). The authors of [66] proposed an architecture, called Secure Fountain (SeF), which is based on fountain codes and enables any full node to encode validated blocks into a small number of coded blocks, thereby reducing its storage costs by orders of magnitude. One of the key idea in SeF is to use the header-chain as a side-information to check whether a coded block is maliciously formed while it is getting decoded. The coded blocks in SeF are called droplets, the nodes storing coded blocks are called droplet nodes, and any new node joining the system is called a bucket node. During bootstrap, a bucket node collects sufficiently many droplets and recovers the blockchain even when some droplet nodes are adversarial, providing murky (malicious) droplets. After validating the blockchain, a bucket node will perform encoding to turn itself into a droplet node. In this way, droplet nodes will slowly replace archival nodes.

**Encoding:** The encoding is performed using a Luby Transform (LT) code [70]. LT codes admit a computationally efficient decoding procedure known as peeling decoder (also known as a belief propagation). SeF exploits peeling process to introduce resiliency against maliciously formed blocks by using the header-chain as a side-information and leveraging Merkle roots stored in block-headers, which is explained as follows.

**Peeling Decoder:** Consider a bucket node that is interested in recovering the blockchain $B$. It contacts an arbitrary subset of $n$ ($n \geq k$) droplet nodes, and downloads the stored data. This includes droplets $C_j$ and vectors $v_j$. Without loss of generality, let us (arbitrarily) label the downloaded droplets as $C_1, C_2, \cdots, C_{ns}$. Note that, since a coded droplet does not have any semantic meaning, the bucket node cannot differentiate between the honest and malicious droplets within the downloaded ones. We assume that the bucket node has access to the honest header-chain. Note that this can be done by contacting several droplet nodes, and obtaining the longest valid header-chain as described in [66]. The decoding proceeds in iterations. In each iteration the algorithm decodes (at most) one block until all the blocks are decoded, otherwise

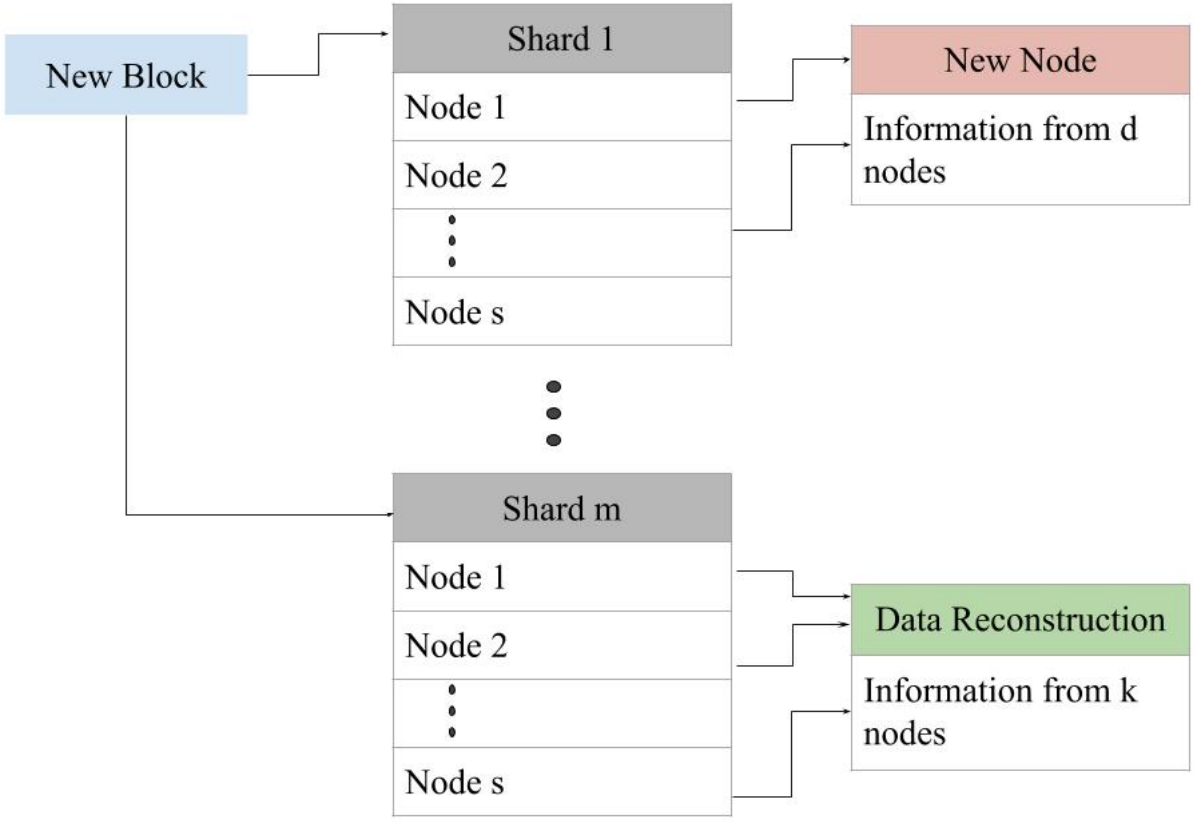



Figure 3.1: Illustration of Secure Repair Block protocol in blockchains.

the decoder declares failure. Note that Step (3) differentiates the error-resilient peeling decoder from the classical peeling decoder for an LT code. More specifically, the classical peeling decoder always accepts a singleton, whereas the error-resilient peeling decoder may reject a singleton if its header and/or Merkle root does not match with the one stored in the header-chain.

SeF codes allow the network to tune the storage savings as a parameter, depending upon how much bootstrap cost new nodes can tolerate. When SeF codes are tuned to achieve $k$-fold storage savings, a new node is guaranteed to recover the blockchain with probability $(1-\delta)$ by contacting $k + O(\sqrt{k}ln^2(k/\delta))$ honest nodes.

## 3.3 Secure Repair Block in Sharded Blockchains

In our work, we assign nodes to shards and perform our coding scheme on individual shards. We use the procedure followed in [1] to execute the random assignment of nodes to shards. In the following, we describe the components of the Secure Repair Block protocol and also algorithms corresponding to encoding, bootstrapping and reconstruction.

### 3.3.1 Components of Secure Repair Block Protocol

We build the secure repair block protocol around the RapidChain protocol. We have the following components:

- Reference committee is elected at the start of the protocol, which help in generation of epoch randomness and creating reference block. The reference block in addition to the epoch randomness and a list of nodes, has additional information of "Node Encoder Coefficient". The node encoder coefficient will be described later.
- Committee reconfiguration is performed using Cuckoo rule described before.
- Intra-shard consensus and cross-shard transactions are performed similar to RapidChain.
- The blocks are encoded in a certain way and multiple coded blocks ($\alpha$ of them) are stored in a single node. If a new node joins the system, bootstrapping the node is considering equivalent to repairing the node as shown in Fig. 3.1. The code construction which will facilitate this process and will be described in the rest of the section.
- It has been claimed in [71] that to achieve scalability, any robust sharded ledger has to perform compaction of state. Hence to perform verification, we propose that a node contacts other nodes to download a certain number of coded blocks and recovers the original blocks.

We would like to note that we do not consider ledger pruning as part of Secure Repair Block protocol.

### 3.3.2 Secure MBR Regenerating Codes

In this section, we describe the framework of secure regenerating codes which guarantee minimum bandwidth during the repair of a single node. These class of codes are known as minimum bandwidth regeneration (MBR) codes. The code construction as well as the security properties of the code have been derived in [72]. These are discussed in detail here for the sake of completeness. These codes will be used in a later section to provide a storage scheme for blockchains with sharding. Consider a file of size $L$ over a finite field $\mathbb{F}_q$ with the corresponding message symbols denoted by $m_1, m_2, \ldots, m_L$. These $L$ symbols are encoded into $n\alpha$ symbols and stored at $n$ nodes each of which can store $\alpha$ symbols over a finite field $\mathbb{F}_q$. We need to be able to perform two functions on this kind of a distributed storage system.

- **Data Reconstruction:** By connecting to any $k$ nodes and downloading all the $\alpha$ symbols present in the $k$ nodes, we have to be able to recover all the $L$ message symbols.
- **Node Repair:** By connecting to any $d$ nodes and downloading $\beta < \alpha$ symbols from the $d$ nodes, we would like to recover the contents of a single node.

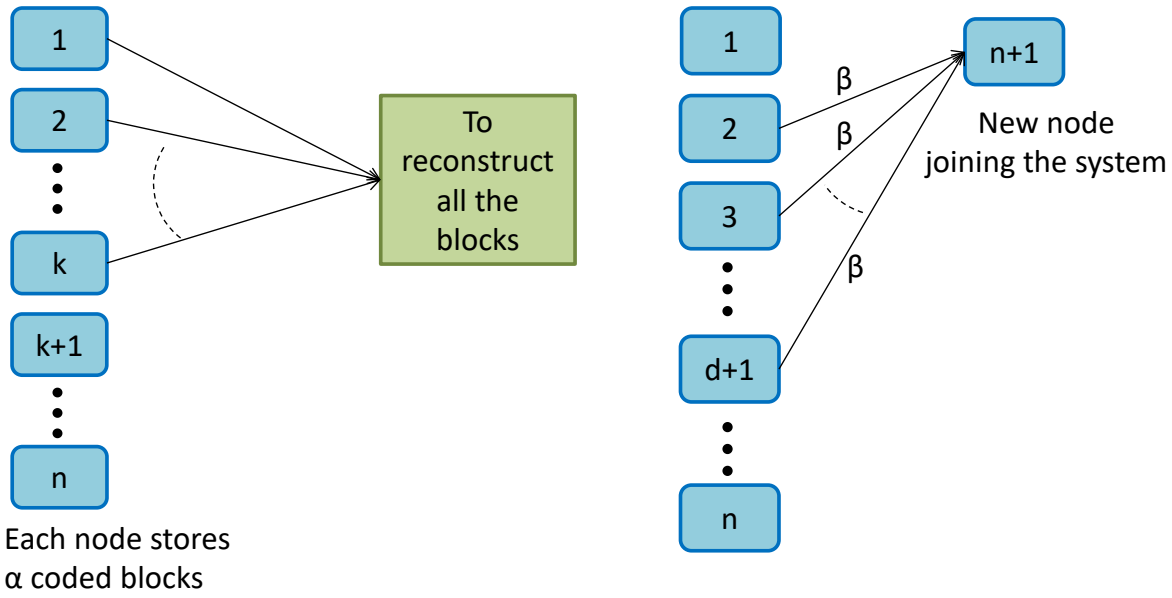



Figure 3.2: Regenerating code framework in the context of blockchains.

The framework of regenerating code in the context of blockchains is shown in Fig. 3.2.

We consider the threat model wherein the contents of one or more nodes may be malicious. A node that is malicious may send arbitrarily corrupted data during data reconstruction and data repair.

For the case of minimum bandwidth regenerating codes $\alpha = d\beta$. In the current code construction, we consider $\beta = 1$ and hence $d = \alpha$. The code construction is given by the following product-matrix framework:

$$C = \Psi M, \tag{3.1}$$

where $\Psi$ is encoding matrix of size $n \times \alpha$ and $M$ is message matrix of size $\alpha \times \alpha$. Each row of $C$ is a set of $\alpha$ symbols which is stored in a node. The message matrix $M$ has the following symmetric structure:

$$M = [m_{i,j}] = \begin{bmatrix} U & V \\ V^T & 0 \end{bmatrix}, \tag{3.2}$$

where $U$ is a symmetric matrix of size $k \times k$ and $V$ is a matrix of size $k \times (\alpha - k)$. The number of variables in the above matrix is thus given by $L = k(\frac{k+1}{2}) + k(\alpha - k)$. One possible choice of the encoding matrix is that of Vandermonde matrix given below.

$$\Psi = \begin{bmatrix} 1 & \gamma_1 & \gamma_1^2 & \cdots & \gamma_1^{\alpha-1} \\ 1 & \gamma_2 & \gamma_2^2 & \cdots & \gamma_2^{\alpha-1} \\ \vdots & \vdots & \ddots & & \vdots \\ 1 & \gamma_n & \gamma_n^2 & \cdots & \gamma_n^{\alpha-1} \end{bmatrix}, \tag{3.3}$$

where all $\gamma_i$ are distinct elements of the finite field $\mathbb{F}_q$. With the encoding matrix chosen as Vandermonde matrix, the above code construction can also be interpreted as being obtained by evaluating polynomials at various points in the finite field $\mathbb{F}_q$. Consider the following set of $\alpha$ polynomials,

$$P^{(i)}(x) = \sum_{j=0}^{\alpha-1} m_{i,j} x^j. \tag{3.4}$$

It can seen that the code can be obtained by evaluating the above polynomials at distinct elements $\gamma_1, \gamma_2 \ldots, \gamma_n$ of the finite field $\mathbb{F}_q$, $q \geq n$. We would like to note that this code construction is very flexible in terms of addition of nodes as long as $k$ and $\alpha$ are kept constant. This is because we just need to add another evaluation point $\gamma_{n+1}$ to the existing code and the resultant code is still an MBR code comprising $n+1$ nodes. This property will be used later in the case of blockchains to bootstrap new nodes.

**Remark 1** *There are also other class of codes known as minimum storage regenerating codes (MSR codes) which can be used to reduce the bootstrap cost as well. These codes offer a different operating point with respect to storage cost and bootstrap cost. These codes have lesser storage cost than MBR codes with an increase in the bootstrap cost. However, the code construction for a range of parameters is also based on product matrix framework and is similar to the MBR code. Thus, we will not discuss this variant of regenerating codes in this work.*

### 3.3.3 Encoding

We analyze the system based on rounds or epochs. At the beginning of each epoch, a reference committee agrees on a reference block consisting of the list of all active nodes for that epoch as well as their assigned shards. In addition to assigning nodes to the shards, in our protocol, the reference committee also assigns $n_S$ distinct coefficients ($\{\gamma_i\}$ termed as node encoder coefficients) from the finite field to every node in the shard. This information is also stored in the reference block. During the epoch, $L = k\alpha - \binom{k}{2}$ blocks in the shard are encoded to obtain $n_S\alpha$ blocks and stored on $n_S$ nodes in the shard. The encoding process is described in Algorithm 1.

To understand the encoding process as the blocks are coming in, the first round of encoding is done when the number of blocks in the epoch is $L$. Then the first $L$ blocks are encoded using the procedure described above and the original blocks are deleted. The second round of encoding is performed when the number of blocks in the epoch is $2L$. In the second round, blocks from $L+1, \ldots, 2L$ are encoded using the procedure described above and the process continues.

**Algorithm 1** Coding the blocks in a shard

1: Input $B_1, \ldots, B_L$ which are $L$ input blocks
2: Stripe the blocks into units of the finite field $\mathbb{F}_q$. The stripes of block $B_\ell$ are denoted by $B_{\ell,1}, \ldots, B_{\ell,Z}$. $Z$ is number of stripes in a block, which can determined by the block size and the field size.
3: Form $\alpha$ message polynomials as described in (3.4) with $\{B_\ell\}$ in place of $\{m_{i,j}\}$.
4: Evaluate the $\alpha$ message polynomials at the coefficient $\gamma_i$ assigned by the reference committee to obtain the coded stripes $C_{i,j,s}, 1 \leq j \leq \alpha, 1 \leq s \leq Z$.
5: Merge the stripes corresponding to a single code block and form $\alpha$ code blocks denoted by $C_{i,1}, C_{i,2}, \ldots, C_{i,\alpha}$.
6: The coded blocks $C_{i,1}, C_{i,2}, \ldots, C_{i,\alpha}$ are stored along with coefficient $\gamma_i$.

### 3.3.4 Boostrapping new nodes to a shard

The code described in Algorithm 1 has the data reconstruction and node repair properties which have been listed in the specifications of a regenerating code. Now, we will perform secure node repair with the same code by contacting more number of helper nodes. Here, we describe a procedure for repairing a single node failure in the presence of $p$ malicious nodes. As can be seen for the code construction, a node $i$ can be identified by its coefficient $\gamma_i$. In order to repair a node whose coefficient is $\gamma_i$, $d + 2p$ other nodes whose coefficients are say $1, 2, \ldots, d + 2p$ (all different from $i$) send the following symbols:

$$[1 \;\; \gamma_i \;\; \gamma_i^2 \ldots \;\; \gamma_i^{\alpha-1}] M [1 \;\; \gamma_j \;\; \gamma_j^2 \ldots \;\; \gamma_j^{\alpha-1}]^T, \tag{3.5}$$

where $j = 1, 2, \ldots, d + 2p$. By collecting the $d + 2p$ symbols, the resultant vector is equivalent to encoding the vector $[1 \;\; \gamma_i \;\; \gamma_i^2 \ldots \;\; \gamma_i^{\alpha-1}]M$ with an MDS code of length $d+2p$ and dimension $d = \alpha$. Hence, it can result of correct output of $[1 \;\; \gamma_i \;\; \gamma_i^2 \ldots \;\; \gamma_i^{\alpha-1}]M$ even if $p$ nodes are malicious and send erroneous symbols.

When a new node joins a shard, the contents of the node are obtained by repair process. This is accomplished by the new node contacting $d+2p$ nodes and downloading one coded block from each one of them. The reference committee keeps track of which nodes have which evaluation points and assigns a unique coefficient $\gamma_i$ to the new node. In the process of reconfiguration of shards/bootstrapping, we are assuming that at any point there are at least $d+2p$ nodes in any shard. Also, this scheme is flexible in terms of the number of malicious nodes we can tolerate. If an estimate of level of security is known at any point of time, then we can contact lesser number of nodes (since $p$ is less) and perform the bootstrapping operation. The data sent by the peers corresponds to the result of one linear combination of the $\alpha$ coded blocks contained in the stored data. Once the new node receives $(d + 2p)$ coded blocks, it applies a decoding procedure to recover its own coded blocks. The algorithm to bootstrap a new node to shard is described in Algorithm 2.

**Algorithm 2** Bootstrapping new node to a shard

1: Input coefficient $\gamma_i$ which corresponding to the evaluation point of the new node $i$.
2: The new node $i$ queries $d+2p$ other nodes in the shard with its coefficient $\gamma_i$.
3: $j^{\text{th}}$ node, $1 \leq j \leq d+2p$ sends the following linear combination to the new node $[1 \;\; \gamma_i \;\; \gamma_i^2 \ldots \;\; \gamma_i^{\alpha-1}]M[1 \;\; \gamma_j \;\; \gamma_j^2 \ldots \;\; \gamma_j^{\alpha-1}]^T$ after the message matrix $M$ is replaced with the corresponding blocks and the process of striping similar to that in the encoding algorithm.
4: After having received $d+2p$ coded blocks from the $d+2p$ nodes in the shard, the new node performs a $(d+2p, d)$ Reed Solomon decoding to obtain the coded blocks which it has to store.

We illustrating the process of bootstrapping a node (in the absence of adversarial nodes) through an example. Let there be 5 nodes in the system such that $k=3$ and $d=\alpha=4$. $M$ is a message matrix consisting of $L = k\alpha - \binom{k}{2} = 9$ blocks arranged as follows:

$$M = \begin{bmatrix} B_1 & B_2 & B_3 & B_7 \\ B_2 & B_4 & B_5 & B_8 \\ B_3 & B_5 & B_6 & B_9 \\ B_7 & B_8 & B_9 & 0 \end{bmatrix} \tag{3.6}$$

The $i^{\text{th}}$ node stores 4 coded blocks $[1 \;\; \gamma_i \;\; \gamma_i^2 \;\; \gamma_i^3]M$. Suppose if a $6^{\text{th}}$ node joins the system, then in order to compute $[1 \;\; \gamma_6 \;\; \gamma_6^2 \;\; \gamma_6^3]M$, the new node downloads one block each from 4 nodes. For example, in Fig. 3.3, nodes $2, 3, 4, 5$ send one block each given by $[1 \;\; \gamma_i \;\; \gamma_i^2 \;\; \gamma_i^3]M \begin{bmatrix} 1 \\ \gamma_6 \\ \gamma_6^2 \\ \gamma_6^3 \end{bmatrix}$ and then the new node computes $[1 \;\; \gamma_6 \;\; \gamma_6^2 \;\; \gamma_6^3]M$ from these 4 blocks.

### 3.3.5 Reconstruction

Here, we describe a procedure for reconstructing data in the presence of $p$ malicious nodes. For data reconstruction, we will contact $k+2p$ nodes and download all $\alpha$ symbols from each of them. Thus, we have access to the following symbols:

$$\Psi_{dr}M = [\Delta_{dr} \;\; \Phi_{dr}] \begin{bmatrix} U & V \\ V^T & 0 \end{bmatrix} \tag{3.7}$$

$$= [\Delta_{dr}U + \Phi_{dr}V^T \;\; \Phi_{dr}V], \tag{3.8}$$

Where $\Psi_{dr}$ refers to an arbitrary $(k+2p)\times\alpha$ submatrix of $\Psi$. The reconstruction of the message matrix involves two steps:

- In the first step, $\Phi_{dr}V$ can be used to be recover $V$ in the presence of $p$ mailcious nodes. This is because $\Phi_{dr}v_j$ for every column vector $v_j$ in $V$ forms an MDS code with parameters

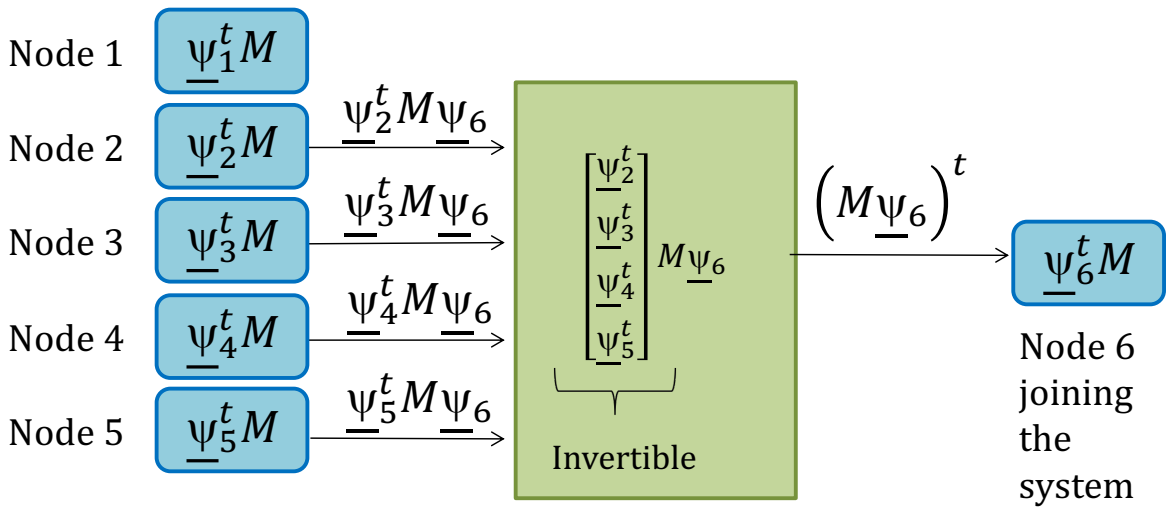



Figure 3.3: Bootstrapping a node is equivalent to repair of product-matrix MBR code.

$(k + 2p, k)$. Hence, $p$ errors can be tolerated. By applying similar procedure to every column of $V$, we can recover $V$ completely.

- In the second step, the contribution due to $V$ is subtracted from the term $\Delta_{dr}U + \Phi_{dr}V^T$ to result in $\Delta_{dr}U$. We can recover $U$ using the property of MDS code similar to that in the above step.

## 3.4 Performance Comparison

In this section, we will compare the performance of three schemes: RapidChain, SeF based sharding, and SRB protocol. The performance metrics include (i) storage overhead, (ii) bootstrap cost, (iii) encoding complexity, and (iv) security guarantee.

### 3.4.1 Storage Overhead

In the case of RapidChain, since each block in the shard is replicated $n_S$ times where $n_S$ is the number of nodes in the shard, the storage overhead is $n_S$. In the case of SeF based sharding, the storage overhead is $(1 + \delta)$ where $\delta$ is some small positive quantity. Here, we would like to note that in SeF based sharding, we need to combine a large number of blocks (order of thousands) in order to obtain a code which can be decoded successfully, i.e., $L$ has to be large. In the case of SRB protocol, the storage overhead is given by $\frac{n_S\alpha}{L}$, which is lower than that of RapidChain and higher than that of SeF-based sharding since $\alpha < L$.

### 3.4.2 Bootstrap Cost

In the case of RapidChain, a new node joining the network has to download the entire blockchain in that shard. So, the bootstrap cost will be $L$ blocks. In the case of SeF based sharding, a new node joining the network has to download at least $L+O(\sqrt{L}\log^2(L/\delta))$ blocks. Bootstrap cost for SeF based sharding is high because the new nodes first recovers all the $L$ blocks and then later computes the linear combination which it has to store. In secure regenerating code based sharding protocol, we have to download $\alpha$ blocks, where $\alpha < L$. We get this advantage since we are considering the bootstrap of a new node as a single node repair problem and we are downloading just as much content needed to recover the coded blocks to be stored in that node.

Consider a case when the block size is 2MB. There are 16000 nodes in the system with 16 shards and hence 1000 nodes per shard. We assume that $L = 1065$ blocks have been processed per shard.The storage per node for RapidChain (without ledger pruning) is 2.13GB and bootstrap cost is also 2.13GB. The storage per node for SeF based sharding is 4MB (Assuming number of coded blocks stored per node $\rho = 2$) and the bootstrap cost is $> 2.13GB$. The parameters for the SRB protocol are given by $k = 30$, $\alpha = 50$, $L = k\alpha - \binom{k}{2} = 1065$. With these parameters, the storage per node is 100MB and the bootstrap cost is also 100MB. Hence, the SRB protocol saves both in terms of storage and bootstrap cost.

### 3.4.3 Epoch Security

Let $t_S$ be the maximum number of malicious nodes within a shard which can be tolerated by the protocol. The number of malicious nodes which can be tolerated in the case of RapidChain is $t_S = \frac{n_S}{2}$ where $n_S$ is the number of nodes within a shard. For the SeF based sharding and SRB protocol, security guarantee is given by the maximum number of malicious nodes in the presence of which a new node joining the shard is bootstrapped with correct coded blocks. In a SeF based sharding scheme, the number of malicious nodes which can be tolerated are $t_S = n_S - \frac{L+O(\sqrt{L}\log^2(L/\delta))}{\rho}$. For the SRB protocol, the number of malicious nodes which can tolerated are $t_S = \frac{n_S-\alpha}{2}$.

Consider a blockchain network with $N$ nodes and $m$ shards. Each shard has $n_S$ nodes in it. Let $T$ be the total number of malicious nodes in the network. A random variable $X$ represents the number of malicious nodes. The probability that the committee election scheme fails in the first epoch be $p_{bootstrap}$. The value of $p_{bootstrap} \leq 2^{-26.36}$ in [1]. In [73], the security analysis of sharding schemes is analyzed assuming a hypergeometric distribution for assigning nodes to the shards. Cumulative hypergeometric distribution, $H(N, P, n_S, p_S)$, is used in calculating the failure probability of one shard and also for failure probability of a given sharding scheme.

$$H(N, T, n_S, t_S) = \sum_{l=t_S}^{n_S} \frac{\binom{T}{l}\binom{N-T}{n_S-l}}{\binom{N}{n_S}} \tag{3.9}$$

Hoeffding bound is proved to provide a tight bound for failure probability in [73], which is given by

$$H(N, T, n_S, t_S) \leq G(x), \tag{3.10}$$

where $G(x) = \left( \left(\frac{g}{r}\right)^r \left(\frac{1-g}{1-r}\right)^{1-r} \right)^{n_S}$, where $r = \frac{t_S}{n_S}$. Let $p_s$ be the probability of a shard failure, which means number of malicious nodes exceed the limit. The failure probability of Rapidchain [73] is upper bounded by

$$p_{bootstrap} + mp_s \leq U(x), \tag{3.11}$$

where $U(x) = p_{bootstrap} + mG(x)$.

The performance metrics of storage overhead, bootstrap cost and epoch security have been summarized in Table 3.1.

### 3.4.4 Encoding Complexity

In RapidChain, there is minimal encoding complexity which involves only copying of blocks from one node to the other. In SeF based sharding, the encoding in every node involves just binary field operations, whose complexity is again very less. In the secure regenerating code based sharding, encoding in the initialization phase involves computing one row of the matrix product which involves taking $\alpha^3$ multiplications over the finite field. In the reconfiguration/bootstrapping phase, computing the contents of a node involves decoding a $(r = d + 2p, d)$ Reed Solomon code whose complexity is given by $r^2\ log^2 r\ loglog r$.

### 3.4.5 Latency and Throughput

Latency of the SRB protocol will be approximately same as that of RapidChain. The reason is two-fold. The consensus protocol is synchronous and happens in 4 rounds for both the protocols. The additional delay incurred by the SRB protocol is due to contacting a set of nodes and downloading coded blocks in order to recover original blocks. This delay is small compared to the $\delta$ set for the synchronous protocol and hence the latency for reaching consensus would be nearly same as that of the RapidChain protocol. Thus, we have that $\tau_{SRB} \approx \tau_{RC}$. The throughput factor for the RapidChain in terms of the latency and the number of shards is given in [71]. The throughput factor for the SRB protocol can be expressed in terms of its latency exactly using the same expression and is given by

$$\sigma_{SRB} < \mu\ \tau_{SRB} \left(\frac{n}{\ln n}\right) \left(\frac{(a_{SRB} - p)^2}{2 + a_{SRB} - p}\right) \left(\frac{1}{v}\right), \tag{3.12}$$

where

- $\mu$ is the ratio of honest blocks in the chain to the total number of blocks,

- $n$ is the total number of nodes and $\ln n$ is the number of nodes in a shard
- $a_{SRB} = \frac{1}{2} - \frac{\alpha}{2 \ln n}$ is the resiliency of the SRB protocol within a shard
- $p$ is the fraction of malicious nodes in the system
- $v$ is the average size of transactions

We can see that the throughput factor of SRB is slightly less than that of RapidChain because $a_{SRB} < \frac{1}{2} = a_{RC}$.

### 3.4.6 Other Aspects

Here, we would like to point out two differences between SeF based sharding and secure regenerating codes based sharding. The encoding process for SeF based sharding is probabilistic in nature and hence there is a chance (though extremely less) that the encoding is such that we cannot recover the original blocks from the encoded blocks. On the contrary, secure regenerating codes based sharding protocol is deterministic and hence the blockchain data is definitely present in the system.

Secondly, SeF based sharding is completely decentralized in nature. Secure regenerating code based sharding is not completely decentralized in the sense that a reference committee has to keep track of the assignment of node encoder coefficients to nodes (negligible overhead is incurred in this process) and if a new node joins the shard, it has to be assigned a coefficient which is not previously assigned to any node in the shard.

Finally, in SeF based sharding, honest header chain is required as side information for applying the peeling based decoder. In secure regenerating code based sharding, there is no need to have the honest header chain as side information as the decoding process is via error correction capability of the code.

In this Chapter, we have presented a secure-repair-block protocol based on regenerating codes to reduce both storage and bootstrap costs. The proposed protocol is based on a sharding approach which uses efficient secure regenerating codes that are not only storage efficient but also bandwidth efficient while repairing failures. Drawing connections between regenerating codes and blockchain, significant storage savings and savings in bootstrapping costs are obtained, which makes it easier for a new miner to enter the system (since storage is one of the key challenge that requires only larger capability miners to enter).

This work demonstrates improvements in storage that allow for easy entry of the lower-capability miners, a detailed implementation of SRB protocol is left for the future. Such implementation can help see the latency and throughput performance of the system.

In our work, we assign nodes to shards and perform our coding scheme on individual shards. We use the procedure followed in [1] to execute the random assignment of nodes to shards.

In the following, we describe the components of the Secure Repair Block protocol and also algorithms corresponding to encoding, bootstrapping and reconstruction.

# Chapter 4

# DRL based optimization framework for Prism Blockchain

In large-scale networks like IoT, blockchain needs to have high scalability as the transactions that are recorded on a blockchain are very high. Recent work has proposed a new approach to decouple the functionality of the blockchain and achieve a much higher performance than the existing schemes [20]. Prism has throughput of 70,000 transactions per sec and latency is the network communication delay, reaching the physical limits of the system [21]. Prism follows the longest chain protocol and achieves good security. In this work, we aim to further improve the security of Prism while maintaining the throughput and latency guarantees.

An intelligent learning-based decision-making approach can optimize the Blockchain configuration adapting to the dynamic nature of incoming transactions, participating nodes while still maintaining the blockchain performance [5]. Artificial Intelligent (AI) has shown its ability to learn from a massive amount of data effectively [74]. Reinforcement Learning (RL) [75] is an AI approach, which has been used in blockchain applications [5]. RL learns the environment dynamics and chooses an action maximizing a long-term reward. Deep Learning was integrated with the traditional RL, called Deep Reinforcement Learning (DRL) to improve the performance of RL and overcome limitations like the curse of dimensionality [33]. Dueling Deep Q Networks (DDQN) [35, 36] is a popular Q-learning algorithm in DRL. Proximal Policy Optimization (PPO) has been shown to achieve better performance than the other policy gradient methods in DRL [53, 76]. This motivates the choice of these two approaches to improve the performance of Prism. DRL is incorporated into blockchain systems to better understand the dynamics of the blockchain system and optimize its performance. According to the blockchain trilemma, any blockchain system can only have at most two of the three features, decentralization, security, and scalability [5]. These three features have a trade-off relationship with each other, where maximizing one feature can drastically degrade the other feature [40].

We propose a DRL-based approach to a high-performance blockchain protocol, Prism. Recent works propose DRL approach to traditional blockchains but Prism has very high performance compared to the traditional blockchain systems. A Deep Reinforcement Learning based Prism Blockchain (DRLPB) scheme is proposed which used to optimize the Prism blockchain performance without compromising on the performance guarantees provided by Prism. We for-

mulate the DRL based approach for Prism and apply DDQN and PPO algorithms to optimize the performance of the Prism blockchain. This is the first work applying PPO to an on chain blockchain-based system to the best of our knowledge, and we show that this approach outperforms DDQN based optimization, which further outperforms standard Prism protocol. We demonstrate that the approach can be improved to have better security for the same throughput and latency. The code is available on github[1]. The security level of Prism is highly dependent on the number of votes received by the leader block. We achieve 84.47% more votes in the PPO based approach and 69.21% more votes in DDQN based approach than Prism.

## 4.1 System Description

The DRLPB consists of a blockchain protocol that runs on a distributed set of nodes connected by a physical network. The blockchain protocol used for DRLPB is discussed in this section along with the assumptions. The performance of Prism [20], is limited by the attributes of the underlying network. Latency is limited by the propagation delay and by the confirmation reliability. In a blockchain protocol, electing a leader block among all the blocks at each level of the block tree has three distinct roles: 1) add transactions to the main chain; 2) election of leaders; 3) vote for ancestor blocks through parent link relationships [21]. This is the main cause of latency and throughput limitations of the longest chain protocol due to the coupling of the roles carried by the blocks. In Prism, these limitations are overcome by factorizing the blocks into three types of blocks. For ease of understanding, we represent them as levels of blocks, which are the proposer blocks, the transaction blocks, and the voter blocks as shown in Fig. 5.2. A miner's block is randomly sortitioned using cryptographic sortition into one of the three types of blocks, and if it is a voter block, it will be further sortitioned into one of the $v$ voter trees.

#### 4.1.0.1 Level 1: Transaction Blocks

To decouple security from throughput, separate transaction blocks are employed in Prism. Transaction blocks are independent, without a parent block, and are mined at a faster rate. The transaction block consists of an ordered list of transactions, drawn from a memory pool. The number of transactions confirmed per sec in Prism depends on transaction blocks.

#### 4.1.0.2 Level 2: Proposer Block

Proposer blocks are the elected leader blocks. Proposer tree is built based on the longest-chain protocol. Proposer blocks choose as their parent block and contain a list of reference links to transaction blocks (which contains transactions), as well as a single reference to a parent

[1]https://github.com/DRLPB/DRLPB

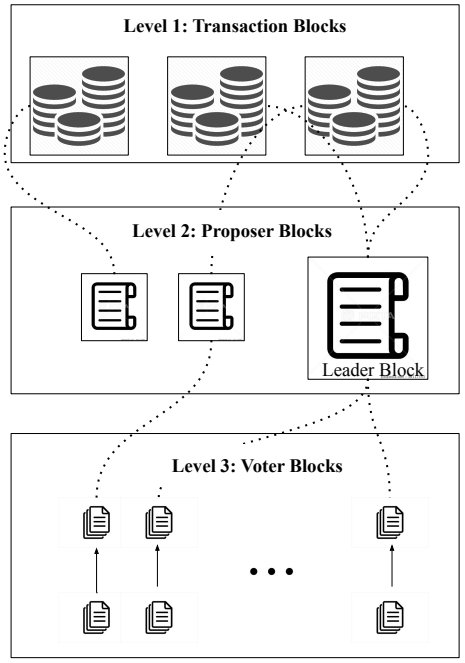



Figure 4.1: Consensus in Prism blockchain

proposer block. The final confirmed sequence of proposer blocks is the leader sequence and is determined by the voter trees. The reliability of Prism depends on Proposer blocks.

#### 4.1.0.3 Level 3: Voter blocks

Voter trees are built using the longest-chain protocol. The parent of voter block is the tip of the longest chain in the voter tree. The content of a voter block is a list of references to proposer blocks called votes. The voter block contains a list of one vote per non-voted level in the block's ancestors as shown in Fig 5.2. The number of voter chains, $v$ considered in Prism experiments is 1000 [21]. The leader voter block at each level is the one that has the highest number of votes among all the proposer blocks at the same level. The voter blocks do not exactly follow the chain structure and hence are referred to as voter trees. However, voter block mining is very similar to the others and we use voter trees and voter chains terminology interchangeably in this chapter. The security of Prism depends on voter blocks and the number of votes received by them.

### 4.1.1 Security

The security of the Prism blockchain depends on the votes from the voter trees, which also secure each leader proposer block. Changing an elected leader requires reversing enough votes to refer them to another proposer block in that level and the voter tree votes are based on the

longest chain protocol. In [21], it is shown that if the adversary has less than 50% hash power ($\beta \leq 0.5$), and the mining rate in each of the voter trees is kept small to minimize forking, then the consistency and liveness of each voter tree guarantee the consistency and liveness of the ledger maintained by the leader proposer blocks. Consistency is a guarantee that all honest parties output the same sequence of blocks throughout the execution of the protocol and liveliness refers to a protocol that can exchange messages among nodes, with the nodes successfully coming to a consensus. A low mining rate would result in high latency to wait for each voter block to get sufficiently deep in its longest chain. When there are many voter chains in parallel, the longest chain protocol guarantee can be achieved without requiring each and every vote to have a very low reversal probability, which is an improvement in latency. So the security of Prism depends on the mining rate of voter blocks, the number of voter chains and their votes.

Consider that there are $n$ proposer blocks at level $i$, denoted by $P = P_{b1}, P_{b2}, \cdots, P_{bn}$. Suppose $P_{bi}$ gets $v_i$ number of votes. A vote for proposer block, $P_{bi}$ is given by a voter block which is on the longest chain of its voter tree. Note that each voter block can vote only on one proposer block. Let $V_{bi}^p = V_1^p, V_2^p, \cdots, V_i^p$ is the set of votes that $P_{bi}$ has received. For every vote, $V_i^p$, let $u_{ij}$ is the depth of voter chain. The block inter-arrivals are exponentially distributed, the number of blocks mined after the block $P_{bi}$ is a Poisson random variable, with rate equal to its mean [21]. This can be related to the time elapsed since $P_{bi}$ was released via the block mining rate. Let the fraction of adversarial hashing power be $\beta$, and we can empirically estimate the average depth of existing public votes as $\bar{u} = \frac{\sum_{ij} u_{ij}}{\sum_i v_i}$ and a forking rate $\alpha$ for voter chains. There are many voter chains so the estimates converge to their true mean. Then, the average depth of a voter chain is,

$$\bar{u}_A = \frac{\beta \bar{u}}{(1-\alpha)(1-\beta)}.$$

The voter chains follow the longest-chain rule, same as in Bitcoin, so the votes reversal probability is a function of adversary hash power and the depth of confirmation [21].

### 4.1.2 Latency

Let the fraction of adversary hashing power be $\beta$ and $v$ be the number of voter trees and $p_\alpha$ is a small value representing the reversal probability of confirmed transactions. Then upper bound on the latency, $\Delta$, is given as follows [21]:

$$\Delta \leq D_a k_1(\beta) + D_a \frac{k_2(\beta)}{v} \log \frac{1}{p_\alpha}, \tag{4.1}$$

where $k_1$ and $k_2$ are constants which depend on $\beta$ [20]. $k_1(\beta) = \frac{5400(1-\beta)}{(1-2\beta)^3 \log \frac{1-\beta}{\beta}}$ and $k_2(\beta) = \frac{5400(1-\beta)}{(1-2\beta)^3} \log \frac{50}{1-2\beta}$. Honest transactions are confirmed in $\max\left(k_1(\beta)D_a, k_2(\beta)\frac{1}{v}\log(\frac{1}{p_\alpha})\right)$ time. For a large number of voter trees $v$, the first term dominates the above equation and therefore

Prism achieves near-optimal latency. This also implies that latency is proportional to the average propagation delay $D$ and independent of the reversal probability. The latency bound considered here is the worst-case latency bound that holds for all attacks described in [21].

The confirmation of a new leader proposer block at a given level, $i$ is determined by the voter trees growth and the votes on level $i$ that are embedded deep into their respective voter trees. The leader block is confirmed when a plurality of voter trees have voted for it, and is guaranteed not to change with probability at least $1 - z$, where $z$ is a user-selected target reversal probability in the presence of adversary that controls a fraction $\beta$ of the total hash power.

### 4.1.3 System Model Assumptions

In the proposed system, it is assumed that there are N nodes and V voter chains. At the beginning of each epoch of the blockchain, (after consensus and adding a block to the blockchain), a node solves a Proof of Work (POW) puzzle to participate in the mining process. After solving the PoW puzzle, each miner mined block will be assigned a proposer or a transaction, or a voter. A mining modifier is defined for transaction block, proposer block and voter block. Changing the mining modifier changes the Prism blockchain performance. The Prism separates the process of confirming blocks and forming a ledger. The functionality of ledger formation is divided into ledger manager, block structure manager and miner. The ledger manager updates the ledger based on the latest blockchain state and the miner assembles new blocks. The block structure manager maintains the clients' view of the blockchain and communicates with peers to exchange new blocks [21]. The block structure manager sanitizes the ledger and orders the transactions periodically. After the consensus, the ledger sanitization is performed during a predefined period, which is set to 1 epoch. The goal of the Prism client is to maintain up-to-date information of the blockchain and the ledger. The number of nodes joining and leaving the network is random and the security of a blockchain protocol like Prism highly depend on the voter blocks mined at each epoch. Our proposed DRL approach can be used to decide the security level of the blockchain by deciding the optimum value of the number of voter trees. The DRL algorithm can be executed at the block structure manager periodically and update the blockchain parameters. The block structure manager follows the same peer-to-peer protocol for updating the DRL parameters as that of updating new blocks. Since the Prism client by default is decentralized, our implementation is also decentralized. The Prism client in our implementation maintains the latest information of the blockchain, the ledger, and the updated parameters of DRL. The malicious activity of block structure manager would be reducing voter chains which would result in more votes received by the the favorable block. The voter chain value cannot be decreased more than the default value of Prism. This would not degrade the performance of the protocol. Another malicious activity would be increasing mining rate very high making it difficult to mine new voter blocks. The upper limit on the mining rate is set to

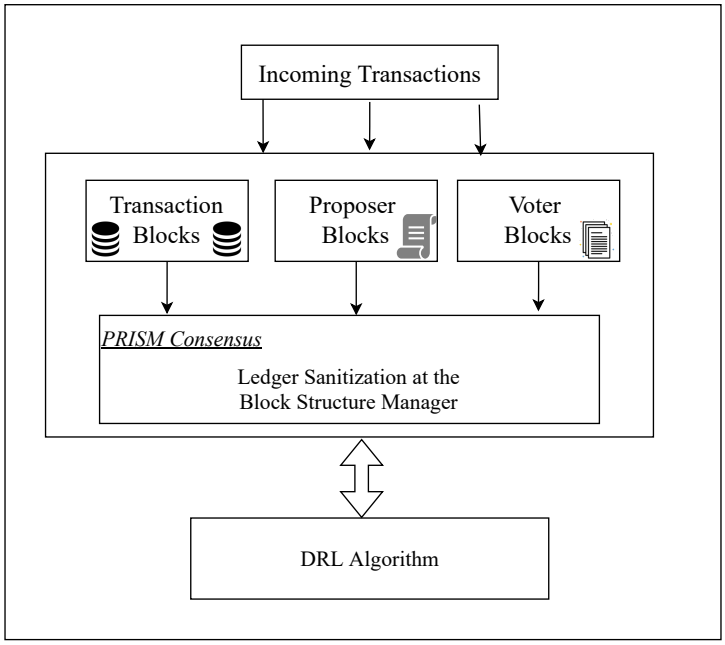



Figure 4.2: System model illustrating Prism blockchain

Prism default value, this helps in maintaining Prism performance even in case of any malicious activity caused during DRL execution. An illustration of the system model is shown in Fig 4.2.

## 4.2 The DRLPB Approach

The objective of this work is to ensure an intelligent, and secure Prism blockchain system by mapping the number of nodes participating in voter chains to the optimized Blockchain configuration considering the trade-off between the three metrics in a blockchain system, security, latency, and cost. In Prism [20], the functionality of blockchain is decoupled and hence, the trade-off is redefined for our analysis. In our work, we try to maintain the preserve the performance of Prism and improve the security level by mainly focusing on the voter chains. We discovered that there is a trade-off between the confirmation latency, the number of voter chains, and the mining rate analogous to the latency, security, and cost trade-off. Two reinforcement learning approaches, DDQN and PPO are adopted to optimize blockchain performance. At each time step, the input transaction data is analyzed to decide the blockchain configuration. The decision is made based on predefined requirements of security level and latency.

### 4.2.1 Markov Decision Process Model

The Markov Decision Process (MDP) environment represents the received voter blocks information from the nodes of the blockchain, where our DRL algorithm needs to interact along with the Blockchain network. The DRL agent is not aware of the environment beforehand. It

uses its experience of interacting and observing the rewards for various decisions in different states and then learning the optimal policy $\pi^*$ to map the states with their best actions.

#### 4.2.1.1 Environment

The optimization problem is formulated as a MDP. The MDP environment is represented by the tuple $< S, A, R, \gamma >$, where, $S$, is the state space, $A$, is the action space, $R$, represents the reward function and $\gamma$ is the discount factor ranging from 0 to 1. At every time step, $t$, the agent receives a state representation, $s$ of the environment. The agent takes an action $a$ in the environment based on a policy $\pi(a|s)$. Then, a transition to the next state, $s'$ occurs with a probability $P(s'|(s, a))$, and a reward, $r_t$ is received by the agent, which is explained in detail in the following subsections. In Prism, each node participating in the mining process solves a PoW puzzle, whose difficulty level is preset based on various parameters like the number of nodes participating in the network. In Prism implementation [21], a mining modifier is used which is used to calculate the overall blocks mined in the blockchain per sec, including transaction, proposer, and voter blocks. In our work, we mainly consider the parameters related to voter block and its mining since we try to focus on the security level of Prism. A DRL method is then used to solve the formulated MDP.

#### 4.2.1.2 State

The state in DRL reflects the environment at a given time, in this case, the consensus information of the blockchain. The state space $S$ consists of a set of vectors representing the number of nodes, the number of voter chains, and the delay corresponding to each voter block added to the chain. The state space is chosen such that we have the information of the entire blockchain for modifying the mining rate and also we have a special focus on the voter chains (the decoupled blockchain). The state is defined as,

$$s_t = [N, V, D]^t, \tag{4.2}$$

where $N$ is the number of nodes participating, $V$ is the number of voter chains, and $D = [d_1, d_2, \cdots, d_V]$ is the delay. The delay vector corresponds to the delay values of the voter block mined.

#### 4.2.1.3 Action

An agent maps the space of states, $S$ to the space of actions, $A$. At every time step $t$, the agent selects an action $a_t \in A$ such that the we have maximum number of voters for that time step. The DRL agent selects the number of voter chains, $V^*$ for each time step $t$ and the mining rate of voters be $M_r$. The number of voter chains in Prism implementation was 300. We initialize the value of $V$ as 300 and the agent can choose the number of voter chains from

a range of 270 to 300, each time with a change of $\Delta V \in \{+10, 0, -10\}$. The voter mining rate in Prism is set to 0.1. We initialize $M_r$ at 0.1 and each time the agent can change the value of $\Delta M_r$ by $\{+0.01, 0, -0.01\}$. The mining rate is a discrete set of values in the range of $[0.01, 0.1]$. Voter mining rate is the rate at which voters mine the blocks. The range is chosen for an acceptable level of performance, which is preserving Prism blockchain performance guarantees. If the mining rate is higher than 0.1, the voter blocks are mined faster, resulting in more voter blocks than required, which is a waste of resources. If it is less than 0.01, they are mined slowly resulting in a very low throughput of voter chain. After choosing the change in action values of $\Delta M_r$, and $\Delta V$, the agent sets new value the number of voter chains and mining rate of the blockchain as $M_r^*$, and $V^*$. For the first iteration, if an action is chosen to decrement both the values, then $M_r^* = M_r - \Delta M_r$ and similarly $V^* = V - \Delta V$. Then, the action selected $a_t$ can be represented as

$$a^t = [\Delta M_r, \Delta V]^t.$$

The action space controls the voter chains in Prism blockchain (decoupled functionality). This allows us to modify the mining rate of just the voter blocks and the number of voter chains without affecting the other decoupled functions of Prism like transaction blocks and proposer blocks. The mining rate of the proposer, transaction and miner can be controlled as discussed in [21]. In this work we control the voter mining rate.

#### 4.2.1.4 Reward

The reward function is the ultimate goal of optimizing the number of votes received in the voter chain, while meeting application-level requirements like security and latency. The reward function equation is obtained from the implantation code of Prism [21], which represents the votes received for each blockchain iteration. Based on the current state and action, the agent obtains a reward from the environment. The security of Prism blockchain depends on the voter trees, in turn, the number of voter blocks confirmed. The reward function represents the number of votes received in the voter chains, which is represented as follows:

$$r_t = \frac{V^* M_r^* \Lambda}{D_a C_p} \quad \text{if } \Gamma < (\tfrac{1-2\beta}{\beta}) \text{ and if } \Delta < D_a k_1(\beta) + \tfrac{D_a k_2(\beta)}{m} \log \tfrac{1}{p_\alpha}$$
$$r_t = 0 \quad \text{otherwise,}$$

where $\Lambda$ is the throughput which is defined as the number of voter blocks confirmed per sec, $M_r$ is the mining rate and $V^*$ is the number of voter chains, $D_a$ is the average of the elements in $D$, which is a vector corresponding to the delay of each voter block. The constraints in our implementation use $D_a$. A throughput parameter $T$, is used in the calculation of overall throughput of the network, including voter, proposer and transaction blocks. $T$ is defined as a network parameter and is used in the calculation of $C_p$ in [21]. $C_p$ is calculated as $\frac{64000}{168}T$, which is given in the [21] implementation. The reward function is chosen in such a way to maximize

the number of votes received by the leader block. The number of votes would depend on the mining rate of voter blocks and number of voter chains. The agent learns to maximize the reward, which corresponds to the maximum achievable security at the given time. The reward is a function of the current state, which is delay corresponding to each voter block. The output of Prism is a log of the propagation delays corresponding to each voter block which is used in our experimentation. This delay depends on number of nodes as the logs are generated at each node. The state also needs the information of current number of voter chains to increment or decrement its action parameters. The reward is also a function of the action, which is the mining rate and maximum number of voter chains.

#### 4.2.1.5 DRL Agent

The main objective of the agent is to find the best mapping between the state and the action that achieves a maximum reward. The agent uses the available state samples and calculates the estimated reward that can be obtained by taking an action $a$ from a state $s$.

The agent tries to maximize the future rewards, which are expressed as the sum of all discounted future returns following time step $t$. The discounted future reward with the discount factor, $\gamma$, is represented as

$$R_t = r_t + \sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau}, \tag{4.3}$$

where $\gamma \in [0, 1]$. The behavior of the agent is described by a policy $\pi(a_t|s_t)$ which is the probability of choosing action $a_t \in A$ condition on the observed state $s_t \in S$. The action-value function under policy $\pi$ describes the expected return of an action for the agent in a given state, $Q_\pi(s, a) = \mathbb{E}_\pi[R_t|s_t, a_t]$, where $\mathbb{E}_\pi$ denotes the expected value given that the agent behaves according to policy $\pi$. The optimal policy $\pi^*$ has the optimal action-value function $Q^*(s, a) = \max_\pi Q_\pi(s, a)$, and satisfies the Bellman optimality equation. Bellman equation states that the optimal action-value function is the expected reward of action a plus the discounted expected value of the best action in the subsequent state $s_{t+1}$,

$$Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t]. \tag{4.4}$$

The optimal action-value function $Q^*(s, a)$ is the maximum expected return achievable under any policy, after seeing a state $s_t$ and taking an action $a_t$ and $\pi$ is a policy mapping states to actions [77]. The objective is to maximize the Q function while satisfying the latency and security constraints.

$$\max_{\pi} Q_{\pi}(s,a) \tag{4.5}$$

$$\text{Security constraint} \quad \Gamma < (\tfrac{1-2\beta}{\beta}) \tag{4.6}$$

$$\text{Latency constraint} \quad \Delta < D_a k_1(\beta) + \tfrac{D_a k_2(\beta)}{m} \log \tfrac{1}{p_\alpha} \tag{4.7}$$

$\beta$ is the adversary power $k_1, k_2$ are dependent constants as defined in Eq. (4.1). $\beta$ is less than 0.5 in the network. At the end of each DRL algorithm iteration, the new state is updated based on the delay vector corresponding to the voter blocks in the voter chains whilst the number of voter chains and mining rate is chosen by the actions with a goal to maximize the reward. The new state is formed by the number of voter blocks in the voter chains and their corresponding delays as an output of the blockchain, based on the action taken in the previous time step. Once the action is decided by the agent, the voter chains and mining rate values are given to the Prism blockchain environment. After execution of the Prism blockchain consensus, the delay values for each voter block along with the number of voter chains are given as next state input to the DRL agent. Given a voter mining rate $M_r$, and the average block propagation delay , $D_a$, then the maximum number of voter block chains possible is [20],

$$V = \frac{0.1CD_a}{\bar{M_r}B_v} - \frac{B_p}{B_v}, \tag{4.8}$$

where, $B_v$ is the block size of the voter block, $C$ is the network capacity and $\bar{M_r} = M_r D_a$ is the voting rate. $B_p$ is the proposer block size.

### 4.2.2 DDQN based Approach

The Prism blockchain output data is given to the DDQN network which outputs the action variables to the blockchain network for better performance. Fig 4.3 shows an illustration of the integration of DDQN and Prism blockchain. DDQN has two DNNs. The first is the evaluate network, which takes the current state-action pair $(s_t, a_t)$ as input, and output is a predicted value of them. The other is a target network, which takes input as the next state $s_{t+1}$ and output is the maximum Q-value of the next state-action pair. The replay memory stores experience tuples which include the current state, the selected action, reward, and next state. The stored experience tuples can be randomly sampled for training the primary network and target network. Randomly sampling experience tuples aim to reduce the effects of data correlation. Learning the optimal action for a particular state starts by taking several random actions for exploration purposes. All previous experiences, including the state, action taken, reward received, and a new state, will be stored in an experience replay memory. The target network parameters, $\theta$, will be updated every certain number of time steps, $U$.

The optimal action-value function obeys the Bellman equation. The optimal value $Q^*(s', a')$ of the next state $s'$ at the next time-step was known for all possible actions $a'$, then the optimal strategy is to select the action $a'$ is to maximize the expected value of $r + \gamma Q^*(s', a')$.

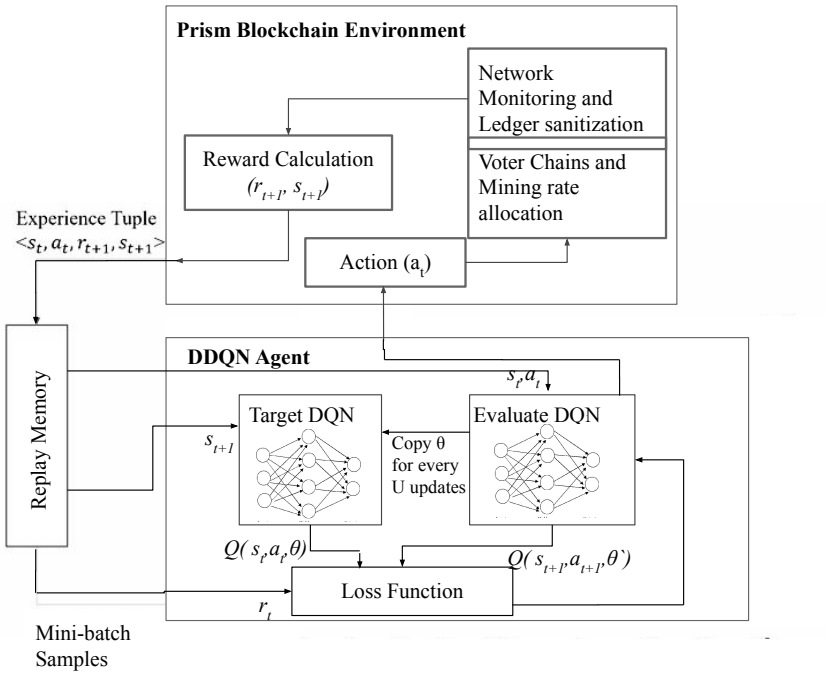



Figure 4.3: Network Architecture of DDQN applied in Prism

**Algorithm 3** DDQN based DRLPB

Initialize the network parameters $Q_\theta$, target network $Q'_\theta$, replay buffer B, and $\tau << 1$
**for** episodes 1,2, …, $E$ **do**
  **if** Eq. (4.6) and (4.7) are satisfied **then**
    Choose $a_t$ with an $\epsilon$ greedy policy from $s_t$
    Get a reward $r_t$ and go to next state $s_{t+1}$
  **else**
    Get a reward 0 and go to next state $s_{t+1}$
  **End If**
  Store the experience $\{s_t, a_t, r_t, s_{t+1}\}$ in B
  **if** B is full **then**
    Randomly select K mini-batch samples from B
    **for** k= 1 to K **do**
      Compute target Q value

$$Q_k^*(s_t, a_t) = r_t + \gamma Q_\theta(s_{t+1}, \text{argmax}_{a'} Q_{\theta'}(s_{t+1}, a'))$$

      Perform stochastic gradient descent on Eq. 4.10
    **End for**
  **End if**
  Update network parameters of the DDQN

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

**End for**

$$Q^*(s,a) = [r_t + \gamma Q(s_{t+1}, \text{argmax}_{a'} Q'(s_t, a_t)]. \tag{4.9}$$

We minimize the mean squared error between $Q$ and $Q^*$, but we have $Q'$ that slowly copies the parameters of $Q$. This is done by periodically copying the parameters:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'.$$

where $\theta'$ is the target network parameter, $\theta$ is the evaluate network parameter, and $\tau$ is the rate of averaging and is set to 0.01. We use a neural network as a function approximator, to estimate the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$, with weights and is referred to as a Q-network. The Q-network is trained by minimising a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration $i$,

$$L_i(\theta) = \mathbb{E}[(y_t - Q(s,a,\theta))^2]]. \tag{4.10}$$

The target used by DDQN is,

$$y_t = R_{t+1} + \gamma Q(s_{t+1}, \text{argmax}_a Q(s_{t+1}, a, \theta); \theta'). \tag{4.11}$$

The replay memory comprised of the agent's last few experience tuples $(s_t, a_t, r_{t+1}, s_{t+1})$. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\Delta_\theta L_i(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a', \theta') - Q(s,a,\theta))\Delta_\theta Q(s,a,\theta)]. \tag{4.12}$$

We optimise the loss function by stochastic gradient descent (SGD). The weights are updated after every time-step, and the expectations are replaced by single samples, like in the Q-learning algorithm [77].

DDQN has a target network which uses the expected Q values for the next state by considering the action that gives the maximum Q-value of the next state and updates the Q values of the evaluated network based on the estimated Q value from the target network. During the training process, the target network is periodically updated, and all the actions selected from Q-values are bounded actions from the constraints defined in Eq. (4.5). Algorithm 3 shows the detailed steps of the training process for $E$ number of episodes. $\epsilon$-greedy is used for exploration. During exploitation, the agent chooses the best action that maximizes the Q function. We use a feed-forward network architecture with two hidden layers. The size of the replay memory is $10^6$ tuples. The probability of selecting a random action, $a_t$, is 1 at the beginning of the training, and reduced linearly in time over the first quarter of training time steps to a value of 0.01 for the remaining training.

### 4.2.3 PPO based Approach

The Proximal Policy Optimization (PPO) algorithm is a state-of-the-art DRL algorithm based on the actor-critic method. In PPO, the primary network consists of two deep neural

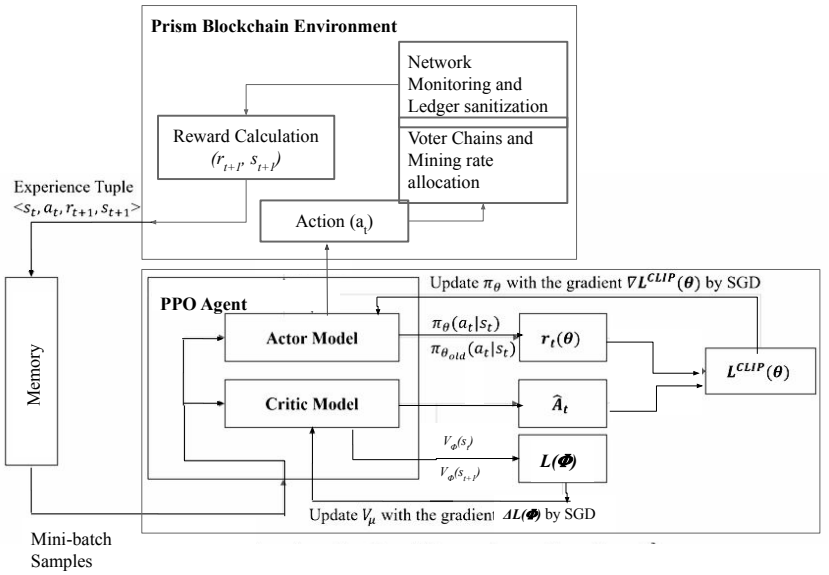



Figure 4.4: Network Architecture of PPO applied in Prism

networks, called the actor-network and the critic network. The actor-network is used to explore the policy, and the critic network estimates the performance. The critic-estimate value helps the actor to learn the gradient of the policy. The target network consists of an actor-network and a critic network. The next state from replay memory is the input of the target network and the output is a critic value for training critic. The replay memory stores experience tuples that include the current state, the selected action, reward, and next state. The stored experience tuples are randomly sampled for training the primary network and the target network.

The current state $s_t$ is given as input to the actor-network and distribution of actions is obtained. The agent gets its action by sampling from the action distribution. The environment gives the agent a reward for the action taken indicating whether the action is beneficial and this reward is used by the critic network. The output of the critic network is a component of the actor's loss function. The actor-network generates the policy and the critic network evaluates the current policy by estimating the advantage function $\hat{A}_t$. The policy is modified according to the advantage function, given as

$$\hat{A}_t^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{4.13}$$

The clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\delta, 1+\delta)\hat{A}_t)], \tag{4.14}$$

where $\rho(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\rho(\theta_{old}) = 1$ is the probability ratio, $\theta$ is the policy parameter, and $\delta$ is a hyper parameter. $\phi$ is the value function parameter. The probability ratio between the action under the current policy and the action under previous policy is represented as $\rho(\theta)$. $\pi_\theta$

**Algorithm 4** PPO based DRLPB

Initialize the policy parameters $\theta_0$, the value function parameters $\phi_0$ and clipping threshold $\delta$
**for** episodes k=1, …, E **do**
    Follow a policy $\pi_{\theta_{old}}$ for $\tau$ time steps
    collect $\mathbb{D} = \tau_i$ set of trajectories
    Compute the rewards $R_t$
    Compute $\hat{A}_t$ in Eq. (4.13) using $V_\phi$
    **for** epoch t:=1, …, $\tau$ **do**
        **if** Eq. 4.6 and 4.7 are satisfied **then**
            Take $a_t$ from $s_t$ based on actor policy
            Get a reward $r_t$
        **else**
            Get a reward $r_t = 0$
        **End if**
        Collect $s_t, a_t, r_t, s_{t+1}$
        Update $s_t$ to next state $s_{t+1}$
        After every K steps
        Update policy by calculating $\theta_{k+1}$ by stochastic gradient ascent to maximize clip objective in Eq. (4.14)
        Update the value function using stochastic gradient descent by calculating $\phi_{k+1}$ by mean squared error

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathbb{D}_k|T} \sum_{\tau \in \mathbb{D}} \sum_{t=0}^{T} (V_\phi(s_t) - R_t)^2$$

    **End for**
    $\theta_{old} \leftarrow \theta$
**End for**

| Parameters | Value |
|---|---|
| Memory pool maximum size | 500,000 transactions |
| Proposer block mining rate | 0.1 blocks/sec |
| Average Transaction size | 20Bytes |
| Maximum block size | 2 MB |
| Maximum Block Interval | 100 ms |
| Maximum voter chains | 1000 |
| Maximum votes | 1000 |
| Data transmission rate | 100 MBPS |
| Batch size | 16 |
| Throughput parameter | 5000 |
| Mining modifier | 2.3 |

Table 4.1: Simulation Parameters

is a stochastic policy. $\theta_{old}$ is the vector of policy parameters before update. The second term, $\text{clip}(r_t(\theta), 1-\delta, 1+\delta)\hat{A}_t$ modifies the objective by clipping the probability ratio, which removes the incentive for moving $\rho_t$ outside of the interval $[1-\delta, 1+\delta]$. Finally, we take the minimum of the clipped and unclipped objective, so the final objective is a lower bound on the unclipped objective. Without a constraint, maximization of $L$ would lead to an excessively large policy update and hence, it is modified to penalize changes to the policy that move $\rho_\theta$ away from 1. The main objective is the following:

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 H[\pi_\theta|s_t]]. \tag{4.15}$$

The advantage estimator $\hat{A}_t$ depends on the state-value function $V^\pi(s_t)$, which is estimated with a second network, the critic network. As parameters are shared between the state-value function estimator and the policy surrogate $\pi(a_t|s_t;\theta)$, a value-function error term $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$ is added to the objective function (scaled by a positive constant parameter $c_1$, where $V_t^{targ}$ denotes the state-value function of training target. The entropy of the policy $\pi$ for state $s_t$ is $H[\pi_\theta|s_t]$ which is scaled by a constant positive parameter $c_2$ in order to encourage exploration. The training process of the overall workflow is shown in Algorithm 4. The policy network is a feed-forward network with two hidden layers. We sample from a single environment and use the default trajectory length of 2048. The weighting factor of the generalized advantage estimator is set to 0.95. The clipping rate is set to 0.2, the value-function error and entropy terms are scaled by $c_1 = 0.5$ and $c_2 = 0.01$, respectively.

## 4.3 Results

The performance of the proposed DRLPB approach is evaluated through simulations in terms of the reward convergence of the proposed approaches.

### 4.3.1 Experimental setup

The simulation environment consists of the Prism blockchain and an initial 300 voters. The proposed DRLPB with DDQN and PPO is implemented with PyTorch. The parameters used in the simulation are summarized in Table 4.1. Block interval is the time taken by a node to solve a nonce, which is the difficulty of the PoW puzzle. In [21], the Prism protocol is run on the AWS servers for 10 minutes and conclusions are derived from the data. Similarly, we run the Prism protocol for 10 minutes to implement our DRL algorithm. The performance is analyzed in terms of the average reward, which represents the number of voter blocks per second of the blockchain under the restrictions of the security constraints. In each episode, the voters are redistributed and the consensus is carried out. A block structure manager described in Section 4.1.3 runs the proposed DRL approach and decides the security level of the blockchain by deciding the optimum value of the number of voter trees. We have run the Prism Blockchain on 5 nodes and collected the data for 10 minutes. The data needed for DRL algorithm is available to all the nodes in the network and does not require additional communication between the node. This data is given to the block structure manager, which runs the DRL algorithm and updates the blockchain parameters.

### 4.3.2 Performance Comparison

We analyze the Prism blockchain with the proposed DRL method for number of voter chains and mining rate. The latency threshold $\lambda$ and security threshold $\Delta$ are derived from Prism analysis in [21]. When DRL is applied, the system maintains Prism performance guarantees. We try to maintain the performance of Prism and improve the security level by mainly focusing on the number of voter chains. We analyze the proposed DRLPB approach with the goal to maximize the rewards subject to the security and latency constraints as described in Eq. (4.5). Therefore, the reward function tries to optimize the objective while considering the constraints.

#### 4.3.2.1 Reward Convergence

In the training process of Algorithm 1, for the DDQN approach, we set the hyperparameter values according to the parameters reported in Table 4.1. These hyperparameters are set based on experiments and observations to maximize performance. The hyperparameter tuning is discussed in Appendix. The reward is observed considering DDQN with a deep neural network of four layers. The reward values shown are smoothed over a window of 20 episodes. Fig 4.5 demonstrates that the proposed DDQN approach converges empirically. The agent explores the environment through random actions for the first 200 episodes and then improves that random policy toward the optimal policy by exponentially decaying the value of $\epsilon$. Prism data is collected and given to the DDQN algorithm. The DDQN agent is trained on the blockchain data and the DDQN algorithm converges with an average reward of 665 for a batch size of 16.

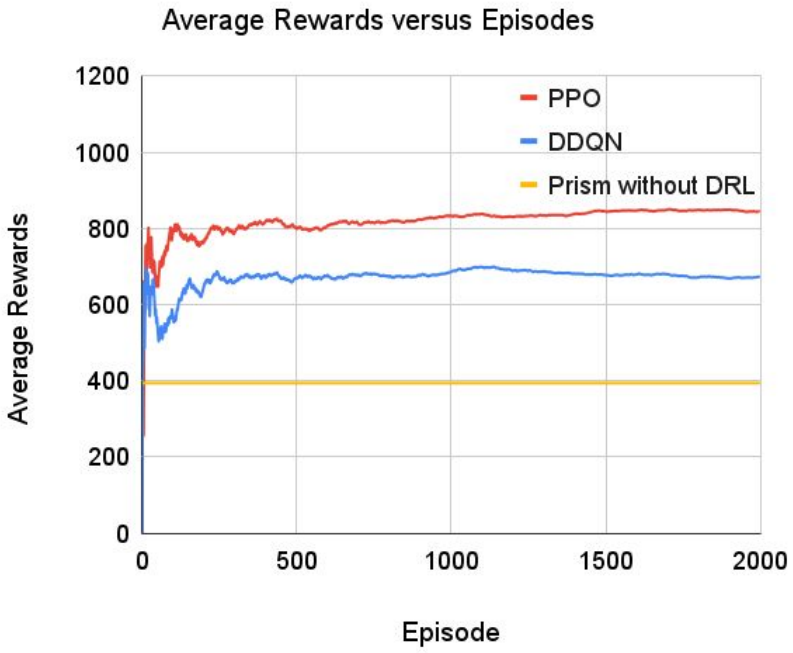



Figure 4.5: Average Reward vs Episodes for DDQN, PPO and Prism is plotted. PPO achieves higher average rewards compared to DDQN and Prism for all the episodes.

In the PPO approach described in Algorithm 4, the hyperparameter values are set according to the parameters reported in Table 4.1. These values were set based on experiments and observations in order to maximize the performance. In PPO, we use a deep neural network with four layers. The input layer is the state and the output layer gives the actions. The hidden layers with 200 neurons is used. Similar to DDQN performance, PPO reward converges in 2000 episodes. The PPO algorithm converges with an average reward of 725 for a batch size of 16.

In Fig. 4.5, we note that the DRL algorithms were able to find the policy that addresses and maximizes the long-term trade-off defined by the reward function in Eq (4.3). The DRL maximize the reward while maintaining the minimum security level and not limiting the transaction confirmation time onto the blockchain. This implies that when the latency is low, the rewards are higher and as the delay increases, the reward received decreases.

Fig. 4.5 shows the overall performance of the approach considered. The average rewards of DDQN, PPO, and Prism without applying DRL are plotted against the testing episodes. The Prism blockchain without RL has a fixed number of voter chains at 300 and a mining rate fixed at 0.1.The total throughput of Prism is measured in transactions per sec. These transactions are grouped into transaction blocks, which are referenced by proposer block and the voter blocks vote on their ancestor blocks. We compare the number of voter blocks per second in Fig. 4.7. By applying DRL, we optimize the number of voter chains and the mining rate. The voter chains can choose values between 270 to 300 with a step size of 10 and mining rate can be in the range of 0.01 to 0.1. The delay varies between 10 ms to 60 ms. Fig 4.5 shows

the average reward in all three schemes. Prism without DRL, DDQN, and PPO achieves an average reward of 393, 670, and 730, respectively.

#### 4.3.2.2 Performance Evaluation for varying blockchain parameters

The throughput parameter $T$, which is a constant is varied and the DDQN and PPO algorithm are executed. The average rewards achievable by DDQN ad PPO are plotted in Fig. 4.6. Another metric we consider while evaluating the performance is number of voter blocks mined per second. Fig. 4.7 shows the voter blocks mined per second in DDQN, PPO, and Prism without DRL. The number of votes at $T = 7000$ are lower than that at $T = 5000$ (Fig. 4.6), while the number of voter blocks confirmed is higher for PPO (Fig. 4.7). This is true for all algorithms comparing between $T = 2000$ and $T = 5000$. Thus, we get more throughput (in terms of voter blocks) compromising on the security level. It is worth noting that the security associated with the voter blockchain defines the security of the entire Prism blockchain. We also note that PPO algorithm has better performance in all the cases.

In Fig. 4.8, the average reward of the DDQN, PPO, and Prism without DRL are plotted for different mining modifier values, $f$. The mining modifier of voter chain decides how fast the mining has to happen. Changing the mining modifier changes the Prism blockchain performance. With a higher $f$, more voter blocks are mined for the same number of participating nodes, resulting in a lower number of votes received at each level of the voter chain. In other words, $f$ is used to calculate the difficulty level of the PoW puzzle. The higher the value of $f$, the faster the blocks are mined, leading to a higher value of the average reward. For $f = 1.3$

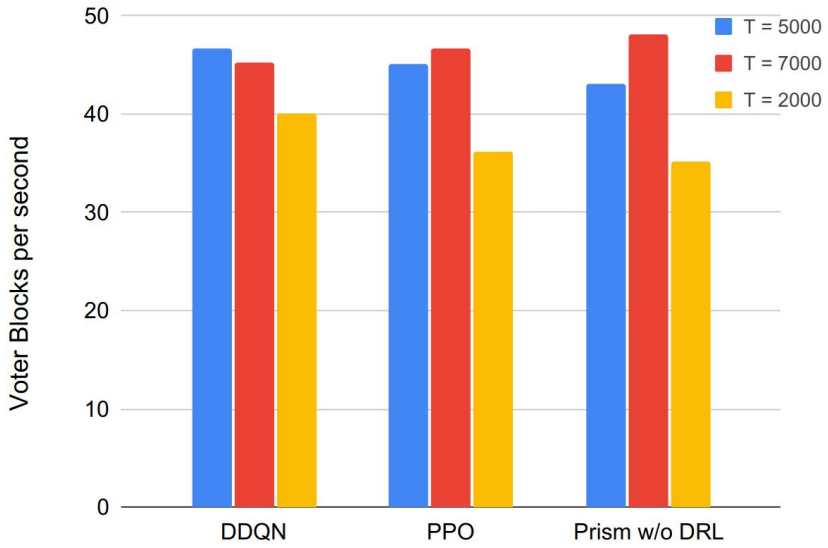



Figure 4.6: Performance comparison of DDQN, PPO and Prism in terms of maximum number of votes per sec. $T = 5000$ acheives better reward compared to $T = 7000$ and $T = 2000$.

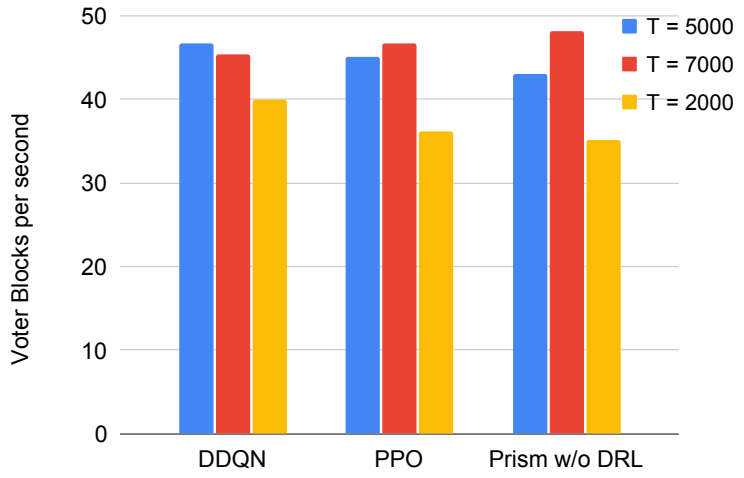



Figure 4.7: Performance comparison of DDQN, PPO and Prism in terms of maximum number of voter blocks per sec for varying $T$. The greater the value of $T$, the higher the number of voter blocks mined per sec in all three schemes.

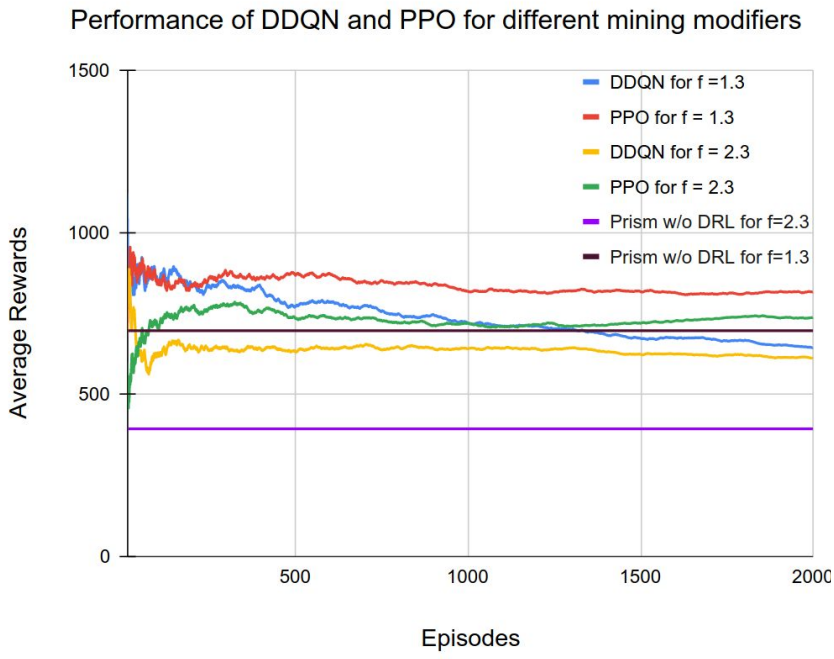



Figure 4.8: Performance comparison of DDQN, PPO and Prism for different mining modifiers. Higher average rewards are observed for $f = 1.3$ in all three schemes.

| | | Prism | DDQN | PPO |
|---|---|---|---|---|
| Total mined blocks (blkps) | f =2.3 | 40 | 40 | 40 |
| | f =1.3 | 43 | 43 | 43 |
| Transactions per sec | f =2.3 | 17949 | 18115 | 18091 |
| | f =1.3 | 22543 | 22414 | 25575 |
| Confirmation latency (s) | f =2.3 | 71 | 97 | 85 |
| | f =1.3 | 49 | 64 | 105 |
| Voter block | f =2.3 | 1669 | 1622 | 1714 |
| propagation delay (ms) | f =1.3 | 1881 | 1635 | 1760 |
| Votes Received | f =2.3 | 393 | 641 | 723 |
| Votes Received | f =1.3 | 696 | 764 | 817 |

Table 4.2: DRLPB Performance for varying mining modifier

| $\beta = 0.2$ | Prism | DDQN | PPO |
|---|---|---|---|
| Transactions confirmed per sec | 10214 | 10325 | 11290 |
| Confirmation latency (s) | 73 | 91 | 82 |
| Voter block propagation delay (ms) | 1558 | 1601 | 1543 |
| Total Votes Received | 393 | 673 | 844 |

Table 4.3: DRLPB Performance for $\beta = 0.2$

| $\beta = 0.3$ | Prism | DDQN | PPO |
|---|---|---|---|
| Transactions confirmed per sec | 1128 | 1081 | 1118 |
| Confirmation latency (s) | 49 | 74 | 87 |
| Voter block propagation delay (ms) | 1884 | 1809 | 1673 |
| Total Votes Received | 294 | 566 | 616 |

Table 4.4: DRLPB Performance for $\beta = 0.3$

Prism without DRL could achieve an average reward of 393, DDQN could achieve 641 and PPO could achieve 723. For $f = 2.3$, Prism without DRL could achieve an average reward of 696, DDQN could achieve 764 and PPO could achieve 817. PPO has much less convergence time compared to DDQN in all the simulations. PPO is also able to reach higher rewards in all the cases compared to DDQN and Prism. Since rewards are the number of votes received by the voter chains, the voter blocks are more secure.

The results in Table 4.2 show the performance of Prism Blockchain without application of DRL and with DQN and PPO approaches. The outputs are taken as an average across all the nodes. This is performed ten to fifteen times and the average values are given in the table. We run the Prism blockchain for about fifteen times for this performance evaluation. During the experimentation, the output log files are fed to the DRL algorithm. The DRL algorithm is

executed at the ledger manager and all the nodes are given the outputs from DRL algorithm like number of voter chains and mining rate as an input. The total mined blocks remain same in all three schemes. The throughput of the entire Prism blockchain including the Transaction blocks, Proposer blocks, and Voter blocks is measured in number of transactions confirmed per sec. The throughput of Prism reaches the physical limit, which means that the bottleneck is in number of transactions that could be generated not in the consensus procedure. We maintain the same performance as in Prism blockchain with respect to number of transactions confirmed per sec. Confirmation latency is the time taken for a final new proposer leader block to be added into the blockchain. This includes the confirmation latency of proposer, transaction and voter chains but these three are processed in parallel. It also includes the propagation delay associated with proposer, transaction and voter blocks across the network. In Prism, the bottleneck of latency was the physical limit, which means the communication bandwidth of the network. Prism output data is considered for determining the confirmation latency and the voter block propagation delay. We are able to maintain the performance of Prism protocol in terms of confirmation latency and voter block propagation delay for mining modifier values 1.3 and 2.3. The number of votes received by the voter block are responsible for the security of the Prism protocol. The number of votes received by the voter chains at each level is highest in PPO than DDQN, which is higher than Prism without DRL. A similar observation is performed for varying adversary power $\beta$ as 0.2 and 0.3. The performance comparison is shown in Tables 4.3 and 4.4. The proposed approaches gives the same overall performance guarantees as Prism while improving the security of Prism protocol.

## Additional Details

### 4.3.3 Effect of Mining Rate

The mining rate of a blockchain and throughput are directly related. Let us consider only the voter blocks. If $B_v$ is the block size in the number of entries in the block and let $M_r$ be the mining rate. Then the number of voter blocks is the voter throughput, which is is at most $B_v M_r$. Note that this is not the overall transactional throughput of the blockchain but just the voter blocks mined per sec.Both the mining rate and the block size are constrained by the security requirement. Increasing the mining rate increases the amount of forking of the blockchain due to multiple blocks that are mined on the same leaf by multiple miners within the network delay. To be secure against an adversary with $\beta$ hash power, the following equation must be satisfied.

$$B_v M_r = \frac{1 - 2\beta}{\beta} \tag{4.16}$$

The value of $B_v M_r$ must be kept smaller than 1 and the security requirement limits bandwidth utilization.

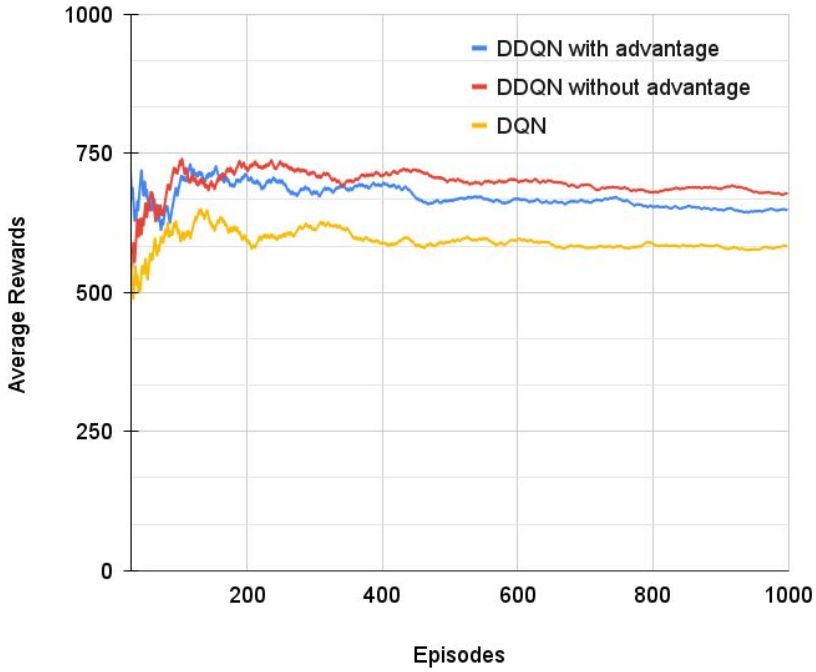



Figure 4.9: Comparison of DDQN with and without advantage function.

### 4.3.4 Selection of DDQN Algorithm

When there is a high-dimension state and/or action space, the tabular representation of Q learning is replaced by a neural network function approximator called Deep Q-Network (DQN). DQN algorithms use Q-learning to learn the best action to take in the given state and a deep neural network to estimate the Q value function. The maximization operator in Q-learning and DQN, use the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in extremely optimistic value estimates. To prevent this, Double Q-learning (DDQN) algorithm decouples the selection from the evaluation. Two value functions are learned by assigning each experience randomly to update one of the two value functions. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. The Dueling Network Architecture has an advantage function which partly helps in its ability to learn the state-value function efficiently. With every update of the Q values in the dueling architecture, the value function is updated.

In Figure 4.9, we compare the DQN and DDQN architectures and note that DDQN provides higher reward, which motivates the choice of DDQN in this work. We further test DRLPB for DDQN with advantage function and without. We note that not choosing an advantage function gives slightly better performance and thus we do not choose advantage function in the work.

# Chapter *5*

# IoT enabled Prism Blockchain using DRL

Blockchain is a distributed ledger technology that enables secure and immutable transactions [3]. It is a decentralized, distributed and public ledger that is used to record transactions across many nodes. It is used for recording and maintaining information about transactions in a secure and immutable manner. In addition to cryptocurrency, blockchain has numerous uses in Internet of Things (IoT), healthcare, and privacy [10]. Blockchain technology provides a secure and reliable way of storing data, making it ideal for IoT applications. With blockchain, data is securely stored and managed in a distributed manner, preventing tampering and allowing for easier data sharing between multiple IoT devices. The combination of IoT and blockchain technology has the potential to revolutionize many industries [74]. IoT devices can collect data from the environment and store it on the blockchain, creating an immutable record of events. This data can be used for a variety of purposes, such as providing insights into the behavior of connected devices, and delay constrained applications [78]. Deep reinforcement learning (DRL) is a type of machine learning that enables an agent to learn through its interaction with the environment by receiving feedback in the form of rewards. DRL algorithms are widely used to solve complex problems in robotics, natural language processing, and autonomous vehicles [33]. Combined with DRL, an IoT enabled blockchain can be used to provide intelligent, automated decision making and behavior optimization [5]. DRL methods learn from the data stored in the blockchain and make decisions that are more accurate and reliable than that without DRL. This can be used to optimize the performance of IoT applications, such as reducing energy consumption or improving the accuracy of data collected [8].

IoT enabled blockchain will require the blockchain nodes to process the high volume of transactions from the IoT devices [5]. This requires scaling of blockchain algorithms for high throughput and low latency without compromising on security and decentralization properties of the blockchain [13]. One recent solution to the scaling problem is Prism blockchain [20]. Prism [20] provides performance guarantees on throughput and latency reaching physical limits of the network. Prism achieves higher performance than the existing schemes by decoupling the functionality of blockchain [21]. In this work, we propose a DRL approach for IoT enabled Prism blockchain. A recent advancement in DRL is Proximal Policy Optimization (PPO)

algorithm. PPO is a policy gradient method that optimizes the policy of an agent and has shown to achieve better performance than the other policy gradient methods in DRL, which motivates the choice of these PPO. In the previous chapter, the performance of Prism consensus protocol was improved using DRL in which PPO has shown to achieve better performance. In the previous chapter, the focus was on providing better security while preserving all the performance guarantees of Prism. In this work, we propose IoT application based intelligent blockchain system. We optimize Prism blockchain to make it suitable for IoT based on DRL algorithm rather than focusing on improving the consensus of Prism blockchain. In Prism smart contracts, Prism blockchain is integrated with Etheruem Virtual Machines (EVM) to remove the consensus bottleneck. This approach seems very useful for smart contracts. In our work, we consider the Prism consensus protocol without Ethereum or smart contracts because in this work, we try to optimize the Prism Blockchain parameters based on the dynamics of the IoT network using DRL approach. The optimization of blockchain parameters is not possible with an additional smart contract layer on the top. Therefore, we select Prism in [21] as our underlying blockchain consensus protocol. First integrate Prism blockchain with IoT and then optimize the performance by leveraging PPO. This is the first work demonstrating the use of Prism blockchain in IoT and also the first work using PPO based approach in an IoT enabled blockchain system to the best of our knowledge. We show that the throughput of IoT enabled blockchain can be improved by using the proposed DRL approach. Our approach can achieve upto 1.5 times higher system rewards compared to IoT integrated Prism without DRL. The performance of the system is observed for varying delay constraints and percentage of adversary in the network.

## 5.1 System Model

The longest chain protocol in Nakamoto consensus is a secure, reliable, and ensures the safety of the network, even in the presence of malicious nodes. Prism is built based on the longest chain protocol by decoupling its functionality. The leader block in Prism is selected based on the number of votes that each block in the blockchain has received. The voting procedure is designed in such a way that it allows the honest nodes to select the longest chain, even in the presence of malicious nodes. The leader block is then added to the main chain, and all the other blocks are orphaned. The nodes that have voted for the leader block are rewarded. The voting procedure also includes a verification process to ensure that the votes are from honest nodes. The main components of the Prism protocol are a proposer block, voter blocks, and transaction blocks. The proposer block contains a set of transactions that are to be included in the ledger. The voter blocks are generated by voters and vote on the validity of the proposed transactions and parent voter blocks. A transaction block contains the list of transactions from the transaction pool. The role of each of the blocks is explained below. Level

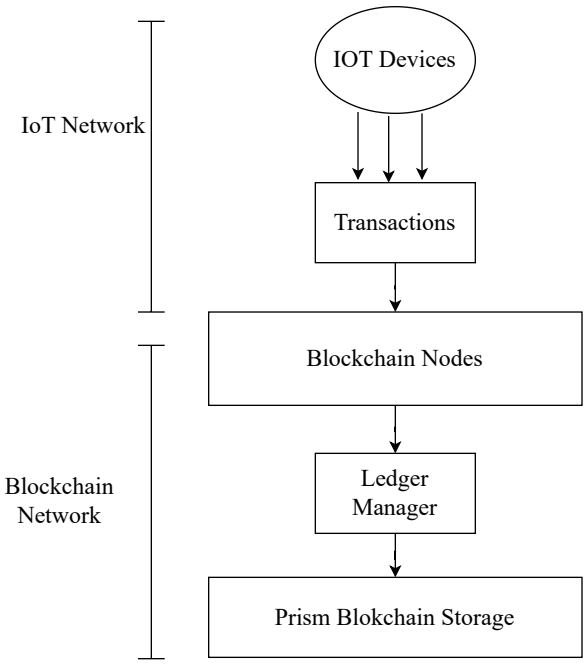



Figure 5.1: An illustration of IoT network integrated with Prism Blockchain Environment.

1 is the transaction block. Prism uses separate transaction blocks to decouple security from throughput. Transaction blocks determine how many confirmed transactions Prism can handle each second. Level 2 is the proposer block which is responsible for reliability of Prism. The elected leader blocks are the proposer blocks and based on the longest-chain protocol. Proposer blocks have a single reference to a parent proposer block as well as a list of reference links to transaction blocks (which contain transactions). Voter trees determine the leader sequence, which is the last confirmed sequence of proposer blocks. Level 3 is the voter blocks which are responsible for the security of Prism. Te parent of the voter block in the voter chains is the tip of the longest chain protocol. The content of a voter block is a list of references to proposer blocks called votes. A miner generates a block with a random sortition procedure that uses a hash function to determine the role of each block. The miner then decides if the block should be a transaction or proposer or voter. If it is a proposer block, it will contain a set of transactions references that can be added to the ledger. If it is a voter block, it will vote on the validity of the proposer transactions. The miner is also responsible for propagating the block to the rest of the network. The proposer block is verified by the voter blocks that have been created. Each voter block will check the transactions referenced in the proposer block and vote on their validity. If the majority of the voter blocks vote in favor of the proposer block, it will be added to the ledger.

### 5.1.1 IoT Enabled Prism Consensus Protocol

An illustration of the network architecture for Prism blockchain- enabled IoT is depicted in Fig. 1, which consists of an IoT network and a blockchain network. Let the number of IoT devices be $O$ and the number of blockchain nodes be $N$. In the $o$th ($o = 1, 2, ..., O$) nodes, we assume that it deploys one wireless access point (AP), which is equipped with a blockchain server node. The blockchain nodes enable computation and storage of computing tasks and can upload data, record transactions, and share information. In addition, we consider that there are $N_o$ IoT devices that can connect the nearest AP to transmit delay-tolerant data randomly, and each IoT device is also equipped a capacity to execute lightweight computing tasks. The delay tolerant data used here is the status data of IoT devices. Typically, the blockchain node is connected to all the APs through a wired link. The transactions from IoT devices are processed by the Prism blockchain nodes. At the end of each blockchain epoch, a new block is added to the blockchain storage by the ledger manager as shown in Fig 5.1.

In Prism, the transaction blocks decouple the security from throughput increases network security while enabling higher transaction throughput. The transaction blocks are independent and without a parent block. These blocks are mined at a faster rate, and consist of an ordered list of transactions from the IoT devices, drawn from a memory pool. The number of transactions confirmed per sec depends on transaction blocks confirmed. After receiving a new transaction block over the network, the miner removes the transactions in the new block from its own memory pool. An illustration of system model is demostrated in Fig 5.2. The IoT transactions are fed to Prism consensus protocol. Prism acheives consensus in a decoupled manner using transaction blocks, proposer blocks and voter blocks. A ledger manager, also called the block structure manager runs the ledger sanitization and adds a new block. The parameters of IoT enabled Prism blockchain are optimized using DRL algorithm for the next epoch in order to maximize the throughput.

### 5.1.2 Throughput

Let $B$ be the block size measured in number of transactions and $M$ be the nimber of blocks, then the throughput of the IoT enabled Prism is at most $MB$ transactions per second (tps). However, the mining rate and the block size are constrained by the security requirement. Increasing the mining rate increases the amount of forking of the blockchain due to multiple blocks being mined by multiple miners within the network delay. Forking reduces throughput as it reduces the block confirmation rate on the ledger. Forking also reduces the security of the protocol because the adversary requires less compute power to attack. Increasing the block size also increases the amount of forking since the network delay increases with the block size [20]. To keep IoT enabled Prism secure, the mining rate and the size of the voter blocks and proposer blocks have to be chosen such that each voter chain and proposer tree has little forking. To decouple security from throughput, transactions are carried by separate transaction blocks, whose

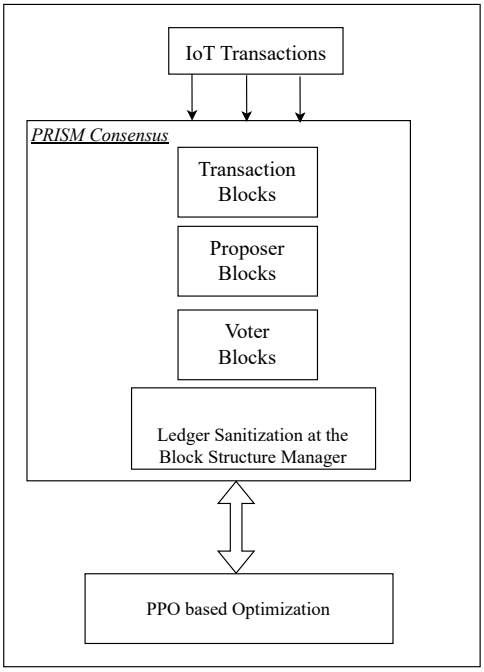



Figure 5.2: An illustration of system model consisting Prism blockchain environment

mining rate can be controlled independent of voter and proposer blocks. For an adversary with $\beta < 50\%$ hash power and network capacity $C$, Prism can achieve its near optimal throughput, $\Lambda$ as follows:

$$\Lambda \leq (1-\beta)C \tag{5.1}$$

### 5.1.3 Security

The security of the IoT enabled Prism blockchain depends on the votes from the voter trees, which also secure each leader proposer block. In Prism implementation [21], if the adversary has less than 50% hash power ($\beta \leq 0.5$), and the mining rate in each of the voter trees is kept small to minimize forking, then the reliable consensus is maintained by the honest leader proposer blocks. A low mining rate would result in high latency and a high mining rate would result in forking. Consider that there are $n$ proposer blocks at level $i$, denoted by $P = P_{b1}, P_{b2}, \cdots, P_{bn}$. Suppose $P_{bi}$ gets $v_i$ number of votes. Each voter block can vote only on one proposer block. For every vote received, $V_i^p$, let $u_{ij}$ be the depth of the voter chain. The block inter-arrivals are exponentially distributed, and the number of blocks mined after the block $P_{bi}$ is a Poisson random variable, with a rate equal to its mean [21]. This can be related to the time elapsed since $P_{bi}$ was released via the block mining rate. Let the fraction of adversarial hashing power be $\beta$, and we can empirically estimate the average depth of existing public votes as $\bar{u} = \frac{\sum_{ij} u_{ij}}{\sum_i v_i}$ and a forking rate $\alpha$. There are many voter chains so the estimates converge to their true mean.

Then, the average depth of a voter chain is,

$$\bar{u}_A = \frac{\beta \bar{u}}{(1-\alpha)(1-\beta)}. \tag{5.2}$$

### 5.1.4 Latency

Let the fraction of adversary hashing power be $\beta$ and $V$ be the number of voter chains and $p_\alpha$ be a small value representing the reversal probability of confirmed transactions. Then the upper bound on the latency, $T_{bc}$, is given as follows [21]:

$$T_{bc} \leq D_a k_1(\beta) + D_a \frac{k_2(\beta)}{v} \log \frac{1}{p_\alpha}, \tag{5.3}$$

where $k_1$ and $k_2$ are constants which depend on $\beta$ [20]. $k_1(\beta) = \frac{5400(1-\beta)}{(1-2\beta)^3 \log \frac{1-\beta}{\beta}}$ and $k_2(\beta) = \frac{5400(1-\beta)}{(1-2\beta)^3} \log \frac{50}{1-2\beta}$. Honest transactions are confirmed in $\max\left(k_1(\beta)D_a, k_2(\beta)\frac{1}{v}\log(\frac{1}{p_\alpha})\right)$ time. This implies that latency is proportional to the average propagation delay $D_a$ and is independent of the reversal probability. The latency bound considered here is the worst-case latency bound that holds for all attacks described in [21].

Latency is permissible for the delay-tolerant data but for the blockchain systems, the latency is considered as an important metric and needs to be minimized. In Prism, the transaction processing of data in has two phases, generates a block at first, and then reaches a consensus on the generated block with a proposer leader selection based on votes received. Consequently, the latency of the data processing in Prism blockchain includes latency of transaction block generation based on IoT device delay and block confirmation, which is represented as

$$T = T_{IoT} + T_{bc} \tag{5.4}$$

where $T_{bc}$ is the average time required for the blockchain to produce a new block, $T_{IoT}$ is the time consumption of IoT data delivery.

In the proposed system, it is assumed that there are N blockchain nodes. At the beginning of each epoch of the blockchain, (after consensus and adding a block to the blockchain), a node solves a Proof of Work (POW) puzzle to participate in the mining process. After solving the PoW puzzle, each miner mined block will be assigned a proposer or a transaction, or a voter. The incoming IoT transactions are accumulated in a transaction pool from which they are selected into transaction blocks. Proposer Blocks reference the transaction blocks and voter blocks vote on their parental blocks and proposer blocks for leader selection. The functionality of ledger formation is divided into ledger manager, block structure manager, and miner. The ledger manager updates the ledger based on the latest blockchain state and the miner assembles new blocks. The block structure manager maintains the clients' view of the blockchain and communicates with peers to exchange new blocks [21]. The block structure manager sanitizes the ledger and orders the transactions periodically. After the consensus, the ledger sanitization

is performed during a predefined period, which is set to 1 epoch. The goal of the Prism client is to maintain up-to-date information on the blockchain and the ledger. The number of nodes joining and leaving the network is random and the security of a blockchain protocol like Prism highly depends on the voter blocks mined at each epoch. Our proposed DRL algorithm can be executed at the block structure manager periodically and update the blockchain parameters. The block structure manager follows the same peer-to-peer protocol for updating the DRL parameters as that of updating new blocks. Since the Prism client by default is decentralized, our implementation is also decentralized. The Prism client in our implementation maintains the latest information on the blockchain, the ledger, and the updated parameters of DRL.

## 5.2 DRL based optimization

The goal of this work is to ensure an intelligent, and secure IoT enabled Prism blockchain. We try to optimize the Prism blockchain parameters based on IoT data using DRL approach. In Prism [20], the functionality of blockchain is decoupled and transactions blocks are responsible for throughput. Hence, we integrate IoT data as the input transactions to Prism blockchain. In order to get the benefits of reliability and security, proposer and voter blocks need to be preserving Prism performance guarantees. In this section, we formulate the DRL problem to maximize the throughput by optimizing Prism blockchain parameters. A reinforcement learning approach, PPO is leveraged to optimize blockchain performance. At each time step, the input transaction data is analyzed to decide the blockchain configuration. The decision is made based on predefined requirements of the application level constraints of security and latency, which are explained in this section.

### 5.2.1 Markov Decision Process Model

The Markov Decision Process (MDP) environment represents the IoT transaction information from the IoT nodes to the blockchain nodes, where our DRL algorithm needs to interact along with the Prism blockchain. The DRL agent is not aware of the environment beforehand and it uses its experience of interacting and observing the rewards for various decisions in different states and then learning the optimal policy $\pi^*$ to map the states with their best actions.

#### 5.2.1.1 Environment

The MDP environment is represented by the tuple $< s, a, r, \gamma >$, where, $s$, is the state space, $a$, is the action space, $r$, represents the reward function and $\gamma$ is the discount factor ranging from 0 to 1. At every time step, $t$, the agent receives a state representation, $s$ of the environment. The agent takes an action $a$ in the environment based on a policy $\pi(a|s)$. Then, a transition to the next state, $s'$ occurs and a reward, $r_t$ is received by the agent, which is

explained in detail in the following subsections. In IoT enabled Prism, each blockchain node participating in the mining process solves a PoW puzzle, whose difficulty level is preset based on various parameters like the number of nodes participating in the network, and the volume of IoT transactions. In Prism implementation [21], a mining rate determines the overall blocks mined in the blockchain per sec, for transaction, proposer, and voter blocks. In our work, we mainly consider the parameters related to Prism and its mining since we try to focus on the optimizing of Prism with respect to IoT. A DRL method is then used to solve the formulated MDP. The initial value of all the decision variables is set to the default values of Prism [21]. After executing the IoT enabled Prism, the proposed DRL algorithm is implemented to optimize the parameters for the next blockchain epoch. The initial IoT enabled Prism state is fed to the DRL algorithm and the output of the DRL algorithm decides the parameters of the Prism blockchain to achieve higher throughput.

#### 5.2.1.2 State

Let $N$ be the number of blockchain nodes and $T_{IoT}$ is the delay associated with the IoT devices. $T_{bc}$ is the delay of the prism blockchain consensus protocol. $T_{IoT}$ is the time consumption of IoT data delivery.

$$s = \{N, T_{IoT}, T_{bc}\}^t \tag{5.5}$$

Note that this state is different from that in the MDP formulation of Chapter 4. This is because the problem in this chapter is to optimize the all blockchain parameters to have a better performance with IoT application. Whereas in Chapter 4, the main focus is on the security of Prism blockchain and the consensus algorithm itself.

#### 5.2.1.3 Action

At every time step $t$, the agent selects an action $a_t \in A$ to maximize the throughput of the system. The DRL agent selects the number of voter chains, $V'$ for each time step $t$. The mining rate of transaction blocks is kept constant to preserve the performance guarantees of Prism. The mining rate of the proposer is $M_p$ and the mining rate of voter blocks $M_v$ is chosen by the DRL agent. The mining rate is the rate at which blocks are mined. The number of voter chains in Prism implementation was 300, which is maintained at 300 in our implementation if not controlled by DRL agent. We initialize the value of $V$ as 300 and the agent can choose the number of voter chains from a range of 270 to 300, each time with a change of $\Delta V \in \{+10, 0, -10\}$. The transaction mining rate $M_t$ in Prism is set to achieve 350 blocks per sec. The value of $M_t$ is mainly responsible for Prism throughput guarantee. We initialize $M_p$ and $M_v$ at 0.1 and each time the agent can change the value of $\Delta M_p$ and $\Delta M_v$ by $\{+0.01, 0, -0.01\}$. The mining rate is a discrete set of values in the range of $[0.01, 0.1]$. The range is chosen for an acceptable level of performance. If the mining rate is higher than

0.1, the blocks are mined faster, resulting in more forks and a waste of resources. If it is less than 0.01, they are mined slower resulting in a very low throughput of the blockchain. After choosing the change in action values of $\Delta M_p$, $\Delta M_v$ and $\Delta V$, the agent sets a new value for the number of voter chains and mining rate of the blockchain as $M'_p$ and $M'_v$, and $V'$. For instance in the first iteration, if an action is chosen to decrement all the values, then $M'_p = M_p - \Delta M_p$, $M'_v = M_v - \Delta M_v$ and similarly $V' = V - \Delta V$. Then, the action selected $a_t$ can be represented as

$$a = \{\Delta M_p, \Delta M_v, \Delta V\}^t \tag{5.6}$$

#### 5.2.1.4 Reward

The reward, $r_t$ is based on the state, $s_t$ and the action, $a_t$ at the current time step, $t$. The DRL algorithm maximizes the discounted future reward, $r_t$. Based on the current state and action, the agent obtains a reward from the environment. The throughput of the Prism blockchain depends on the voter blocks, mining rates and the number of IoT transactions successfully added to the blockchain. Let $\Lambda$ be the throughput of the entire system which is defined as the number of blocks confirmed per sec, and $K$ be a constant calculated based on the network parameters of the blockchain [21]. Let $\Lambda_v$ and $\Lambda_p$ be the voter and proposer throughput. The reward function represents the number of blocks.

If the security constraint in Eq. (5.2) and the latency constraint in Eq. (5.3) are satisfied then, the reward received by the agent is defined as follows.

**5.2.1.4.1 Voter block reward:** The voter block reward $r_v$ is calculated based on the parameters of the voter block like the number of voter chains $V'$ and the voter mining rate $M'_v$.

$$r_v = V' M'_v T_{bc} \tag{5.7}$$

Otherwise, if Eq. (5.2) and Eq. (5.3) are not satisfied then, the reward received by the agent is zero.

$$r_v = 0 \tag{5.8}$$

**5.2.1.4.2 Proposer block reward:** The proposer block reward $r_p$ is based on the proposer mining rate $M'_p$.

$$r_p = M'_p T_{bc} \tag{5.9}$$

Otherwise, if Eq. (5.2) and Eq. (5.3) are not satisfied then, the reward received by the agent is zero.

$$r_p = 0 \tag{5.10}$$

**5.2.1.4.3 IoT based Transaction block reward:** This reward $r_t$ is based on the parameters of transaction block like transaction block mining rate $M_t$ and the blockchain and IoT delay $T_{IoT} + T_{bc}$.

$$r_t = M_t(T_{IoT} + T_{bc}) \tag{5.11}$$

Otherwise, reward received by the agent is zero.

$$r_t = 0 \tag{5.12}$$

Total reward is a weighted sum of the rewards.

$$r = x_1 r_v + x_2 r_p + x_3 r_t \tag{5.13}$$

$r$ is the average number of blocks at a given time. In addition, $x_1$, $x_2$, and $x_3$ are the weights that help the differentiated system rewards. All three coefficients should add up to 1 and their values are defined in the experimental setup.

### 5.2.1.5 Agent

The agent uses the available state samples and calculates the estimated reward that can be obtained by taking an action $a$ from a state $s$. Agent builds a precise estimation of the action-value function $Q^*(s, a)$. Due to the high dimensionality of the problem and the complexity in calculating the Q-function, we implement deep neural networks as function approximators in the implementation. The PPO agent is a policy gradient approach and uses neural networks as a function approximator. The agent tries to maximize the future rewards, which are expressed as the sum of all discounted future returns following time step $t$. The discounted future reward with the discount factor, $\gamma$, is represented as

$$R_t = r_t + \sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau} \tag{5.14}$$

where $\gamma \in [0, 1]$. The behavior of the agent is described by a policy $\pi(a_t|s_t)$ which is the probability of choosing action $a_t \in A$ condition on the observed state $s_t \in S$. The action-value function under policy $\pi$ describes the expected return of action for the agent in a given state, $Q_\pi(s, a) = \mathbb{E}_\pi[R_t|s_t, a_t]$, where $\mathbb{E}_\pi$ denotes the expected value given that the agent behaves according to policy $\pi$. The optimal policy $\pi^*$ has the optimal action-value function $Q^*(s, a) = \max_\pi Q_\pi(s, a)$, and satisfies the Bellman optimality equation. Bellman equation states that the optimal action-value function is the expected reward of action $a$ plus the discounted expected value of the best action in the subsequent state $s_{t+1}$,

$$Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a_{t+1})|s_t, a_t] \tag{5.15}$$

The optimal action-value function $Q^*(s, a)$ is the maximum expected return achievable under any policy, after seeing a state $s_t$ and taking an action $a_t$ and $\pi$ is a policy mapping states to actions [77]. The objective is to maximize the Q function while satisfying the latency and security constraints. The objective function and constraint are as follows:

$$\max_\pi Q_\pi(s, a), \tag{5.16}$$

$$\text{Security constraint} \quad \beta < 0.5, \tag{5.17}$$

$$\text{Latency constraint} \quad T_{bc} < D_a k_1(\beta) + \frac{D_a k_2(\beta)}{m} \log \frac{1}{p_\alpha} \tag{5.18}$$

where $\beta$ is the adversary power $k_1, k_2$ are dependent constants as defined in Eq. (5.3) and $p_\alpha$ is a small value of the reversal probability of confirmed transactions. $D_a$ is the average delay, which is the average propagation delay. The initial state fed to the DRL agent is from Prism and all the parameters and control variables are set to the default values of Prism [21]. In this case, the constraints are satisfied and hence the initial rewards received by the agent are non-zero. During the initial state $s_0$ the value of $M_p$ and $M_v$ is 0.1 and $V$ is 300. At time $t$, the state $s_t$ is the number of nodes, the number of voter chains, and the delay corresponding to each block. The action $a_t$ is to increment or decrement the values of the mining rates and the number of voter chains. At $t + 1$, the state $s_t + 1$ depends on the number of voter chains in the previous time $t$ and the mining rates at $t$. At the end of each DRL algorithm iteration, the new state is updated based on the delay corresponding to the Prism consensus and the delay associated with IoT transactions. The new state $s_{t+1}$ is formed by the output of the blockchain, based on the action taken in the previous time step. Once the action is decided by the agent, the parameters are given to the Prism blockchain environment. The reward is calculated with a goal to maximize the throughput of the entire system.

### 5.2.2 Proximal Policy Optimization based Approach

The Proximal Policy Optimization (PPO) algorithm is a state-of-the-art DRL algorithm based on the actor-critic method [53] . In PPO, the actor-network and the critic network are two deep neural networks that make up the main network. The critic network gauges performance, while the actor network is used to learn the policy. The actor can learn the gradient of the policy by using the critic-estimate value. The actor network and the critic network are both part of the target network. The target network can be viewed as an old version of the primary network, which is used to generate the target value for training critic. It includes a target actor-network and a target critic network. The next state from replay memory is the target network's input, and the output is a critic value for training critic. The present state, the chosen action, the reward, and the subsequent state are all stored as experience tuples in the replay memory. For training the main network and the target network, random samples from the stored experience tuples are taken. Fig. 5.3 shows the PPO based algorithm used in our approach. The current state $s_t$ is given as input to the actor-network and the distribution

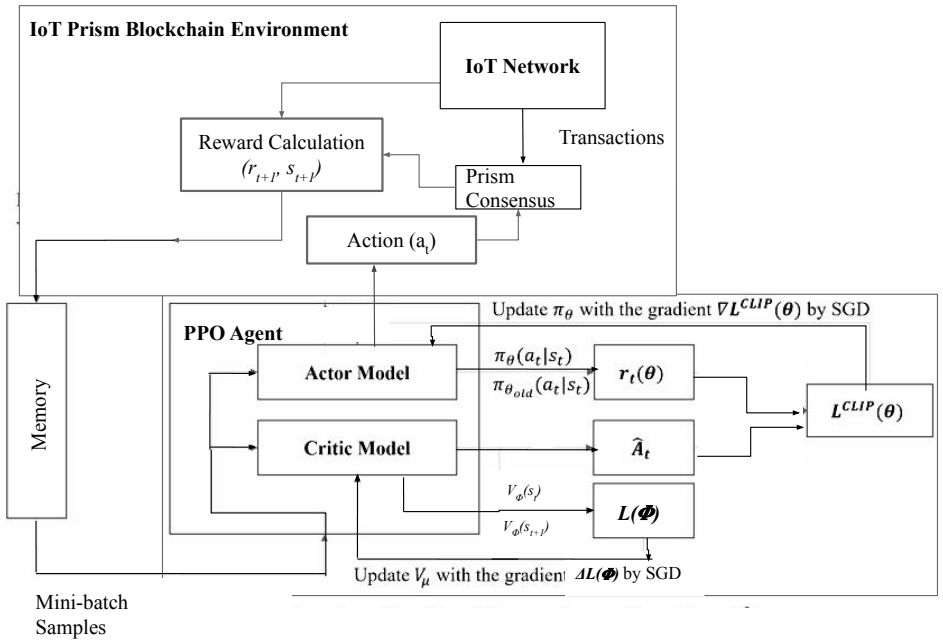



Figure 5.3: An illustration of Proximal Policy Optimization based Approach using IoT Prism Blockchain

of actions is obtained. The agent gets its action by sampling from the action distribution. The environment gives the agent a reward for the action taken indicating whether the action is beneficial and this reward is used by the critic network. The output of the critic network is a component of the actor's loss function. The actor-network generates the policy and the critic network evaluates the current policy by estimating the advantage function $\hat{A}_t$. The policy is modified according to the advantage function, given as

$$\hat{A}_t^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{5.19}$$

If the advantage of the optimal action in $s$ is equal to zero then the expected return in $s$ is the same as the expected return when being in state $s$ and taking an action $a$. This is because the optimal policy will always choose $a$ in $s$. The advantage of all other actions is negative which means they bring less reward than the optimal action, so they are less advantageous.

The clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\delta, 1+\delta)\hat{A}_t)], \tag{5.20}$$

where $\rho(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\rho(\theta_{old}) = 1$ is the probability ratio, $\theta$ is the policy parameter, and $\delta$ is a hyper parameter. The probability ratio between the action under the current policy and the action under previous policy is represented as $\rho(\theta)$. $\pi_\theta$ is a stochastic policy. $\theta_{old}$ is the vector of policy parameters before update. The second term, $\text{clip}(r_t(\theta), 1-\delta, 1+\delta)\hat{A}_t$ modifies the objective by clipping the probability ratio, which removes the incentive for moving $\rho_t$ outside of

**Algorithm 5** Proximal Policy Optimization based Approach using IoT Prism Blockchain

Initialize IoT Prism parameters to default values
Initialize agent parameters $\theta_0$, $\phi_0$ and clipping threshold $\delta$
**for** Episodes e=1, 2, ..., E **do**
    Observe the state $s_t$
    Follow a policy $\pi_{\theta_{old}}$ for $\tau$ time steps
    Collect $\mathbb{D} = \tau_i$ set of trajectories
    Compute the rewards $r$
    Compute $\hat{A}_t$ in Eq. (5.9) using $V_\phi$
    **for** epoch t:=1, ..., $\tau$ **do**
        **if** IoT enabled Prism constraints hold **then**
            Choose $a_t$ for Prism based on actor policy $\theta$
        **End if**
        Compute rewards $r$
        Update $s_t$ to next state $s_{t+1}$
        **for** Every $K$ steps **do**
            Update the policy using stochastic gradient ascent

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathbb{D}_i|T} \sum_{\mathbb{D}} \sum_{t} \min(\rho_t(\theta)\hat{A}_t,$$

$$\text{clip}(r_t(\theta), 1-\delta, 1+\delta)\hat{A}_t)$$

            Update value function using stochastic gradient descent by mean squared error

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathbb{D}_k|T} \sum_{\tau \in \mathbb{D}} \sum_{t=0}^{T} (V_\phi(s_t) - R_t)^2$$

        **End for**
        $\theta_k \leftarrow \theta_{k+1}$
    **End for**
**End for**

| Parameters | Value |
|---|---|
| Memory pool maximum size | 500,000 transactions |
| Transaction mining rate | 350 blocks/sec |
| Proposer block mining rate | 0.1 blocks/sec |
| Voter block mining rate | 0.1 blocks/sec |
| Average Transaction size | 20Bytes |
| Maximum block size | 2 MB |
| Maximum Block Interval | 100 ms |
| Maximum voter chains | 1000 |
| Maximum votes | 1000 |
| Data transmission rate | 100 MBPS |
| Batch size | 16 |

Table 5.1: Simulation Parameters

the interval $[1-\delta, 1+\delta]$. Finally, we take the minimum of the clipped and unclipped objective, so the final objective is a lower bound on the unclipped objective. Without a constraint, maximization of $L$ would lead to an excessively large policy update and hence, it is modified to penalize changes to the policy that move $\rho_\theta$ away from 1. The main objective is the following:

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 H[\pi_\theta|s_t]]. \tag{5.21}$$

The advantage estimator $\hat{A}_t$ depends on the state-value function $V^\pi(s_t)$, which is estimated with a second network, the critic network. As parameters are shared between the state-value function estimator and the policy surrogate $\pi(a_t|s_t;\theta)$, a value-function error term $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$ is added to the objective function (scaled by a positive constant parameter $c_1$, where $V_t^{targ}$ denotes the state-value function of training target. The entropy of the policy $\pi$ for state $s_t$ , $H[\pi_\theta|s_t]$, scaled by a constant positive parameter $c_2$, is added to the objective to encourage exploration. The training process of the overall workflow is shown in Algorithm 5. The policy network is a feed-forward network with two hidden layers. We sample from a single environment and use the default trajectory length of 2048.

## 5.3 Results and Discussion

### 5.3.1 Experimental Setup

The environmental setup consists of 1000 IoT nodes sending transactions to the Prism blockchain. Prism blockchain nodes are initialized with 300 voters and later this value is selected by the DRL algorithm randomly for every epoch. The proposed scheme leveraging PPO is implemented with PyTorch. At the beginning of the experiment, the default value of Mining rate of proposer, voter and transaction blocks denoted by $M_p$, $M_v$ and $M_t$ respectively is 0.1. The parameters used in the simulation are summarized in Table 5.1. In [21], the Prism protocol

is run on the AWS servers for 10 minutes on which a detailed analysis is performed. Similar to that we run the Prism protocol integrated with IoT data for 10 minutes and then implement our DRL algorithm. The performance is analyzed in terms of the average reward, which tries to maximize the number of blocks confirmed per second in the blockchain under the restrictions of the security and latency constraints. In each episode, the voters are redistributed and the consensus is carried out. The block structure manager runs the proposed DRL approach and optimizes the Prism blockchain parameters. We ran our IoT integrated Prism Blockchain on 5 nodes and collected the data for 10 minutes. The data required for DRL algorithm is available to all the nodes in the network and does not require additional communication between the node. This data is given to the block structure manager, which runs the DRL algorithm and updates the blockchain parameters. The proposed PPO algorithm is implemented using Pytorch. For PPO algorithm, the default trajectory length is 2048. $x_1$, $x_2$, and $x_3$ are set to 0.3, 0.3 and 0.4 respectively. The weighting factor of the generalized advantage estimator is set to 0.95. The clipping rate is set to 0.2, the value-function error and entropy terms are scaled by $c_1 = 0.5$ and $c_2 = 0.01$, respectively. The entropy coefficient is set to $10^{-3}$ with a decay of 0.99.

#### 5.3.1.1 Reward Convergence

In the training process of Algorithm 1, for the PPO based approach, we set the hyperparameter values according to the parameters reported in Table 5.1. These hyperparameters are set based on experiments and observations to maximize performance. The reward is observed considering DDQN with a deep neural network of four layers. The reward values shown here are smoothed over a window of 20 episodes. Fig. 5.4 demonstrates that the proposed PPO approach converges empirically. The agent explores with a probability of $\epsilon$, which is initially set to 1 to ensure random action selection. The value of $\epsilon$ is decayed per episode.The $\epsilon$ value is decayed upto $\epsilon = 0.01$. After running the experiment for 1000 episodes, the agent explores for 100 episodes and the epsilon decay value is set at $10^{-6}$. The PPO algorithm converges with an average reward of 573 for a batch size of 16. In Fig. 5.5, the proposed algorithm performance is observed for different learning rates. The proposed algorithm converges for all the learning rates, we choose the learning rate of $10^{-4}$ for our performance comparison.

#### 5.3.1.2 Performance Comparison

We compare our proposed approach to IoT enabled prism approach. In addition, we also consider our DRL based IoT Prism with fixed voter chains for comparison. In this case, the value of $V$ is set to the default value and is not changed during the experiment. We consider a fixed voter chains approach to highlight the importance of optimizing all the system parameters for a better performance. By fixing the value of $V$, we are trying to analyze the IoT enabled system for its effectiveness.

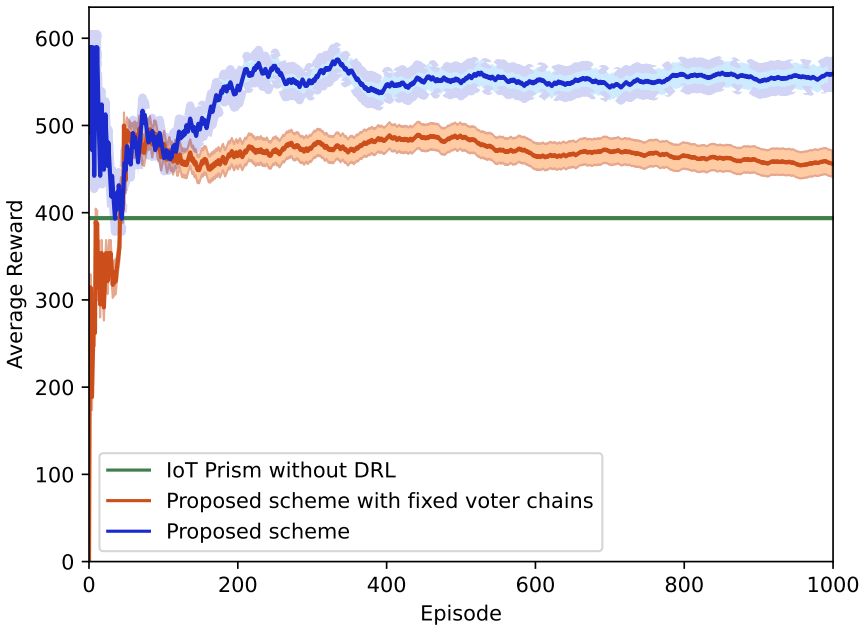



Figure 5.4: Average Reward vs Episodes for the proposed scheme, proposed scheme with fixed voter chains and IoT Prism without DRL is plotted with standard deviation. The proposed scheme achieves higher average rewards compared to other two for all the episodes.

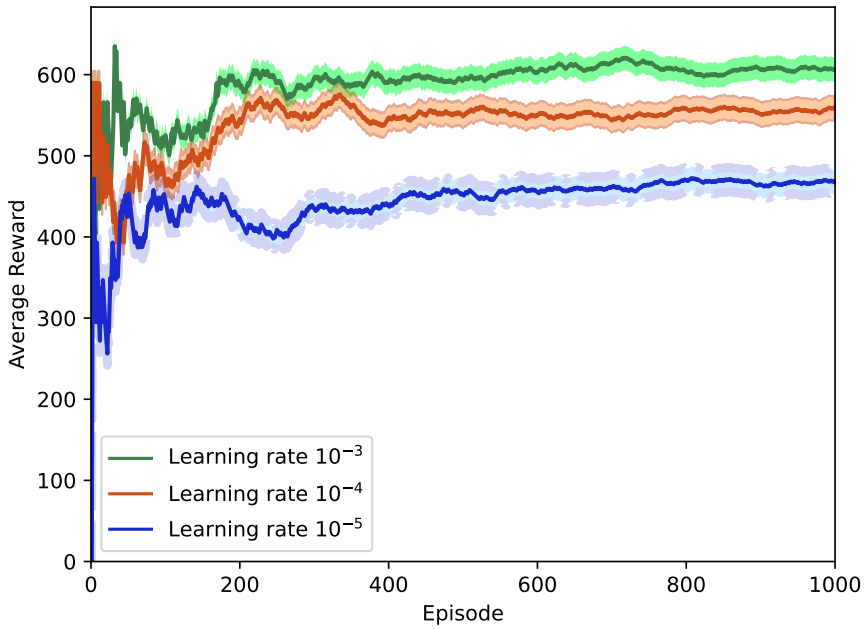



Figure 5.5: Average Reward vs Episodes for the proposed scheme is shown when the learning rate is varied for the proposed scheme.

We compare the performance when PPO algorithm is executed at 5 different blockchain nodes. At all the nodes, the proposed scheme has higher average rewards than the proposed scheme with fixed voter chains and IoT Prism without DRL. This shows that any node can perform the ledger manger duties and the proposed scheme will have higher rewards.

In Fig. 5.4, we note that the DRL algorithms were able to find the policy that addresses and maximizes the long-term reward function. The DRL maximize the reward while maintaining the minimum security level and not limiting the transaction confirmation time onto the blockchain. This implies that when the latency is low, the rewards are higher and as the delay increases, the reward received decreases. The proposed scheme could achieve upto 1.5 times higher system rewards in all the simulations.

In Fig 5.7, the performance of the proposed scheme is observed for varying values of average transaction block size. The transaction block size is related to the amount of transactions referenced from the transaction pool. In this work, the input transactions to Prism blockchain are from the IoT network. With higher transaction block size, higher number of transactions are processed per second resulting in a higher throughput. While higher transaction block size can give higher throughput, it has a risk of increasing the delay in the consensus. Therefore, we evaluate our scheme with block size limit varying from 500 KB to 2 MB. The proposed scheme outperforms proposed scheme with fixed voter chains and IoT Prism without DRL.

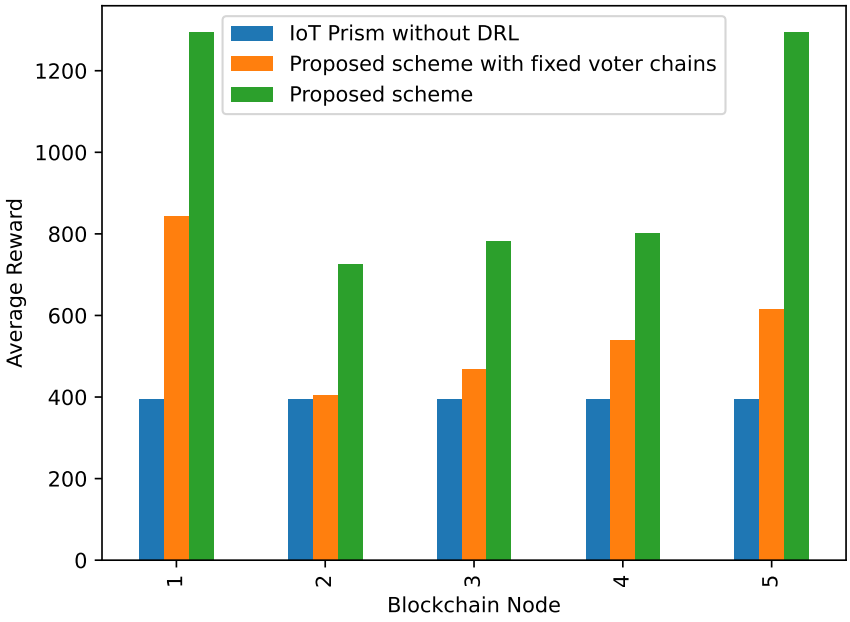



Figure 5.6: The maximum value of average reward for all the episodes is shown at each node. The proposed scheme, proposed scheme with fixed voter chains and IoT Prism without DRL are compared and it is observed that the proposed scheme achieves higher average rewards.

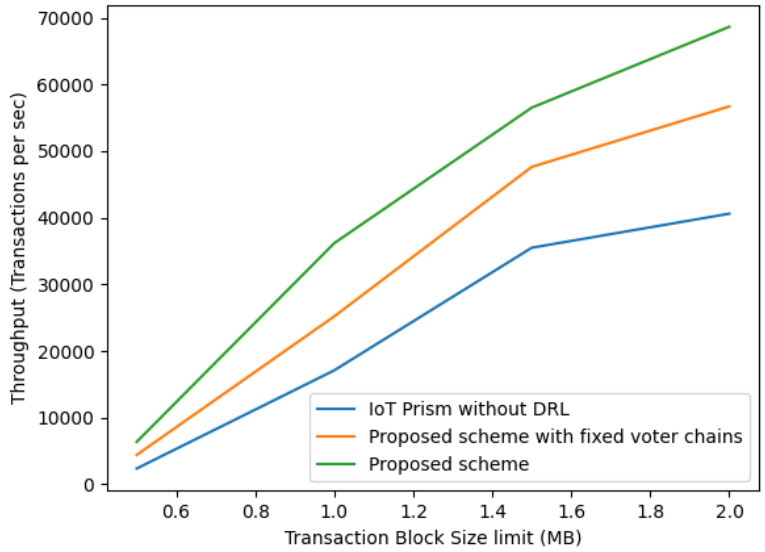



Figure 5.7: Performance comparison of the proposed scheme, proposed scheme with fixed voter chains and IoT Prism without DRL in terms of throughput (transactions per sec) with varying transaction block size limit. The proposed scheme has better throughput than others.

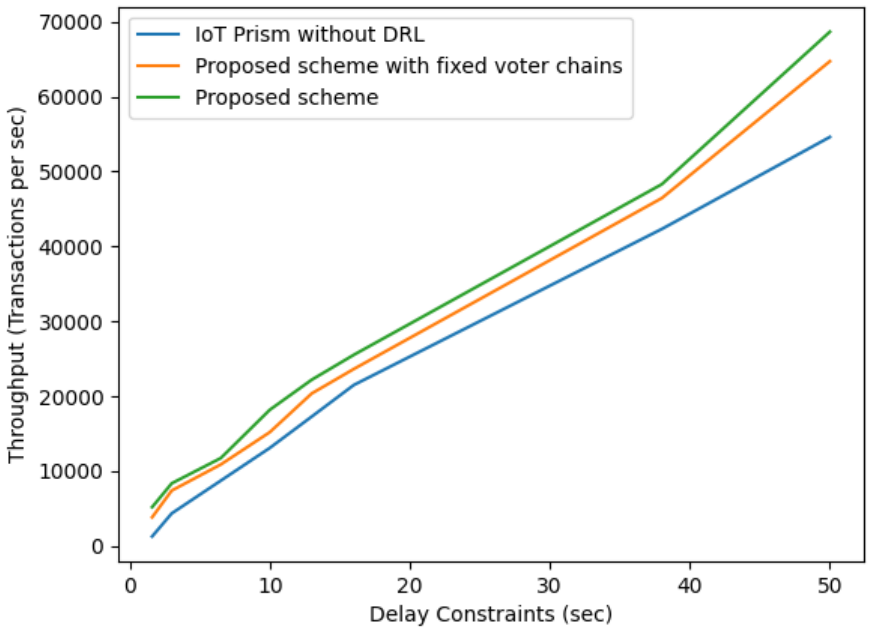



Figure 5.8: Performance comparison of the proposed scheme, proposed scheme with fixed voter chains and IoT Prism without DRL in terms of throughput (transactions per sec) with varying delay constraints is shown. The proposed scheme has better throughput than others for all delay constraint values.

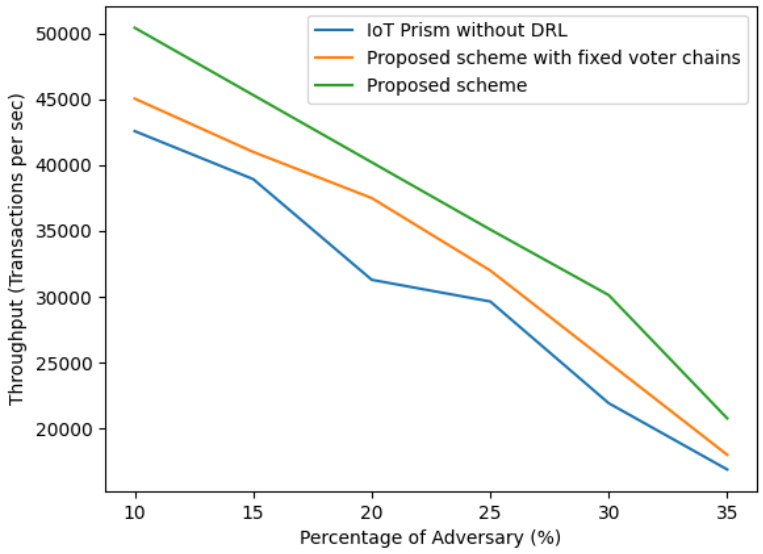



Figure 5.9: Performance comparison of the proposed scheme, proposed scheme with fixed voter chains, and IoT Prism without DRL in terms of throughput (transactions per sec) with varying percentage of adversary is shown. The proposed scheme has better throughput than others.

In Fig. 5.8, the throughput of the proposed scheme, proposed scheme with fixed voter chains and IoT Prism without DRL are plotted for different delay values. The delay here is the total delay $T$ which is a combination of the IoT delay and the blockchain confirmation delay. With a higher delay, more blocks are mined for the same number of participating nodes. The higher the value of delay constraint, the faster the blocks are mined, leading to a higher value of the throughput. With varying delay constraints, the proposed scheme on an average could achieve 6,000 tps higher throughput. Proposed scheme has much less convergence time compared to proposed scheme with fixed voter chains and IoT Prism without DRL in all the simulations. PPO is also able to reach higher rewards in all the cases compared to IoT enabled Prism. The results in Fig 5.9 show the performance of IoT enabled Prism Blockchain without application of DRL, proposed scheme with fixed voter chains, and proposed PPO approaches. The outputs are taken as an average across all the nodes. The DRL algorithm is executed at the ledger manager and all the nodes are given the outputs from DRL algorithm. The throughput of the entire IoT enabled Prism blockchain including the Transaction blocks, Proposer blocks, and Voter blocks is measured in number of transactions confirmed per sec. We vary the percentage of malicious nodes in the entire network and observe the performance. The security of Prism blockchain has a constraint on the percentage of malicious nodes to be less than 50%. We observe that the throughput decreases with an increase in the malicious nodes percentage in the network. However, the proposed scheme can achieve higher throughput compared to IoT

prism without DRL. The proposed approaches achieves the better overall performance than without DRL based IoT Prism.

# Chapter *6*

# MARL approach for coordinated UAV action control

Unmanned aerial vehicles (UAVs) are widely used for missions in dynamic environments. Deep Reinforcement Learning (DRL) can find effective strategies for multiple different agents that need to cooperate to complete the task. In this work, the challenge of controlling the movement of a fleet of UAVs is addressed by Multi-Agent Deep Reinforcement Learning (MARL). The collaborative movement of the UAV fleet can be controlled centrally and also in a decentralized fashion, which is studied in this work. We consider a dynamic military environment with a fleet of UAVs, whose task is to destroy the enemy targets while avoiding obstacles like mines. The UAVs inherently come with a limited battery capacity directing our research to focus on the minimum task completion time. We propose a continuous-time based Proximal Policy Optimization (PPO) algorithm for multi-aGent Learning In Dynamic Environments (GLIDE). In GLIDE, the UAVs coordinate among themselves and communicate with the central base to choose the best possible action. The action control in GLIDE can be controlled in centralized and decentralized way based on which two algorithms called Centralized-GLIDE (C-GLIDE), and Decentralized-GLIDE (D-GLIDE) are proposed. We developed a simulator called UAV SIM, in which the mines are placed at randomly generated 2D locations unknown to the UAVs at the beginning of each episode. The performance of both the proposed schemes is evaluated through extensive simulations. Both C-GLIDE and D-GLIDE converge and have comparable performance in target destruction rate for the same number of targets and mines. We observe that D-GLIDE is up to 68% faster in task completion time compared to C-GLIDE and could keep more UAVs alive at the end of the task.

Unmanned aerial vehicles (UAVs), are employed extensively in missions involving navigating through unknown environments, such as wildfire monitoring [22], target tracking [23], and search and rescue [24], as they can host a variety of sensors to measure the environment with relatively low operating costs and high flexibility. Most research on UAVs depends on the target model's accuracy or prior knowledge of the environment [25]. However, this is extremely difficult to achieve in most realistic implementations because environmental information is typically limited. Deep Reinforcement Learning (DRL) has made incredible advances in recent years in many well-known sequential decision-making tasks, including playing the game of Go [79],

playing real-time strategy games [80], controlling robots [34], and autonomous driving [81], especially in conjunction with the development of deep neural networks (DNNs) for function approximation [82]. A Deep Reinforcement Learning (DRL) agent autonomously learns an optimal policy to maximize its rewards through its interaction with the environment. The flight environment for a UAV is usually local or completely unknown during online path planning. Hence, the agent has to react to the dynamic environment using incomplete information, which is the key challenge in UAV action control [26, 27]. In [28], DRL was used with independence on environment model, and prior knowledge, solving the online path planning problem by trial and error interactions with its environment. Model-free RL methods [29] and the Q learning methods [30] have gained recent popularity. We focus on one of the most important applications of UAV, which is target tracking and obstacle avoidance.

The application of UAVs in military or civil fields such as reconnaissance, strike, rescue, and early warning [22] involves path planning in dynamic environments and is challenging as the UAV has to avoid obstacles that are not known to it beforehand [31]. A single agent in such an environment might lack the battery capacity to accomplish the tasks efficiently. Hence multiple UAVs are employed to coordinate and complete the task. Interestingly, the majority of successful applications in DRL literature involve the participation of multiple agents, requiring the use of multi-agent reinforcement learning (MARL). MARL specifically addresses the sequential decision-making problem of multiple autonomous agents operating in a shared environment, which together seek to maximize their own long-term return by interacting with the environment as well as other agents [32]. MARL algorithms can be classified into different categories based on the types of situations they handle as fully cooperative, fully competitive, and a mix of the two. This allows a variety of new solutions that build on concepts such as cooperation or competition [32]. However, multi-agent settings also introduce challenges, including inadequate communication, difficulties in reward assignment to agents, and environment non-stationarity.

We consider a military environment application for multiple UAVs, where each UAV can act independently of the other agents' actions. The environment contains a set of targets that the UAVs need to coordinate and destroy. There are several mines in the field that are capable of destroying the UAVs. These mines are placed at any random location which is unknown to the UAVs. We assume that the mines have a sensing radius in which they can detect the UAV and destroy it. In this scenario, the UAV must learn to detect and avoid mines within a minimum distance from the sensing radius of the mine. The goal here for each UAV is to destroy the enemy target, by avoiding the mines and reaching the base safely. The challenge for a UAV is that it has no prior knowledge of the location of the mines and has to find a suitable policy to adapt to the dynamic environment. We propose a multi-aGent Learning In Dynamic Environments (GLIDE), which uses MARL to control the action of a set of UAVs. In GLIDE, the MARL based approach aims to find optimal strategies for agents in settings where multiple agents interact in the same environment. In GLIDE, we adopt Proximal Policy Optimization

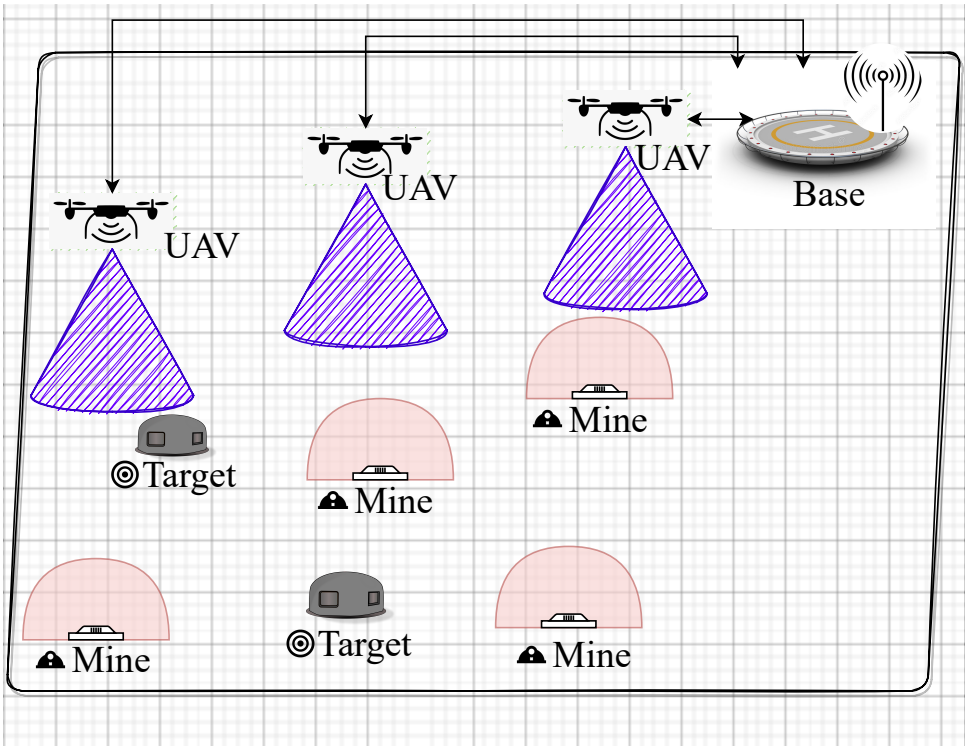



Figure 6.1: An illustration of the System Model

(PPO) to train our agents, which is a well-known state of art continuous control algorithm. The aim of the UAV fleet is to explore the entire field of operation as quickly as possible so that the task can be completed with a minimum time. We compare the performance of centralized action control based GLIDE, called C-GLIDE with decentralized action control based GLIDE, called D-GLIDE. The results show that the UAVs can accomplish minimum task completion time with D-GLIDE.

The main contributions of this work are as follows:

1. The coordinated UAV action control problem is formulated as a Markov Decision Process (MDP) with the action space involving accelerations of all the UAVs. Previous works use fixed length strides for the movement of UAV, due to which the UAVs moves to a fixed distance at every time step. In GLIDE, at each time step, the UAVs can choose to change their acceleration. This gives freedom for the UAVs to move to any amount of distance intended per time step. We observed that this approach also used lower parameters for training the model since we just have 3 continuous values in $x$, $y$, and $z$ directions. The state space takes into account the global situational information and also the local situation faced by each UAV during the time of operation.

2. We propose two MARL algorithms based on PPO for coordinated UAV action control namely centralized GLIDE (C-GLIDE) and Decentralized GLIDE (D-GLIDE) with a

Table 6.1: A table showing the comparison of GLIDE system model and other works in literature

| Paper | Single or Multiple UAVs | Obstacles and Mines | Assumptions | Environment | Task |
|---|---|---|---|---|---|
| [83] | Multiple | None | All the UAVs are connected to a cellular network | Ground | Finding the best path |
| [62] | Multiple | Preset obstacle areas in grid map | UAVs follow the assigned path | Ground patrol area | Target searching and tracking |
| [84] | Multiple | None | All the UAVs are connected to a cellular network | Ground based dynamic users | QoE driven UAVs assisted communications |
| **This work, GLIDE** | Multiple | Mines | UAVs periodically communicate with the base | Military environment created with our simulator, UAV SIM | Target destruction |

continuous action space. In C-GLIDE, the action control is performed based on the combined state space information available at the base. This keeps all the UAVs updated with global and local information. However, this slightly hampers the task completion time. Whereas D-GLIDE is based on centralized training and decentralized execution. The UAVs have access to their local data and once in a few time-steps get updated with global information. This resulted in faster task completion time.

3. We build a simulator for our experimentation called UAV SIM. Our experimentation results show that both these algorithms converge and have a comparable target destruction rate and mine discovery rate. With low number of targets and mines, both C-GLIDE and D-GLIDE perform equally. C-GLIDE is useful for lower number of targets, mines and UAVs. As the number of targets and mines increases to the maximum limit, D-GLIDE completes the task with up to 68% less time compared to C-GLIDE. We also observe that the target destruction rate of D-GLIDE is upto 21% higher and upto 42% more UAVs are alive with the same number of mines in the field than C-GLIDE.

A summary of related works comparison with respect to the system model is presented in Table 6.1. Also, please refer to Table 2.1 for the comparison of the proposed algorithm with related works. In this work, we introduce a novel formulation of the UAV action control problem. Following that, we propose GLIDE, which leverages continuous-time PPO-based approach for

the action control of a group of UAVs in a dynamic environment. A centralized action control algorithm C-GLIDE is compared with a decentralized action control algorithm D-GLIDE. We implement the simulation environment called UAV SIM for our experimentation.

## 6.1 System Description

This section presents the system model for the multi-UAV action control problem. Each UAV scans a large area divided into multiple grids. The grid coordinates are used to simplify the positioning of the UAVs and a set of $K$ targets. A group of UAVs is assigned to accomplish the task in this area. Consider a set of $N$ UAVs, that intend to complete the task within a time $T$. The time of operation, $T$ consists of several time slots $(1, 2, \cdots, t)$, each of length $\delta$. All the UAVs start from the base, and return to the base upon task completion. We assume that all the UAVs have sufficient battery capacity for task completion. We consider a military environment where the UAVs need to navigate and destroy the target. The targets are immobile and are placed in the area of operation. A single UAV would have limited battery and cannot span the entire area especially when field of operation is large. Whereas a group of UAVs, can coordinate and accomplish the task of destroying the targets more efficiently. The action control becomes challenging due to the presence of mines, which have a destruction range within which they can destroy the UAVs. Hence, the UAVs need to detect and avoid the mines with a distance greater than the destruction range of the mine. All the UAVs communicate with the base and get regular updates about the mines. A UAV upon detecting a mine communicates the location of the mine to the base. The base then updates all the UAVs with the coordinates of the mines. All the UAVs have knowledge of all the target positions at the beginning of each experiment. We consider a square grid world of size $m \times m \in N^2$ with each cell of size $c$ and the set of all possible positions of UAV, $m$. We assume the continuous movement of UAVs in the environment and incorporate the map-based state space. Next, the details of the UAV simulator are given in the subsection.

## 6.2 UAV Simulator

A set of $N$ UAVs are considered to move in the environment which is modelled as a grid of dimensions $m \times m$ and of height $f$. Each UAV's current position in the grid is represented by $p_i^t = [x_i^t, {y_i}^t, z_i^t]$ with altitude ranging from between 0 and $f$. The operational status $b_i^t \in \{0, 1\}$, indicate whether the UAV is inactive or active. The environment contains of a bse which is the start and end position for all the UAVs. Each UAV has a battery as its fuel and a radio antenna for communication. The UAV can communicate to the base, and internally with other UAVs. There is a base set up, from where the UAVs are deployed and they are supposed to complete their task and come back to the base, from where the UAVs are monitored. In addition to

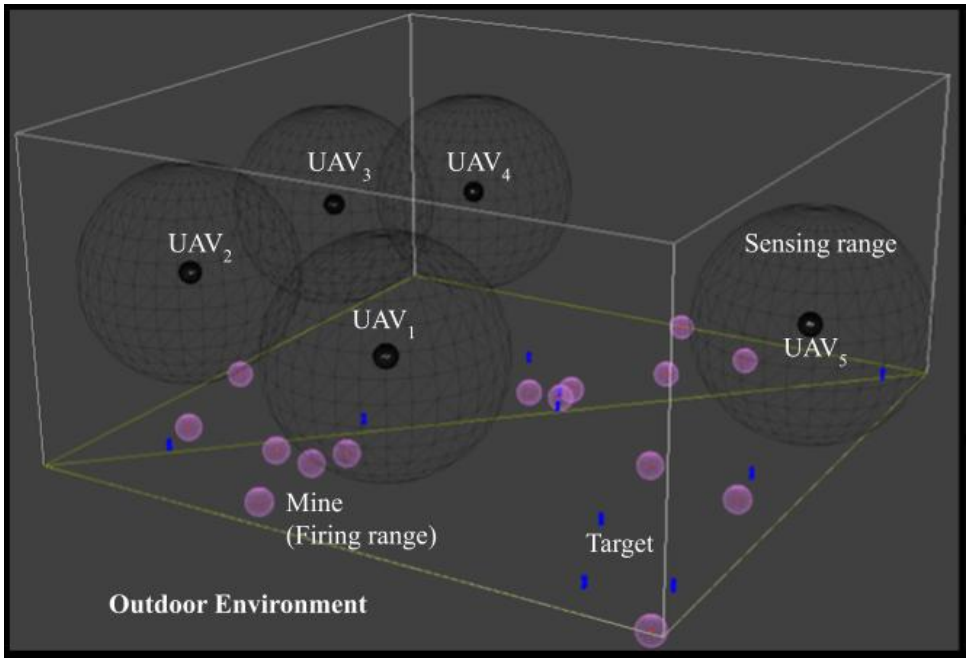



Figure 6.2: An illustration of the outdoor environment with n UAVs and Minies and targets placed on the ground.

this, each UAV has sensing capability with a sensing range. When the UAVs are functioning, we consider the following parameters: the battery consumption of the UAV, the current UAV location which is given in coordinates, the target coordinates that need to be destroyed, and the discovered mines on the field. Other factors that affect UAV performance are distance traveled and directional movement which are also taken into account. The proposed algorithm uses this information to determine the UAV's plan of action. We assume the following: (i) Each UAV will move to the position that has been planned for the epoch. (ii) Each UAV can monitor their battery levels and calculate how long they can fly with the remaining battery. (iii) Each UAV communicates information on its own location, as well as the location of the mine if it is discovered. In C-GLIDE, the UAVs communicate continuously with the base. In D-GLIDE, the UAVs communicate whenever a mine is discovered otherwise, the communication is once in every $K$ time steps. The grid clearly defines the surveillance area. The simulator code will be made available on github upon acceptance. [1]

[1] https://github.itap.purdue.edu/Clan-labs/GLIDE.git

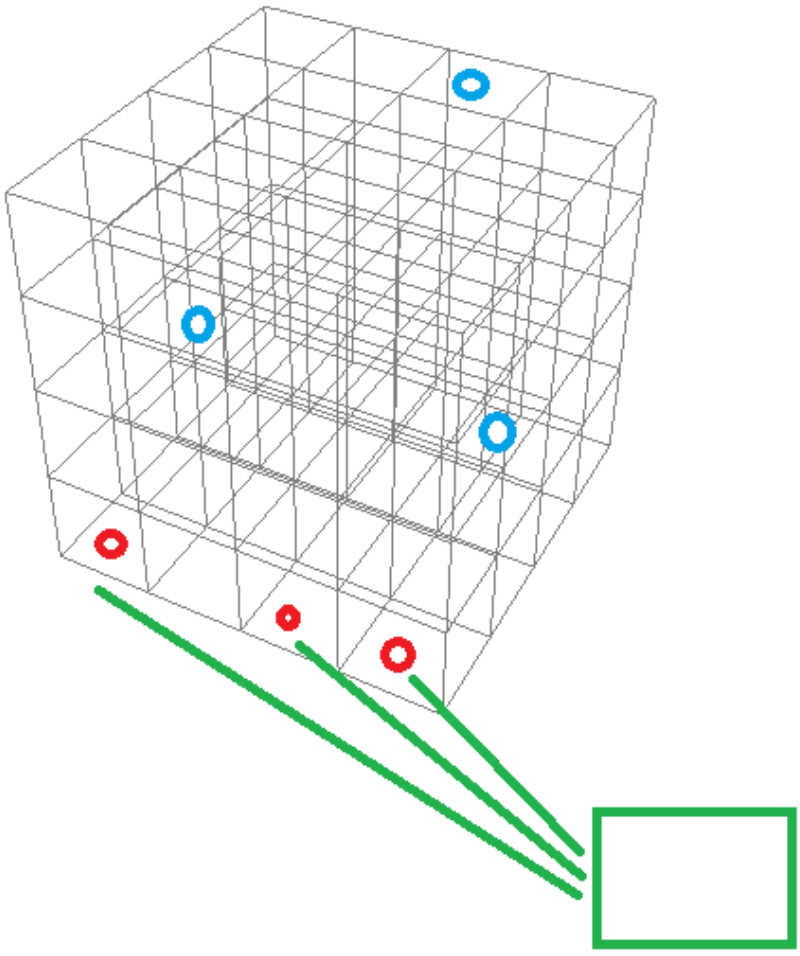


Figure 6.3: Coordinate-based system for the field of operation

## 6.3 GLIDE: MARL Approach

### 6.3.1 Markov Decision Process Model

The objective of this work is to ensure the task completion with a group of UAVs in an adaptable and intelligent way, while taking into account various factors like mine avoidance, target destruction, and distance traveled by each UAV. At each time step, the input data is analyzed to understand the dynamics of the environment. The action control is performed based on the requirement of minimum distance traveled and obstacle avoidance for task completion. The optimization problem is formulated as an Markov Decision Process (MDP). The MDP environment is represented by the tuple $< S, A, R, \lambda >$, where, $S$ is the state space, $A$ is the action space, $R$ represents the reward function, and $\lambda$ is the discount factor ranging from 0 to 1. At every time step $t$, the agent receives a state representation, $s$, of the environment. The agent takes an action $a$ in the environment based on a policy $\pi(a|s)$. Then, a transition to the next state $s'$ occurs with a probability $P(s'|s, a)$, and a reward, $r_t$ is received by the agent, which is explained in detail in the following subsections.

#### 6.3.1.1 State

The state reflects the environment at a given time, in this case, the dynamic military environment information at each time step. The state space, $S$, consists of the target coordinates, mine coordinates, and various attributes of the UAV. We consider a set of $N$ UAVs communicating with a base. Since the mines are capable of destroying the UAV, we consider a binary vector $B^N$ with $N$ elements with each entry indicating the live status of the UAV i.e. 1 denoting that the UAV is alive and 0 denoting that the UAV is destroyed. The target coordinates are represented in a two-dimensional grid, called the target map denoted by $T^{m\times m}$. If the target is present in the field and is active, then the grid coordinates corresponding to the target locations are set to +1. If the target is down and destroyed by one of the UAVs then, those coordinates are represented by -1. Every other situation is represented by 0 in the target map. Similarly, the mine map indicates whether the mine is discovered or destroyed. If the mine is detected and is active, then its corresponding coordinates in the mine map are set to +1. If the mine is destroyed, it is marked by -1 and every other situation is represented by 0. Note that, we only keep the non-zeros entries of target map and mine map to reduce the size of the state space. The position of each UAV is a three-dimensional vector. Since, we have $N$ UAVs, all the UAV positions are considered in the state space and is represented by $P^{N\times 3}$. The three-directional velocity of each UAV, is also considered in the state space, denoted by $V^{N\times 3}$. The combined state information is used to choose the actions of each UAV.

$$S_t = \big[\{x : x \neq 0 \wedge x \in T^{m\times m}\}, \{y : y \neq 0 \wedge y \in M^{m\times m}\}, P^{N\times 3}, V^{N\times 3}, B^N\big] \tag{6.1}$$

The base has the entire information of the environment, that is, the target locations, detected mine location and state information for each UAV position, velocity and liveliness. The base communicates the state with the UAVs and receives the UAV specific information from each UAV during the communication. In C-GLIDE, the UAVs are in continuous communication with the base whereas in D-GLIDE, the UAVs communicate whenever a mine is discovered or once in every $K$ time steps.

#### 6.3.1.2 Action

The objective of an agent is to map the state space, $S$ to the action space, $A$. A set of $N$ deployed UAVs move within the limits of the grid of the environment. At every time step $t$, an agent $i \in N$, selects an action $a_t^i \in A$ for each UAV. The action is the decision of movement towards the next target. The position of each UAV, $i$ is described using $p_t^i = [x_{i(t)}, y_{i(t)}, z_{i(t)}]^T \in R^3$ with altitude at ground level or any value up to an altitude $f$. The operational status of a UAV represents its action which is either taking an action to move or staying in the current position (no movement). The UAV can choose to move towards the next target or stay in the current position. A set of $N$ deployed UAVs move within the limits of the grid of the environment. The action space is continuous acceleration in 3 directions for each UAV. The initial speed considered for each UAV is 0 m/s. With every time step, the UAV changes its speed, moves a certain distance and calculates the velocity for next time step. The maximum acceleration for each UAV is 50 m/s$^2$.

The action set is a choice of acceleration in each direction, i.e, x, y, and z. For instance, with the current UAV position, a particular acceleration can be chosen as $(a_x, a_y, a_z)$ as (-5, 0, 5) then the UAV would move in the direction of the resultant of the three chosen vectors covering a distance of $5\sqrt{2}$. Here, the UAV might face an obstacle or reach the end of the grid in y direction, hence its acceleration in y direction is 0. This helps us bound the movement of UAV within the grid. Using this action space helps the UAV to not only move in linearly but also take a curvature motion. Note that $N$ UAVs are present so we predict $N$ resultant acceleration vectors at each time step.

All the UAVs start at the base with an initial speed 0 and depending on the current acceleration, direction of movement towards the target and time, choose their next acceleration. In C-GLIDE, every agent has a different value of acceleration based on a centralized architecture whereas in D-GLIDE, every agent chooses a different value of acceleration in a decentralized architecture. These methods are explained in detail in the next section.

#### 6.3.1.3 Reward function

The reward function $r_t$, is based on the state $s_t$ and the action $a_t$ at the current time step $t$. The DRL algorithm maximizes the discounted future reward, $R_t$. Based on the current state and action, the agent obtains a reward from the environment. We consider the reward function

a combination of (a) proximity based reward which takes into account the distance traveled by each UAV, (b) mine detection and avoidance, (c) target destruction, and (d) liveliness reward. Each reward is normalized between 0 and 1 and added in the total reward.

**6.3.1.3.1 Proximity based reward** Assuming that the UAV scans at a constant distance, we can define a proximity-based reward that will motivate the UAV to move closer to the target and maintain a safe distance from the mines. First we will discuss avoiding mines and then moving closer to the target. If the UAV moves too close to the mine, it will get destroyed within the destruction range of the mine. If the mine is destroyed by the UAV, the UAV will get zero rewards since it does not have to avoid it anymore. Consider a binary variable indicating the liveliness of the mine as $l_m$ which is set if the mine is alive. If the mine is alive, we calculate the reward based on the minimum distance, $d_{min}$ and least distance, $d_l$. The value of $d_l$ is the destruction range of the mine which needs to be avoided by the UAV to stay safe. The minimum distance $d_{min}$ in this case is calculated as the minimum distance between a particular UAV and a detected mine. As the UAV moves closer to this range, it is negatively rewarded. Without the proximity based reward for mine, the UAV will not be motivated to maintain distance from the mine. Let the mine proximity reward be represented as $r_{mp}$ for each UAV and is is clipped between 0 and 1.

$$r_{mp} = \begin{cases} -(clip(1/d_{min}, 0, 1/d_l) * d_l), & \text{if } l_m = 1 \\ 0, & \text{if } l_m = 0 \end{cases} \tag{6.2}$$

The reward $r_{mp}$ is calculated at each UAV and is communicated to the base. Since we have $N$ UAVs communicating their reward to the base. At the base, the mine reward is normalized and expressed as,

$$R_{mp} = \frac{\sum r_{mp}}{N} \tag{6.3}$$

Similarly, if a target is destroyed, the agent receives a positive reward, otherwise, it receives a reward based on minimum distance $d_{min}$ and the least distance $d_l$. The minimum distance $d_{min}$ in this case is calculated as the minimum distance between a particular UAV and a target. The value of $d_l$ is the firing range of a UAV which indicates the least distance between the UAV and a target so that the UAV can destroy the locked target. The more closer the UAV gets to the target, the more it is positively rewarded. A binary variable indicating the liveliness of the target, $l_t$ is set if the target is alive. Let the target proximity reward for each UAV be represented as $r_{tp}$ and clipped between 0 and 1.

$$r_{tp} = (clip(1/d_{min}, 0, 1/d_l) * d_l) - 1, \quad \text{if } l_t = 1. \tag{6.4}$$

All the UAVs communicate the value of $r_{tp}$ to the base. Since we have $N$ UAVs, at the base, the target reward is normalized and expressed as,

$$R_{tp} = \frac{\sum r_{tp}}{N} \tag{6.5}$$

**6.3.1.3.2 Target destruction reward** The target destruction reward, $r_{td}$ is received by an agent when a UAV successfully destroys a target. By doing so, the agent has completed a part of the assigned task and gains a positive reward. Let $z_d$ be the reward received when a UAV destroys one target from a total of $Z$ targets. In our implementation, the value of $z_d$ is set to 1. At each UAV, $r_{td}$ is calculated as follows:

$$r_{td} = \begin{cases} z_d, & \text{if } l_t = 1 \\ 0, & \text{if } l_t = 0 \end{cases} \tag{6.6}$$

At the base, the target rewards communicated by each UAVs are summed to check if all the targets are destroyed. Since we have $Z$ targets, the target destruction reward at the base is expressed as,

$$R_{td} = \frac{\sum r_{td}}{Z} \tag{6.7}$$

**6.3.1.3.3 Mine detection reward** A UAV detects a mine within its sensing radius and communicates the location of the mine to the base in addition to avoiding it. The base can then update the rest of the UAVs in the next time step about the location of the detected mine. A mine detection is denoted by $m_d$ and is given to each UAV when it detects a mine $m$ from a total set of $M$ total mines. In our implementation, the value of $m_d$ is set to 1.

$$r_{md} = \begin{cases} m_d, & \text{if } m \in M \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

The number of mines present in the field is known but the locations of those mines is unknown. Consider there are $M$ mines in the field, then the mine detection reward calculated at the base can be expressed as follows,

$$R_{md} = \frac{\sum r_{md}}{M} \tag{6.9}$$

**6.3.1.3.4 Time based reward** The time-based reward $R_\tau$ is calculated at the base to ensure that all the UAVs are negatively rewarded with each passing time step. Our aim here is to finish the task as quickly as possible so that we can save the battery consumption of the UAVs. The total time for task completion for all the UAVs is $T$ and with every time step $\tau$, the combined UAVs' time based reward is penalized.

$$R_\tau = \begin{cases} -1, & \text{if } 0 < \tau < T \\ 0, & \text{otherwise} \end{cases} \tag{6.10}$$

A time based reward motivates the fleet of UAVs to finish the task in minimum time, this does not essentially mean that the task aims at keeping the maximum number of UAVs alive. Hence, we introduce a liveliness reward.

**6.3.1.3.5 Liveliness reward** We try to maintain as many UAVs alive as possible upon task completion. If the UAV is destroyed by the mine, it is rewarded 0. If it destroys the targets and returns to the base, it is alive and is rewarded positively. The reward $R_l$ is calculated at the base as the ratio of number of UAVs alive, $u_a$ by maximum number of live UAVs possible, $N$. This is given as follows,

$$R_l = \begin{cases} \frac{u_a}{N} - 1 \end{cases} \tag{6.11}$$

If all the UAVs are alive, then the value of $R_l$ is 0. If none of the UAVs are alive then $R_l$ is -1.

**6.3.1.3.6 Total Reward** The total reward is the reward given to each UAV but can be calculated at the base as and when the UAvs communicate to the base. We add all the above normalized rewards of proximity based, target destruction, mine detection, time based and liveliness reward. Then we train the DRL agent at the base based on the total reward and is communicated to each UAV.

$$R = R_{tp} + R_{mp} + R_{td} + R_{md} + R_{\tau} + R_l \tag{6.12}$$

The frequency of communication with the base is algorithm specific in GLIDE, and is discused in detail in the next section. The DRL agent is unaware of the environment beforehand and needs to interact with the dynamic environment. It uses its experience of interacting and observing the rewards for various decisions in different states and then learn the optimal policy $\pi^*$ to map the states with their best actions. In a military environment, all UAVs have participate in the task completion process. The number of UAVs is chosen based on the number of targets, and area of the field. If a UAV is destroyed by the mine during the episode, it is penalized by the liveliness reward. The subsequent action space would not include the non alive UAVs.

## 6.4 DRL-Based UAV Action Control

The main objective of the DRL agent is to find the best mapping between the state and the action that achieves a maximum discounted average reward. In this work, we leverage PPO algorithm, which is a policy gradient algorithm. Although conventional policy-gradient algorithms, such as advantage actor-critic (A2C) [37], have successfully produced strong control effects in a variety of decision-making situations, they continue to encounter a number of issues, such as the challenging choice of iteration step size and poor data usage. Hence, PPO algorithm which is a state-of-the-art DRL algorithm based on the actor-critic method is chosen for our approach.

The information structure of MARL, or who knows what during training and execution, is more complex than it is in the single-agent situation. In this work, the agents do not have explicit information exchange with each other. Instead, each agent makes decisions based on its local observations, and information communicated from the base to UAV. The local observations

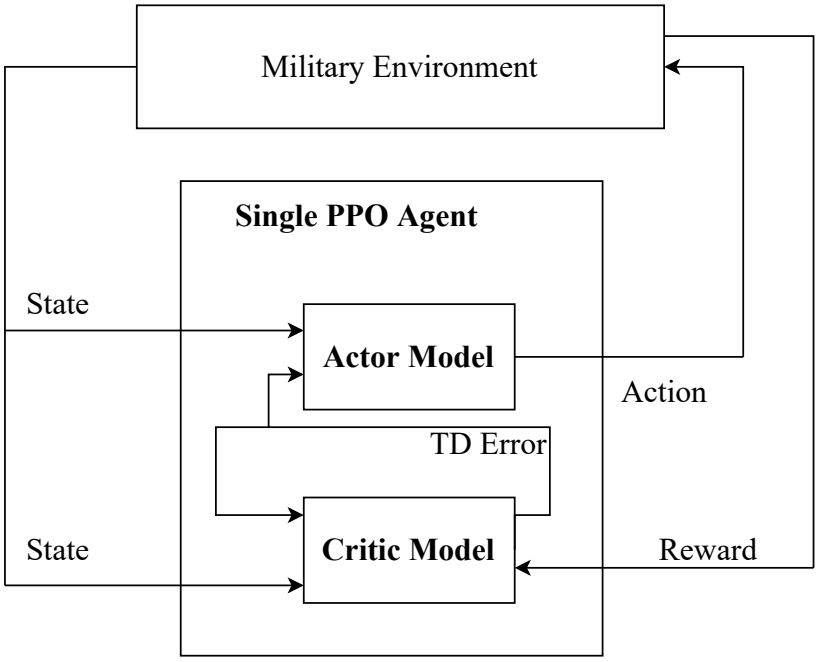



Figure 6.4: An illustration of each interaction with the environment

vary across agents and may contain some global information such as the joint actions of other agents, because of the control sharing information structure at the base. We propose two PPO-based multi-agent algorithms called continuous centralized GLIDE (C-GLIDE) and continuous decentralized GLIDE (D-GLIDE).

The current state $s_i^t$ of a UAV $i$ is given as input to the actor-network and the distribution of actions is obtained. Each agent chooses its action $a_i^t$. The environment gives each agent a reward $r_i^t$ for the action taken indicating whether the action is beneficial and this reward is used by the critic network. The actor-network generates the policy and the critic network evaluates the current policy by estimating the advantage function $\hat{A}_t^\pi$.

$$A_t^\pi = Q_\pi(s_t, a_t) - V_\pi(s_t) \tag{6.13}$$

The policy is modified according to the advantage function. Figure 6.4 demonstrates the agent environment interaction of single agent of the PPO algorithm with an actor network and a critic network. During training, a batch of samples is selected from the buffer to update network parameters. In order to improve the sampling efficiency, PPO adopts the importance sampling method. the probability ratio between old policy $\pi_{\theta_{old}}$ and new policy $\pi_\theta$ is denoted by $r_t(\theta)$ and is expressed as,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{6.14}$$

PPO has to restrict the policy's updating window and uses the clip method to directly limit the update range to $[1-\epsilon, 1+\epsilon]$. In PPO, the clip function is used to limit the update to the

**Algorithm 6** C-GLIDE

Initial actor network parameters $\theta_0$, critic network parameters $\phi_0$ and clipping threshold $\epsilon$
**for** episodes $1, 2, \cdots, M$ **do**
  **for** actor $1, 2, \cdots, N$ **do**
    Follow central actor policy $\pi_{\theta_{old}}$ for $T$ timesteps
    Collect $\mathcal{D}_k$ set of trajectories
    **for** epoch $t := 1, \cdots, T$ **do**
      Base updates all UAVs with state $s_{t-1}$
      **if** No obstacle in detection range **then**
        Take action $a_t^i$ based on $\pi_{\theta_{old}}$
      **else if** Mine detected **then**
        Choose $a_t^i$ to avoid the mine
        Update base with the mine location
      Update $s_t^i$ to next state $s_{t+1}^i$
      Compute the rewards
      Compute advantages $\hat{A}_1, \cdots, \hat{A}_T$ in (6.13)
  **if** Timesteps == K **then**
    Update $\theta$ by SGD to maximize $\mathcal{L}_{clip}(\theta)$ in (6.15)

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{clip}(\theta)$$

    Update $\phi$ by SGD and minimize $\mathcal{L}(\phi)$ (6.24)

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

    $\theta_{old} \leftarrow \theta$

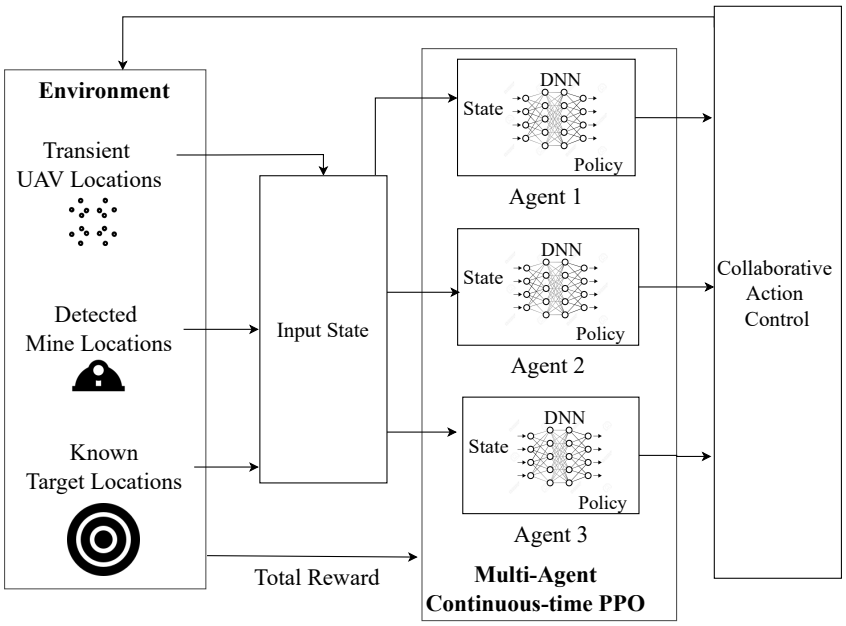



Figure 6.5: An illustration of DRL agents' interaction with the environment

policy parameters during policy optimization. To prevent large policy updates that could lead to instability in the policy distribution, the clip function constrains the probability ratio term in the surrogate objective function to a specified range. Applying clip function to the probability ratio term, the objective function ensures that policy updates stay close to the original policy while still allowing for some exploration. The loss function of PPO is:

$$\mathcal{L}_{clip}(\theta) = \mathbb{E}(\min r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{6.15}$$

where $\epsilon$ is a hyper-parameter. The objective function in Equation (6.15) only uses the policy of the actor network. For the critic network update, another term that uses the value function is needed. The centralized critic networks with weights $\phi$ is at the base. This centralized state consists of a fully-observable environment state which is common to all the agents, and also each agent's partial local observation. The agents behavior optimizes the critic loss $L_{VF}(\phi)$ based on the squared difference between the actual return and the value estimation. The critic estimation bootstraps the next state's expected value at the end of a mini-batch of size $NT$. This continues until the episode terminates.To minimize the difference between the estimated value and the actual value, the squared loss is used.

$$L_{VF}(\phi) = (V_\phi - V^{Target})^2 \tag{6.16}$$

An entropy term is added to the objective to encourage exploration. Then, the final objective function for critic network is:

$$L(\phi) = \mathcal{L}_{clip}(\theta) - c_1 L_{VF}(\phi) + c_2 S[\pi_\theta(s_t)] \tag{6.17}$$

$L(\phi)$ is calculated at the base, which is the critic network. In multi-agent DRL, each agent learns by interacting with the environment. The agents learn and optimize their behavior, by observing a state $s_t^i \in S$ and each performs an action $a_t^i \in A$ at time $t$ and receives an instantaneous total reward. The agent goes to the next state $s_{t+1}^i$. For each agent, the goal is to learn a policy that maximizes their reward. The centralized and decentralized action control of the agents is discussed in detail in the next subsection.

### 6.4.1 C-GLIDE

In a multi-agent system, each agent's reward depends on both its own actions and those of the other agents. It would be challenging to guarantee the algorithm's convergence since altering one agent's policy will have an impact on how other agents choose their optimal course of action and cause erroneous value function estimates. This research adopts a MARL approach to address the issue, as shown in Figure 6.5. In C-GLIDE, the action control of the UAV fleet is executed centrally. The concept of centralized training and distributed execution in D-GLIDE.

In the continuous centralized GLIDE approach all the UAVs' action is considered as a single agent and is controlled by the base. The state space $s$ is a combined input of all the UAV's current positions and the detected mines and the locations of undestroyed targets. The core value function is learned by the critic network. The combined agent in the actor-network simply uses its observations to determine the policy. A multi-agent algorithm introduces a scalability issue. Each agent needs to take into consideration the joint action space, whose dimension grows exponentially with the number of agents, in order to address non-stationarity. So, with increasing number of UAVs, the joint action space grows exponentially. The theoretical study of MARL is complicated by the presence of many agents, particularly the convergence analysis [32]. An approach to addressing scalability is to employ the mean-field regime with a large number of homogeneous agents [85, 86]. In mean-field reinforcement learning [86], the interactions within the population of agents are approximated by those between a single agent and the average effect from all the agents. The interplay between the two entities is mutually reinforced. Thus, the impact of each agent on the overall multi-agent system might diminish to the point where all agents are interchangeable or indistinguishable. However, a mean-field quantity, such as the average state or the empirical distribution of states, can accurately describe the interaction with other agents. This greatly simplifies the study because each agent merely needs to discover its optimal mean-field response. Our model considers each UAV shares a common reward function depending only on the local state and the mean-field, which encourages cooperation among the agents.

The mean field action distribution represents the joint probability distribution of all agents' actions in our multi-agent system.

$$\pi(a_t|s_t) = \prod_i \pi^i(a_t^i|s_t), \tag{6.18}$$

where, $\pi(a_t|s_t)$ is the mean field action distribution. $\pi^i(a_t^i|s_t)$ is the individual policy of agent $i$, which determines the probability of agent $i$ taking action $a_t^i$ in state $s_t$.

The surrogate objective function for multi-agent centralized PPO with a mean field action space is similar to the standard PPO, but it includes parameters for each agent and incorporates the mean field action distribution.

$$\mathcal{L}_{clip}(\theta) = \sum_t \sum_i [\min(r_t^i(\theta^i) * A_t^i, clip(r_t^i(\theta^i), 1-\epsilon, 1+\epsilon) * A_t^i)], \tag{6.19}$$

where, $r_t^i(\theta^i) = \pi^i(a_t^i|s_t^i, \theta^i)/\pi_{old}^i(a_t^i|s_t^i)$ is the probability ratio of the new policy for agent $i$ over its old policy. $A_t^i$ is the advantage estimate for agent $i$, which measures how much better or worse an action is compared to the average action in a given state. The value function in the PPO algorithm is estimated using a neural network function approximator. The value function approximator takes the state as input and outputs an estimate of the expected cumulative reward, also known as the state value. The centralized value function is updated by minimizing the mean squared error between the predicted value and the target value:

$$L_{VF} = \sum_t [(V_\phi - V^{target})^2]. \tag{6.20}$$

Note that the values of $V_\phi$ and $V^{target}$ are calculated for every $t$. The above expression does not have the subscript $t$ to avoid confusion. The objective function combines the surrogate objective function, the value function loss, and an entropy regularization term, weighted by their respective coefficients:

$$L(\phi) = \mathcal{L}_{clip}(\theta) - c_1 L_{VF}(\phi) + c_2 H[\pi_\theta(s_t)].. \tag{6.21}$$

$c_1$ and $c_2$ are coefficients that control the balance between the surrogate and value function losses. $H[\pi_\theta(s_t)]$ is an entropy regularization term that encourages exploration by maximizing the policy's entropy for agent $i$.

In C-GLIDE in Algorithm 6, the actor network and the critic network are initialized with weights $\theta_0$ and $\phi_0$ respectively, along with a clipping thershold $\epsilon$. The experiment is run for $M$ episodes. In C-GLIDE all the UAVs have aggregated state and aggregated action. There are N UAVs, so each UAV has an actor network and all the actor networks follow the old policy for $T$ time-steps. The actors collect $\mathcal{D}_k$ set of trajectories with the central actor policy. During each time-step, base updates the UAVs with the current state $s_{t-1}$ and the best action $a_t^i$ is taken by each UAV towards the target while avoiding the mines. If a mine is detected, its location is updated to the base. The rewards and advantage function is calculated and the state is updated to the next state. Once in every $K$ time-steps, the critic network parameters are updated based on the experience of all the actors stored in a local buffer. The detailed steps of C-GLIDE algorithm are presented in Algorithm 6.

### 6.4.2 D-GLIDE

D-GLIDE is based on centralized training and distributed execution. A core value function is learned by the critic network. It has access to data from the base, such as other agents' location information and environmental information. Each agent in the actor-network simply uses its own local observations to determine the policy. Assuming that every agent shares the same state space $S$, the algorithm allows any number of agents to be used in task execution. instead of using the agent's own value function, this method uses the Critic network of each agent to fit the global value function. Only the agent's policy has to be changed in order to optimize the global value function in this manner. The multi-agent policy gradient may also be obtained directly using the chain rule, similar to the single-agent deterministic policy gradient. In a dynamic environment containing several UAVs, for each UAV, $i$, its observation value at time $t$ is $s_t^i$, and the actor-network outputs the mean and variance of the corresponding action probability distribution according to $s_t^i$, then constructs a normal distribution, sampling which we obtain an action. Through the above methods, UAVs learn the cooperation policy between agents. During the execution phase, it exclusively depends on its own local perspective to make decisions, resulting in a collaborative policy that does not rely on communication. Additionally, UAVs with the same purpose share the same actor network settings to lower the cost of network training.

The surrogate objective function for centralized training and distributed execution is similar to the single-agent PPO, but it incorporates the centralized value function. It also includes parameters of each agent like $\theta$. The surrogate objective function is given by:

$$\mathcal{L}_{clip}(\theta) = \sum_t \sum_i [\min(r_t^i(\theta^i) * A_t^i, clip(r_t^i(\theta^i), 1-\epsilon, 1+\epsilon) * A_t^i)] \tag{6.22}$$

where, $r_t^i(\theta^i) = \pi^i(a_t^i|s_t^i, \theta^i)/\pi_{old}^i(a_t^i|s_t^i)$ is the probability ratio of the new policy for agent $i$ over its old policy. $A_t^i$ is the advantage estimate for agent $i$, which measures how much better or worse an action is compared to the average action in a given state.

The centralized value function $V_\phi$ is updated by minimizing the mean squared error between the predicted value and the target value:

$$L_{VF} = \sum_t [(V_\phi - V^{target})^2] \tag{6.23}$$

The objective function combines the surrogate objective function, the value function loss, and an entropy regularization term, weighted by their respective coefficients:

$$L(\phi) = \mathcal{L}_{clip}(\theta) - c_1 L_{VF}(\phi) + c_2 H[\pi_\theta(s_t)]. \tag{6.24}$$

$c_1$ and $c_2$ are coefficients that control the balance between the surrogate and value function losses. $H[\pi_\theta(s_t)]$ is an entropy regularization term that encourages exploration by maximizing

the policy's entropy for agent $i$. The multi-agent algorithm optimizes the policy parameters $\theta^i$ for each agent $i$ by sampling trajectories, computing advantages using the centralized value function, and performing gradient updates to maximize the surrogate objective function. The centralized value function is used to estimate the advantage values for each agent, while the decentralized execution allows each agent to execute its policy independently during interaction with the environment.

In D-GLIDE in Algorithm 7 the actor network and the critic network are initialized and the experiment is run for $M$ episodes similar to C-GLIDE. In D-GLIDE, the actor network is decentralized and there is one critic network which is centralized. Using different critic network for each actor network will be noisy. So, the training is centralized with one critic network. Each UAV has an actor network follows the old policy collecting $\mathcal{D}_k$ set of trajectories for $T$ time-steps. The policy followed by each actor here is a distributed policy unlike the centralized policy in C-GLIDE. During each time-step, base updates the UAVs with the detected mines. The best action $a_t^i$ is taken by each UAV towards the target while avoiding the mines based on the distributed policy. The rewards and advantage function is calculated and the state is updated to the next state . Once in every $K$ time-steps, the local mini-batch experience samples are used to update the critic network parameters based on the experience of all the actors. The algorithm procedure is discussed in Algorithm 7.

## 6.5 Results

In this section, the convergence analysis of the proposed C-GLIDE and D-GLIDE algorithms is presented. The D-GLIDE analysed in various scenarios with respect to the C-GLIDE. Then a summary of the analysis is provided based on the impact of several hyper-parameters on both C-GLIDE and D-GLIDE algorithms.

### 6.5.1 Simulation Setting

We consider field size of 1000 m $\times$ 1000 m within which the UAVs can fly upto a maximum altitude of 500 m. In this implentation, by default there are 4 UAVs at the base and 4 mines that are placed at unknown locations in the ground as enemy defences to protect 4 strategic targets from the attacks of UAVs. Unless otherwise stated, the aforementioned scenario is considered as the default setting. The destruction range of the UAV and the mine, and detection range of the UAV along with other environment specific parameters are mentioned in Table 6.2.

The proposed C-GLIDE and D-GLIDE algorithms are implemented on a workstation with a 48 core Intel Xeon CPU, 32 GB DDR4 primary memory and 4×Nvidia RTX 2080 Ti with 12 GB video memory for running CUDA-accelerated Tensorflow. The GPUs are interlinked by the NVlink interface and communicate with the CPU by the PCIe×16.0. We used software packages based on `Python3.8.12`, `Tensorflow v1.14.0`, `stable-baselines v2.10.2`, and

**Algorithm 7** D-GLIDE

Initial actor network parameters $\theta_0$, critic network parameters $\phi_0$ and clipping threshold $\epsilon$
**for** episodes $1, 2, \cdots, M$ **do**
  **for** actor $1, 2, \cdots, N$ **do**
    Follow an actor policy $\pi_{\theta_{old}}$ for $T$ for each agent
    Collect $\mathcal{D}_k$ set of trajectories
    **for** epoch $t := 1, \cdots, T$ **do**
      Base updates all UAVs the detected mines
      **if** No obstacle in detection range **then**
        Take action $a_t^i$ based on $\pi_{\theta_{old}}$ from state $s_t^i$
      **else if** Mine detected **then**
        Choose $a_t^i$ to avoid the mine
        Update base with the mine location
      Update $s_t^i$ to next state $s_{t+1}^i$
      Compute the rewards
      Compute advantages $\hat{A}_1, \cdots, \hat{A}_T$ in (6.13)
  **if** Timesteps == K **then**
    Update $\theta$ by SGD to maximize $\mathcal{L}_{clip}(\theta)$ in (6.15)

$$\theta_{k+1} = \text{argmax}_\theta \mathcal{L}_{clip}(\theta)$$

    Update $\phi$ by SGD and minimize $\mathcal{L}(\phi)$ (6.24)

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

    $\theta_{old} \leftarrow \theta$

Table 6.2: Environmental Parameters

| **Parameters** | **Value** |
|---|---|
| Length $l$ | 1000 m |
| Breadth $b$ | 1000 m |
| Height $f$ | 500 m |
| Height of the Target $f_T$ | 40 m |
| Radar range of the mine $D_1$ | 50 m |
| Destruction range of the mine | 50 m |
| Detection range of UAV $d_1$ | 300 m |
| Destruction range of UAV $d_2$ | 100 m |
| Maximum speed of the UAV $v_{max}$ | 90 m/s |
| Maximum acceleration of the UAV $a_{max}$ | 50 m/s$^2$ |

Table 6.3: PPO Hyper-parameters

| **Parameters** | **Value** |
|---|---|
| Neurons in hidden layer 1 | 256 |
| Neurons in hidden layer 2 | 256 |
| Relay memory size $\|\mathcal{B}\|$ | 3072 |
| Minibatch size $\|b_m\|$ | 768 |
| Learning rate $\alpha$ | $2.5 \times 10^{-4}$ |
| Discount factor $\gamma$ | 0.99 |
| GAE parameter $\lambda$ | 0.95 |
| Activation function | ReLU |
| Clip range $\epsilon$ | 0.2 |
| Optimizer | Adam |
| Epochs $\eta$ | 10 |
| Total episodes $\mathcal{E}$ | $5 \times 10^5$ |

`gym v0.19.0` to implement the C-GLIDE and D-GLIDE algorithms. The hyper-parameter values are described in Table 6.3.

### 6.5.2 Convergence Analysis

In this subsection, a comparison between the proposed D-GLIDE and C-GLIDE are presented to highlight the benefits of decentralization. In Figure 6.6 the episode reward stabilizes as the D-GLIDE agent interact with the environment for around 25 million time slots. However, that is not the case for the C-GLIDE algorithm in all scenarios. Note that the in the time based reward negatively rewards the agent to encourage lower task completion time. Other rewards are small positive values. Hence the reward values are negative in the convergence plots. However the performance of the UAV is not effected by this. We observe that both the algorithms perform well when there are fewer enemy defenses or strategic targets to destroy, as shown in Figure 6.6a and 6.6c. Because the scope of collaboration among individual UAVs is limited

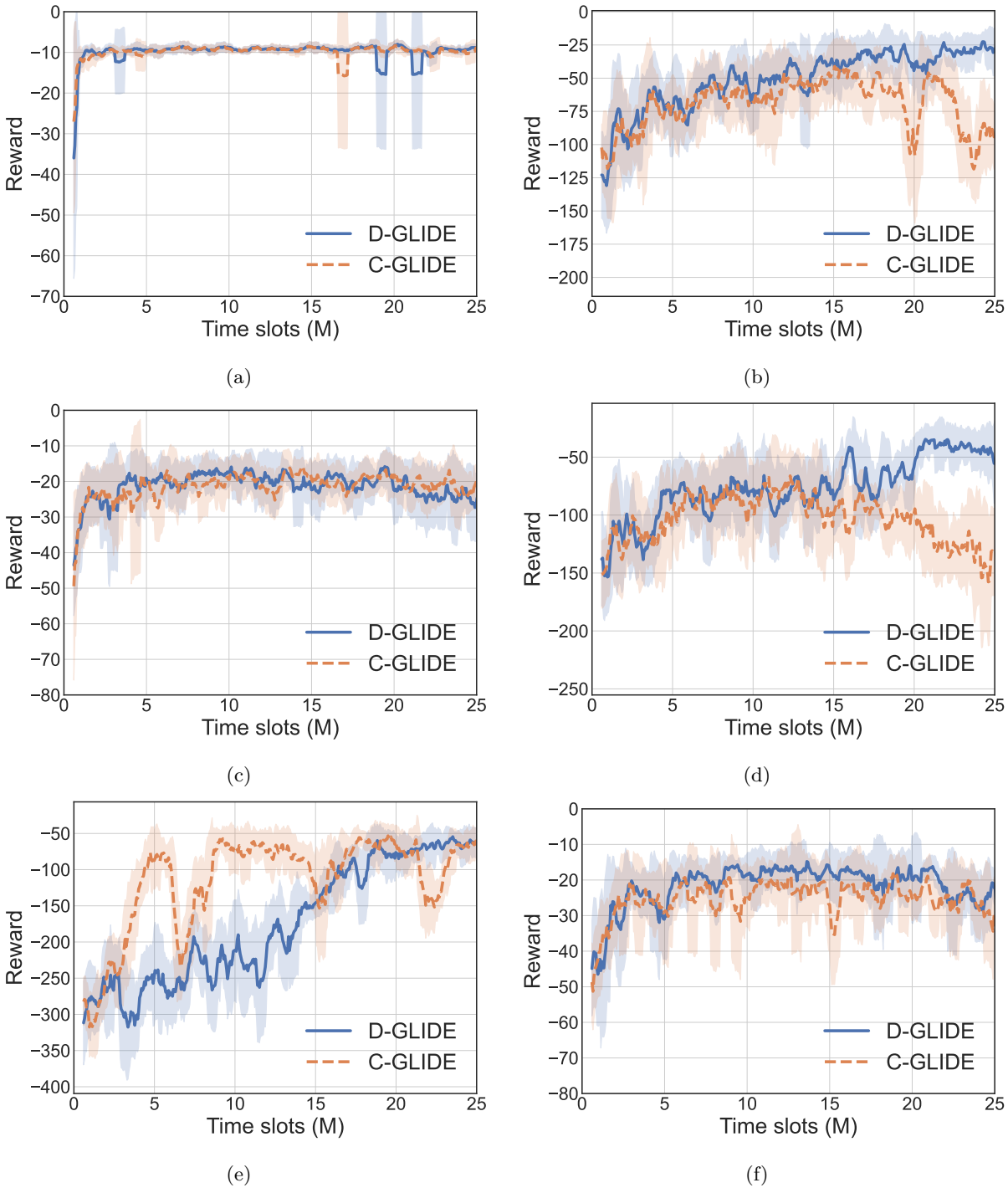



Figure 6.6: Convergence of the C-GLIDE and D-GLIDE algorithm under various settings - (a) 1 target (b) 6 target (c) 1 mine (d) 6 mine (e) 1 uav (f) 6 uav, where other objects are as per the default scenario.

and less essential to destroy all targets. Thus the UAVs do not necessarily have personalized strategies, enabling the C-GLIDE algorithm to comprehend a general policy.

However, as the number of mines or targets grows, the UAVs have to work together and divide the undertaken task among themselves to achieve the objective in minimum time, which requires personalized learning in the UAVs. Therefore, D-GLIDE learning helps in such cases as each UAV can optimize its strategy without affecting others. We show the benefits of decentralization in Figure 6.6b and 6.6d where the C-GLIDE implementation cannot facilitate personalized UAV behavior due to a common set of hidden layers and underperform as compared to the distributed implementation. Furthermore, we observe from Figure 6.6e and 6.6f that increasing the number of UAVs in the environment reduces the complexity of the resultant strategy due to redundant UAVs. Hence both the algorithms perform equivalently as we increase the number of UAVs beyond the number of targets or mines. Please note that C-GLIDE learning becomes challenging as we increase the environment's targets, mines, and UAVs.

In summary, D-GLIDE algorithm scales well with the increasing number of objects in the environment and encourages UAV-specific policy optimization for improved performance in contrast to the C-GLIDE implementation.

### 6.5.3 Performance Analysis

In this subsection, we discuss about the effectiveness of the proposed solution in various scenarios and analyse the scalibility of the system with increasing number of targets, mines, and UAVs in the simulated area of operation.

#### 6.5.3.1 Increasing Targets

Here, we analyse the effect of increasing the number of targets from 1 to 6, keeping the mines and UAVs constant as per our default scenario. From Figure 6.7a, we observe that both the distributed and C-GLIDE algorithm destroys all the targets in the environment with 4 deployed UAVs, thus algorithms are scalable with number of targets in the environment. However, significantly less number of mines are triggered by the UAVs in case the of D-GLIDEas compared to that of C-GLIDE see Figure 6.7b. The destroyed targets are integer values. The error bars indicate the difference. As a result, UAVs are less intercepted and therefore more UAVs are alive at the end of the episode as shown in Figure 6.7c.

With increasing targets, the UAVs need to collaborate effectively among themselves and learn group along with individual skills to destroy all targets in minimum time. The C-GLIDE algorithm suffers from increasing targets as it has common hidden layers to process the input state of the environment, limiting the scope of individual skill adoption by the UAVs. Whereas, in the distributed algorithm each UAV optimizes a separate neural network and thus the UAVs do not interfere in each other's learning, enabling higher degree of personalizing. We further observe a steady increase in the time taken to destroy all targets with the increase in the number

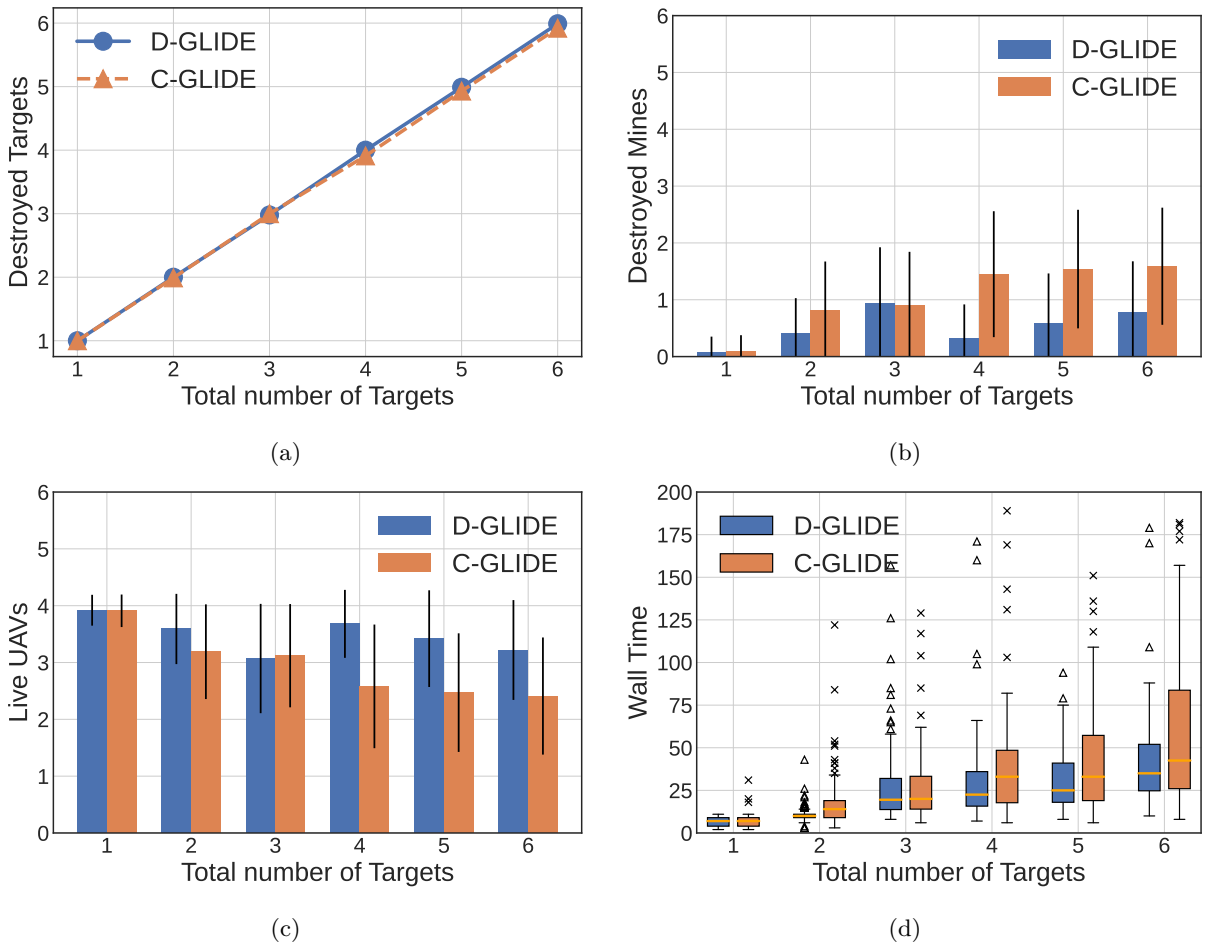



Figure 6.7: Correlation among increasing number of strategic targets vs - (a) destroyed targets (b) destroyed mines (c) live UAVs (d) Wall time or time taken at the end of each episode.

of targets as the number of UAVs is fixed. However, the distributed algorithm outperforms the C-GLIDE and take significantly less time to destroy the targets as shown in Figure 6.7d. The Figure 6.7d has outliers which are shows as markers (cross and triangle) in the plot.

In summary, number of targets influences the degree of collaboration among the UAVs that is required to destroy all the targets in minimum time and also increase the chance of UAV interception by the mines as the UAVs have to traverse more of the operational area to reach each target.

#### 6.5.3.2 Increasing Enemy Mines

As number of mines grows, probability of UAV interception increases, which results destruction of all 4 UAVs from time-to-time. In Figure 6.8a, we observe that some targets are missed as the number of mines in the environment are more than 4 deployed UAVs. However, the median destroyed targets is very close to 4 targets that shows the scalibility of both the algorithms with increasing mines. The distributed algorithm outperforms the C-GLIDE by a slight margin in destroying targets. Moreover, the C-GLIDE algorithm performs poorly in terms of avoiding mines resulting in UAVs getting destroyed during the operation as compared to the D-GLIDE. This is depicted in Figure 6.8b.

Subsequently, for the distributed algorithm, more UAVs are alive after destroying all the targets. In Figure 6.8c, we see a downward trend in live UAVs as number of mine increases. This is because the location of the mines are not known beforehand; thus the UAVs have to detect and dodge the mines dynamically during their lifetime. More mines mean there is larger probability of UAV interception, resulting less alive UAVs after destroying all targets. Moreover, from Figure 6.8d we observe a steady increase in wall time with more mines as it reduces the possible safe paths from a position and may force an UAV to take long alternate paths to reach the same destination in contrast to the scenario with no mines. The distributed algorithm takes significantly less time to destroy all the targets in the environment.

In short, increasing the mines, increases the hardness of the joint path planning of UAVs as there will be less safe trajectories for each UAV. However, the distributed algorithm scales effectively when 4 UAVs are deployed with mines ranging from 1 to 6.

#### 6.5.3.3 Increasing UAVs

Similarly in this subsection, the number of UAVs are varied in the simulated area of operation from 1 to 6, keeping mines and targets constant as per the default scenario. We observe in Figure 6.9a that some targets may remain alive with 1 UAV where, all targets are destroyed every time with 6 UAVs. Hence, destroying targets becomes easier as we increase the number of UAVs in the environment. However, the probability of the task being intercepted by the mines increases with more UAVs when traversing through the area of operation. Therefore, more mines are destroyed as shown in Figure 6.9b.

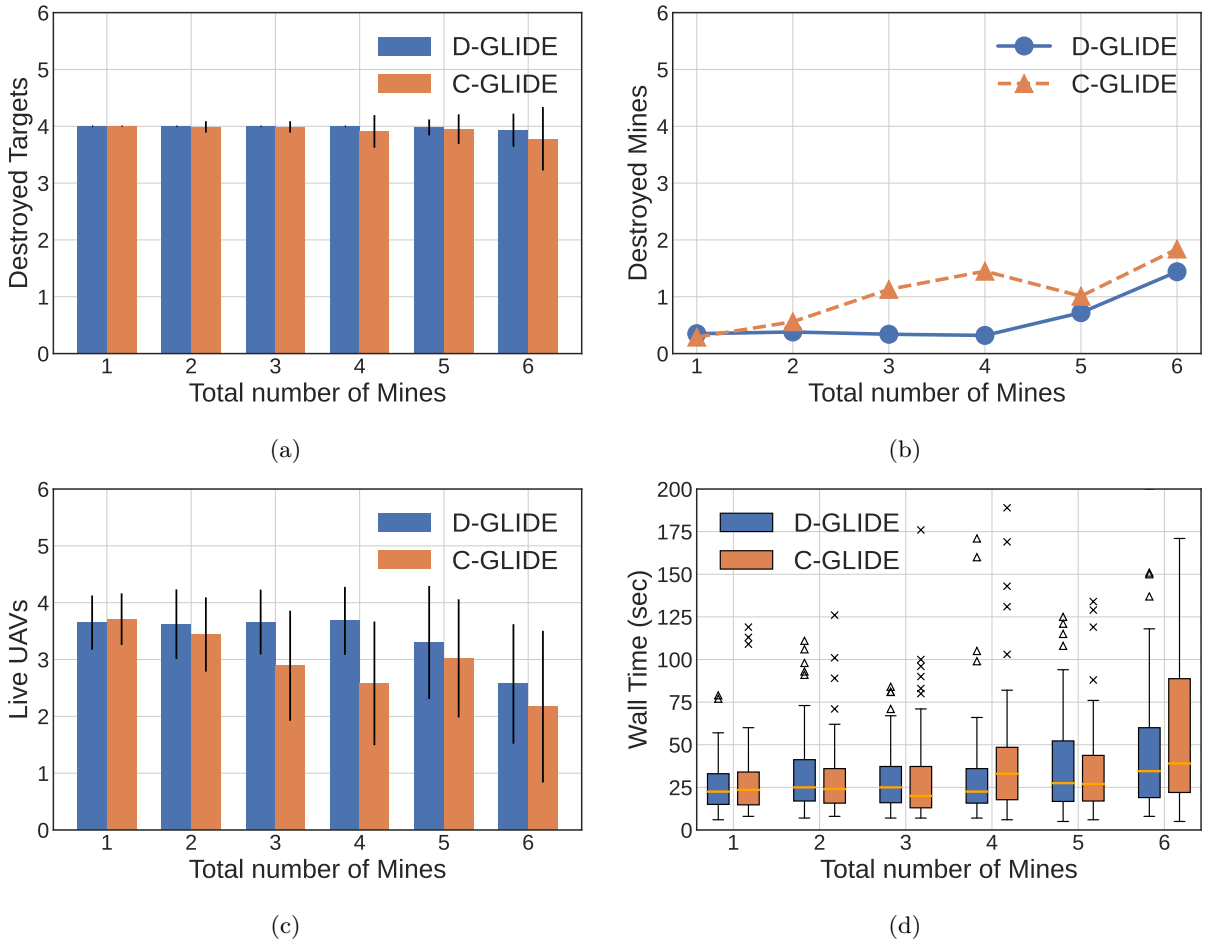



Figure 6.8: Correlation among increasing number of mines vs - (a) destroyed targets (b) destroyed mines (c) live UAVs (d) Wall time or time taken at the end of each episode.

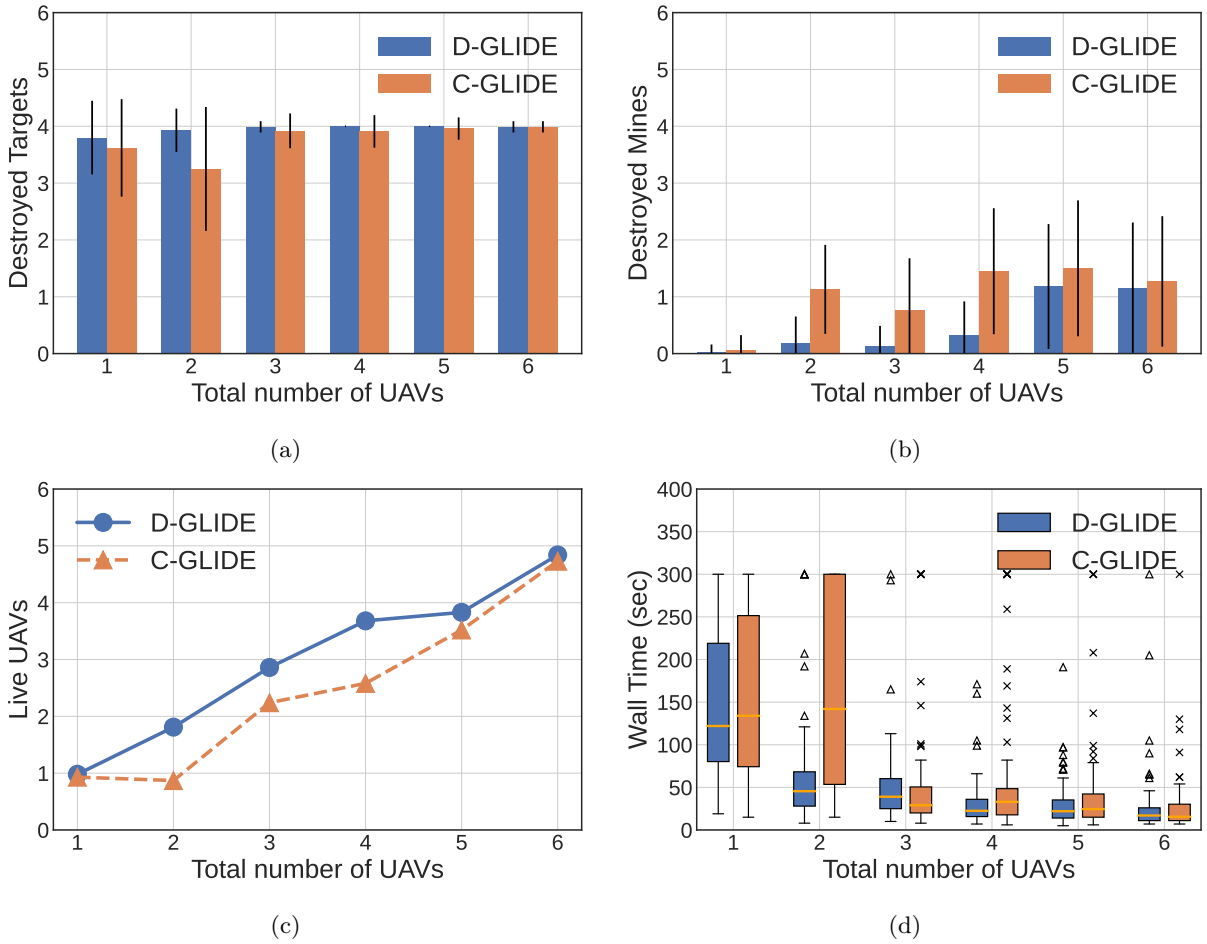
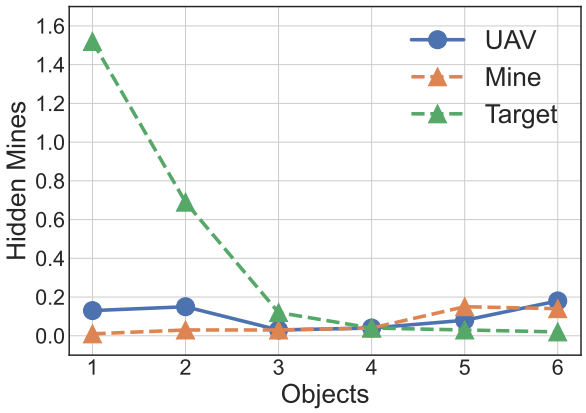
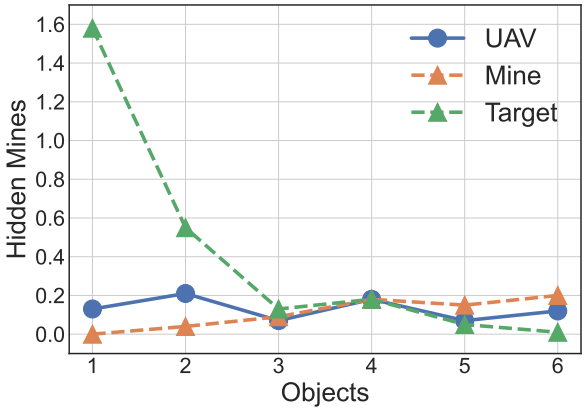



Figure 6.9: Correlation among increasing number of UAVs vs - (a) destroyed targets (b) destroyed mines (c) live UAVs (d) Wall time or time taken at the end of each episode.




(a)




(b)

Figure 6.10: Exploring the operational area to find mines with increasing number of targets, mines and UAVs - (a) D-GLIDE (b) C-GLIDE

The C-GLIDE algorithm suffers from inadequate capacity to facilitate personalized UAV behaviour and thus performs relatively badly than the distributed algorithm in avoiding interception by the mines. As a result, less UAVs are alive after destroying the targets, see Figure 6.9c for more details. Interestingly, from Figure 6.9d, we see a steady decrease in the wall time or time required in destroying the targets with increase in number of UAVs. The UAVs must learn cooperative strategy along with an independent strategy to reduce the wall time (e.g., as a group they have to explore the operational area to discover the hidden mines and thereafter, individual UAV must select an appropriate target and move in the shortest safe trajectory to reach that target, resulting destruction of all targets in minimum time). The D-GLIDE algorithm adapts more group and personalized skills over time. It outperforms the C-GLIDE algorithm in terms of time taken to destroying the targets.

In summary, increasing UAVs, helps in destroying the strategic targets. All the UAVs collaborate most effectively in order to avoid mines and divide the task among themselves for completion in minimum time.

#### 6.5.3.4 Exploring Area of Operation

In this subsection, we analyse the behaviour of the D-GLIDE and the C-GLIDE algorithm in exploring the operational area to find hidden mines with increasing number of UAVs, targets and mines individually, keeping others as per the default scenario. As seen in Figure 6.10a and 6.10b, less mines are discovered by the UAVs as the number of targets decreases. Therefore, both the algorithms can prioritize to destroy the targets in minimum time (see Figure 6.7d) when target count is less and compromise on discovering the mines. This results in less exploration over the area of operation. However, as the number of target increases, the UAVs take more time and discover the hidden mines for safest path planning to destroy the targets.

We vary the number of mines in the area of operation and observe that the UAVs find most of the mines before destroying the targets. Discovering the mines becomes crucial as the mines are increased, otherwise the UAVs may be intercepted. Next, we increase the number of UAVs and observe that most of the mines are discovered during the operation.

In summary, exploring the operational area is crucial in discovering the mines and therefore helps the UAVs to plan safest paths to the targets and destroy them in minimum time.

In this chapter we discussed a DRL based approach for action control of a group of UAVs. We use MARL for the collaborative movement of the UAV fleet in a dynamic military environment. We propose two MARL algorithms called C-GLIDE and D-GLIDE. We developed a simulator called UAV SIM, in which the mines are placed at random locations unknown to the UAVs at the beginning of each episode. The performance of the proposed scheme is is presented and the results show that the D-GLIDE approach significantly outperforms the C-GLIDE approach.

# Chapter 7

# Summary

In this thesis, the importance of scaling blockchain protocols and recent techniques like sharding which improve the throughput of the blockchain protocol are discussed. A coding theory based approach for sharding with the advantage of reduced storage cost and bootstrap cost is introduced. We discussed Prism Blockchain, one of the promising blockchain protocol with good performance guarantees. An approach to optimize the performance of Prism Blockchain by leveraging DRL techniques like DDQN and PPO is discussed. Next, we have presented a multi-agent reinforcement learning based action control for UAVs. A UAV simulator called UAV SIM is developed for our implementation and is discussed. Important results for the above discussed techniques are presented to demonstrate the usefulness.

## 7.1 Conclusion

- To scale blockchains, recent advances in sharding and Prism are considered as baseline for this thesis.
  - A coding theory based approach called secure repair block protocol is presented. The SRB protocol leverages MBR codes.
  - SRB has an improvement in the storage cost and bootstrap cost comparison with the existing schemes.
- This thesis presented a framework leveraging two popular DRL approaches for optimizing Blockchain performance called the Deep Reinforcement Learning-based Prism Blockchain (DRLPB). This scheme dynamically optimizes the parameters of Prism blockchain and helps in achieving a better performance.
  - DRLPB has a DDQN based algorithm is presented to achieve better performance guarantees in Prism Blockchain
  - Later in DRLPB scheme, a PPO based algorithm is compared with the proposed DDQQN based algorithm and the baselines.

  - The analysis shows that the DRLPB scheme adapts the Prism blockchain parameters to enhance the security upto 84% more than Prism, while still preserving the performance guarantees of Prism.

- The main challenge in IoT enabled blockchain is to integrate the metadata into the blocks and store them efficiently. Blockchain-integrated IoT has a challenge of low throughput. In order to improve the throughput and maintain the guarantees of the blockchain-based IoT system, DRL techniques are explored. In this work we extend the Prism based blockchain leveraging DRL for an application use case like IoT. For this, we are implemented the Prism based IoT network in Rust language and PPO based DRL algorithm in Python.

- This thesis also presents a multi agent reinforcement learning approach for UAV action control in a dynamic military environment.

  - A novel problem formulation of the MDP is presented which takes into account the acceleration of the UAVs.
  - The UAV fleet coordinated control is achieved by using two continuous -time PPO based approaches called C-PPO and D-PPO. The C-PPO executes the action control centrally whereas D-PPO involves centralized training and decentralized execution.
  - A UAV simulator called UAV SIM is developed for the performance analysis.

## 7.2 Future works

In this thesis we have compared Rapidchain, SeF and SRB on measures such as storage overhead, encoding complexity and bootstrap cost. In the future, the run time estimates of SRB protocol can be estimated which is based on number of encoding and decoding operations.

The application of blockchain in fields like IoT and supply chain is of particular interest because of the security guarantees provided by the blockchain system. Each application comes with a new set of challenges. The proposed approach of DRLPB and DRL based IoT enabled blockchain can be used with smart contracts. This will be a useful direction to explore the benefits of smart contracts alog with DRL based Prism blockchain.

A continuous-time PPO for UAV action control is implemented so far and we have built our own UAV simulator. In the literature of DRL, the shared experience actor-critic approach has gained popularity recently. In this approach, all the agent's experiences including failed episodes can be used to train the model. This is very advantageous for several applications. This algorithm can be explored for UAV action control. After the completion of the current PPO based MARL for UAV control, we plan to extend it using shared experience actor critic method.

Sharding has been a powerful tool in the context of blockchain. We plan to focus on the merkle tree structure of blockchain to propose an efficient scaling solution. Recent work in coded merkle tree and Rainblock have modified the block structure and addressed challenges like data availability attack. We are currently analyzing the works in literature for solving the data availability attack.

# Related Publications

1. **Divija S.G**, V.Lalitha and V.Aggarwal, "*Secure Regenerating Codes for Reducing Storage and Bootstrap Costs in Sharded Block Chains*," Blockchain'20, November, 2020.

2. **Divija S.G**, V. Lalitha, V.Aggarwal, *"An Optimization Framework based on Deep Reinforcement Learning Approaches for Prism Blockchain," in IEEE Transactions on Services Computing, vol. 16, no. 4, pp. 2451-246, doi: 10.1109/TSC.2023.3242606.*

3. **Divija S.G**, V.Aggarwal, *"Prism Blockchain Enabled Internet of Things with Deep Reinforcement Learning " submitted in IEEE Transactions on Mobile Computing, 2023*

4. **Divija S.G**, Prasenjit.K, Vedant.P, Vijay.S, V.Aggarwal, *"GLIDE: GLIDE: Multi-Agent Deep Reinforcement Learning for Coordinated UAV Control in Dynamic Military Environments"* submitted in ACM Journal on Autonomous Transportation Systems, 2023.

# Bibliography


[1] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 931–948.

[2] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Sok: Consensus in the age of blockchains," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 183–198.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[4] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[5] 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, 2019.

[6] S. Li, M. Yu, C.-S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, "Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 249–261, 2020.

[7] S. Kadhe, J. Chung, and K. Ramchandran, "Sef: A secure fountain architecture for slashing storage costs in blockchains," *arXiv preprint arXiv:1906.12140*, 2019.

[8] J. Yun, Y. Goh, and J.-M. Chung, "Dqn based optimization framework for secure sharded blockchain systems," *IEEE Internet of Things Journal*, 2020.

[9] R. Ma, Z. Yi, Y. Xiang, D. Shi, C. Xu, and H. Wu, "A blockchain-enabled demand management and control framework driven by deep reinforcement learning," *IEEE Transactions on Industrial Electronics*, 2022.

[10] S. Bano, M. Al-Bassam, and G. Danezis, "The road to scalable blockchain designs," *USENIX; login magazine*, vol. 42, no. 4, pp. 31–36, 2017.

[11] ""blockchain luxembourg s.a.","  *https://www.blockchain.com/charts/blocks-size, [Online; Accessed on 10/13/2019].*

[12] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 45–59. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal

[13] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22 328–22 370, 2019.

[14] N. Chawla, H. W. Behrens, D. Tapp, D. Boscovic, and K. S. Candan, "Velocity: Scalability improvements in block propagation through rateless erasure coding," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 447–454.

[15] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[16] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[17] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[18] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, "Erasure code-based low storage blockchain node," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1622–1627.

[19] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded merkle tree: Solving data availability attacks in blockchains," 2019.

[20] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 585–602.

[21] L. Yang, V. Bagaria, G. Wang, M. Alizadeh, D. Tse, G. Fanti, and P. Viswanath, "Prism: Scaling bitcoin by 10,000 x," *arXiv preprint arXiv:1909.11261*, 2019.

[22] J. Cui, Y. Liu, and A. Nallanathan, "The application of multi-agent reinforcement learning in uav networks," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2019, pp. 1–6.

[23] C. Yan and X. Xiang, "A path planning algorithm for uav based on improved q-learning," in *2018 2nd International Conference on Robotics and Automation Sciences (ICRAS)*, 2018, pp. 1–5.

[24] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen, "Reinforcement learning for autonomous uav navigation using function approximation," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2018, pp. 1–6.

[25] S. Islam and A. Razi, "A path planning algorithm for collective monitoring using autonomous drones," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*, 2019, pp. 1–6.

[26] Y. Li, S. Zhang, F. Ye, T. Jiang, and Y. Li, "A uav path planning method based on deep reinforcement learning," in *2020 IEEE USNC-CNC-URSI North American Radio Science Meeting (Joint with AP-S Symposium)*. IEEE, 2020, pp. 93–94.

[27] S. Rahim, M. M. Razaq, S. Y. Chang, and L. Peng, "A reinforcement learning-based path planning for collaborative uavs," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 1938–1943.

[28] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[29] M. T. Mamaghani and Y. Hong, "Intelligent trajectory design for secure full-duplex mimo-uav relaying against active eavesdroppers: A model-free reinforcement learning approach," *IEEE Access*, vol. 9, pp. 4447–4465, 2020.

[30] Z. Yijing, Z. Zheng, Z. Xiaoyi, and L. Yang, "Q learning algorithm based uav path learning and obstacle avoidence approach," in *2017 36th Chinese Control Conference (CCC)*. IEEE, 2017, pp. 3397–3402.

[31] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[32] L. M. Schmidt, J. Brosig, A. Plinge, B. M. Eskofier, and C. Mutschler, "An introduction to multi-agent reinforcement learning and review of its application to autonomous mobility," *arXiv preprint arXiv:2203.07676*, 2022.

[33] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[34] C. Zhou, H. He, P. Yang, F. Lyu, W. Wu, N. Cheng, and X. Shen, "Deep rl-based trajectory planning for aoi minimization in uav-assisted iot," in *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2019, pp. 1–6.

[35] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[36] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[37] X. Bai, C. Lu, Q. Bao, S. Zhu, and S. Xia, "An improved ppo for multiple unmanned aerial vehicles," in *Journal of Physics: Conference Series*, vol. 1757, no. 1. IOP Publishing, 2021, p. 012156.

[38] C. Yan, X. Xiang, and C. Wang, "Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, pp. 297–309, 2020.

[39] U. Ates, "Long-term planning with deep reinforcement learning on autonomous drones," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020, pp. 1–6.

[40] J. Karamačoski, N. Paunkoska, N. Marina, and M. Punčeva, "Blockchain for reliable and secure distributed communication channel," in *2019 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 2019, pp. 91–97.

[41] N. Mhaisen, N. Fetais, A. Erbad, A. Mohamed, and M. Guizani, "To chain or not to chain: A reinforcement learning approach for blockchain-enabled iot monitoring applications," *Future Generation Computer Systems*, vol. 111, pp. 39–51, 2020.

[42] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2188–2204, 2018.

[43] 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, 2019.

[44] J. Li and D. Guo, "Liveness and consistency of bitcoin and prism blockchains: The non-lockstep synchronous case," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–9.

[45] ——, "On analysis of the bitcoin and prism backbone protocols in synchronous networks," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 17–24.

[46] G. Wang, S. Wang, V. Bagaria, D. Tse, and P. Viswanath, "Prism removes consensus bottleneck for smart contracts," in *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2020, pp. 68–77.

[47] G. Zyskind, O. Nathan, *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 2015, pp. 180–184.

[48] T. Alam, A. Ullah, and M. Benaida, "Deep reinforcement learning approach for computation offloading in blockchain-enabled communications systems," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–14, 2022.

[49] F. Jameel, U. Javaid, W. U. Khan, M. N. Aman, H. Pervaiz, and R. Jäntti, "Reinforcement learning in blockchain-enabled iiot networks: A survey of recent advances and open challenges," *Sustainability*, vol. 12, no. 12, p. 5161, 2020.

[50] A. Z. Al-Marridi, A. Mohamed, and A. Erbad, "Reinforcement learning approaches for efficient and secure blockchain-powered smart health systems," *Computer Networks*, vol. 197, p. 108279, 2021.

[51] Y. Wu, Z. Wang, Y. Ma, and V. C. Leung, "Deep reinforcement learning for blockchain in industrial iot: A survey," *Computer Networks*, vol. 191, p. 108004, 2021.

[52] Z. Yang, R. Yang, F. R. Yu, M. Li, Y. Zhang, and Y. Teng, "Sharded blockchain for collaborative computing in the internet of things: Combined of dynamic clustering and deep reinforcement learning approach," *IEEE Internet of Things Journal*, 2022.

[53] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep rl: A case study on ppo and trpo," in *International conference on learning representations*, 2019.

[54] C. Yan, X. Xiang, and C. Wang, "Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, pp. 297–309, 2020.

[55] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "Multi-uav path planning for wireless data harvesting with deep reinforcement learning," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171–1187, 2021.

[56] B. Li and Y. Wu, "Path planning for uav ground target tracking via deep reinforcement learning," *IEEE Access*, vol. 8, pp. 29 064–29 074, 2020.

[57] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV path planning using global and local map information with deep reinforcement learning," *CoRR*, vol. abs/2010.06917, 2020. [Online]. Available: https://arxiv.org/abs/2010.06917

[58] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, "Path planning for uav-mounted mobile edge computing with deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5723–5728, 2020.

[59] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "Uav path planning for wireless data harvesting: A deep reinforcement learning approach," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[60] ——, "Multi-uav path planning for wireless data harvesting with deep reinforcement learning," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171–1187, 2021.

[61] C. H. Liu, Z. Chen, J. Tang, J. Xu, and C. Piao, "Energy-efficient uav control for effective and fair communication coverage: A deep reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 2059–2070, 2018.

[62] T. Wang, R. Qin, Y. Chen, H. Snoussi, and C. Choi, "A reinforcement learning approach for uav target searching and tracking," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 4347–4364, 2019.

[63] B. Zhang, Z. Mao, W. Liu, and J. Liu, "Geometric reinforcement learning for path planning of uavs," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[64] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 523–533.

[65] S. Liu, "Bitcoin blockchain size 2010-2020, by quarter," https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/, jun 2020.

[66] S. Kadhe, J. Chung, and K. Ramchandran, "Sef: A secure fountain architecture for slashing storage costs in blockchains," *arXiv preprint arXiv:1906.12140*, 2019.

[67] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for blockchains," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2619–2623.

[68] M. Dai, S. Zhang, H. Wang, and S. Jin, "A low storage room requirement framework for distributed ledger in blockchain," *IEEE Access*, vol. 6, pp. 22 970–22 975, 2018.

[69] W. Liang, Y. Fan, K.-C. Li, D. Zhang, and J.-L. Gaudiot, "Secure data storage and recovery in industrial blockchain network environments," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6543–6552, 2020.

[70] M. Luby, "Lt codes," in *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002, pp. 271–280.

[71] G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer, "Divide and scale: Formalization of distributed ledger sharding protocols," *arXiv preprint arXiv:1910.10434*, 2019.

[72] N. B. Shah, K. Rashmi, and P. V. Kumar, "Information-theoretically secure regenerating codes for distributed storage," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, Houston, Texas, USA, 2011, pp. 1–5.

[73] A. Hafid, A. S. Hafid, and M. Samih, "New mathematical model to analyze security of sharding-based blockchain protocols," *IEEE Access*, vol. 7, pp. 185 447–185 457, 2019.

[74] S. K. Singh, S. Rathore, and J. H. Park, "Blockiotintelligence: A blockchain-enabled intelligent iot architecture with artificial intelligence," *Future Generation Computer Systems*, vol. 110, pp. 721–743, 2020.

[75] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[76] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

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

[78] X. Lan, X. Tang, D. Zhai, D. Wang, and Z. Han, "Blockchain-secured data collection for uav-assisted iot: A ddpg approach," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[79] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[80] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[81] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[82] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[83] U. Challita, W. Saad, and C. Bettstetter, "Deep reinforcement learning for interference-aware path planning of cellular-connected uavs," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[84] X. Liu, Y. Liu, and Y. Chen, "Reinforcement learning in multiple-uav networks: Deployment and movement design," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8036–8049, 2019.

[85] D. Chen, Q. Qi, Z. Zhuang, J. Wang, J. Liao, and Z. Han, "Mean field deep reinforcement learning for fair and efficient uav control," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 813–828, 2021.

[86] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 5571–5580.