Optimized Multi-Agent Deep Reinforcement Learning for Target Search and Localization

Ahmed Alagha

A Thesis

in the Concordia Institute

for

Information and Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montréal, Québec, Canada

November 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Ahmed Alagha

Entitled: **Optimized Multi-Agent Deep Reinforcement Learning for Target Search and Localization**

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

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Rodolfo Coutinho

_____ External Examiner
Dr. Georges Kaddoum

_____ Arm's Length Examiner
Dr. Lyes Kadem

_____ Internal Examiner
Dr. Farnoosh Naderkhani

_____ Internal Examiner
Dr. Manar Amayri

_____ Supervisor
Dr. Jamal Bentahar

_____ Co-supervisors
Dr. Rabeb Mizouni & Dr. Shakti Singh

Approved by _____
Dr. Farnoosh Naderkhani, Graduate Program Direction

October 30, 2024
_____
Date of Defense

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Optimized Multi-Agent Deep Reinforcement Learning for Target Search and Localization

Ahmed Alagha, Ph.D.

Concordia University, 2024

In a world that increasingly relies on autonomous systems, swarm robotics hold the promise of revolutionizing how complex tasks, such as target search and localization, are approached. The ability of multiple autonomous agents, such as robots and UAVs, to work together, exchange information, and adapt to dynamic environments is critical for target localization applications ranging from search and rescue missions to environmental monitoring. However, efficiently coordinating a swarm of robots to search for targets in uncertain and complex environments poses significant challenges. Most existing solutions for target search and localization still possess challenges and limitations in terms of adaptability to different environments and scalability. These challenges become even more intricate when the target may not exist (i.e. false alarms) or is unreachable.

In this thesis, the main motivation is to leverage AI, specifically Multi-Agent Deep Reinforcement Learning (MDRL), to address the target search and localization problem. The aim is to develop MDRL solutions where the agents intelligently and autonomously learn to tackle the problem and its different complexities, with minimum human intervention. The capacity of MDRL in producing agents capable of learning from their experiences in the environment proves efficient in handling complex and dynamic scenarios, such as coordinating with other agents, translating data readings into actions that lead to the target, and

navigating obstacles in the environment. This research is motivated by four main needs: (1) Adaptable solutions for collaborative target search and localization for varying environment complexities; (2) autonomous and intelligent sensing agents with decision-making that addresses scenarios like false alarms and target unreachability; (3) scalable and efficient AI-based learning process for the sensing agents, and (4) mechanisms for knowledge exchange across different users and parties from different domains for better accessibility to AI solutions.

The aforementioned needs are addressed in this thesis by: (1) Developing novel MDRL algorithms for collaborative target search in both simple and cluttered environments by modeling the problem as a Markov Decision Process (MDP), (2) designing novel methods based on ideas from MDRL, Imitation Learning (IL), and reward shaping for enhanced and quick learning performance, (3) enhancing the proposed MDRL algorithms by integrating complex decision-making, where agents can take actions ranging from mobility to deciding on the existence and reachability of the target, (4) developing a blockchain-based platform for Deep Reinforcement Learning as a Service (DRLaaS) allowing collaborative training and better accessibility to DRL solutions for target localization problems, and (5) ensuring the scalability of all the proposed solutions through the use of concepts such as Centralized-Learning and Distributed Execution (CLDE) MDRL methods coupled with Convolutional Neural Networks (CNNs) for optimized analysis of the agents' collected observations. Besides these contributions, we present several experimental studies and simulations that validate the proposed methods and compare them against existing state-of-the-art benchmarks in the literature.

Acknowledgments

First and foremost, I am grateful to God for granting me the strength, patience, and guidance throughout this challenging journey.

I would like to extend my sincere gratitude to my supervisor, Dr. Jamal Bentahar, for his unwavering support and guidance throughout my research. I am also deeply thankful to my co-supervisors, Dr. Rabeb Mizouni and Dr. Shakti Singh, for their expertise, constructive feedback, and constant support. I would like to offer special thanks to Dr. Hadi Otrok, who has been not only a mentor but also an older brother to me. His guidance and belief in me have been a source of strength throughout this path. Looking back at how much I have grown and developed over the past years, I feel immensely fortunate and grateful to have crossed paths with such incredible mentors, whose guidance and support have been pivotal to my journey.

I extend my heartfelt gratitude to the esteemed members of my PhD committee: Dr. Lyes Kadem, Dr. Farnoosh Naderkhani, Dr. Manar Amayri, and Dr. Georges Kaddoum. Their insightful feedback, constructive criticism, and invaluable guidance throughout the evaluation process have greatly enriched this work. I deeply appreciate their time and effort in reviewing my research and contributing to the success of this thesis.

The journey through this PhD would not have been the same without the incredible support and friendship of those around me. To my dear friends: Ahmad Hammoud, Kareem Hamdash, Mario Chahoud, Hani Sami, Mouhamad Arafeh, Osama Wehbi, Mohamad

Wazzeh, and my dear brother Ali Alagha, your companionship made even the most challenging days manageable. I am truly blessed to have such a remarkable group of friends who have stood by me, and I am deeply grateful for the laughter, support, and strength you have provided along the way.

Last but not least, none of this would have been possible without the love and support of my family. To my parents, your endless sacrifices, guidance, and constant faith in me have laid the foundation for everything I have achieved. To my brothers, your constant support has been invaluable throughout this journey.

This research was made possible through the support of the Fonds de recherche du Québec - Nature et technologies (FRQNT) and Concordia University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid development of sensing technologies, *target search and localization* has emerged as a problem of interest in tasks related to homeland security and environmental monitoring [1, 2]. The problem is defined as the process of identifying the location of a certain target using sensory data readings collected by sensing agents. Examples of such tasks include radiation monitoring [1, 3], forest fires detection [4], intruder tracking [5, 6], and search and rescue missions [7, 8]. In such applications, a set of sensing nodes collaborate to identify the location of (i.e. localize) a target in an Area of Interest (AoI).

A traditional approach to estimate the target location is to aggregate and fuse data readings from a group of sensing nodes in a central unit. Such data fusion methods include Maximum Likelihood Estimate [9], Bayes' theorem [10, 11], and Copula theory [12]. Existing proposals using this approach mainly aim at optimizing either the deployment of sensing nodes or the selection of active nodes [1, 13]. An alternative approach is to specifically deploy a group of mobile sensing agents, such as UAVs or robots, to perform surveillance and localize the target. Most works adopting this method optimize target detection time by surveying some pre-defined or data-driven paths [3, 14, 15]. However, localizing the target through location estimates or pre-defined survey paths has several limitations. The proposals in data fusion mainly deal with stationary sensing nodes, and hence the estimated

target location comes with uncertainty, since no physical agents are actually at the target location. On the other hand, the existing works using pre-defined survey paths with mobile agents suffer from adaptability issues, as further re-modeling and supervision are required with the increasing complexity and dynamicity of the environment.

To tackle the aforementioned issues, Reinforcement Learning (RL) comes as an efficient method in obtaining agents that are capable of building their own intelligence, with limited supervision. RL has been proposed to tackle problems in several domains, including robotics [7, 16, 17], online games [18], and wireless channel access [19]. In robotics, the goal of RL is to endow robots with the ability to learn, improve, adapt, and reproduce tasks with dynamically changing constraints [16, 20]. In RL, an agent learns a policy for decision-making based on its experience in the environment, guided by a numerical reward signal that the agent tries to maximize [21]. This makes RL suitable for localization tasks, since localization agents continuously interact with the environment and collect observations (sensor readings), which can be used to determine their next actions.

While many of the initial works in RL utilize a single agent; several real-world decision-making problems, like target localization, naturally fall in the realm of *Muti-Agent Systems* and *Multi-Agent RL (MARL)*. In MARL, multiple autonomous agents act in a common environment aiming to maximize their reward, through the interaction with the environment and other agents [22]. Agents could be cooperative, i.e. aiming to achieve a common goal, or competitive, i.e. each with its own goal. MARL has applications in video games [23], autonomous driving [24], resource allocation problems [25], traffic control [26], and robot swarms [17], to name a few.

Amongst the main challenges faced in MARL is the curse of dimensionality, which refers to the increasing calculations that need to be made with increasing number of inputs/agents [22, 27]. The recent breakthroughs in Deep Learning [28], and its integration in Deep Reinforcement Learning (DRL) [18], have helped in partially overcoming the curse

of dimensionality. As a result, several proposals have successfully employed Multi-Agent Deep Reinforcement Learning (MDRL), such as OpenAI Five, which is a multi-agent AI that won against the world champions in the competitive five-on-five video game Dota 2 [23]. Another challenge is that of non-stationarity. While an agent is only concerned with the outcome of its own actions in single-agent settings, agents in the multi-agent domain need to observe the behavior of other agents. The agents here learn concurrently, and hence the environment is in constant reshaping, thus becoming non-stationary from the perspective of each agent [22, 29].

In this thesis, we aim to address the target search and localization problem and its complexities using MDRL, while also tackling the challenges associated with MDRL. In this context, we develop scalable MDRL methods in which sensing agents autonomously learn how to execute the target localization problem in a timely and cost-effective manner, and in different scenarios of varying complexities. The agents not only learn how to search for the target, but also learn to coordinate to manage resources. We also focus on the efficiency of the learning algorithm and ensure its convergence by proposing novel methods combining ideas from MDRL, Imitation Learning (IL), and reward shaping to speed up the learning. These methods are further enabled through the development of a novel blockchain-based knowledge sharing platform that allows users and entities to request tasks related to designing and training Deep Reinforcement Learning (DRL) models, increasing the accessibility to DRL solutions for target search and localization. In summary, this thesis develops novel scalable MDRL methods, integrating IL and reward shaping, to efficiently solve the target search and localization problem under varying complexities, while also enabling collaboration and accessibility through a blockchain-based knowledge-sharing platform.

1.1 Target Search and Localization: An Example

The target localization problem tackled in this work is represented in Fig. 1.1 using a radioactive environment with 3 agents and 3 walls. The readings collected by each agent (in photon counts per minute), at each time step t_i, are given along the paths. The problem is defined as follows: given N agents and a certain area of interest (AoI) within which an unknown target is located, the agents are to cooperate to search the area and find this target. An agent should use the collected observations, including the data readings and information about the environment and other agents, to take movement actions in the environment aiming to find the target. The observations are collected in a progressive manner and are updated after each step in the environment. Data readings might not always be sufficient, which requires the agents to cooperate in exploring the environment to collect more readings, which can be then used for better decision-making. It is assumed that agents are capable of storing their history in terms of data readings and locations visited. Agents are also capable of communicating with each other to share information. Obstacles in the environment increase the complexity of the problem, as they attenuate the signal from the source (i.e. the radiation), affect the mobility of the agents, and could potentially lead to the target being unreachable.

The desired behavior in such environments is for the agents to quickly reach the unknown target location while cooperating to preserve resources. The idea is, even though the target location is unknown, the data readings induced by the target should help the agents take the right actions that lead to exploring the environment in a way that leads to quick localization. The agents should be able to tackle the localization problem for any target location and any initial distribution of agents and walls. Initially, a coordinated exploration is needed for the agents collect some readings and build an intuition about the possible target locations. Cooperation should also be seen in resource management, as agents should only attempt to contribute to the task (by searching) if they can add benefit to the team. These

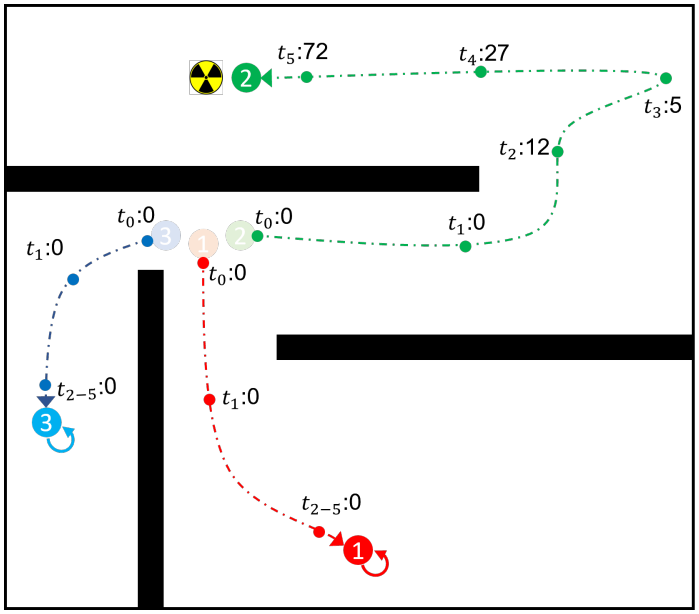


Figure 1.1: A representation of the target localization problem.

desired behaviors are illustrated in Fig. 1.1, which presents a simplified scenario. Initially (t_0-t_1), the three agents have similar low readings, and hence the ideal situation would be to coordinate and split to cover and explore a wider area. At t_2, and given the readings and location histories of the 3 agents, it is evident that agent 2 has come closer to the target, and hence agents 1 and 2 choose to stay idle to preserve resources. At the same time step, agent 2 is uncertain as to which direction to move, and hence chooses to explore the area to the right, before realizing that the better action is to move left. This example summarizes three desired behaviors in the agents: 1) To cooperate and explore when data readings are insufficient, 2) to incorporate the history of readings/locations into the decision-making process with the aim of moving towards areas with potentially higher readings, and 3) to preserve resources when the agents cannot be beneficial to the target localization task. This is while considering the existence of obstacles/walls that increase the complexity of the problem. These behaviors are to be developed by the agents themselves through rewarded interaction with the environment using the proposed MDRL models. The problem increases in difficulty in scenarios such as false alarms (i.e. the target does not exist) and unreachable

targets, requiring higher coordination between the agents.

The aforementioned behaviors are difficult to achieve through the existing target search works in the literature. Works using pre-defined survey paths might guarantee target localization since the entire area is scheduled for scanning, but the lack of coordination or smart decision-making results in a high localization time and a waste of resources. On the other hand, data-driven methods use guided decision-making based on the collected readings, where agents move towards higher readings. Such methods fall inefficient in complex environments that cannot be modeled. For example, in environments with obstacles (Fig. 1.1), agents might not always need to move towards higher readings due to the obstacles blocking the direct path. Additionally, obstacles attenuate the collected data readings, making it difficult to fully depend on the readings. It is also difficult to incorporate obstacles, their locations, and their features into decision-making mathematical models. Alternatively, Deep Neural Networks (DNNs) have been common in modeling complex environments in DRL. Some works adopted DRL to tackle the target search environment in single-agent settings [2]. The DNNs in DRL allow modeling complex environments in different formats, making it possible to develop complex behaviors. However, these works only tackle single-agent environments, and cannot be directly extrapolated to multi-agent environments due to the lack of coordination in single-agent settings and the curse of dimensionality in multi-agent settings.

In summary, existing methods for target localization and MDRL suffer from the following limitations:

- Traditional target estimation methods suffer from uncertainty when localizing the target, since they rely on the use of previously placed stationary sensing nodes that only give an estimation to the target location.
- Existing target search and localization methods cannot be adapted to complex environments where coordination and optimized decision-making is needed.

- Single-agent DRL method for target search and localization cannot be directly extrapolated to multi-agent settings, due to the lack of coordination in single-agent settings and the curse of dimensionality in multi-agent settings.

1.2 Problem Statement and Research Questions

Target search and localization using collaborative sensing agents presents several challenges. The main challenge is to develop a system that is adaptable to different environments without the need of human intervention to re-design solutions. This has pushed researchers to explore the use of DRL in addressing the problem in single-agent settings, where an algorithm is developed that the agent follows to build its own intelligence by experiencing the environment. However, the target search and localization problem is naturally a multi-agent problem, and extrapolating existing single-agent DRL solutions into multi-agent settings bring many issues in terms of the curse of dimensionality [22, 27]. This refers to the exponential growth of the state/action search space of the problem with the increasing number of agents. Additionally, having multiple agents collaborating in the same environment increases the difficulty of decision-making per each agent, as they need to consider the behaviors of other agents for optimal collaboration and quick target localization process. This leads to the first research question (RQ1) of this thesis:

- **RQ1: How to design MDRL solutions that are scalable with the number of agents and the observations they collect, while also ensuring efficient collaboration?**

The decision-making by an agent in the target search and localization problem relies mainly on the observations collected by the agent and how they translate into actions. In this problem, these observations are usually related to the complexity of the environment, sensory data, as well as observations about other agents. Since the problem is a sequential

decision-making one, these spatial observations accumulate over time, which calls for efficient representations of the collected observations to ensure the agents can capture spatial and temporal features without the need of constant re-design of the learning model. With the increase of the number of agents in the environment, the number of observations increases, which is another issue to be addressed by the representation task. In addition to scalability issues regarding the observations, the action space of the problem exponentially grows with the number of agents. Given a single agent with $\mathcal{A}$ possible mobility decisions/actions in a given timestep, the action space becomes $\mathcal{A}^N$ if the problem extends to multi-agent settings with N agents. Finally, DRL is based on rewarded interaction with the environment, where an agent aims to maximize its own reward. Without proper considerations to collaborative behaviors, an agent in a multi-agent environment could act selfishly to maximize its own reward without considering the performance of the team, which is another major issue. All the aforementioned are challenges associated with the adoption of MDRL for target search and localization problems even in simple environments. However, how does that change for more complex environments? How is the problem affected by scenarios of environments with obstacles, false alarms, and unreachable targets? Henceforth, our second research question (RQ2) is:

- **RQ2: How to extend MDRL solutions to consider complex environments and varying target localization scenarios?**

In environments with obstacles, agents must navigate through obstructed spaces where mobility is significantly affected, often requiring more complex paths to reach the target. These obstacles not only hinder the movement of the agents but also attenuate the data readings that reach their sensors, introducing more uncertainty into the observations. Additionally, having obstacles could lead to scenarios where the target exists but is unreachable, which requires more complex decision-making where the agents need to flag the existence

of the target and attempt to estimate its location. All of the above adds additional layers of difficulty to the task, as agents must account for inaccurate data while analyzing the environment and coordinating with one another. The increased complexity of both the observation and action spaces in these scenarios could lead to slower learning and longer convergence times. Therefore, addressing the challenges of such environments requires not only extending MDRL to handle these scenarios, but also optimizing the learning process to ensure faster convergence. This leads to the third research question (RQ3):

- **RQ3: How to optimize MDRL for target localization to ensure efficient exploration, fast convergence, and robust performance in complex environments with obstacles?**

Training MDRL solutions is notoriously demanding due to the complexity of multi-agent interactions, which require the agents to learn both individually and collaboratively. Due to the increased state and action spaces in multi-agent settings, and the inherent complexity of the target search and localization problem in complex environments, there is a dire need to optimize the learning process and ensure convergence in a timely manner. Inadequately modeling the problem or the observations can lead to a misrepresentation of the environment, resulting in sub-optimal policies and reduced agent performance. Furthermore, poorly designed or sparse reward functions can slow down the learning process, as agents struggle to associate their actions with meaningful feedback. Even if the reward function and the observations are well-designed, the inherent complexity of the target search and localization problem could potentially lead to slow convergence. Therefore, optimizing the MDRL learning process to ensure efficient exploration, faster convergence, and robust performance across complex environments is essential. Successfully addressing these challenges requires expertise not only in DRL but also in the specific domain of the problem, such as target localization and environmental dynamics. Given the complexity and multi-disciplinary nature of the task, knowledge sharing and collaboration are essential

to develop robust DRL solutions. This necessitates leveraging crowdsourced DRL services and expertise to accelerate innovation. Hence, the final research question (RQ4) is:

- **RQ4: How can we facilitate knowledge sharing and provide DRL services to enhance MDRL solutions for complex environments?**

Developing effective DRL solutions requires a wide range of expertise, including knowledge of DRL algorithms, multi-agent systems, reward engineering, and domain-specific understanding of the problem at hand. For many users, especially those without a strong technical background, accessing this expertise is a significant barrier. Existing DRL services tend to be centralized, proprietary, and often prohibitively expensive, limiting accessibility for researchers, small organizations, or individuals. This creates a demand for more democratized and crowdsourced solutions, where experts can contribute their knowledge, share models, and offer DRL services on an open platform. Crowdsourced DRL not only facilitates knowledge sharing but also provides a more affordable and scalable way to access ready-made solutions for inexperienced users. Moreover, this collaborative approach can ease the training of other models by offering pre-trained models, best practices, and domain-specific insights, ultimately lowering the entry barrier for those wishing to implement MDRL in complex multi-agent environments like target search and localization.

1.3 Research Objectives and Contributions

The ultimate goal of this thesis is to develop novel scalable MDRL methods, integrating IL and reward shaping, to efficiently solve the target search and localization problem under varying complexities, while also enabling collaboration and accessibility through a crowdsourced blockchain-based knowledge-sharing platform. To achieve this, we have outlined specific objectives as follows:

- **Objective 1**: Model the target search and localization environment in MDRL. This includes the careful design of the agents' observations in the environment, in addition to the interactions between the agents and the environment and between the agents themselves, in simple and complex environments.

- **Objective 2**: Design optimized and scalable MDRL methods for target search and localization, which achieve the desired performance in terms of localization time and resource management.

- **Objective 3**: Develop and implement MDRL solutions that consider scenarios such as false alarms (target non-existence) and unreachable targets, which demands complex decision-making.

- **Objective 4**: Design methods to speed up the learning in MDRL, for the target localization problem and other similar problems, using ideas from IL and knowledge sharing.

- **Objective 5**: Design a framework to increase the accessibility to DRL solutions by utilizing the expertise of the crowd for complex problems such as target localization.

To achieve the stated objectives, and to fill the gaps in existing literature (Section 1.1) and answer the presented research questions (Section 1.2), this thesis makes the following significant contributions:

(1) **Contribution 1**: We propose novel MDRL methods for cooperative target search and localization in simple environments, with emphasis on solution scalability, optimized representation of observations, and efficient collaboration. The proposed method proves efficient in producing intelligent agents capable of achieving the desired behaviors in terms of quick, cooperative, and cost-efficient target localization. **(This contribution is discussed in Chapter 3)**.

(2) **Contribution 2**: We extend the proposed MDRL methods to account for complex environments with obstacles. We address the increased complexity of the problem by proposing a novel method, named Demonstration Cloning (DC), that combines ideas from MDRL and IL for fast learning convergence. **(This contribution is discussed in Chapter 4)**.

(3) **Contribution 3**: We propose a novel blockchain-assisted Multi-Expert Demonstration Cloning (MEDC) method that allows for the sharing of pre-trained MDRL models to assist in training new solutions for target localization, using an upgraded version of the previously proposed DC method **(This contribution is discussed in Chapter 5)**.

(4) **Contribution 4**: We develop new MDRL methods that account for target search and localization scenarios with uncertainties, such as false alarms where the target does not exist and unreachable targets. The proposed methods combine actions on different dimensionalities, such mobility, flagging the reachability and existence of the target, and estimating its location. Multiple models are combined into one using ideas from Transfer Learning (TL) combined with MDRL **(This contribution is discussed in Chapter 6)**.

(5) **Contribution 5**: We design a novel crowdsourced blockchain-assisted Deep Reinforcement Learning as a Service (DRLaaS) framework that addresses the lack of diverse expertise and computational capabilities for typical users, by crowdsourcing DRL design and training tasks to experts instead of relying solely on centralized platforms, which is proven to be essential for complex problem like target search and localization **(This contribution is discussed in Chapter 7)**.

1.4 Thesis Organization

The thesis is organized as follows:

Chapter 2 presents a detailed background on topics such as target search and localization, Markov Decision Processes (MDPs), MDRL, CNNS, and Proximal Policy Optimization (PPO), which is the used DRL algorithm in this work. It also includes a literature review on existing solutions for target localization, MDRL, reward shaping, IL-assisted DRL, Federated Reinforcement Learning (FRL), Machine Learning as a Service (MLaaS), and crowdsourcing for machine learning, which are important topics related to the proposed solutions.

Chapter 3 presents and discusses the proposed scalable MDRL solutions for target search and localization in simple environments. It focuses on modeling the problem in MDRL settings, designing the observations and reward function for optimized and collaborative learning, and uses methods in CNNs and PPO to define and optimize the agents' policies. We show the efficiency and scalability of the proposed methods and compare them to several benchmarks.

Chapter 4 presents an extension of our MDRL methods to account for more complex environments, specifically those filled with obstacles that hinder mobility and attenuate sensor data. We introduce a novel method named Demonstration Cloning (DC), which integrates principles from MDRL and IL to accelerate learning convergence in these challenging environments. Several experiments and benchmarks are used to validate the proposed methods.

Chapter 5 expands on the DC method by incorporating a novel blockchain-assisted framework which facilitates the sharing of pre-trained MDRL models across users. The chapter discusses the integration of blockchain for secure and decentralized sharing of models, addressing scalability and collaboration challenges in the training of MDRL agents. Experiments and implementations of smart contracts are discussed, and comparisons with

existing literature are conducted.

Chapter 6 focuses on addressing uncertainties such as false alarms and unreachable targets in target search and localization tasks. We propose novel methods that allow agents to make decisions across multiple dimensions, including mobility, flagging unreachable or non-existing targets, and estimating target locations. By combining TL with MDRL, the chapter highlights how various models are integrated to tackle complex uncertainties, ensuring robust agent behavior in unpredictable environments, with several experiments conducted to validate the proposed methods.

Chapter 7 introduces a novel crowdsourced blockchain-assisted DRL as a Service (DRLaaS) framework, which provides a decentralized platform for users to access diverse DRL expertise and computational resources. This framework allows for the outsourcing of complex DRL design and training tasks to experts, enabling scalable and efficient solutions for target search and localization. The chapter explores the benefits of moving beyond centralized services and how crowdsourcing ensures greater accessibility and innovation in DRL/MDRL applications like target search and localization.

Finally, we summarize the thesis contributions in **Chapter 8** and highlight on the existing research gap that require further consideration by the research community and highlight some potential future directions.

Chapter 2

Background and Literature Review

This chapter presents a detailed background on topics such as target search and localization, Markov Decision Processes (MDPs), MDRL, CNNS, and Proximal Policy Optimization (PPO), which is the used DRL algorithm in this work. It also includes a literature review on existing solutions for target localization, MDRL, Federated Reinforcement Learning (FRL), reward shaping, IL-assisted DRL, Machine Learning as a Service (MLaaS), and crowdsourcing for machine learning, which are important topics related to the proposed solutions. These serve as foundations for the understanding of this thesis, the used methods, as well as related works and their limitations.

2.1 Background

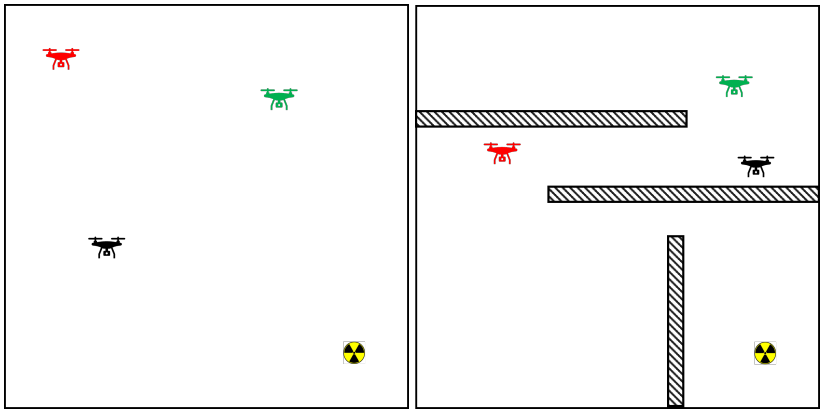
2.1.1 Target Search and Localization

The target search and localization problem has been previously defined and explained in Chapter 1. In this section, we give more elaboration on the different scenarios to be addressed in this thesis. As discussed earlier, the target localization problem is defined as follows: given N agents and a certain AoI within which a target is located, the agents

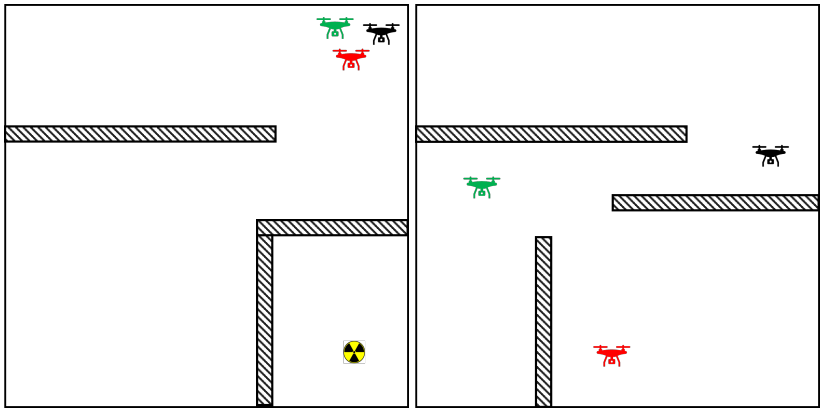
are to cooperate to localize this target. Two main goals are usually associated with such tasks: fast localization and low cost. Here, cost could be in terms of power consumption or incentives paid to individuals to carry out the tasks (as in the case of crowdsensing). At a given step, an agent observes its own location and reading, in addition to the location of other agents and the readings collected by them. In such an environment, the agents need to cooperate to localize the target and manage their resource, and this cooperation can be seen in two forms: 1) in case all agents initially have low readings, they are expected to split and give good coverage of the area, aiming to gather more informative readings, and 2) if some agents cannot contribute to the localization process (i.e. the agents are far from the target and hence keep getting very low readings), they are expected to become idle at some point to save resources, given that other agents have more informative readings. These cooperative behaviors are expected to be developed by the agents themselves, through rewarded interaction with the environment. The complexity of the problem significantly increases with the existence of the obstacles, which hinder the movement of the sensing agents and attenuate their readings. The existence of obstacles could also lead to scenarios of unreachable targets, requiring additional decisions about estimating its location. Additionally, in many scenarios of continuous surveillance, false alarms could arise about the existence of the target, which requires the agents to be able to quickly identify such alarms for better resource management. Figure 2.1 shows the different scenarios to be addressed by the sensing agents.

2.1.2 Markov Decision Process

An MDP is a mathematical framework used to model the decision-making in environments where outcomes are partly random and under the control of a certain decision-maker. It provides a formalism for modeling problems where an agent interacts with the environment in discrete time steps, making decisions that influence future states. An MDP is



(a) Simple environments **(b)** Complex environments (obstacles)



(c) Unreachable targets **(d)** False alarms

Figure 2.1: The different scenarios to be addressed by the agents.

defined by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, where:

- $\mathcal{S}$ represents the state space, which defines all the possible situations an agent can encounter in the environment.
- $\mathcal{A}$ is the action space, defining the set of actions available to the agent in each state.
- $\mathcal{P}(s^{'}|s, a)$ is the transition probability function that gives the probability of transitioning to a state $s^{'}$ given the current state s and the action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function that assigns a scalar reward to the agent based on the action a taken in a state s.

- $\gamma \in [0, 1]$ is the discount factor for the trade-off between instantaneous and future rewards.

Generally, an MDP exhibits the Markov Property, which implies that the future state depends only on the current state and the action taken, and not on the sequence of previous states and actions. The agent's goal in an MDP is to find a policy $\pi(a|s)$ that defines the probability of selecting action a in state s and that maximizes the expected cumulative return. This return is typically defined as the sum of discounted future rewards, given as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \tag{1}$$

2.1.3 Multi Agent Deep Reinforcement Learning (MDRL)

One way to generalize MDP to account for multiple agents is using Markov Games [30], which redefines the game-theoretic stochastic games in the RL context. In target localization problems, the state space of the environment is defined by the target location and the distribution of agents throughout the area, i.e. each target location combined with a distribution of agents gives a unique state. Since the target location is unknown, the problem is modeled as a Partially Observable Markov Game (POMG). POMG can be defined as a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{N} = 1, ..., N$ denotes the finite set of $N \geq 1$ agents;
- $\mathcal{S}$ denotes the finite set of states;
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N$ is the set of joint actions, where $\mathcal{A}_i$ denotes the finite set of actions available for agent i, and $\mathbf{a} = (a_1, ..., a_N)$ denotes a joint action;
- $\mathcal{O} = \mathcal{O}_1 \times \mathcal{O}_2 \times ... \times \mathcal{O}_N$ is the set of joint observations, where $\mathcal{O}_i$ denotes the finite set of observations for agent i, and $\mathbf{o} = (o_1, ..., o_N)$ denotes a joint observation;

- $\mathcal{P}$ is a set of Markovian state transition and observation probabilities, where $\mathcal{P}(s', \mathbf{o} \mid s, \mathbf{a})$ denotes the probability that taking joint action $\mathbf{a}$ in state s results in a transition to state s' and joint observation $\mathbf{o}$;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function for agent i;
- $\gamma \in [0, 1]$ is the discount factor for the trade-off between instantaneous and future rewards.

2.1.4 Convolutional Neural Networks (CNNs)

CNNs are a class of deep learning models designed to process and analyze grid data, such as images. The have proven to be successful in computer vision applications, especially for capturing spatial correlations in images, and became the cornerstone of computer vision tasks like image classification, object detection, and segmentation. Typically, a CNN consists of an encoder followed by fully connected layers. The encoder contains multiple convolution and pooling layers, which are responsible for the extraction of features and the dimensionality reduction of the data. The learned features are then fed into the fully connected layers which perform classification or regression tasks. One key advantage of CNNs is their ability to automatically learn relevant features without the need for manual feature engineering, making them highly effective in various vision-related applications, including medical imaging, autonomous driving, and facial recognition.

2.1.5 Proximal Policy Optimization (PPO)

The curse of dimensionality renders tabular RL methods, such as Q-Learning, infeasible when dealing with multi-agent systems. Alternatively, function approximators, such as deep neural networks, come as efficient tools to generalize from seen to unseen states, which has made them widely used recently in MDRL systems. Here, a policy π_θ,

parametrized by θ, is to be optimized with the objective of maximizing the cumulative reward throughout the episode. The policy maps the current state to a probability distribution over possible actions. Amongst the several RL techniques used to optimize the policy are Policy Gradient (PG) methods, which use the rewards in computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm.

In this work, we use Proximal Policy Optimization (PPO) [31]; a state-of-the-art policy gradient method for RL that uses the actor-critic structure. In this structure, the policy (actor) is used to select actions, while the estimated value function (critic) is used to criticize the actions made by the actor. PPO uses the current experiences, combined with the critique, and tries to take the biggest improvement step to update the current policy, without moving far from it. This tackles the issue of destructively large policy updates in traditional PG methods, which could cause considerable loss of performance. PPO alternates between sampling data through interaction with the environment, and optimizing a clipped policy surrogate, which is given as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t) \right] \tag{2}$$

where ε is a clipping hyperparameter. In this function, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of taking certain actions between the old and the current policy. $r_t(\theta)$ is greater than 1 if a_t is more likely to be taken in s_t after the latest policy update, and less than 1 if the opposite is true. $\hat{A}_t$ corresponds to an estimator of the advantage function; a function that measures how good a certain action is, given a certain state. In this work, we use Generalized Advantage Estimate (GAE) [32] to estimate $\hat{A}_t$. For each update, PPO uses a fixed length trajectory H, known as the horizon length, which runs the policy for H time steps to collect experiences. The estimator of the advantage function is then given as:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{H} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_{t+l} := r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l}) \tag{3}$$

where δ_{t+l} is the Temporal Difference (TD) residual, γ,$\lambda \in [0, 1]$ are discount factors, and $V(s_t)$ is the value predicted by the value function network (critic). The original work in [32] uses a summation that goes to ∞ in Eq. 3, however this can be replaced with H instead since the PPO update occurs after H time steps.

In Eq. 2, the first term in the min $(r_t(\theta)\hat{A}_t)$ pushes the policy towards actions that yield high positive advantage, while the second term $(\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)$ clips $r_t(\theta)$, which removes the incentive of moving outside the interval $[1-\varepsilon, 1+\varepsilon]$. The minimum is taken over both terms, resulting in the final objective being a lower bound on the unclipped objective. This helps in improving the current policy without moving far from it.

The objective function in Eq. 2 is combined with two other terms, $L_t^{VF}(\theta)$ and $S[\pi_\theta](s_t)$, to give the following final objective function that is maximized at each iteration:

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{4}$$

where c_1 and c_2 are coefficients. In this function, $L_t^{VF} = (V_\theta(s_t) - V_t^{\text{targ}})^2$ denotes the squared-error loss (value function error), while S denotes an entropy bonus that ensures sufficient exploration.

PPO is known for its simplicity, low computational complexity, and balance between sample efficiency and wall-time, especially when compared to other state-of-the-art methods such as Trust Region Policy Optimization (TRPO)[33].

2.2 Literature Review

In this section, we overview the existing literature work for (1) Target Localization, (2) MDRL, (3) Reward Shaping, (4) Imitation Learning (IL)-assisted DRL, (5) Federated Reinforcement Learning (FRL), (6) Machine Learning as a Service (MLaaS), and (7) Crowdsourcing for Machine Learning.

2.2.1 Target Localization

Existing works for target localization mainly fall into one of two categories: 1) mathematical data fusion models and 2) node placement/selection and path planning. In both categories, several approaches are presented with the aim of finding or estimating the target location as quick as possible, based on collected readings. Data fusion models propose mathematical solutions for integrating the data readings from multiple sources in order to reach a final estimate. The works in [10] and [11] proposed a data fusion algorithm based on Bayesian methods in which readings are iteratively fused to update the prior belief about the target location, while taking radiation localization as a running example. Other algorithms are based on the Time Difference of Arrival (TDoA) and Direction of Arrival (DoA), with exemplary scenarios of sound event localization [34, 35]. Maximum Likelihood Estimation (MLE) and Inverse Square Law are other mathematical methods used to fuse the data readings for the aim of localizing certain events [9, 36].

Works that fall in the second category are mainly concerned with the design of sensing systems that optimize resources while localizing the target. Such efforts focus on the optimized deployment of sensing nodes, or the selection of active nodes which collect readings for the localization process. The works in [11, 37, 38] consider the placement of stationary and mobile nodes, in a certain AoI, using greedy and genetic methods, aiming to minimize the localization time, minimize the number of deployed nodes, and maximize the localization accuracy. In [1, 13, 39], the authors propose data-driven mechanisms for selecting active nodes for localization tasks, while considering parameters such as the nodes' residual energy, data confidence, and area coverage. These works target either stationary sensing nodes or mobile sensing devices in crowdsensing systems. Other works build survey paths, which are either pre-defined or data-driven, that agents follow to find the target. The traditional uniform search method [40] is extrapolated in many works due

to its implementation simplicity, where a uniform survey path of different forms is predefined. The work in [3] proposed circular path planning strategies utilizing the directional characteristics of sensors. A data-driven surveying approach, using Bayesian methods to build probability density functions (PDFs) about the target location, is used in [15]. Local PDFs at the individual agents are shared to obtain a global PDF, which is then used in an optimization objective that decides what actions to take, aiming to find the target while balancing resource consumption between agents.

While these works prove efficient at providing estimates for the target location, while minimizing time and error and optimizing resources, they lack flexibility in adapting to different environments without the need of significant supervision. RL comes as an alternative with increased adaptability, where with the right learning algorithm, agents can learn to localize the target in varying environments. The work in [2] introduced the use of RL in tackling the problem of target localization, specifically in radioactive environments, for single agent systems. The proposed algorithm utilizes Double Deep Q-learning (DDQN) with a CNN to train a sensing agent to navigate in a 10m x 10m area looking for the radiation source. While the work is the first of its kind, it faces scalability issues, as the algorithm cannot be adapted for multi-agents, and the training process would be very complex for big inputs.

2.2.2 Multi-Agent Deep Reinforcement Learning (MDRL)

Following the recent advances in deep learning and its integration in RL, MARL solutions re-emerged, as MDRL, after being idle for years due to the increasing complexity of their state and action spaces [27]. Since then, several works showed promising results using MDRL for different complex applications, such as games and autonomous driving. In OpenAI's Hide & Seek problem [41], agents in two opposing teams are trained to maximize

their own team's reward, using Recurrent Neural Networks (RNNs), which are updated using PPO, an actor-critic method. Through the use of self-play, agents on one team develop policies, which are later countered by the other team with new policies, and so on. The work shows cooperation between agents, where hiders split and collect objects to block routes that lead to their whereabouts. Another similar approach by OpenAI is used to train a team of five agents playing the Dota 2 game, which was able to beat a team of professionals [23]. Another work in [24] proposes a framework for autonomous driving using Deep Q-Learning with RNNs and CNNs, in which agents displayed successful behavior in staying in lane. In [42], the authors propose a traffic control model using cooperative multi-agent reinforcement learning for the optimization of traffic systems. The proposed model shows great improvements to traffic control, and it proves it can realize real-time dynamic traffic control. The authors in [43] propose a multi-agent coordination framework based on deep reinforcement learning for traffic signal control. A spatial differentiation is designed for cooperation, where temporal-spatial information in the Q-learning replay buffer is used to amend the reward for each action. The problem of multi-agent path finding is addressed in [17] using a combination of multi-agent reinforcement learning and imitation learning. In this work, agents (robots) plan paths in a partially observable world while exhibiting coordination. Once a policy is learned, it can be copied onto any number of agents, showing scalability with team size.

2.2.3 Reward Shaping

In DRL, a sparse reward is a case where the environment rarely produces a useful reward signal. Sparse rewards are the easiest and most common form of rewards, as the desired goals in most applications naturally induce a sparse reward, such as achieving a checkmate in chess. However, due to the complexity of DRL problems, especially in MDRL, sparse rewards induce difficulty in learning, especially during the exploration stage

where the agents initially act randomly in the environment and barely collect rewards. Several works have introduced shaped reward functions, which distribute the reward over the course of the learning. One common form of shaped rewards are distance-based rewards, where agents get rewarded in each step of an episode if they get closer to achieving the goal. For example, in the problem of target localization [2, 44, 45], agents get rewarded at each step if they get closer to the target, where measures like Euclidean Distance, Manhattan Distance, or Breadth-First Search are used in each step to compute the distance. In [41], the authors tackle the problem of Hide-and-Seek, where a vision-based reward is designed that rewards seekers in each step if they keep hiders within their sight, and rewards hiders in each step otherwise. Other proposed reward shaping methods alter the original reward with values generated from a shaping function. The authors in [46] propose a scheme for reward shaping based on Graph Convolutional Recurrent Networks to predict and produce reward shaping values. Another work in [47] proposes a reward shaping method based on Lyapunov stability theory, which tempts the RL process into maximal reward region by driving the reward to make the Lyapunov function.

Despite the fact that reward shaping is seen to speed up the learning in DRL and MDRL, designing shaped rewards requires considerable engineering, and could still lead to local optima [48, 49]. Additionally, many of the shaped rewards or reward shaping methods are computationally expensive, such as search methods or graph neural networks, which adds additional overhead to the learning.

2.2.4 Imitation Learning-assisted RL

A recent alternative method to speed up the learning in RL, instead of reward shaping, is to combine RL with Imitation Learning (IL). In this approach, previously obtained experts (or expert demonstrations) are used partially to help in training new agents. Here, the demonstrations are used in a supervised learning method where the goal is to minimize the

loss between the agent's actions and the expert's demonstrations. During IL, the agents do not collect rewards, and the learning is entirely based on the expert's demonstrations which act as labeled data in supervised learning. In [48, 49], the authors propose methods that alternate between RL and IL for off-policy RL. In their case, the expert demonstrations are stored in a buffer, and for a portion of the RL period, the RL agent is trained with Behavioral Cloning (BC), where the aim is to exactly mimic the behavior of the expert with no reward feedback. The authors in [17, 50] extrapolate these works into MDRL, where agents alternate between MDRL and behavioral cloning (IL) from an expert.

The above works prove efficient in speeding up the learning, under the assumption that an expert model that is proficient with the environment exists. However, if there is a slight variance in the expertise of such an expert (i.e. the expert is familiar with a similar environment that is not exactly the same as the agent's environment), this introduces difficulties in the learning convergence. This is because, in behavioral cloning, the agents have no way of determining whether the expert demonstrations are good or bad, and are just tasked to mimic the expert's behavior into their own policies.

2.2.5 Federated Reinforcement Learning

Federated Reinforcement Learning (FRL) brings RL into the realm of FL. FRL aims to build a better policy from multiple RL agents without requiring them to share their raw experiences. Several works have adopted FRL, especially in the domain of MDRL, with the aim of increasing the sample efficiency of the training process, by aggregating models from different users trained on different experiences. In mobile edge computing, the authors in [51] propose a multi-agent framework for data offloading. The problem of data allocation is formulated as a multi-agent Markov Decision Process (MDP), and a joint cooperation algorithm that combines the edge federated model with the multi-agent RL is proposed. Another work in [52] proposes a FRL framework for collaboration among edge nodes to

exchange learning parameters, with the aim of better training and inference. FRL has also been combined with Blockchain, where the authors in [53] propose a framework that trains DRL models for computation offloading and resource allocation in 5G ultra-dense edge computing networks. The DRL models are trained in a distributed manner via a FL architecture, in which the communication is done securely over the Blockchain. In robotics applications, the authors in [54] propose a FRL architecture for cloud robotic technologies, in which a shared model on the cloud is upgraded with knowledge from different robots performing autonomous navigation. In autonomous driving, the authors in [55] propose an online FRL transfer process for real-time knowledge extraction, where agents take actions based on their own knowledge and the knowledge shared by others.

Despite the several advantages of FRL, it comes with several drawbacks. In a realistic scenario, if the models are trained in environments that inherit different dynamics, the learning convergence of the aggregated model could face issues [56]. Additionally, in most FRL frameworks, the global model is obtained by averaging the shared models, which requires all the models to have the same architecture in the case of neural networks. While this could be tackled by other additional steps, such as model compression (knowledge distillation), it introduces additional overhead and risk of losing information. Moreover, FRL is vulnerable to random failures or adversarial attacks, in which the shared models give harmful behavior that could affect the aggregated global model [57].

2.2.6 Machine Learning as a Service

The advent of MLaaS provided several platforms in the industry that allow users with varying skill sets to leverage its capabilities. Several MLaaS providers offer pre-built ML models and/or tools that enable MLaaS users to obtain ML models. Google Cloud is one platform providing users with services, such as AutoML and Vertex AI. AutoML provides a user-friendly interface for users with limited experience, where users can upload their

labeled datasets and the service takes care of model training and optimization. AutoML offers services for a varying set of tasks, including image recognition, natural language processing, and video analysis. Vertex AI, on the other hand, is an end-to-end ML platform that offers a set of tools for building and training ML models. Similar to AutoML, Amazon SageMaker and Microsoft Azure facilitates the end-to-end ML process by offering a range of tools that cater to different skill levels. Google Colab provides a cloud-based development environment for ML through a Jupyter Notebook interface. Colab is mainly known for providing access to powerful, but limited without paid subscription, GPUs and TPUs (Tensor Processing Units) for faster training. While SageMaker, Azure, and Colab can be used to train DRL systems, the higher complexities of DRL problems hinder their usability, especially for inexperienced users.

In terms of research, MLaaS attracted many scientists over the past few years. The proposal in [58] is amongst the first works addressing MLaaS, where the authors present an architectural design for a MLaaS platform. In the proposed design, the data is received and processed by a Data Gatherer, and then a model is built and trained by a Modeler Composite. The authors in [59] propose Acumos, a platform capable of packaging ML models into portable containerized microservices, which can be easily integrated into different business applications. The main aim is to reduce the technical burden on developers when applying ML models to their applications. The authors in [60] propose a privacy-preserving MLaaS on resource-constrained devices at the pervasive edge. The proposed framework uses methods such DNN splitting and quantization, enclave parallelization, and resource-aware offloading policies to protect clients' private data while using computing resources in the pervasive edge ecosystem. The authors in [61] propose a MLaaS framework that optimizes the allocation of limited ML resources by intelligently considering attributes such as service profile, region-wise resource usage patterns, and current ML resource usage. In [62], the authors propose Cashew, a service for ML data processing that caches common

input data pipelines across jobs from different clients to optimize training throughput.

The aforementioned platforms and research proposals prove efficient automating and increasing accessibility to ML. However, they cannot be adapted to DRL problems due to the higher complexity of such problems and the lack of human expertise to design and train DRL solutions.

2.2.7 Crowdsourcing for Machine Learning

The intersection between Crowdsourcing and ML is twofold: some works utilize ML and DRL in designing crowdsourcing systems [63, 64, 65], while other works crowdsource certain ML tasks [66, 67, 68, 69, 70]. We present an overview of the second set of works in the literature, since our work aims to crowdsource DRL tasks. Existing works using crowdsourcing for ML mainly use the crowd for data collection and processing. In [66], a dataset of breathing and coughing sounds that are collected via crowdsourcing is used for COVID-19 diagnosis. The authors in [67] propose a ML framework for flood forecasting systems using data collected through crowdsourcing, where workers report data about actual flooding incidents, such as rainfall intensity levels, continuing rainfall duration, and drainage ability. The authors in [68] develop a ML model to evaluate the performance of workers in categorizing neurotypical and autistic children. The data are collected through a crowdsourcing platform, where workers watch videos of children and fill out a series of questions about the child's behavior. In [69], the authors propose a collaborative crowdsourcing system for ML data labeling. In the proposed system, groups of workers collaborate in labeling data through three stages: Vote (choosing the label), Explain (reasoning behind the label), and Categorize (review other workers' explanations). The authors in [70] propose a crowdsourced annotation framework for data in sound event detection applications. The goal is to estimate strong labels, i.e. data labels with high confidence, using weak labels that go through methods such as majority voting.

Despite the numerous research put into crowdsourcing data-related tasks for ML, there are no proposals, to our knowledge, that crowdsource the ML (or DRL) training process. This work aims to propose a comprehensive framework that addresses this issue by utilizing the expertise of the crowd in training DRL solutions.

Chapter 3

Target Localization using Multi-Agent Deep Reinforcement Learning with Proximal Policy Optimization

3.1 Introduction

In this chapter, we tackle the target localization problem by designing novel MDRL models. In target localization tasks, agents equipped with sensing devices are deployed in an area of interest to identify the location of the target. The intended *cooperation* among such agents, along with the need for automated learning that is adaptable to different complex environments, motivates the use of MDRL. We first propose initial models that directly extend single agent RL into the multi-agent domains, and show their scalability issues. A final novel model is then proposed, based on the concept of centralized learning and decentralized execution, which is scalable in terms of the number of agents. Moreover, the proposed model is optimized with regard to the input size, where image processing techniques are used to efficiently downsize the input observations to focus on the important

features, resulting in a reduced number of trainable parameters. The proposed models utilize PPO to optimize actor and critic networks that are based on CNNs. In summary, the contributions of this chapter are as follows:

(1) The formulation and modeling of the multi-agent target localization problem in MDRL.

(2) The design of MDRL models using PPO and CNNs, in which agents show *cooperation* in localizing the target, achieved through a team-based reward function.

(3) The design of a distributed target localization approach, which ensures scalability, through the use of centralized learning with distributed execution, and the use of downsized observations.

The proposed approach is tested and evaluated for the scenario of radioactive target localization. The environment is modeled using radiation physics, where the agents are to localize a radioactive source of a given intensity. The proposed models are analyzed in terms of the learning performance, showing scalability and adaptability to variations of the environment, with a human-like performance developed by the localization agents in cooperating to find the target. The performance of the resultant agents is compared against existing localization benchmarks, namely Bayesian-based survey methods [15], uniform survey methods [40], and single-agent RL-based target localization using Double Deep Q-Networks [2]. The final proposed model shows better performance and resources management, proving the efficacy of the achieved cooperation between the agents.

3.2 MDRL formulation for Target Localization

Following the MDRL formulation presented in Section 2.1.3, and for the target localization problem, the game unfolds over a finite sequence of steps, i.e. the game has a finite horizon. At every step, each agent i analyzes its observation $o_i \in \mathcal{O}_i$ and takes action

$a_i \in \mathcal{A}_i$ based on a policy $\pi_i : \mathcal{O}_i \times \mathcal{A}_i \rightarrow [0, 1]$ and receives a reward r_i. When looking at the full picture, all agents simultaneously select an action $\mathbf{a} = (a_1, ..., a_N)$ and receive a reward and observation $\mathbf{o} = (o_1, ..., o_N)$. The objective for each agent is to maximize the expected sum of rewards it receives during the game.

3.3 Proposed Approach

This section first presents the initial proposed approach, and then discusses improvements in an incremental manner.

Localizing the target is a challenging task, which becomes more complex to model when cooperation between agents is expected. To push the agents to cooperate, we propose a *team-based* (joint) reward [41, 71]. Following the individual actions taken by the agents, the environment releases a shared reward to all agents based on the combined/joint actions. This helps the agents realize the advantage of taking actions that benefit the team. This is due to the fact that, with a team-based reward, each agent's reward is affected by the actions of other agents, and hence an agent learns to adapt its actions according to the environment and the behaviors of other agents, resulting in the desired cooperative behavior. Given a joint action $\mathbf{a}$ at step t, the team reward at that step is given as:

$$R_t = \begin{cases} 1 - b & \text{if } \min(D_t) < \min(D_{t-1}) \\ -1 - b & \text{otherwise} \end{cases} \tag{5}$$

where b is the number of agents who took a moving action, i.e. did not stay idle, and D is the set of distances between the agents and the target. The first term of the equation above, i.e. $+1$ or -1, depends on whether the team has moved towards the target (+1) or not (-1). The team is considered to have moved closer to the target if the closest agent in step $t-1$ makes a move towards the target in step t, i.e. when $\min(D_t) < \min(D_{t-1})$. The term

($-b$) in the equation above represents the cost of movement. This reward representation is a form of *shaped rewards* (i.e. the reward is not sparse) [72]. Agents get feedback/rewards throughout the episode steps, and not only when the target is successfully localized, which helps in learning even if the episode terminates without finding the target. The maximum possible reward in a step is 0, which corresponds to the case where a single agent moves towards the target while all other agents are idle. The maximum reward is not expected, mainly because agents need initially to take costly "exploration" steps to gather data that help decide consequent actions. Agents also are not motivated to collectively stay idle, since this results in accumulating a -1 reward throughout the entire episode. Hence, agents are motivated to finish the localization episode as fast as possible, to accumulate less negative reward. Generally, this reward function incentivizes agents to find the target as soon as possible, while also incentivizing cooperation in managing resources. The cooperative behavior developed due to this reward function will be further analyzed and discussed in Section 3.7.5.

For the agents to make beneficial decisions, it is important to store previous observations. For this reason, the agents' observations are modeled as 2D maps containing information related to the readings collected and the agents' locations. This helps in accumulating information, if needed, in the same 2D maps without the need of changing their sizes. Additionally, observations like readings and agents' locations are spatially correlated in localization tasks, and hence can be maintained and analyzed through 2D maps. Furthermore, 2D representations are more scalable in comparison with 1D representations, while increasing the number of agents. 1D representations would require increased policy input size with the increase of the number of agents, which may require modifying the policy network architecture or its hyperparameters. Alternatively, the input size could be maintained using 2D representations, and this will be shown later in Section 3.6. For these reasons, CNNs are used for the actor and critic networks. More details regarding the structure of

these networks and the observations will be discussed in the following sections. Thus, three models are proposed in this section, where each incrementally improves the previous one. These models are based on the actor-critic structure, using CNNs, and optimized by PPO. The three models are as follows:

(1) Centralized Multi-Agent Target Localization (CMTL):

This model directly extends single-agent RL into multi-agent domains, where multiple agents are treated as one big agent with complex combinations of actions.

(2) Distributed Multi-Agent Target Localization (DMTL):

This model uses the experiences of multiple agents to model the behavior of a single agent around other agents. In other words, this teaches an agent how to act based on its observations, which include the sensor readings and the locations of other agents.

(3) Optimized DMTL (ODMTL):

This model extends DMTL by optimizing the agents' observations and focusing on the important features, using image processing techniques.

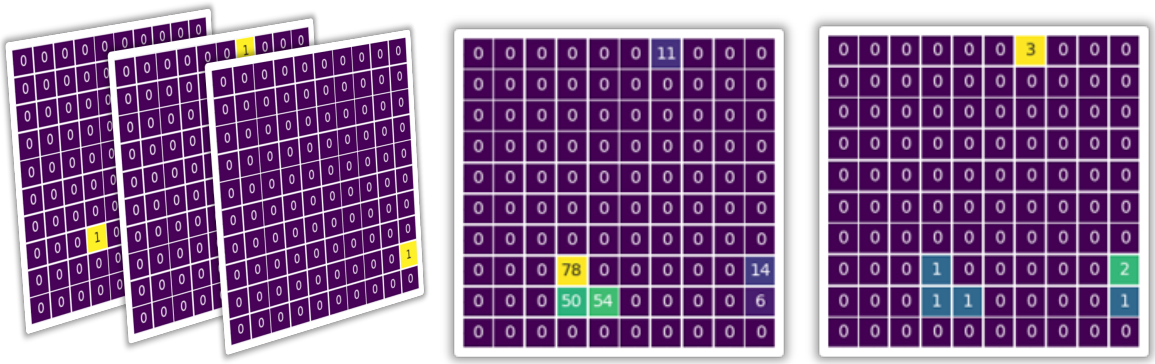
3.4 Centralized Multi-Agent Target Localization (CMTL)

In this model, the multi-agent environment is treated as a single-agent one, and learning happens in the joint observation-action space. This model extrapolates *centralized learning centralized execution* methods, sometimes referred to as the Joint Actor Critic (JAC) methods [73]. This method is a straightforward adaptation of single-agent algorithms in multi-agent settings, and it completely avoids the non-stationarity issue. This is because a centralized actor learns a policy that controls all agents.

3.4.1 Observation Space

The centralized actor network here takes as input the combined observations from all agents, and gives a joint action. In this work, observations are modeled as a stack of 2D maps of size $m \times n$, each representing an observation. Here, $m \times n$ represents the size of the AoI in terms of number of grid elements. When mapped to a real-life application, these dimensions could represent any kind of measurements, i.e. meters or kilometers, depending on the application. Given N agents, the actor network here takes 2+N observations, Fig. 3.1 illustrates all the input observations for the case of 3 agents, which are:

- Location Maps (Fig. 3.1a): N maps reflecting agents' locations.
- Readings Map (Fig. 3.1b): a 2D map showing the last reading collected in each grid element. Grid elements that have not been visited are given a reading of 0.
- Visit Counts Map (Fig. 3.1c): a 2D matrix showing the number of visits each grid element has received from all agents combined.



(a) Location Maps **(b)** Readings Map **(c)** Visit Counts

Figure 3.1: An example of the observation set for the case of 3 agents under the CMTL model, consisting of 5 (2+N) observations.

This way of modeling observations helps preserve information from previous experiences. The readings map helps in guiding agents towards the target location, where the

agent is expected to learn to follow high sensor readings to reach to the target. The visit counts map is important for exploration, especially in cases where collected readings are very low (agents are very far from the target). In such cases, keeping track of previously visited locations would help agents learn to explore unvisited areas for better information gathering. It is worth mentioning that each of these observations is normalized before being fed to the CNN.

3.4.2 CNN architecture and learning process

As discussed earlier, CNNs are used for the actor and critic networks. For the actor network, at each time step, the 2+N joint observations of all agents are fed to a CNN that gives 5^N outputs, each corresponding to a possible combination of actions. A Softmax function is used at the last layer so that the outputs correspond to a probability distribution over the possible action combinations. For example, in the case of 2 agents, the network takes 4 observations at each step and outputs 25 values that add up to 1, each corresponding to a combination of actions, given that each agent has 5 possible actions (move in one of the 4 directions or stay idle). The architecture of the actor network is shown in Fig. 3.2. This architecture is similar to the LeNet-5 architecture [74] which is widely used in image analysis tasks. The critic network has a similar structure, with only one output representing the state value, which is used in computing the advantage function (Eq. 3).

The training process for the CMTL model using PPO is summarized in Algorithm 3.1. Each episode starts with a reset that returns the initial observations ($\mathbf{o}^0 = (o_1^0, ..., o_N^0)$). Given these observations, the centralized actor network gives a set of joint actions ($\mathbf{a}^j = (a_1^j, ..., a_N^j)$). The agents act in the environment, which returns a new set of observations ($\mathbf{o}^{j+1}$), a reward value (r^j), and a termination flag (d^j) indicating whether the episode is finished or not. These items are stored in their corresponding buffers. After a certain number of steps (H), the buffered observations and rewards are used to update the actor

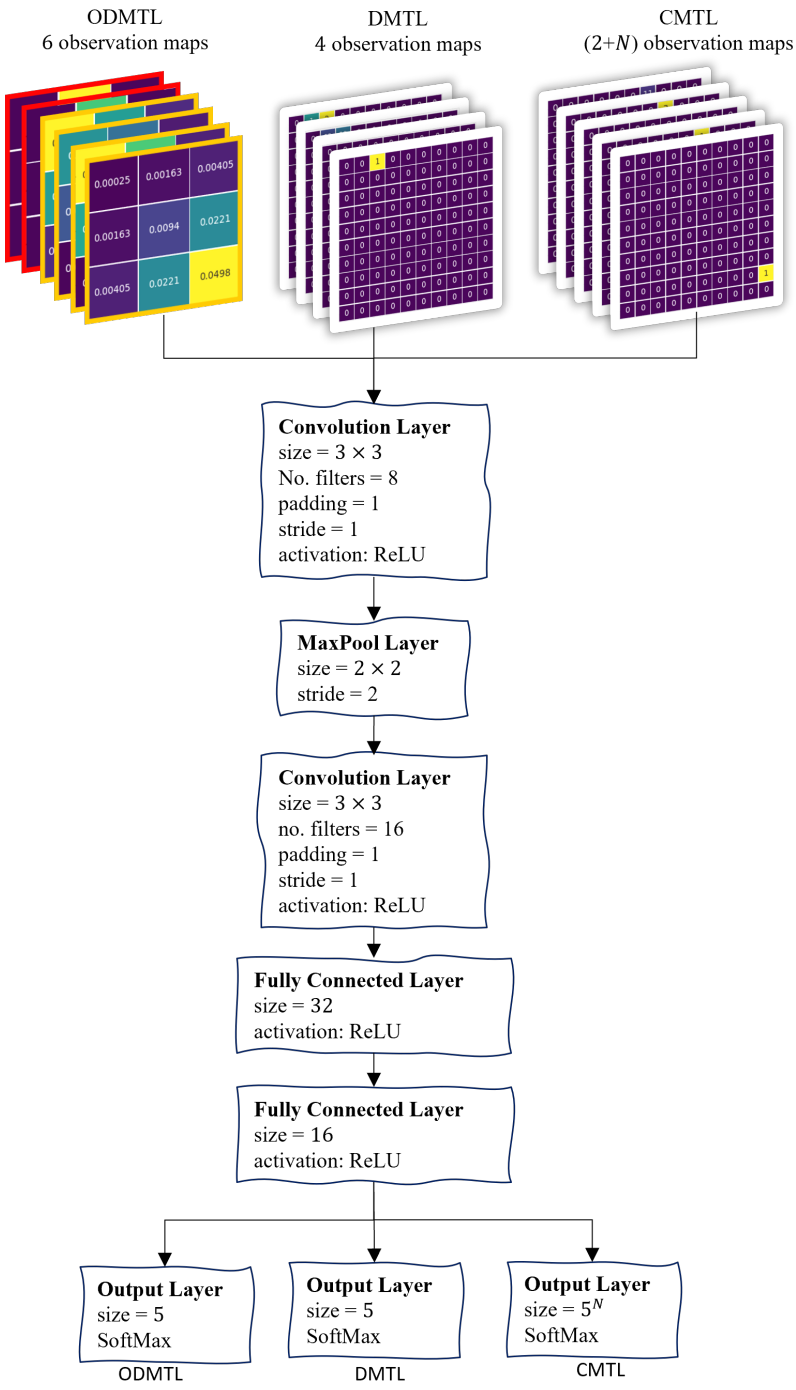


Figure 3.2: The architecture of the actor network (CNN) used in the proposed models. The parameters of the network are optimized using PPO.

and critic networks, using the objective function in Eq. 4. The inference of the CMTL model is shown in Algorithm 3.2, which only uses the actor network.

Algorithm 3.1: CMTL training with PPO

Input: Initial actor and critic networks

while Step $\leq$ MaxNumOfSteps **do**
  $\mathbf{o}^0$ = Env_Reset() #initial observations
  for i = 0, 1, 2, ..., Episode_Length:
    $\mathbf{a}^i$ = actor($\mathbf{o}^i$) #get actions
    $\mathbf{o}^{i+1}, r^i, d^i$ = Env_Step($\mathbf{a}^i$)
    Store $\mathbf{o}^{i+1}$, $\mathbf{a}^i$, and r^i in their corresponding buffers
    Step = Step + 1
    if Step % H == 0 **then**: #PPO update
      Compute the advantage estimate $\hat{A}$ using the rewards and the value function V (critic)
      Update the actor (policy) and critic networks using the loss $L^{CLIP+VF+S}(\theta)$
      Empty all buffers
    end if
    if $d^i == 1$ **then** break #if episode terminates
  end for
end while

Algorithm 3.2: CMTL Inference

Input: Trained actor network

$\mathbf{o}^0$ = Env_Reset() #initial observations
for i = 0, 1, 2, ..., Episode_Length:
  $\mathbf{a}^i$ = actor($\mathbf{o}^i$) #get actions
  $\mathbf{o}^{i+1}, r^i, d^i$ = Env_Step($\mathbf{a}^i$)
  if $d^i == 1$ **then** break #if episode terminates
end for

PPO trains a stochastic policy in an on-policy way. This means that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. During training, the choice of action through the actor policy is stochastic over the probability distribution. Over the course of training, the policy typically becomes progressively less random, as the PPO update encourages the network to exploit good actions. During execution, i.e. when the learning stops, the action selection is greedy, where the output with the maximum value is chosen.

As can be noticed, this way of modeling the multi-agent problem as a big single agent

has significant scalability issues during the training phase. The observation space increases linearly, and the action space increases exponentially, with the number of agents. Additionally, even during execution (after the learning is done), agents still receive actions from a central unit, i.e. the single actor network.

3.5 Distributed Multi-Agent Target Localization (DMTL)

To tackle the scalability limitations in the previous model, this section proposes a DMTL model, which is based on the *centralized learning distributed execution* (CLDE) approach [41]. In DMTL, the actor network is trained for a single agent, based on the experiences of all agents. In other words, the experiences of each individual agent are used to update a single network, which is then used by each agent to make decisions around other agents. This is viable since agents are cooperating and have the same goal. This method uses local observations for each agent to update the networks during training. A centralized critic network is used to avoid the non-stationarity issue, which is caused due to the environment being influenced by other agents from a single agent's perspective. Policy gradient methods in multi-agent settings are known to exhibit high variance gradient estimates, since an agent's reward depends on the actions of other agents. Hence, a centralized critic (value function) which has access to the observations and actions of all agents helps in tackling the variance issue [73, 75]. This is viable since the critic network is only used during training in the process of updating the actor network, and is omitted during execution. During execution, each agent gets a copy of the actor network, and they act independently in a distributed manner.

3.5.1 Observation Space

Similar to Model 1, observations are modeled as a stack of 2D maps of size $m \times n$, equivalent to the AoI size, each representing an observation. For a given agent i, the actor network takes 4 observations, illustrated in Fig. 3.3, which are:

- Location Map (Fig. 3.3a): a 2D matrix showing the agent i's location.
- Map of Other Locations (Fig. 3.3b): a 2D matrix showing the number of agents located in each grid element (excluding agent i).
- Readings map (Fig. 3.3c): a 2D matrix showing the last reading collected in each grid element. Grid elements that have not been visited are given a reading of 0.
- Visit Counts Map (Fig. 3.3d): a 2D matrix showing the number of visits each grid element has received from all agents combined.

As discussed in Section 1.1, it is assumed that agents can communicate, and hence experiences are shared to update the observation of each agent. However, each agent takes an action on its own. Given that an agent observes the locations of other agents, the desired outcome is for an agent to incorporate this information into making a decision that is beneficial to the team as a whole.

3.5.2 CNN architecture and learning process

The CNN architecture used for this model is shown in Fig. 3.2. It is similar to the one used for CMTL but with different input and output sizes. Here, the actor network takes 4 observations and gives 5 possible actions. Similar to what has been discussed in Section 3.4.2, action selection is stochastic during training, and greedy during execution. Each agent gets a copy of the latest network, upon which they act independently. The individual experiences, along with the team-based rewards obtained, are then used to update

(a) Location map

(b) Map of Other Locations

(c) Readings map

(d) Visit counts map

Figure 3.3: An example of the observation space for an agent in a team of 3 agents under the CLDE model.

the network using PPO. The centralized critic network takes the joint observations of all agents and outputs the state value. The joint observations here can be represented in 3 maps: 1) All Locations Map which shows the locations of all agents, 2) Readings Map, and 3) Visit Counts Map. These 3 maps summarize the observable state of the environment considering all agents.

The training and inference processes for the DMTL model are summarized in Algorithms 3.3 and 3.4. This process is similar to that for CMTL (Algorithms 3.1 and 3.2),

however the actor network (shown in Fig. 3.2) is called for each agent separately.

Algorithm 3.3: DMTL (and ODMTL) training using PPO

Input: Initial actor and critic networks

while Step $\leq$ MaxNumOfSteps **do**
  $\mathbf{o}^0$ = Env_Reset() #initial observations
  for i = 0, 1, 2, ..., Episode_Length:
    for j = 1, 2, ..., Team_Size:
      a^i_j = actor(o^i_j) #get action for agent j
    end for
    $\mathbf{a}^i$ = $[a^i_1, a^i_2, ...]$
    $\mathbf{o}^{i+1}, r^i, d^i$ = Env_Step($\mathbf{a}^i$)
    Store $\mathbf{o}^{i+1}$, $\mathbf{a}^i$, and r^i in their corresponding buffers
    Step = Step + 1
    if Step % H == 0 **then**: #PPO update
      Compute the advantage estimate $\hat{A}$ using the rewards and the value function V (critic)
      Update the actor (policy) and critic networks using the loss $L^{CLIP+VF+S}(\theta)$
      Empty all buffers
    end if
    if $d^i == 1$ **then** break #if episode terminates
  end for
end while

Algorithm 3.4: DMTL and OMDTL Inference

Input: Trained actor network

$\mathbf{o}^0$ = Env_Reset() #initial observations
for i = 0, 1, 2, ..., Episode_Length:
  for j = 1, 2, ..., Team_Size:
    a^i_j = actor(o^i_j) #get action for agent j
  end for
  $\mathbf{a}^i$ = $[a^i_1, a^i_2, ...]$
  $\mathbf{o}^{i+1}, r^i, d^i$ = Env_Step($\mathbf{a}^i$)
  if $d^i == 1$ **then** break #if episode terminates
end for

During execution, agents act in a fully decentralized manner, as each agent acts based on its copy of the actor network combined with the observations. Unlike the case in CMTL, for a given agent, the actor network takes the observations and gives only one of the 5 actions.

The cooperation is induced here through considering other agents and their readings in the agent's observations. As a result, this model can be scaled up to accommodate many agents. This model, however, has a limitation when it comes to the size of each observation, as increasing the observation size (i.e. increasing $m \times n$) would increase the learning complexity. The next model tackles this issue as discussed next.

3.6 Optimized DMTL (ODMTL)

The previous model helps in scaling up the solution to accommodate multiple agents. However, as the size of each observation increases, the model's complexity (number of trainable parameters) increases. To tackle this, ODMTL aims at reducing the input dimensionality, while maintaining the important features of the observations.

3.6.1 Observation Space

In this model, the observations an agent collects are divided into two sets: Local and Global, as shown in Fig. 3.4. Local observations help an agent make a local decision based on the area surrounding them. This is equivalent to a real-life scenario of a person carrying a detector and adjusting their actions based on the immediate readings he/she obtains. Global observations help in cooperation between agents, as they give a summarized view of the entire AoI. In this model, given an AoI that is gridded into $m \times n$ grid elements, observation maps are resized to $w \times w$, where $m, n > w > 1$ and w is an odd number. w here is a hyperparameter, where smaller values result in less network parameters (and hence reduced learning complexity), but also lead to higher loss of information, when compared to higher values. The aim here is to capture the essential features of the observations in the resized version, hence reducing the learning required by the network. To achieve this, the ***local observations*** at step t are given as:

- Windowed Location Map (Fig. 3.4a): a 2D matrix showing the agents' location in a $w \times w$ window centered around the agent's location in step $t - \frac{w-1}{2}$. This map is used to correlate the agent's location with respect to the windowed readings map.

- Windowed Readings Map (Fig. 3.4c): a 2D matrix showing the last reading collected in each grid element in a $w \times w$ window centered around the agent's location in step $t - \frac{w-1}{2}$.

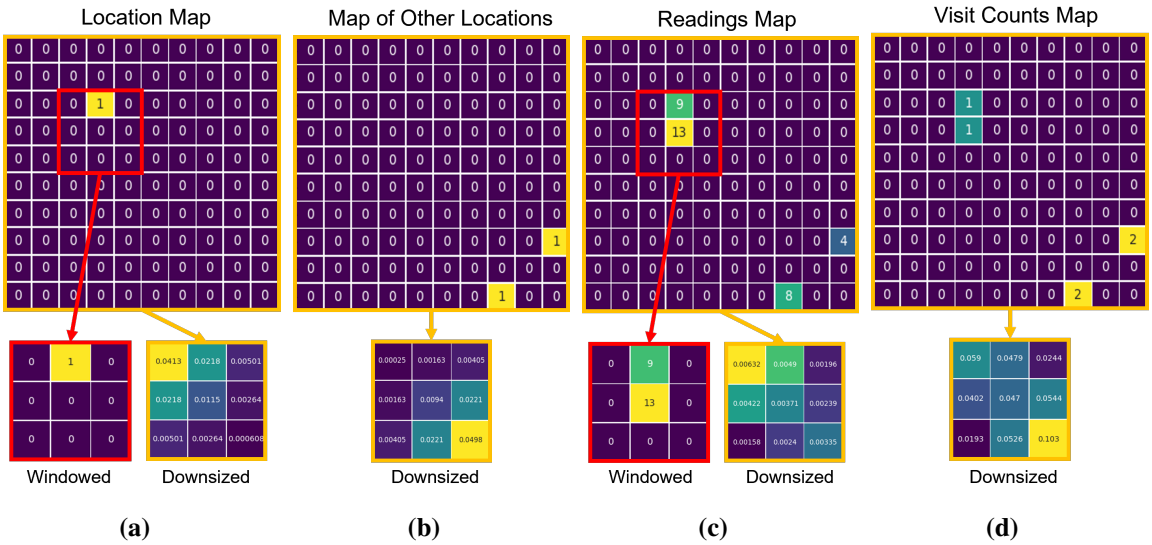


Figure 3.4: An example of the observation space for the case of 3 agents. Local observations are highlighted in red, while global observations are highlighted in yellow.

The windows here are centered around the agent's location in step $t - \frac{w-1}{2}$ to capture as many of the previously collected readings as possible. Specifically, this ensures that at least the readings collected in the interval $[t - w - 1, t]$, i.e. the last w readings, are captured in the window. For example, Fig. 3.5 shows the readings collected over 6 time steps, where the agent's current location is at time step t. If a window of size $w = 3$ is centered at the agent's current location at time step t (red window), only the last 2 readings can be captured (readings in in the interval $[t - 1, t]$). However, if the window is centered at the location in step $t - 1$ (blue window), the last 3 readings can be captured (readings in the interval

$[t-2, t]$). This is sufficient for an agent to make a local decision based on the readings in the area surrounding them.

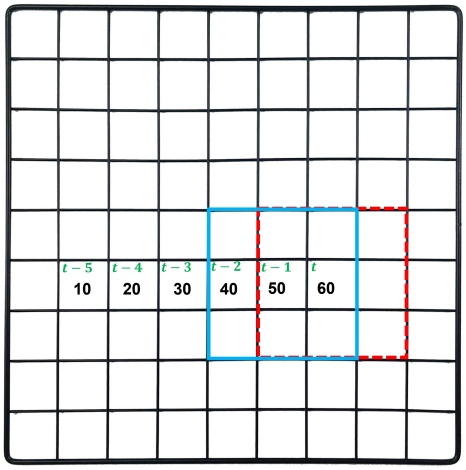


Figure 3.5: An example showing the different placement of the window.

On the other hand, the ***global observations*** are given as:

- Downsized Location Map (Fig. 3.4a): a 2D matrix of size $w \times w$, which is the result of downsampling the original location map of size $m \times n$.
- Downsized Map of Other Locations (Fig. 3.4b): a 2D matrix of size $w \times w$, which is the result of downsampling the original map of other locations of size $m \times n$.
- Downsized Readings Map (Fig. 3.4c): a 2D matrix of size $w \times w$, which is the result of downsampling the original readings map of size $m \times n$.
- Downsized Visit Counts Map (Fig. 3.4d): a 2D matrix of size $w \times w$, which is the result of downsampling the original visit counts map of size $m \times n$.

The maps are downsampled using bi-linear interpolation. A Gaussian filter with standard deviation (σ) of 1 is used before downsampling the maps, which helps in reducing the loss of information when downsampling inputs with high frequency components. Generally, the minor loss of information in the downsized maps through downsampling is insignificant here, as long as the trends in the maps are preserved. This is mainly because

these maps are intended for the cooperation between agents, which only requires a summarized overview of the entire area.

With these modified inputs, the actor network takes 6 observations of size $w \times w$. Irrespective of the area size, w could have small odd values, which significantly reduces the amount of learning needed. The local observations preserve only the surrounding readings, and help correlate them with the agent's location in order to determine the direction of movement. The global observations preserve a summary of the distribution of other agents, the readings collected, and the exploration done throughout the AoI, with respect to the agent's location, which is sufficient for the purpose of cooperation. This model preserves the informative part of the full observations, which are sufficient for each agent to make a decision that helps the entire team.

3.6.2 CNN architecture and learning process

The CNN architecture used for this model is shown in Fig. 3.2. The actor network takes 6 input observations and gives 5 outputs, one for each action. Similar to the previous two models, action selection is stochastic during training, and greedy during execution. The critic network takes the joint observations of all agents and outputs the state value. The joint observations here can be represented in 3 maps: 1) Downsized Locations which reduces a map showing all agents locations, 2) Downsized Readings Map, and 3) a Downsized Visit Counts Map. These 3 maps summarize the observable state of the environment considering all agents.

The learning and inference processes for the ODMTL model is the same as described in Algorithms 3.3 and 3.4 for DMTL, with only the observations being optimized. It is important to note that using centralized learning with decentralized execution is essential in ODMTL for the reduced observations to show effectiveness. For example, if reduced

Table 3.1: Actor and critic networks architectures.

Layer	CMTL		DMTL		ODMTL	
	actor	**critic**	**actor**	**critic**	**actor**	**critic**
Input (observations)	$2 + N$ maps		4 maps	3 maps	6 maps	3 maps
Convolutional Layer	size = 3×3, No. filters = 8, padding = 1, stride = 1, activation = ReLU					
MaxPool Layer	size = 2×2, stride = 2					
Convolutional Layer	size = 3×3, No. filters = 16, padding = 1, stride = 1, activation = ReLU					
Fully Connected Layer	size = 32, activation = ReLU					
Fully Connected Layer	size = 16, activation = ReLU					
Output (actions)	5^N	1	5	1	5	1

observations were introduced in CMTL, the model would still suffer from scalability issues, since the observation/action spaces increase linearly/exponentially with the number of agents. Table 3.1 summarizes the architectures of the actor and critic networks for all three models.

3.7 Evaluation

In this section, several experiments are conducted to assess and evaluate the performance of the proposed approaches. The simulations for these experiments have been performed using an Intel E5-2650 v4 Broadwell workstation equipped with 128 GB, 800 GB SSD, and NVIDIA P100 Pascal GPU (16G HBM2 memory). The implementation of PPO is similar to [76], which has been modified to incorporate discrete actions, GAE, and Centralized Learning for Distributed Execution, when needed.

The performance of each of the proposed models is analyzed in terms of episodic cumulative reward, episode length, and cost. The reward reflects the team reward obtained in each episode. The episode length reflects the number of steps needed to finish the localization process. In a step, agents simultaneously act in the environment by taking one of the five actions: moving up, down, right, left, or staying idle. The cost is reflected in the total distance traveled by all agents, since a moving action by an agent is accompanied with consumption of resources. Further tests are performed on scenarios with varying agents' group size, input size, area size, and target strengths. A behavioral analysis is conducted in Section 3.7.5, discussing the different behaviors developed by the agents. Finally, the performance of the resultant agents is compared against existing localization benchmarks, such as Bayesian-based survey methods [15], uniform survey methods [40], and single-agent RL-based target localization using Double Deep Q-Networks (DDQN) [2].

3.7.1 Simulation Environment

For all the experiments, radioactive target localization is used as an environment, where the aim is to localize a radioactive target of strength 1×10^9 photons/minute, in an area of size 1km $\times$ 1km. The data readings are generated using the principles of radiation physics, as explained in [10, 11]. Assuming agents are carrying radiation detectors, the radiation readings at each detector follow a Poisson distribution that is given as [77]:

$$CPM_i \propto \frac{I}{d_i^2} \tag{6}$$

where CPM_i is the counts per minute at agent i's detector, I is the source intensity in photons per minute, and d_i is the distance between agent i and the source. The background radiation is ignored since it is negligible when compared to the radioactive target. Additionally, the background radiation is assumed to be uniform throughout a given area, and hence it would not affect the localization task which relies mainly on the difference between

readings at different locations [10, 11].

Although radiation localization is used as an example to evaluate the proposed models, the applicability of the proposed system is valid for any localization task, under the same assumptions of having a single target. This is due to the fact that a sensor reading in localization tasks is given as a function of the distance between the agent and the target; the closer the agent is to the target, the higher the reading. For example, in several environmental applications, this idea is given as the Inverse Square Law, which states that a physical quantity at a certain location is inversely proportional to the square of the distance between the source of that physical quantity and that location. This is used to represent quantities such as radiation, heat, and sound [36, 78]. In applications where sensing agents are equipped with cameras, images can also be translated into sensor values depending on the proximity of the agent to the target, if present in the field of view.

For all the following experiments, each of the models was trained for 10 million steps. In each training episode, the environment is reset with a randomized target and agents' locations. It is assumed here that an agent cannot start in the same location as the target or another agent. An episode terminates when the target is found, or when a limit of 100 timesteps is reached. After every 1000 training episodes, the average results of 100 testing episodes, where the agents act greedily based on the latest policy update, are recorded. The list of PPO hyperparameters used in training are shown in Table 3.2. The values for ε, c_2, γ, and λ are as suggested in the corresponding PPO work [31].

3.7.2 Cumulative Testing Rewards

Fig. 3.6 shows the cumulative reward obtained per episode while running the three models for the cases of 1-4 and 10 agents, in an AoI of size 1km$\times$1km, gridded into a 10$\times$10 grid (each grid element has a size of 100m$\times$100m). The ODMTL model is used here with $w = 3$. The cumulative reward in an episode represents the accumulation of the

Table 3.2: Hyperparameters used for PPO.

Hyperparameter	Value
Learning rate	3×10^{-4}
PPO clipping parameter ε	0.2
Entropy coefficient c_2	0.01
Discount factor γ	0.99
Discount factor λ	0.95
Timesteps per update (Horizon H)	4000
Number of epochs per update	50

reward at each time step, i.e. Eq. 5, throughout the episode. Table 3.3 summarizes the key points in Fig. 3.6 (convergence step and the max reward) for all team sizes given the 3 different models. For the case of one agent (Fig. 3.6a), it can be noticed that all models eventually achieve the same performance, with an average episodic reward of -3.5. This is expected since DMTL reduces to CMTL (by omitting the map of other locations). However, ODMTL is seen to converge faster than the other approaches. Specifically, ODMTL achieves $\sim$1.8 and $\sim$1.7 times faster convergence when compared to CMTL and DMTL, respectively. This is attributed to the reduction in observation sizes, which leads to less trainable parameters in the CNN networks, as seen in Table 3.4. For the case of a single agent, the networks in CMTL and DMTL have $\sim$3.51 times the number of trainable parameters in ODMTL. As the number of agents increases (Figs. 3.6b-3.6d), it can be noticed that the performance of CMTL significantly deteriorates, which due to the increase in dimensionality of the observation and action spaces, resulting in the exponential increase in trainable parameters, as seen in Table 3.4. For the cases of 3-4 agents, CMTL fails to localize the target, where most episodes terminate after reaching the limit. On the other hand, while DMTL and ODMTL both show convergence in all scenarios, ODMTL always has faster convergence, due to the reduced observations and the lower number of trainable parameters. Specifically, ODMTL is $\sim$25%, $\sim$12%, $\sim$41%, and $\sim$62% faster when compared to DMTL for team sizes of 2, 3, 4, and 10, respectively. However, despite the faster

convergence, ODMTL has slightly lower maximum reward, when compared to DMTL. On average, ODMTL achieves a maximum reward that is lower than DMTL by 0.7, 1.5, 1.8, and 0.7, for the cases of 2-4 and 10 team sizes. For example, for the case of 4 agents, DMTL eventually reaches an average reward of -1.9 per episode, while ODMTL reaches -3.7. This means that, while the agents in both cases learn to correctly localize the target, agents in ODMTL take nearly 2 additional exploration steps in the environment (in total). This is attributed to the minor loss of information resulting from the reduction of observations. In terms of localization success rate, both ODMTL and DMTL achieve a success rate of 100% in all scenarios (a-e), while CMTL achieves a success rate that drops from 100% to 45% as the team size goes from 1 to 4 agents. The success rate is computed by averaging the outcome of 1000 testing episodes conducted after the 10 million training steps, where success represents the case of localizing/finding the target within the 100 episodic steps.

Table 3.3: Summary of the key results in Fig 3.6. Convergence refers to the time step ($\times 10^6$) at which the max reward is first achieved.

Team Size	**Attribute**	**Model**		
		CMTL	**DMTL**	**ODMTL**
1	Convergence	5	4.5	1.8
	Max Reward	-3.5		
2	Convergence	10	6	4.8
	Max Reward	-12	-3.1	-3.8
3	Convergence	N/A	5.5	4.5
	Max Reward		-2.3	-3.8
4	Convergence		3.1	2.2
	Max Reward		-1.9	-3.7
10	Convergence		3.4	2.1
	Max Reward		-2.1	-2.8

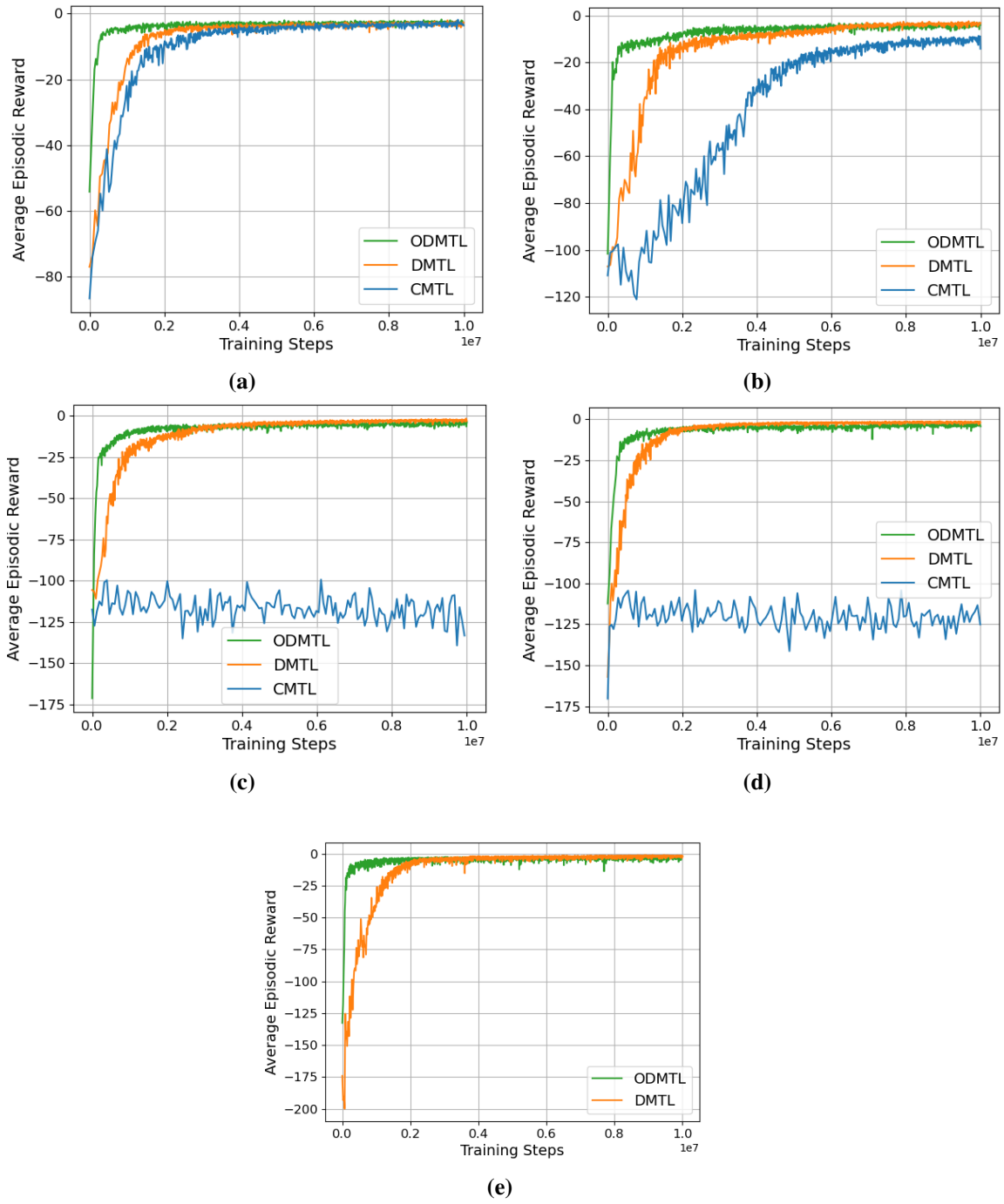


Figure 3.6: The average episodic reward for the three models, for a system of (a) one agent, (b) two agents, (c) three agents, (d) four agents, and (e) ten agents.

Table 3.4: Number of trainable parameters for each model for different team sizes. DMTL and OMDTL have slightly smaller values for team size = 1 compared to other team sizes because the Map of Other Locations is ignored.

Model	Team Size				
	1	**2**	**3**	**4**	**5**
CMTL	14837	15249	17021	25593	68165
DMTL	14837	14909			
ODMTL	4229	4301			

3.7.3 Episodic Length and Cost

The general aim of localization tasks is to achieve fast localization at low cost. Localization time is measured by the number of time steps it takes to localize the target, i.e. for an agent to step into the grid element that has the target. The cost represents the total number of movement steps in an episode. In a single time step, the number of movement steps could be as low as 0 (none of the agents has moved) or as high as number of agents (all agents have moved). As discussed in Section 3.7.1, each point in the following results is the average of 100 testing episodes conducted at the corresponding training step. Figs. 3.7 and 3.8, along with Table 3.5, compare the three models in terms of the localization time and cost per episode, respectively. In both cases, CMTL shows poor performance as the number of agents increases, due to its scalability issues. When comparing DMTL and ODMTL in terms of localization time, it can be seen that both have a similar performance, with an advantage for ODMTL in terms of convergence time, and a slight advantage for DMTL in terms of final localization time. On average, DMTL achieves $\sim$9% less localization time and $\sim$8% less cost after the training is concluded, when compared to ODMTL, while ODMTL achieves $\sim$34% faster convergence. The higher difference between DMTL and ODMTL in terms of cost, when compared to time, means that cooperation between agents in ODMTL is slightly affected by the loss of information in the reduced observations. Considering the case of 4 agents (Fig. 3.7d), for example, DMTL converges to an

average localization time of 4.5 steps, as opposed to 5.5 for ODMTL. This means that, on average, the agents in ODMTL take an additional exploration step (in total), when compared to DMTL. This is not significant, given that ODMTL reduces the learning required and converges faster. As for the cost (Fig. 3.8d) at step 1M, it can be seen that ODMTL has less than half the cost of DMTL. This is although both models converge in terms of localization time by the same step (Fig. 3.7d). This means that while agents in DMTL learn to localize the target by step 1M, cooperation between them is still not fully learned, unlike the case on ODMTL. DMTL eventually converges to a cost of 4.0, as opposed to 4.9 for ODMTL. More analysis on the cooperative behavior is to be discussed in Section 3.7.5.

Table 3.5: Summary of the key results in Fig. 3.7 and Fig. 3.8, representing the lowest time and cost achieved by each model, for each team size, on average. Time is given in (timesteps) and cost is given in (Moving steps).

Team Size	**Attribute**	**Model**		
		CMTL	**DMTL**	**ODMTL**
1	Min. Time	9.8	9.0	8.3
	Min. Cost	9.8	9.0	8.3
2	Min. Time	7.7	5.5	5.7
	Min. Cost	13.1	7.8	8.1
3	Min. Time	N/A	4.7	5.8
	Min. Cost		5.1	6.3
4	Min. Time		4.5	5.5
	Min. Cost		4.8	5.8
10	Min. Time		2.8	3.2
	Min. Cost		6.2	6.8

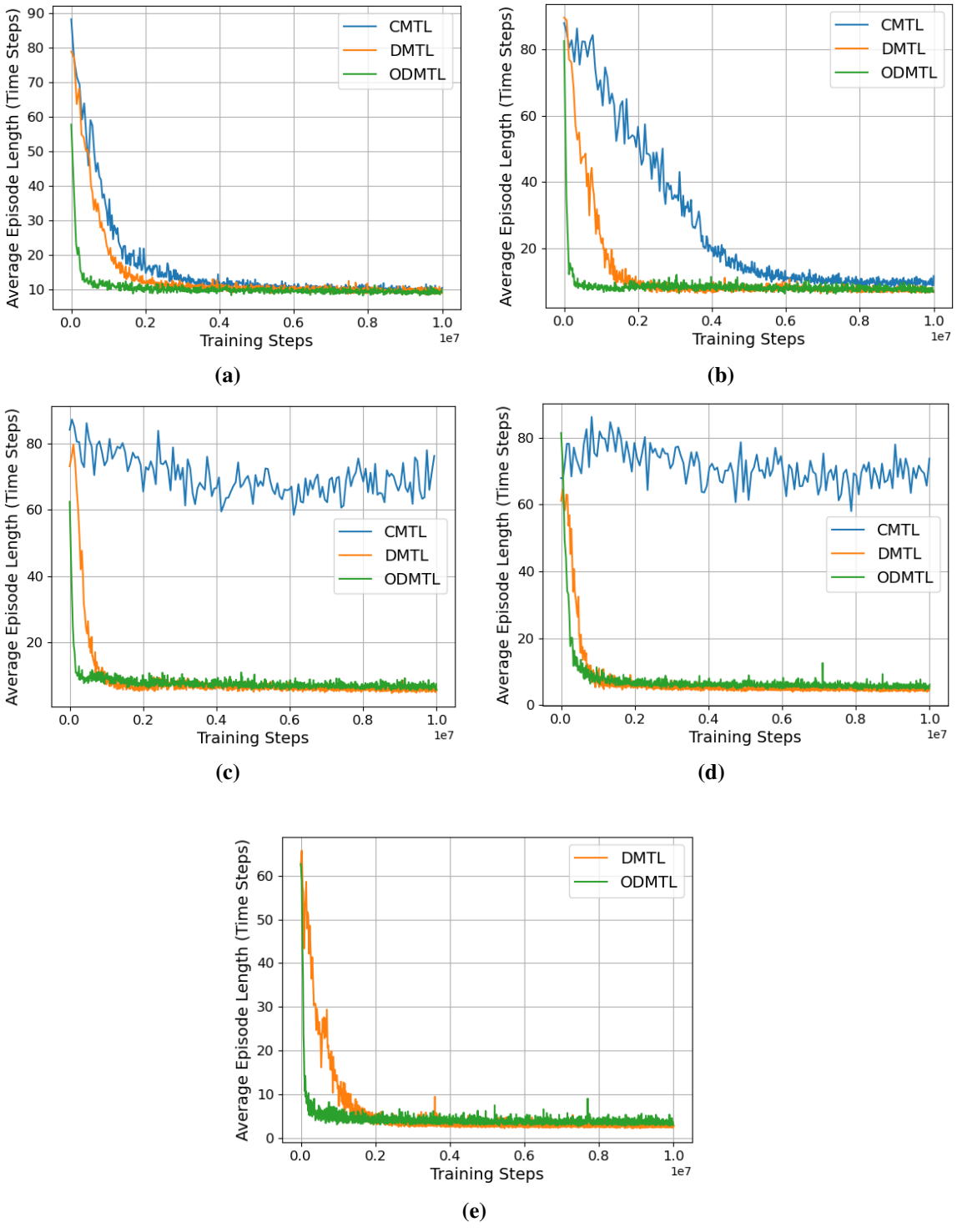
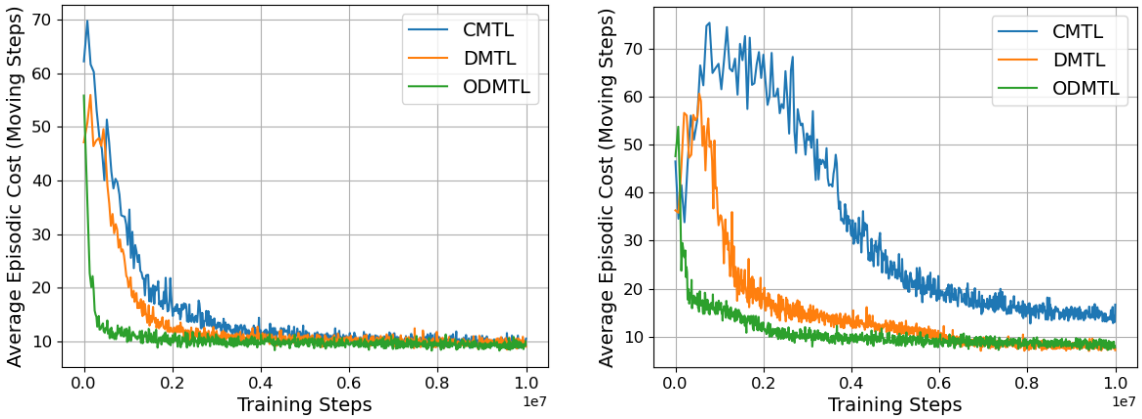
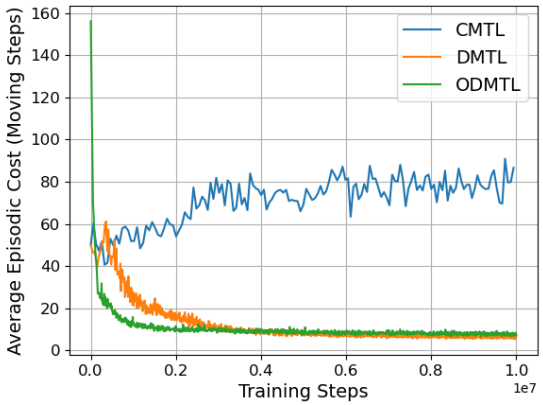


Figure 3.7: The localization time achieved, per episode, for a system of (a) one agent, (b) two agents, (c) three agents, (d) four agents, and (e) ten agents.

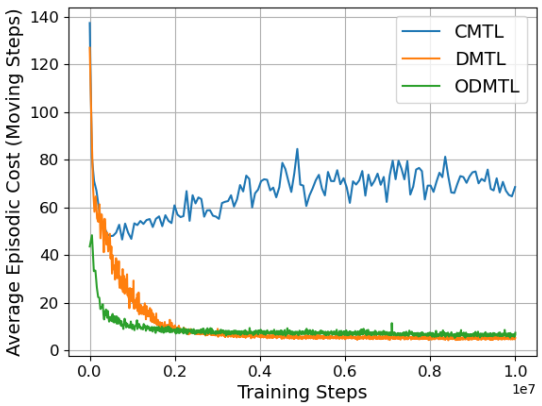


(a)

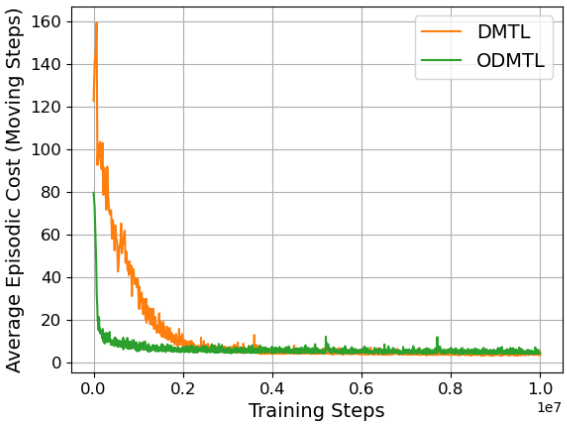
(b)



(c)



(d)



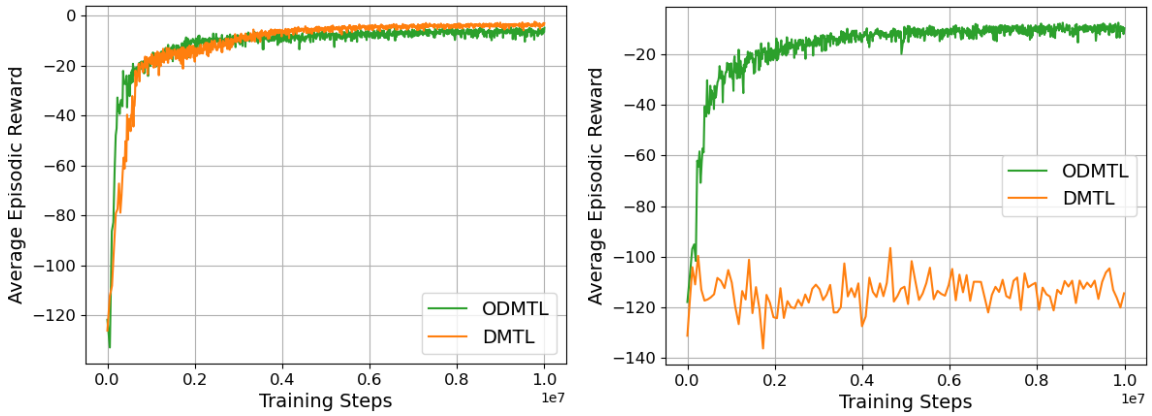
(e)

Figure 3.8: The cost of localization, for the three models, for a system of (a) one agent, (b) two agents, (c) three agents, (d) four agents, and (e) ten agents.

3.7.4 Varying Environments

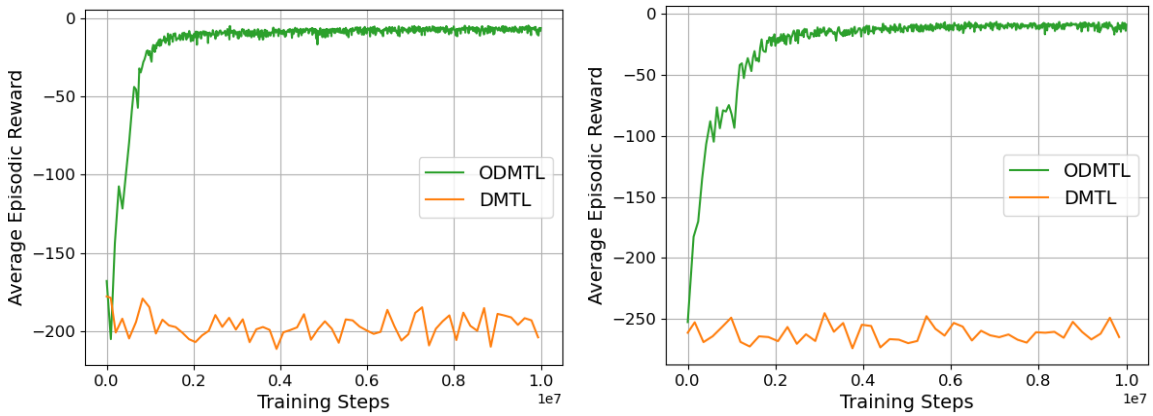
As discussed in Section 1.2, the ability to learn in different environments, without human supervision, motivates the use of MDRL. Here, this ability is tested while using the proposed MDRL models. For each of the following environment variations (different area/input sizes), the learning process is initiated and the actor/critic networks are retrained following the same MDRL models proposed in this work, i.e. DMTL and ODMTL. CMTL is not considered since it already displayed scalability issues, as seen in Figs. 3.6-3.8.

Fig. 3.9 shows the rewards obtained while using DMTL and ODMTL for 3 agents, as the area size is increased to 2km $\times$ 2km and 3km $\times$ 3km, while the gridding is kept at 10 $\times$ 10. Although both models displayed similar performance for an area size of 2km $\times$ 2km, DMTL's performance deteriorates as the area size is increased, while ODMTL maintains high performance. Since the localization becomes harder with increased area size, DMTL would need significantly more training to reach convergence, which is achieved early through ODMTL. In terms of localization success rate, ODMTL maintains a success rate of 100% in both scenarios, while DMTL achieves a success rate of 100% in Fig. 3.9a that drops to 24% in Fig. 3.9b. On the other hand, Fig. 3.10 shows the results as the grid size ($m \times n$) is increased to 13 $\times$ 13 and 15 $\times$ 15, while maintaining the area size at 1km $\times$ 1km (i.e. each grid element has a size of 80m$\times$80m and 67m$\times$67m, respectively). Increasing the grid size, while fixing the area, makes the localization process more detailed, i.e. agents make decisions more frequently. Since DMTL deals directly with the entire grid, increasing the size makes the learning more complex, as more network parameters need to be trained/learned. As a result, its performance is poor, as seen in Figs. 3.10a and 3.10b, which is unlike ODMTL that maintains high performance, since its input is not affected by the grid size due to observations reduction. In both scenarios, ODMTL achieves a localization success rate of 100%, while DMTL achieves a low rate of 22%, which emphasizes on the struggle DMTL faces as the localization problem gets harder.



(a) area = 2km×2km **(b)** area = 3km×3km

Figure 3.9: Testing the learning process of the MDRL models on environments with different area sizes.



(a) Grid size = 13×13 **(b)** Grid size = 15×15

Figure 3.10: Testing the learning process of the MDRL models on environments with different grid sizes.

In all the previous experiments, agents had to localize a radioactive target of intensity 1×10^9 photons/minute. In the following experiment, the agents learn to localize in an environment of varying target intensities. In each training episode, the environment resets with one of 3 target intensity values from $[1 \times 10^8, 5 \times 10^8, 1 \times 10^9]$ photons/minute, which correspond to targets of weak, medium, and high intensities, respectively. These three values are enough to represent the different patterns of data readings that could be obtained throughout the AoI. During inference/testing, the agents are placed in environments with a

target of intensity in the range $[1 \times 10^8, 1 \times 10^9]$, i.e. the target could have any value in this continuous uniform range. Fig. 3.11 shows the results of this experiment, for a group of 3 agents in a 1km$\times$1km area, gridded into 10$\times$10. Each point in this figure is the average of 100 testing episodes, where in each episode the target intensity is randomly set to a value in the range $[1 \times 10^8, 1 \times 10^9]$, which follows a uniform distribution. As can be seen, given 10 million training steps, DMTL struggles to converge, while ODMTL converges early with the same amount of experience. Additionally, ODMTL maintains a localization success rate of 100% after the learning is concluded, which is unlike the success rate of 76% achieved by DMTL. This is attributed to the reduction of observation dimensionality, which leads to faster learning. Thus, these experiments validate that ODMTL is easily adaptable to different environments. It also shows that the resultant agents can handle any target intensity given that they are trained on the three intensity types: low, medium, and high.

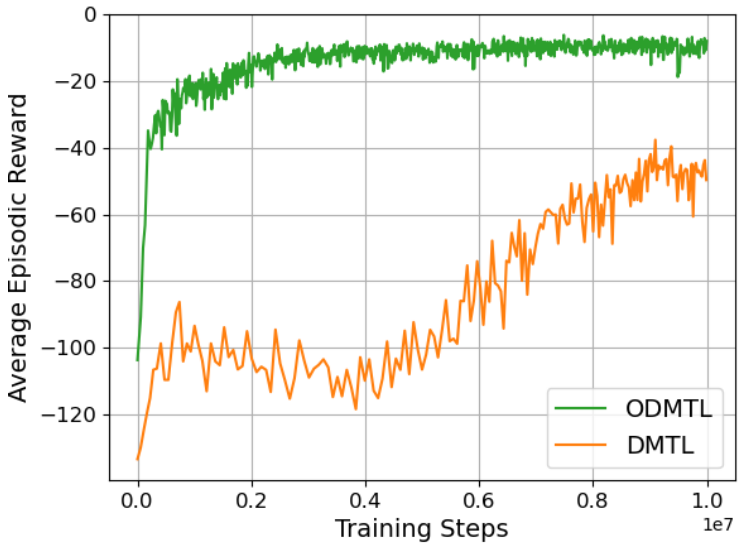


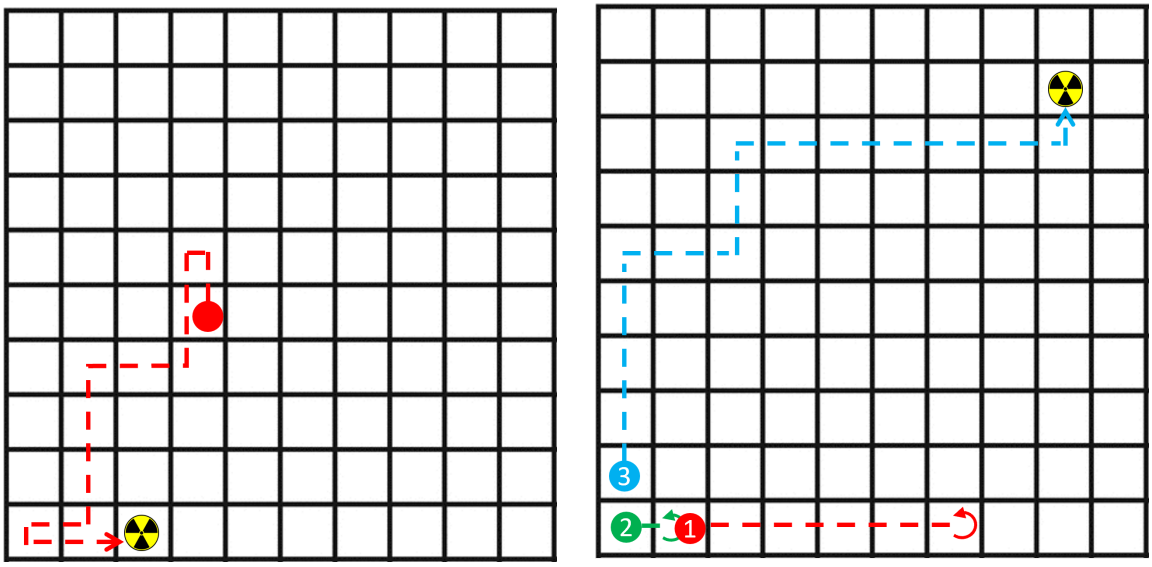
Figure 3.11: Testing the MDRL models on an environment with varying target intensities.

3.7.5 Behavioral Analysis

This section analyzes the different behaviors developed by the agents in the target localization problem. These behaviors mainly result from the nature of the observations fed to the actor network, in addition to the reward function. It is worth mentioning that in all proposed models, the agents develop the same behavior described below, given that the learning converges. Hence, we here report the behaviors obtained through the ODMTL approach.

Fig. 3.12 shows scenarios illustrating the behaviors developed by the agents, using the ODMTL model. In Fig. 3.12a, a human-like behavior by the agent is noticed. Initially, a single reading is not enough to determine which direction to move towards, and hence the agent explores by going up. With a second reading collected, the agent determines that the target is more likely downwards, and hence moves towards that direction. Similarly, another exploration step is taken when the agent is near the target, as the previously collected readings could indicate multiple possible target locations. On the other hand, 2 different behaviors are developed for the case of multi-agents, in which cooperation can be noticed, as seen in Fig. 3.12b. The first behavior can be seen initially, where all agents have very low readings and cannot tell which direction to move towards. Hence, they decide to split to cover a bigger area. The second behavior is a conservative one, and can be seen in Agent 2's actions, as it sees that the other two agents are sufficient to cover the area, and hence decides to stay idle after one exploration step. This behavior can also be seen as agent 1 decides to stop after few exploration steps, since it could be noticed that agent 3 is getting increasingly higher readings, and hence is getting closer to the target. It is also worth mentioning that there are rare cases where multiple agents end up in the same grid. In such scenarios, cooperation is hard to achieve, mainly because each agent has the same copy of the actor network, and hence both would end up taking the same action. Nonetheless, it was noticed that in all such rare scenarios, the target is still localized (by both agents), but

cooperation is not fully achieved.



(a) (b)

Figure 3.12: Examples of the different behaviors developed by the agents for the cases of (a) a single agent and (b) multi agents.

Another interesting aspect is related to the different stages during the learning process at which the agents develop the aforementioned behaviors. When studying the progression of behavior throughout the learning process, it was noticed that agents initially learn to localize, without focusing on cooperation. For example, when considering Fig. 3.6c, it is noticed that agents develop the localization behavior nearly by step 1M with minimal cooperation, where agents sometimes could take unnecessary actions. The cooperative behavior is noticed to be developed later, which explains the slight increase in the reward afterwards.

3.7.6 Benchmarks: Localization Methods

In this section, the performance of the localization agents obtained by ODMTL is benchmarked against several localization methods from the literature, which are given as:

(1) DDQN: a localization approach based on Double Deep Q-learning for a system with

a single agent [2], which is trained for 10 million steps.

(2) DDQN-2: an extrapolation of DDQN [2] in multi-agent settings, using centralized execution, which is trained for 10 million steps. The policy network here takes the observations of all agents, and gives a joint action vector.

(3) Bayesian: a multi-agent cooperative localization approach using a Bayesian approach [11, 15]. In [15], a Bayesian framework is used to build and update local probability density functions (PDFs) of the target location, which are then shared between the agents to obtain a global PDF. This is then used in an optimization objective that aims to find the target while balancing resources consumption between agents. We do not focus on the different types of resource consumption in this work, and assume each agent has 1 cost unit if decided to take a moving action.

(4) Uniform: a traditional target search method where each agent moves in a pre-defined path that uniformly covers the entire area [40]. Here, agents are placed in different starting positions, and each agent covers the area in a zigzag fashion.

For the ODMTL, DDQN-2, Bayesian, and Uniform approaches, a group of 3 agents is used for comparison. Each of the results is an average of 1000 testing episodes, where in each episode the target is placed in a random location. All the aforementioned benchmarks succeed in localizing the target in all scenarios, with differences in terms of localization time and cost.

Table 3.6 shows a comparison between single-agent DDQN and multi-agent ODMTL, in terms of localization time and total cost, for varying target strengths. The use of DDQN as a benchmark is to show the trade-off between localization time and the cost of using multiple agents instead of a single agent. The total cost here represents the total number of moving steps taken by the team of agents. For the single-agent DDQN, the localization time and cost are equal, since each time step corresponds to one moving action by the

agent. When compared to the single DDQN agent, it is expected that ODMTL agents perform faster localization, as seen in the table. However, what is worth highlighting is the fact that the 3-agent ODMTL has less cost when compared to the single-agent DDQN. Specifically, 3-agent ODMTL costs 7.9% less, on average, when compared to a single-agent DDQN. While ODMTL has triple the number of agents, the cooperation between agents in achieving fast localization and managing their resources leads to low cost in total, which motivates the use of multiple cooperative agents for such problems.

Table 3.6: Comparison between ODMTL and DDQN in terms of localization time (time steps) and cost (moving steps).

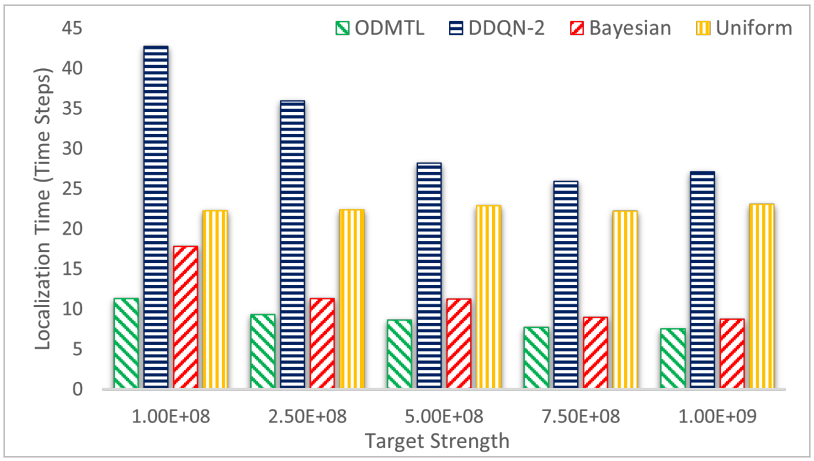
Attribute	**Model**	**Target Strength (photon/min)**				
		1×10^8	2.5×10^8	5×10^8	7.5×10^8	1×10^9
Loc. Time	ODMTL	11.27	9.27	8.61	7.71	7.54
	DDQN	14.12	12.3	11.52	9.31	9.58
Loc. Cost	ODMTL	13.3	10.99	9.76	9.38	8.78
	DDQN	14.12	12.3	11.52	9.31	9.58

Fig. 3.13 shows the results of comparing ODMTL to the multi-agent DDQN-2, Bayesian, and uniform approaches. When compared to DDQN-2, ODMTL achieves up to 2.9 times faster localization. When it comes to the cost, both models are on par. This shows that, given the same amount of experience for both models (10 million steps), the learning in DDQN-2 is incomplete, and agents prefer the idle action most of the time, hence resulting i slow localization and low cost. This is mainly because of the scalability issues faced in DDQN-2 due to the centralized execution. ODMTL agents also outperforms Bayesian-based agents, with up to 58% faster localization and up to 62% reduction in cost. This can be primarily attributed to two reasons. Firstly, the Bayesian approach heavily depends on the data readings collected by the agents, and hence struggles when such readings are insignificant. This can be seen for the case of the weakest target strength, where there is a noticeable difference between ODMTL and Bayesian, in terms of localization time and

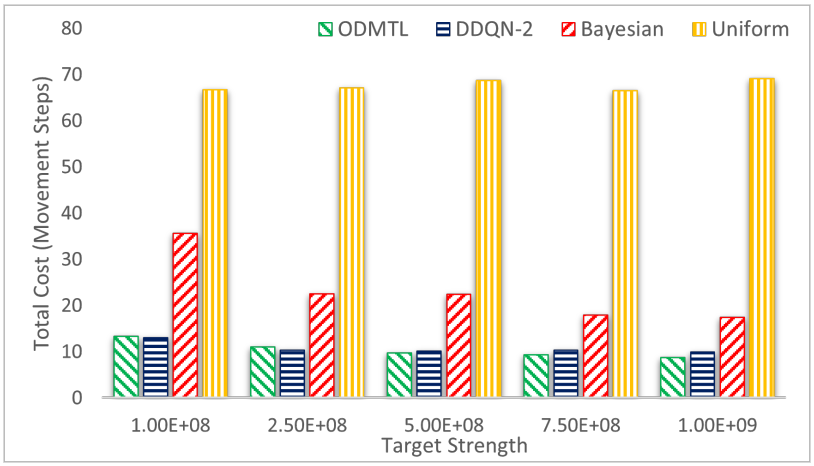
cost. Secondly, the Bayesian approach focuses on balancing resources between agents. As a result, some agents could be of use to the localization task, but decide not to move if the amount of resources they have is much less than the average of the team, and this affects the localization time. In terms of cost, while the Bayesian approach achieves balance between agents in terms of resource consumption, the total consumption is still high, due to the same aforementioned reason. The traditional uniform search approach is the simplest in terms of implementation, yet it has relatively the worst performance, which is attributed to the absence of cooperation between agents, and not using any data-driven methodologies to localize the target. ODMTL agents achieve up to 3 times faster localization with up to 87% reduction in cost, when compared to uniform methods.

3.8 Conclusion and Discussion

In this chapter, the problem of target localization is tackled using Multi-Agent Deep Reinforcement Learning (MDRL) models, where agents learn to cooperate to localize a target. Observations were modeled as stacks of 2D heatmaps, representing agents' locations and readings. Proximal Policy Optimization (PPO) was adapted to optimize an actor-critic structure, represented by a Convolutional Neural Network. Three models were designed and analyzed: 1) a centralized model (CMTL) which extrapolates single-agent deep RL into multi-agents settings, 2) a decentralized model (DMTL) where agents act independently during execution, and 3) an optimized decentralized model (ODMTL) which builds on DMTL by reducing the dimensionality of agents' observations to focus on important features. CMTL was shown to have scalability issues, where increasing the number of localization agents increases the dimensionality of the state and action spaces, and renders the model poor. DMTL was shown to tackle the scalability issues with CMTL, and was able to prove efficient in achieving cooperation between increasing number of agents



(a)



(b)

Figure 3.13: Comparison between the different benchmarks in terms of (a) localization time and (b) total cost, for varying target strength.

while localizing the target. ODMTL was shown to achieve faster learning and prove adaptable in varying localization environments. The agents were found to develop human-like behavior in cooperating to localize the target. When compared to a single-agent Double Deep Q-learning (DDQN) benchmark, ODMTL was found to achieve an average of 7.9% reduction in cost, which proves the efficiency of the cooperation between agents in managing their resources, while achieving fast localization. The localization agents obtained by ODMTL were also found to outperform existing multi-agent localization approaches, such as Bayesian- and uniform-based methods, with up to 3 times faster localization and 87%

reduction in cost.

While this chapter serves as an introduction to MDRL for target localization problems, there are certain improvements to be considered for future work. The complexity of the environment is to be further increased, through introducing obstacles that could hinder the movement of the agents and affect their readings. Additionally, communication between agents should be restricted, to increase the realism of the proposed models. Moreover, recent works on Federated RL [79] could prove efficient in handling distributed learning, which could be suitable for localization systems where the learning occurs in the physical world.

Chapter 4

Multi-Agent Deep Reinforcement Learning with Demonstration Cloning for Target Localization in Complex Environments

4.1 Introduction

In this chapter, we extend the proposed MDRL methods to address the challenges posed by complex environments filled with obstacles. While the previous chapter focused on collaborative and scalable target search and localization in simpler and more controlled settings, real-world environments are often cluttered with physical barriers that impede agent mobility and attenuate sensor data. These obstacles introduce significant complexities, as agents must navigate through restricted spaces, avoid collisions, and maintain accurate data interpretation despite noisy or incomplete observations.

To tackle these challenges, this chapter discusses two novel MDRL methods that tackle

the target localization through search in complex environments. The main goal is to utilize agents' observations, i.e. data readings and information about the other agents and the environment, to decide on movement actions to optimally search the area for the target. Initially, a model is designed which utilizes PPO with CNNs to model, train, and optimize the agents' policies. Agents' observations are modeled as 2D heatmaps capturing information about their own, and the locations of other agents, in addition to the collected readings and the distribution of walls in the environment. A shaped team-based reward function, using Breadth First Search (BFS), is designed to guide the agents in *cooperating* to localize the target. Convolutional Autoencoders (CAE) are used to create embeddings that efficiently represent the walls in the environment. A centralized-learning and decentralized-execution (CLDE) method is used to train the agents, which helps in tackling the curse of dimensionality issue in MDRL. This model is then further improved in a second model by replacing the shaped reward function with a sparse reward to speed up the learning. This is combined with *Demonstration Cloning (DC)*, a novel approach that utilizes demonstrations from expert or semi-expert agents, similarly to Imitation Learning (IL), to guide the MDRL agents in the learning process, aiming to achieve faster and more resilient learning. This results in two efficient methods that could be used depending on the availability of expert demonstrations. In summary, the contributions of this chapter are as follows:

(1) The formulation of the multi-agent target localization problem in complex environments using MDRL.

(2) The modeling of the agents' observations as 2D heatmaps, and using CAEs to create embeddings for walls. Observations are fed to CNNs that specify the actions, which are optimized using PPO, and guided by a shaped reward function that is designed using BFS.

(3) The design of Demonstration Cloning (DC); a method that utilizes expert experiences

in guiding agents into faster and more resilient on-policy MDRL in the presence of sparse rewards.

4.2 General Overview of the Proposed Solutions

This section presents the two proposed MDRL models for target localization tasks. Fig. 4.1 gives a general overview of the two proposed models. In the first model (MDRL-SR), agents' policies translate observations into actions that are executed in the environment to obtain a shaped reward and the next observations, which are used for the following step. PPO incorporates the collected rewards to update the agents' policies. Given the availability of expert demonstrations, the second model (MDRL-DC) builds on the first model by using demonstration cloning (DC) in guiding the agents towards collecting better experiences in the environment. The proposed models share the same observation and action spaces, as well as the same policy structure, but differ in the reward function and the learning process.

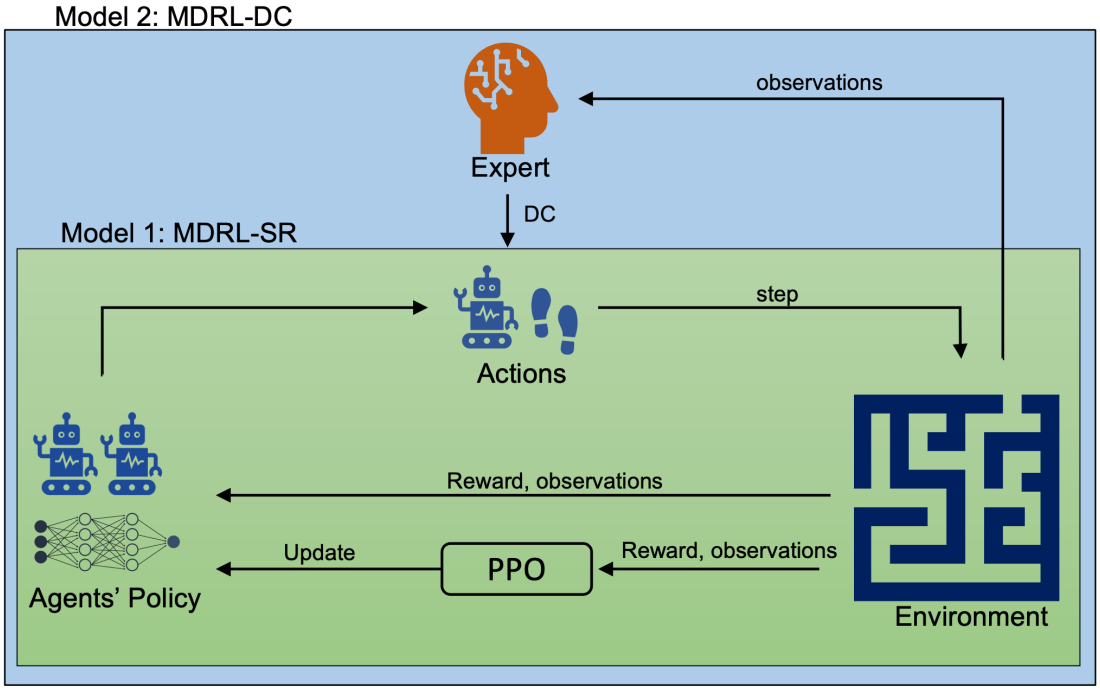


Figure 4.1: General overview of the proposed models.

4.3 Observation Space

The process of localizing the target requires knowledge of current and historical data readings, in addition to the locations where these readings were collected. For this reason, the agents' observations are modeled as 2D maps containing information related to the readings collected, in addition to the agents' locations. To obtain this, the AoI is represented as a grid of size $m \times n$, and the observations are modeled as a stack of 2D maps, where each of them has a size of $m \times n$. The choices of m and n affect the details of the representation, where higher values allow representing finer details at the cost of higher computational complexity. The grid dimensions are not to be confused with the actual dimensions of the AoI. The dimensions of the AoI could be in meters or kilometers, depending on the application, which would be divided into the corresponding $m \times n$ grid. Each of the 2D maps is normalized before being fed to the neural networks, which is a standard practice in deep learning that ensures faster convergence [80]. Here, the normalization of a 2D map is done by dividing all of its elements by the maximum pixel value, resulting in values between 0 and 1. Fig. 4.2 shows the 5 normalized observations collected by each agent at a given timestep t, which are:

- Location Map: a 2D map showing the agent's own location.
- Team Locations Map: a 2D map showing the locations of all the other $N - 1$ agents.
- Readings Map: a 2D map showing the last data reading collected in each grid element by any of the agents. Grid elements that have not been visited are assigned a data reading of 0.
- Visit Counts Map: a 2D map that keeps track of the frequency at which each grid element has been visited by any of the agents.
- Walls Map: a 2D map showing the walls in the area.

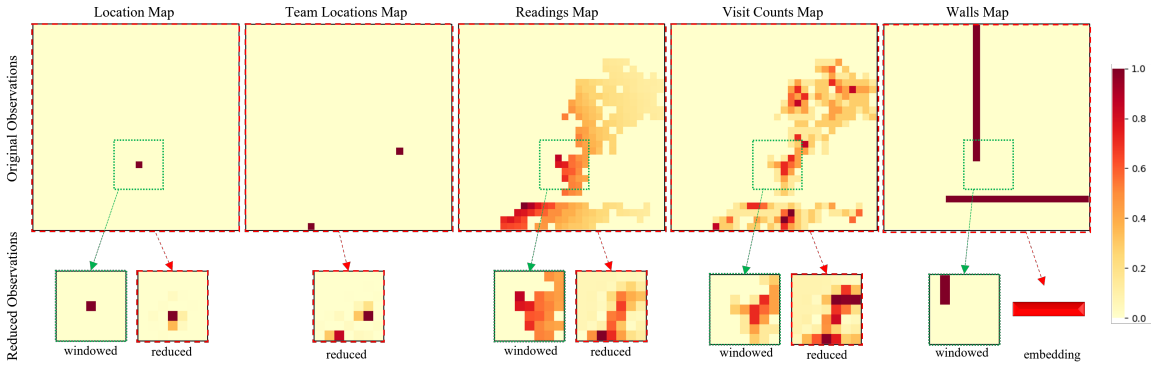


Figure 4.2: The set of original and reduced observations collected by an agent, in a team of 3 agents. Amongst the reduced observations, local observations are highlighted in green (windowed), while the global observations are highlighted in red.

Modeling the observations as 2D maps helps in preserving information. This way, the information could be accumulated in a progressive manner on the same 2D maps as the target localization process carries on. Additionally, agents' locations and the readings collected are spatially correlated in target localization tasks, and such spatial features can be maintained and analyzed when the observations are modeled as 2D maps.

In the target localization problem, each agent bases its actions solely on the aforementioned observations. In these observations, the readings map is used to guide the agents towards the target location, where the agents are expected to follow the paths leading to higher readings. The readings map is built in a progressive manner, where the map initially could have insufficient and uninformative readings (due to the agents being far from the target or due to signal attenuation because of walls). At each step, the agents take actions and update the readings map with the collected readings, after which it can be better used to guide the agents to reach the target. With the existence of walls in the area, the agents could initially need to move away from the target (and hence move towards low data readings), as seen in Fig. 1.1 for agent 2. This could be achieved when incorporating the readings and walls maps together. In cases where the agents fail to collect good readings initially, they are expected to explore unvisited areas, which is enabled through keeping track of the visited locations in the Visit Counts map. To help with cooperation and coordination between

the team members, the Team Locations map is used. The agents are assumed to have full communication, and hence are capable of sharing their own locations and readings.

Before feeding the observations to the actor and critic networks, they undergo pre-processing steps to reduce dimensionality while maintaining the important features. This helps with speeding up the learning process, as the number of trainable parameters is reduced. To do so, the 5 aforementioned original observations are converted into the 9 reduced observations, as shown in Fig. 4.2. Each of the first 8 reduced observations has a size of $w \times w$, where $m, n > w > 1$ and w is odd. w is a hyperparameter, where smaller values lead to less trainable parameters in the policy network (and hence reduced learning complexity), but also lead to higher loss of information, when compared to higher values. The main goal is to obtain reduced observations that still maintain the essential information from the original observations. To achieve this, the reduced observations are divided into two sets: local and global observations. The local observations help an agent make local decisions regarding the areas surrounding them. Such decisions include overcoming nearby obstacles and moving towards unvisited areas or areas with higher readings within the small region around them. On the other hand, the global observations give the agent a summary of the information in the AoI, which helps in coordinating with other agents and planning ahead.

The local observations capture the information surrounding the agent, in a $w \times w$ map centered around the agent's current location, and are given as follows at step t:

- Windowed Location Map: a 2D map showing the agent's location.
- Windowed Readings Map: a 2D map showing the latest data readings collected in a small region around the agent.
- Windowed Visit Counts Map: a 2D map showing the frequency of the visited areas around the agent.

- Windowed Walls Map: a 2D map showing the walls near the agent.

The global observations have a size of $w \times w$ and are as follows for a given agent, at timestep t:

- Reduced Location Map: a 2D map obtained by downsampling the agent's original location map.
- Reduced Team Locations Map: a 2D map obtained by downsampling the agent's original team locations map.
- Reduced Readings Map: a 2D map obtained by downsampling the agent's original readings map.
- Reduced Visit Counts Map: a 2D map obtained by downsampling the agent's original visit counts map.
- Walls Embedding: an embedding of size $1 \times d$ obtained through encoding the agent's original walls map using a pre-trained Convolutional AutoEncoder (CAE).

The first 4 global observations are obtained through downsampling using bi-linear interpolation. While generic downsampling is effective for these observations, the loss of information would be notable when applied to the walls map. This is mainly because, in some cases, the agents need to navigate through narrow areas/entrances, and information may be lost if traditional downsampling techniques are used. To overcome this, Convolutional AutoEncoders (CAE) are used to embed the walls map into an embedding (1D vector) of reduced dimensions. The structure of the used CAE is shown in Fig. 4.3. Given a dataset of walls, which is synthetically created using variations of wall placements and wall lengths, the CAE is trained to embed the walls map into a vector of length d. A typical autoencoder consists of two components: an encoder and a decoder. The encoder maps the

input to an embedding (a code), and the decoder aims to reconstruct the original input from the code. Typically, the training loss is based on the difference between the input and the reconstructed map, which is to be minimized, indicating that the code has enough information to reconstruct the original image from. Once an autoencoder is trained, the encoder is then used to create embeddings for the wall maps observations.

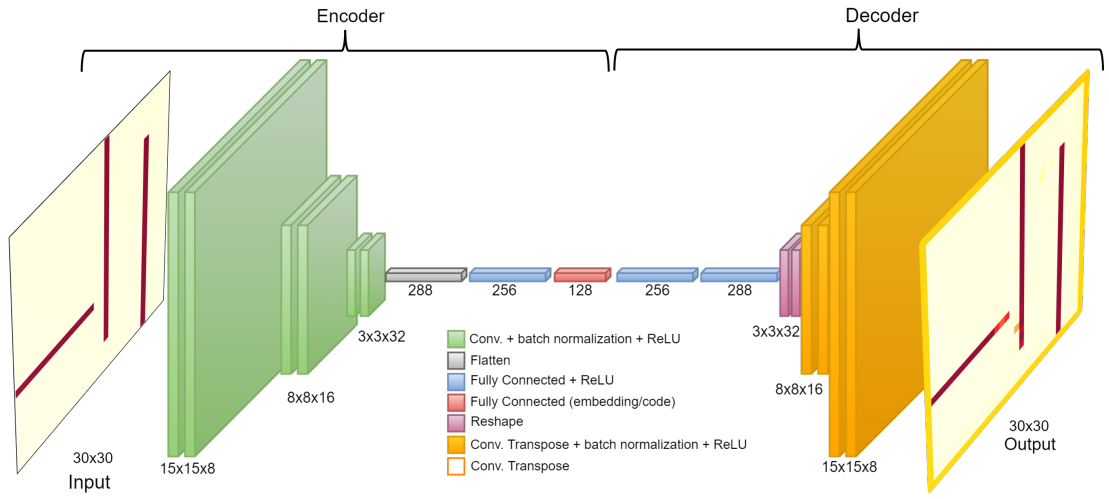


Figure 4.3: The architecture used for the CAE. The CAE is pre-trained using a dataset of walls, after which the encoder is used to embed the wall maps as part of the proposed MDRL models.

4.4 Action Space

In the proposed models, the agents take discrete actions in the AoI: to either move in a certain direction or stay idle. The choice of staying idle helps in preserving resources, while moving affects the contribution the agent has towards the target localization task. The direction of movement is discretized into B possible directions {1, 2, ..., b_i, ..., B}, where the direction (or angle) of movement is given as:

$$\theta = 2\pi \frac{b_i}{B} \tag{7}$$

B here is a hyperparameter that could be determined based on the desired details and the available computational capabilities. Higher values of B allow more mobility freedom for the agents, but also increase the learning overhead as the policy network would have more trainable parameters. In this work, the agent has 9 possible actions, being 8 possible cardinal directions (B=8) and one action for staying idle. This has been proved sufficient in this work, as it covers the horizontal, vertical, and diagonal directions. It is assumed that all agents have the same fixed speed. At each time step, certain actions may be invalid (such as moving into a wall or outside the boundaries of the AoI). In such cases, these actions are masked out, and the agent has to choose from the set of viable actions. This is applied during the learning and the inference processes. Additionally, during learning and inference, if all agents choose to stay idle for 3 successive timesteps, the idle action is masked out for the next step for all agents, to encourage exploration and to speed up the learning.

4.5 Actor and Critic Networks

The proposed models use an actor-critic structure in the learning process, optimized by PPO. The actor (policy) network translates input observations into actions, while the critic network is used to estimate the value function. Since the observations in this work are modeled as 2D maps, Convolutional Neural Networks (CNNs) are used to represent the actor and critic. CNNs help in capturing spatial features in input maps, which are common in the aforementioned observations.

The structures of the actor and critic networks are shown in Fig. 4.4. This architecture is similar to the LeNet-5 architecture [74] which is widely used in image analysis tasks. At each time step, for a given agent, the actor network takes as input the 9 reduced observations, shown previously in Fig. 4.2, and produces B+1 actions. The first 8 observations (2D maps) are fed through convolutional and Max-Pooling layers to extract features.

The resultant maps are flattened and concatenated with walls embedding, and fed to the fully-connected layers. The 1D walls embedding does not contain any spatial information, and hence is not passed through the convolutional layers. A SoftMax function is used at the last layer so that the outputs correspond to a probability distribution over the possible actions. The proposed models in this work follow a Centralized-Learning & Decentralized-Execution (CLDE) [41] method. Each agent has the same copy of the actor network during learning and execution, and acts based on its own observations in a decentralized manner. However, during the learning process, the critic network has access to all observations. A centralized critic helps in tackling the non-stationarity issue, which is caused due to the environment being influenced by other agents from a single agent's perspective [73]. This is viable since the critic is not needed and emitted during the execution/deployment stage. For this reason, the critic CNN takes as input the following 4 observations:

- Reduced All-Locations Map: a downsampled version of a 2D map showing the locations of all agents in the AoI.
- Reduced Readings Map
- Reduced Visit Counts Map
- The walls embedding

These observations are sufficient for the critic to give a value to the current state of the environment, which is a singular output given by the critic network. The critic considers the global observations and gives a value representing how good the current state is. This value is used in the PPO process to update the networks, as discussed later in Section 4.6. The input 2D maps to the neural networks are normalized beforehand, which is a typical process when using CNNs.

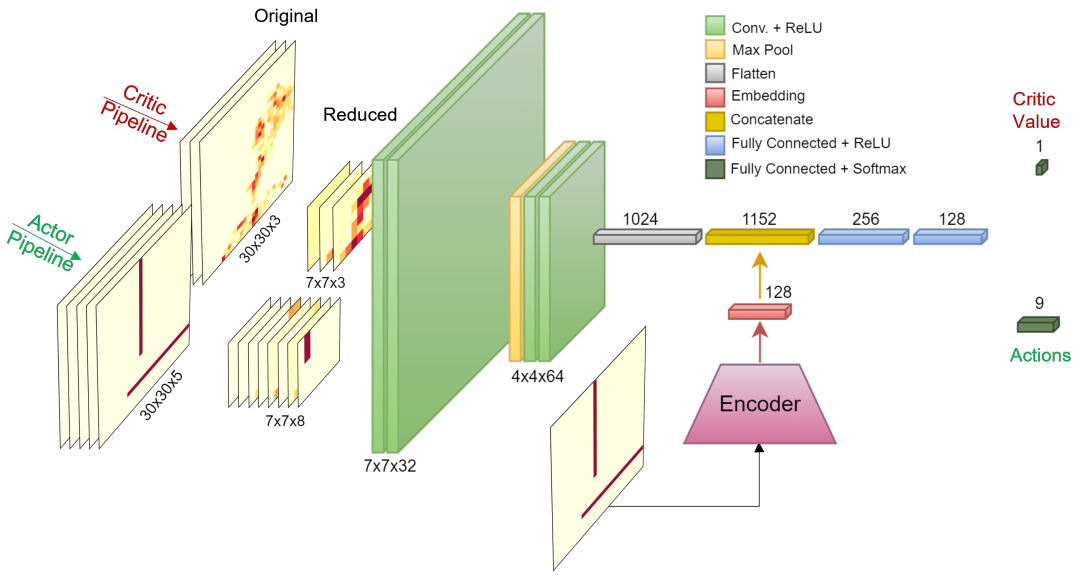


Figure 4.4: The actor and critic networks (CNN) used in the proposed models.

4.6 Reward Function and Learning Process

This section presents and discusses the two proposed models. The two models share the same previously presented observation and actions spaces, and the same network architectures, but differ in terms of the reward function and the learning process.

4.6.1 Model 1: MDRL with Shaped Rewards (MDRL-SR)

This section presents the first model tackling the target localization problem in complex environments.

Reward Function: This model uses a *shaped* reward function to guide the learning process. To push the agents to cooperate, the reward function is also *team-based* (joint) [41]. Following the actions taken individually by the agents, the environment releases a shared (equal) reward to all the agents based on the joint actions. This helps the team in realizing the advantage of taking actions that benefit the team.

At a given time step, and given N agents with the joint action $\mathbf{a}^t$ at step t, the reward function is given as:

$$R_t = \begin{cases} -b + 1 & \text{if } \min(D_t) < \min(D_{t-1}) \\ -b - 1 & \text{otherwise} \end{cases} \tag{8}$$

where D is the set of distances between the agents and the target, and b be the number of agents that took a moving action (i.e. did not choose to stay idle). The $(-b)$ term represents the cost of movement, being -1 for each agent that moves. The second term (± 1) assesses if the agents have gotten closer to the target or not. At step t, the team is considered to have gotten closer to the target if the closest agent (or one of the closest agents) to the target at step $t-1$ gets closer to the target at step t, i.e. if $\min(D_t) < \min(D_{t-1})$. For example, considering the scenario in Fig. 1.1, the team would get a reward of -2 at t_0 (3 agents moved and the team got closer to the target), a reward of -2 at t_2 (1 agent moved and the team did not get closer to the target), and a reward of 0 at t_3 (one agent moved and the team got closer to the target). It is worth mentioning that the reward function is used only during the training stage (offline in simulations) as a feedback mechanism to assess the actions taken by the agents. While the reward function is based on the distance to the target, the agents solely act based on their observations, with no knowledge of the target location. Once the training is completed, i.e. during the inference/deployment stage, the reward function is omitted, and the agents exploit whatever policy they have already learnt. Distance-based reward functions are common in similar works, as in [2, 44, 81].

Using this shaped reward, the agents get feedback throughout the episode, and not only when the target is localized, which helps in learning even if the episode terminates without finding the target. The agents are motivated to finish the localization quickly, to accumulate less moving cost. Additionally, the agents are motivated to cooperate in order to move only when necessary (i.e. when they can contribute to the task), which helps in performing the task efficiently while managing resources.

To compute the distance between the agents and the target (during learning), conventional methods like Euclidean/Manhattan distance cannot be used due to the existence of walls. To tackle this, the proposed model uses the $m \times n$ gridded AoI presented in Section 4.3, along with the Breadth First Search (BFS) method to compute the shortest distance between each agent and the target. BFS starts from an initial node, and gradually explores paths starting from this node until the goal is reached. A path here is given as a set of grid elements that lead the agent to the target. It is computationally expensive to perform BFS at each time step, for each agent. To expedite the process, at the beginning of each episode, a *distance map* is built by computing the distance between the target and all the grid elements in the AoI grid. This can be done with a single BFS process by setting the starting node as the target's location and ignoring the goal check in the BFS algorithm. As a result, BFS would continue to explore all nodes, and hence compute the distance to all the grid elements. A sample of the distance map is shown in Fig. 4.5.

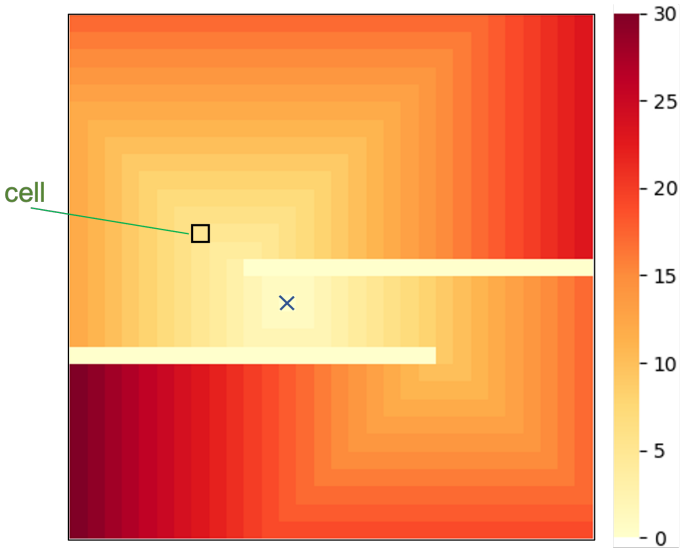


Figure 4.5: An example of the distance map obtained at the beginning of an episode. Each cell contains a value representing the shortest distance (in number of steps) between the cell and the target (marked with $\times$).

Learning Process: This model follows a Centralized-Learning & Decentralized-Execution (CLDE) [41] method, with PPO to train the actor/critic networks. The training process is

similar to that explained in Algorithm 3.3. Each episode starts with a reset that returns the initial observations for each agent ($\mathbf{o}^0 = (o_1^0, ..., o_N^0)$). Each agent gets a copy of the actor network, which takes a set of observations for agent j and returns an action a_j. Each of the agents acts in the environment and receives a set of new observations, a shared reward (the same for all agents), and a termination flag (d) indicating whether the episode is finished or not. These items are stored in their corresponding buffers, and later used (after H timesteps) to update the actor and critic networks. H here represents the horizon, i.e. the number of steps in the environment after which a PPO update occurs. During deployment/inference, only the actor network is used, where each agent gets a copy of the network and acts in a distributed manner. This model proves efficient in obtaining agents that are capable of localizing the target in complex environments, as will be shown in Section 4.7. The proposed reward function helps in pushing the agents to quickly localize the target, while cooperating to manage resources. However, the complexity overhead of using BFS in each episode could be reduced, which is the purpose of the second model.

4.6.2 Model 2: MDRL with Demonstration Cloning (MDRL-DC)

This model builds on the first model by introducing a novel concept: Demonstration Cloning (DC). The main idea here is to use an existing expert to *guide* the agents into collecting better experience. An expert is an existing agent that has expertise in tackling the same environment. In this work, a semi-expert is defined as an agent that has expertise in tackling a similar environment that slightly differs from the current one.

Reward Function: As discussed in Section 4.6, the reward function in Eq. 8 is computationally expensive, as it requires performing BFS in each episode. Alternatively, this model uses a sparse reward function given as:

$$R_t = \begin{cases} -b + S & \text{if target is localized} \\ -b & \text{otherwise} \end{cases} \tag{9}$$

where S is a constant representing the sparse reward given to the agents if the target is found. This reward function maintains the same cost penalties as the ones in Eq. 8, but gives a large positive reward only when the target is localized.

Learning Process: The idea of using expert demonstrations with RL is derived from Imitation Learning (IL). In IL, agents learn to mimic the behavior of an expert by using expert demonstrations as a labeled dataset in a supervised learning process, in which the aim is to reduce the loss between the agent and expert actions. Unlike RL, the concept of rewards does not exist in IL, as agents solely consider the expert as the main reference, without any input from their own experience. The work in [49] proposed using expert demonstrations with off-policy RL, where demonstrations are combined in the same buffer as agents' experiences and used to update the networks. Other works combine IL with off-policy [48] and on-policy [17] RL. In these works, the proposed models alternate between using the agent's own experience (i.e. rewards) and IL (i.e. mimicking an expert with no rewards). One issue in such works is the assumption that an expert, who is fully suited for the environment, exists. This limits the usability of such methods, as such experts may not always be available.

In this work, we propose MDRL-DC, a method that utilizes expert demonstrations while always maintaining the "learning from rewards" concept in RL. The proposed model alternates between RL and DC as follows: during RL, the agents explore the environment using the actions suggested by their own actor networks. During DC, the agents explore the environment based on the actions suggested by the expert. This way, while the expert guides the learning during DC, the agents are still learning based on their own experience, i.e. based on the rewards collected. This could result in better experiences, where the

agents are more exposed to the sparse reward. This allows the use of semi-experts, i.e. experts that are familiar with similar environments but not necessarily the same environment. Bad actions suggested by the semi-expert will be identified through the collected rewards. Using the aforementioned works (RL+IL) with such semi-experts would not be efficient, since they are not fully suitable for the current environments, and agents during IL have no inputs in deciding whether an experience is good or not. As a result, the agents during the IL phase may learn what is against the learnt policy during the RL phase, resulting in a slower and potentially unstable learning.

Figure 4.6 and Algorithm 4.1 illustrate and summarize the learning process of this model, which alternates between RL and DC. At the beginning of each episode, an expert probability E determines whether the episode will follow RL or DC. During RL, the process is similar to that described in Algorithm 4.1, where agents act based on their copies of the actor network, and their experiences (observations, rewards, idle flags) are buffered. When the expert is switched on, the expert takes the observations seen by the agents and suggests actions. These actions are then followed by the agents and the corresponding experiences are stored in the same buffers as the previous experiences. One key point here is that the actions suggested by the expert are *conditioned* on being probable under the agents' actor networks. This is because PPO is an on-policy algorithm, which means that the sampled actions need to follow the most recent version of the actor network. To handle this, a hyperparameter l is proposed, which represents the minimum probability needed, under the agent's own policy, for an action to be considered. In the proposed model, instead of suggesting the best action, the expert ranks all possible actions from best to worst, given higher values to better actions. An agent then takes the best action a, as suggested by the expert, that satisfies $P(a) > l$, where P is the probability distribution over the actions given by the agent's own policy. This way, the demonstrations given by the experts can help the agents get more exposed to rewards, while not violating the agents' own actor networks

for on-policy learning. It is also worth mentioning that, despite using a fixed expert rate E throughout the learning, the effect of the expert is automatically annealed as the learning progresses. This is because the actor network becomes increasingly more confident, which results in less actions meeting the $P(a) > l$ condition.

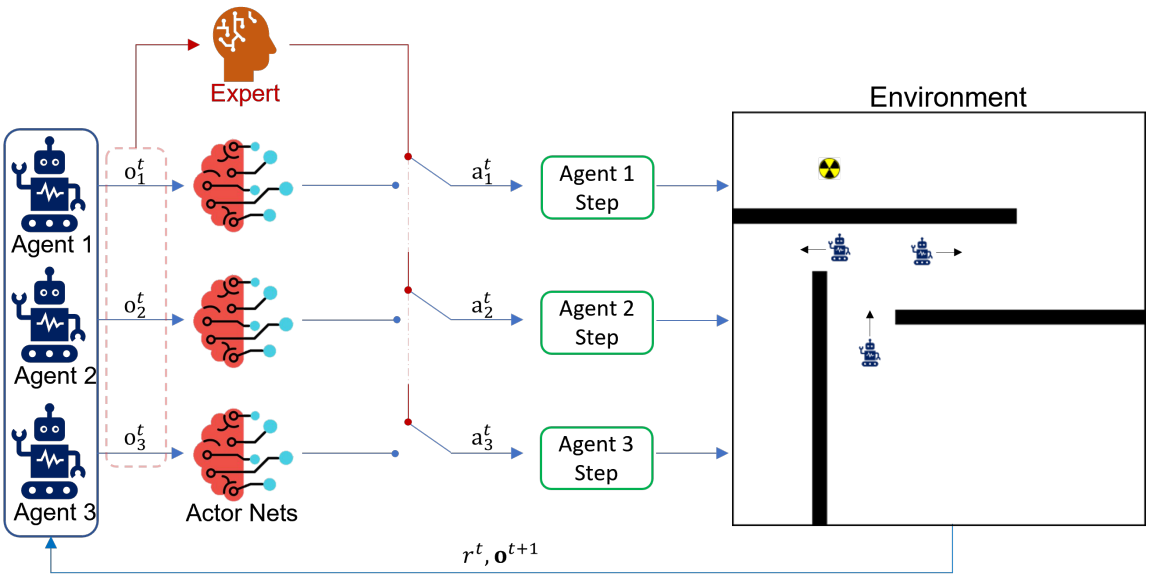


Figure 4.6: MDRL with Demonstration Cloning.

It is assumed that there is access to the expert policy, and hence expert demonstrations can be generated online as needed. This is suitable for problems like target localization, where the expert could be previously trained on simpler environments (with no walls or with a single agent).

4.7 Experiments and Evaluation

This section presents several experiments conducted to evaluate the performance of the proposed methods. Simulations have been conducted using an Intel E5-2650 v4 Broadwell workstation equipped with 128 GB RAM, 800 GB SSD, and NVIDIA P100 Pascal GPU (16 GB HBM2 memory).

The performance of the proposed models is analyzed in terms of episodic length and

Algorithm 4.1: The training process for the second model

Input: Initial actor and critic networks, Expert policy

while Step $\leq$ MaxNumOfSteps **do**:
  $\mathbf{o}^0$ = Env_Reset() #initial observations
  ExpertProb = rand() #generate a random probability
  for i = 0, 1, 2, ..., Episode_Length:
    for j = 1, 2, ..., Team_Size:
      P^j = actor(o^i_j) #get prob. dist. under actor network for agent j
      if ExpertProb $< E$:
        $\mathbf{a}_e$ = Expert(o^i_j) #get the action values from the expert, sorted from best to worst
        $\mathbf{a}_e[P^j[\mathbf{a}_e] < l]$ = 0 #mask out actions that do not meet the threshold l.
        a^i_j = max($\mathbf{a}_e$)
      else:
        a^i_j = sample(P^j) #sample an action from the actor distribution
    end for
    $\mathbf{a}^i$ = $[a^i_1, a^i_2, ...]$
    $\mathbf{o}^{i+1}, r^i, d^i$ = Env_Step($\mathbf{a}^i$)
    Store $\mathbf{o}^{i+1}$, $\mathbf{a}^i$, and r^i in their corresponding buffers
    Step = Step + 1
    if Step % H == 0 **then**: #PPO update
      Compute the advantage estimate $\hat{A}$
      Update the actor and critic networks using $L^{CLIP+VF+S}(\theta)$
      Empty all buffers
    end if
    if $d^i == 1$ **then** break #if episode terminates
  end for
end while

cost. As discussed in Section 1.1, the main aim of the target localization methods is to achieve fast localization with low cost. The localization time is measured here by the episodic length, i.e. the number of steps it takes the agents from the beginning of the episode until the target is localized. The cost is given as the total number of movement steps taken by the agents, since a moving action reflects consumption of resources. In a single episodic step, the cost could be as low as 0 (no agents move) or N (all agents moved). At each step, each of the agents takes one of the 9 actions discussed in Section 4.4. A moving action moves the agent 50m in the chosen direction, after which another step (action) is required to carry on, based on newer observations. The maps dimensions

($m \times n$) are set to 30×30, while the reduced dimensions are with a window size ($w \times w$) of 7×7. For all the experiments, a problem of radioactive target localization is used as an environment, as discussed in Section 3.7.1.

For all the following experiments, each of the models was trained for 50 million steps. Each step represents a single interaction between the agents and the environment. In each training episode, the environment is reset with a randomized target and agents' locations, and with randomized walls placements and lengths. The number of walls is fixed for a single experiment. Walls here are always originating from the sides of the AoI. It is assumed that an agent cannot start in the same location as the target or another agent. An episode terminates when the target is found, or when a limit of 100 timesteps is reached. After every 40,000 training steps, the average results of 4,000 testing steps, where the agents act greedily based on the latest policy update, are recorded and plotted. The list of hyperparameters used in training PPO, CAE, and DC, are shown in Table 4.1. The values for ε, c_2, γ, and λ are set as suggested in the corresponding PPO work [31].

Table 4.1: Hyperparameters used for PPO and CAE training.

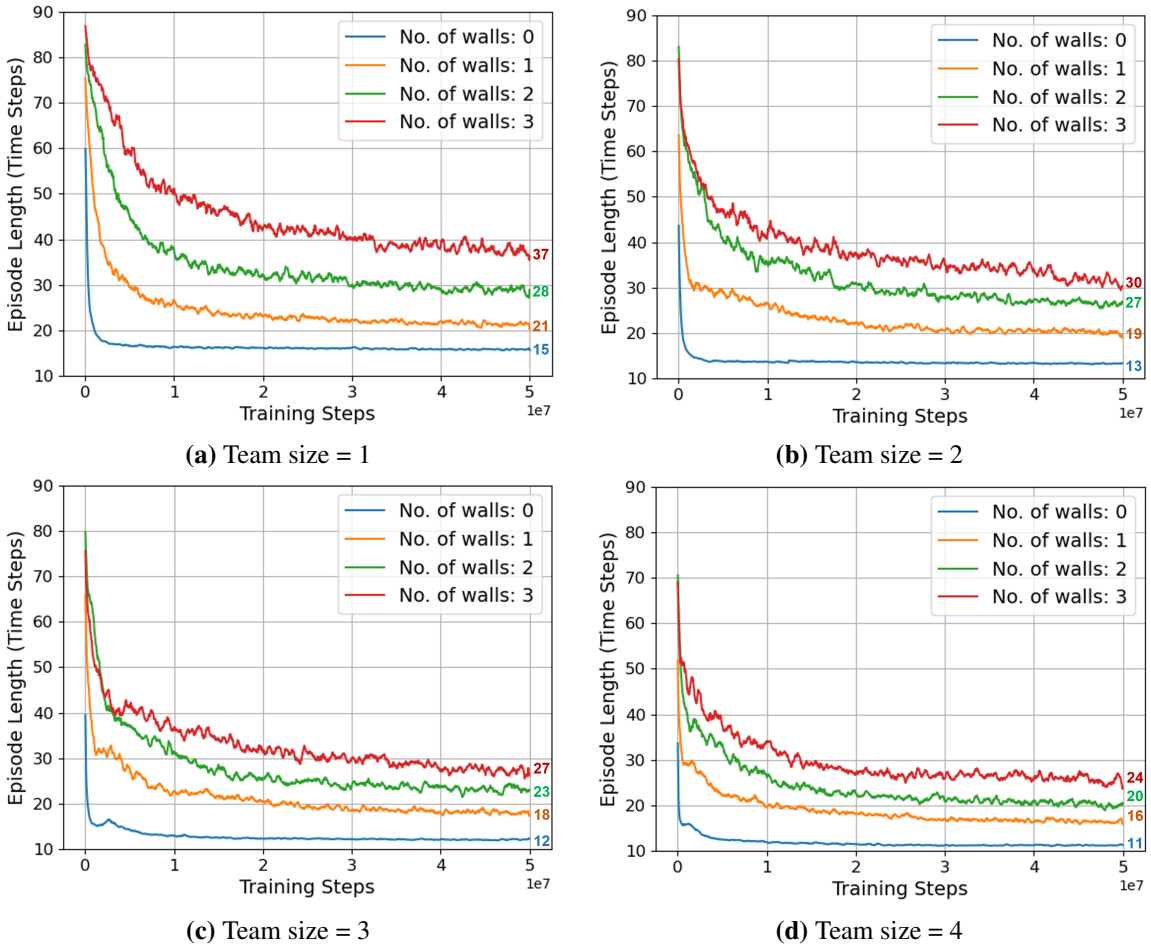
PPO Hyperparameters	Value
Learning rate	3×10^{-4}
PPO clipping parameter ε	0.2
Entropy coefficient c_2	0.01
Discount factor γ	0.99
Discount factor λ	0.95
Timesteps per update (Horizon H)	4000
Number of epochs per update	20
CAE Hyperparameters	**Value**
Learning rate	1×10^{-3}
Embedding Size d	128
Dataset Size	100000
DC Hyperparameters	**Value**
Expert Rate E	0.2
Action Probability Threshold l	0.05

4.7.1 Performance of MDRL-SR

This section studies the performance of the MDRL-SR model in varying scenarios of team size and environment complexities. Specifically, the number of agents is varied from 1 to 4, while the number of walls is varied from 0 to 3. Fig. 4.7 and Fig. 4.8 show the episodic length and cost obtained throughout the learning process. As seen in all figures, the learning converges regardless of the team size or the environment complexity, with faster convergence in simpler environments. For all of these figures, the localization success rate is 100% at the end of the learning process, showing the effectiveness of the proposed approach. Intuitively, and as seen in the figures, longer episodes with higher cost are obtained as the environment gets more complex (more walls), since agents need more steps to navigate through the environment, and since more walls result in more attenuation of the radioactive readings, and hence harder localization.

As seen in Fig. 4.7, faster localization is achieved as the team size is increased from 1 (Fig. 4.7a) to 4 (Fig. 4.7d), for all environment complexity levels. On the other hand, the cost follows a slightly different behavior in Fig. 4.8. The episodic cost is affected by two factors: the episode length and the team size. Though having a bigger team results in more cost per step, on average, yet bigger teams result in faster localization, as per Fig. 4.7, and hence less total cost per episode. This trade-off depends on the complexity of the problem. For example, in the case of simple environments with 0 or 1 wall, it can be seen that the cost is slightly reduced as the team size increases. That is, even when increasing the number of agents, the cooperation achieved by the agents results in faster localization at lower cost, as agents learn to prefer the IDLE action if they are not beneficial to the localization process (as discussed in Section 1.1). For more complex environments (2 or 3 walls), more exploration is needed since readings are significantly attenuated. As a result, increasing the number of agents could result in increasing the total cost, as seen when comparing Fig. 4.8a with Fig. 4.8b for the cases of 2 and 3 walls. This cost saturates at a

certain team size, indicating that enough agents are available to handle the task and balance their resources without the need of unnecessary exploration. For example, for the case of 2 walls, the cost slightly increases from 20 to 38 when going from a team of 1 member to two members, then saturates around the same cost for higher team sizes. This saturation point happens at a later team size for a more complex environment of 3 walls, as the cost increases from 36 to 53 as the team size is increased from 1 to 3, and saturates for higher team sizes. This saturation, in all the aforementioned scenarios, is an indication that the agents have achieved good cooperation and resource management, even when the team size is increased.



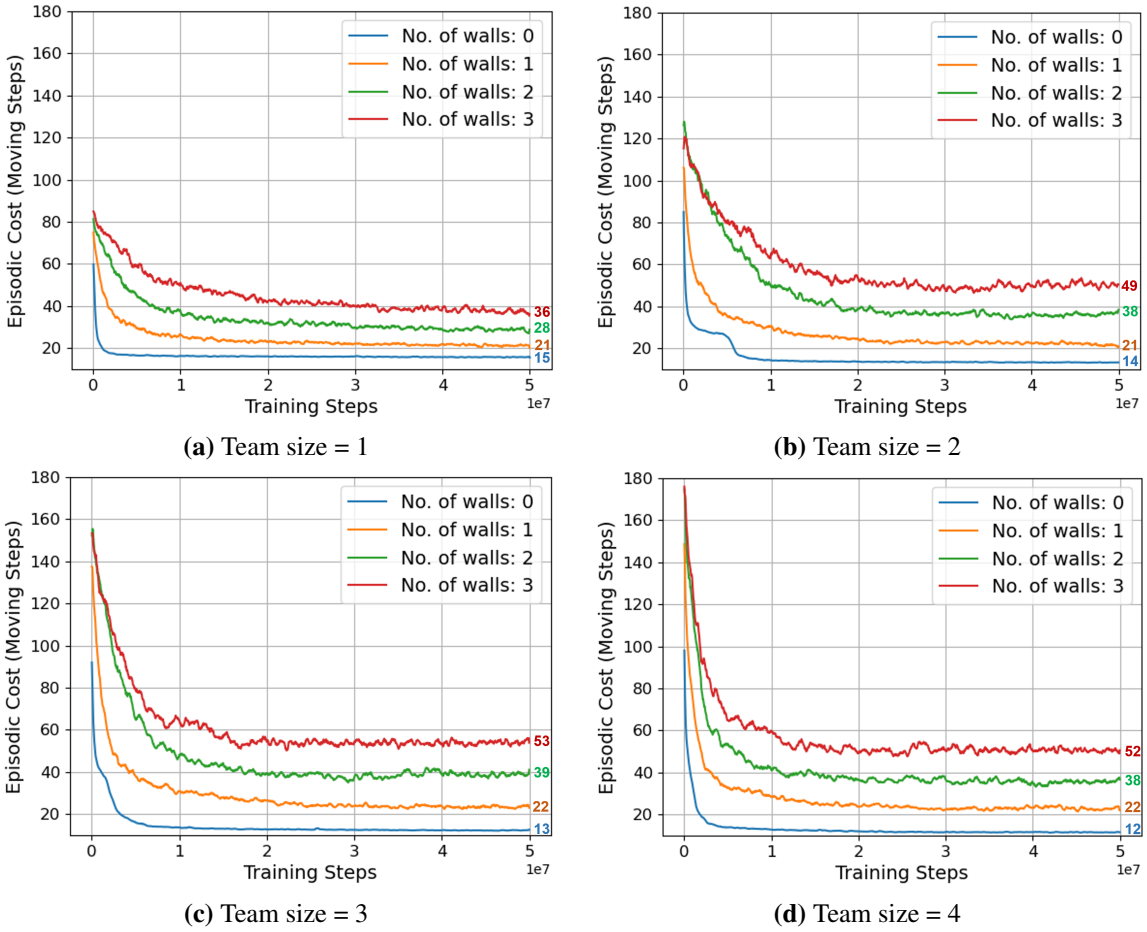
(a) Team size = 1

(b) Team size = 2

(c) Team size = 3

(d) Team size = 4

Figure 4.7: The episodic length throughout the learning for a system of (a) one agent, (b) two agents, (c) three agents, and (d) four agents, for varying numbers of walls within the environment



(a) Team size = 1

(b) Team size = 2

(c) Team size = 3

(d) Team size = 4

Figure 4.8: The episodic cost throughout the learning for a system of (a) one agent, (b) two agents, (c) three agents, and (d) four agents, for varying numbers of walls within the environment

4.7.2 Performance of MDRL-DC

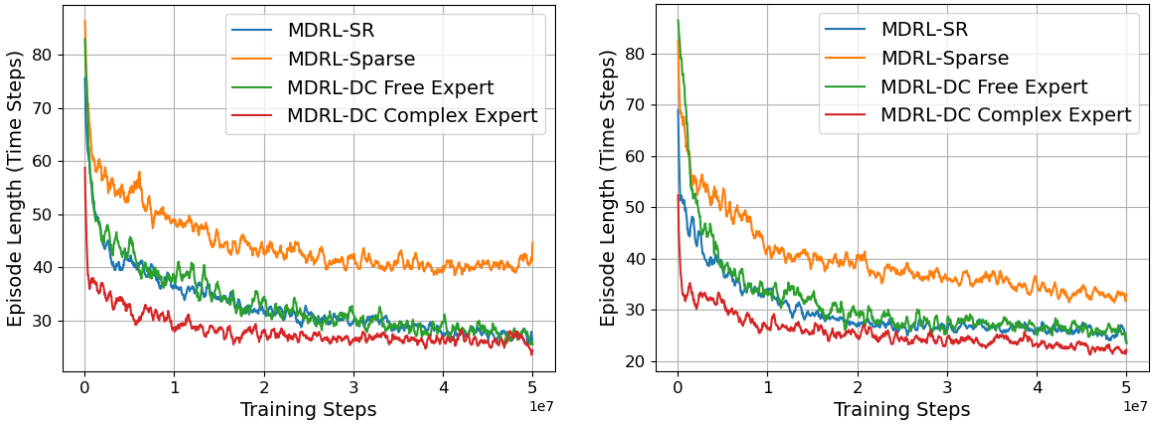
This section analyzes the performance of the MDRL-DC model in comparison with the MDRL-SR model. This is done for teams of size 3 to 4 agents, in a complex environment of 3 walls. The performance of MDRL-DC with two different semi-experts is studied. The first is an expert in a multi-agent free environment (no obstacles) while the second is an expert in a single-agent complex environment. The first expert is familiar with cooperation (and not complex environments), while the second expert is familiar with complex environments and not cooperation. Additionally, to show the drawback of using sparse rewards (without experts), the model MDRL-Sparse is included in the comparison. A sparse reward

with the value S =50 is used, as this was found to be sufficient to incentivize the agents.

Figures 4.9 and 4.10 compare MDRL-SR with MDRL-Sparse and MDRL-DC in terms of episode length and cost. It is evident, when comparing MDRL-Sparse to the rest of the models, that using a sparse reward results in poor performance, which is due to the fact that agent is rarely exposed to the reward. It can also be seen that, when using a free expert, MDRL-DC is at least as good as MDRL-SR, if not better. This is significant since MDRL-DC is able to overcome the reward sparsity using expert demonstrations, and achieve a performance similar to a MDRL-SR that uses a shaped reward. Additionally, since the computationally expensive shaped reward is emitted in MDRL-DC, the learning process of 50M steps consume nearly 47% less wall-clock time when compared to MDRL-SR. When looking at the behavior of MDRL-DC with a free expert in Fig. 4.9, it can be seen that the performance is similar to MDRL-SR in terms of episode length. However, it can be seen in Fig. 4.10 that MDRL-DC with a free expert has a slightly better performance when compared to MDRL-SR, as it achieves faster convergence. This is attributed to the fact that the free expert is familiar with cooperation, and hence is capable of guiding the new agents into learning this attribute quicker. On the other hand, MDRL-DC with a complex expert slightly outperforming MDRL-SR in terms of time and cost, while maintaining the same 47% less wall-clock time for the learning process. This shows that, despite using a sparse reward and semi-experts that are not fully familiar with the same environments, the proposed DC method is able to achieve similar, if not better, performance to a system with shaped reward at almost half the runtime.

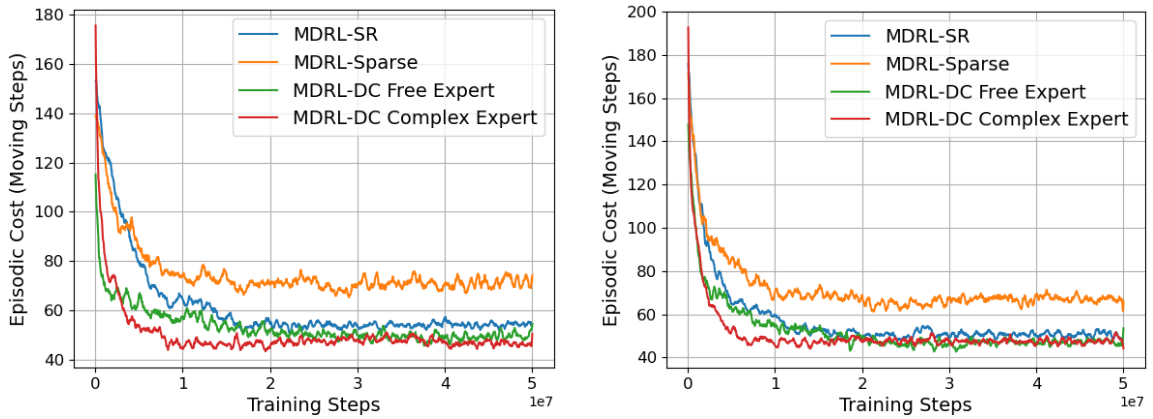
4.7.3 Benchmarks

This section compares the performance of MDRL-DC with several benchmarks. These benchmarks are either MDRL-based target localization or traditional target localization



(a) Team size = 3 **(b)** Team size = 4

Figure 4.9: The episodic time throughout the learning for a system of (a) 3 agents and (b) 4 agents, in an environment of 3 walls, for different learning models



(a) Team size = 3 **(b)** Team size = 4

Figure 4.10: The episodic cost throughout the learning for a system of (a) 3 agents and (b) 4 agents, in an environment of 3 walls, for different learning models.

methods. In all of the following results, a simulation environment of 3 agents with a varying complexity of 1-3 walls is used to compare the performance in terms of episode length (time) and cost. The produced agents, following the training process, are placed in random environments for inference, where the episode length and cost is averaged over 1000 different episodes.

Figure 4.11 compares the performance of MDRL-DC with 3 other deep RL models and 2 traditional methods, which are:

- ODMTL: the MDRL approach for target localization [44] in free environments (no obstacles), which was developed in Chapter 3. This model ignores the existence of walls and attempts to localize the target without considering them in agents' observations.

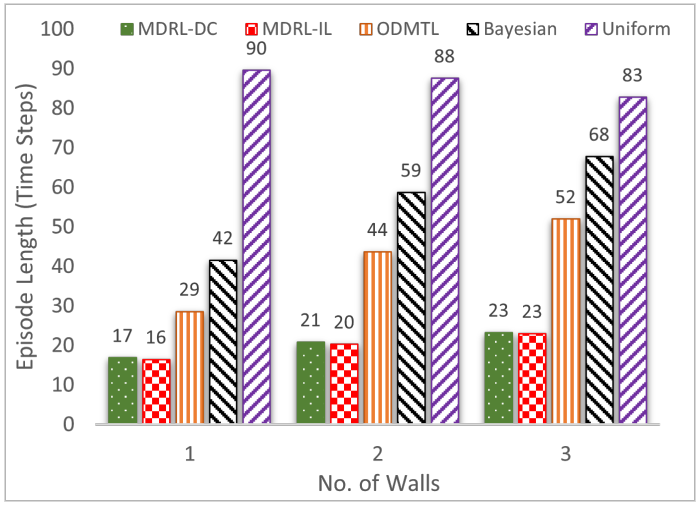
- MDRL-IL: a MDRL model that alternates between RL and Imitation Learning (IL) [17]. Unlike our proposed model, the agents in MDRL-IL during the IL stage blindly copy the behavior of the expert into their actor networks using Behavioral Cloning (BC), without having to experience any rewards in the environment.

- Bayesian: a multi-agent cooperative approach using Bayesian methods [11, 15]. In [15], a Bayesian framework is used to build and update local probability density functions (PDFs) of the target location, which are shared between the agents to build a global PDF. This is used in an optimization objective that aims to find the target while balancing resources consumption between agents.

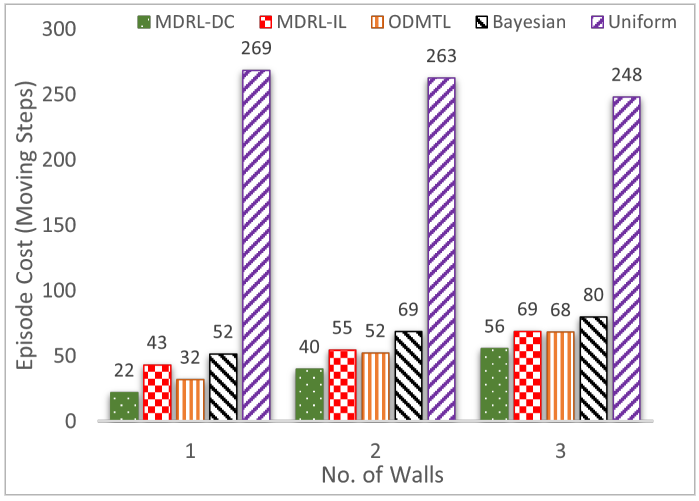
- Uniform: a traditional method where each agent moves in a pre-defined path that uniformly covers the area [40].

As can be seen in the figure, the performance of the ODMTL model significantly falls behind the proposed MDRL-DC. Specifically, MDRL-DC has 50% less localization time and 24% less cost, on average, when compared to ODMTL. This shows the significance of considering walls/obstacles, as they have effects on sensor readings and agents planned paths. While both MDRL-IL and MDRL-DC use experts' demonstrations during the training, their performances are different. Here, both models use an expert that is familiar with single-agent complex environments. When analyzing Fig. 4.11a, similar performance in terms of episodic length is noticed when comparing both models. However, when analyzing Fig. 4.11b, the significance of MDRL-DC can be noticed. Since the expert is not familiar with cooperation, the agents in MDRL-IL struggle to cooperate. Even though these

agents could learn cooperation during the RL stage (which is 80% of the learning), this is overwritten by BC in the IL stage, since the expert always acts greedily and has no sense of the other agents. On the other hand, since the agents in MDRL-DC learn based on their own experience/rewards, they can identify through rewards if such experiences are good or bad, even if suggested by the expert. As a result, the model outperforms MDRL-IL, and shows its ability of using demonstrations from semi-experts. Specifically, the model achieves 32% less cost, on average, when compared to MDRL-IL.



(a)



(b)

Figure 4.11: Performance comparison between the benchmarks in terms of (a) episode length and (b) episode cost, throughout the learning process.

When compared to traditional target localization methods, MDRL-DC shows dominance. Traditional methods could be efficient in handling free environments, but struggle with complex environments with walls, as such environments are difficult to define and represent. When compared to Bayesian methods, MDRL-DC localizes the target with 63% less time and 43% less cost, on average. On the other hand, uniform search methods do guarantee finding the target, since the agents scan the entire area, however this comes at huge costs, since all the agents are always active.

4.8 CONCLUSION

In this chapter, the problem of target localization in complex environments is tackled, where two MDRL models are proposed. Agents' observations are modeled as 2D heatmaps that capture essential information about their locations and readings, in addition to information about the environment complexity in terms of walls distribution. Convolutional Autoencoders are used to create embeddings to represent walls, while Convolutional Neural Networks are used for the actor-critic structures in the Proximal Policy Optimization algorithm that is used to optimize the learning. The first model, MDRL-SR, uses a shaped reward function, based on breadth first search, to guide the agents into cooperating to localize the target. The second model, MDRL-DC, builds on the first model by using a sparse reward instead, combined with a novel Demonstration Cloning method that utilizes expert demonstrations to guide the learning of the new agents. MDRL-DC was found to achieve similar, if not better, performance as MDRL-SR despite using a sparse reward, and while consuming 47% less wall-clock time. MDRL-DC has also shown dominance when compared to existing benchmarks, including traditional localization methods and DRL-based methods. Specifically, MDRL-DC is found to achieve up to 63% less localization time and 43% less localization cost, when compared to traditional methods. When compared against works that combine reinforcement and imitation learning, it was seen to achieve 32% less

cost during the localization process, indicating its ability to produce better cooperative agents.

Chapter 5

Blockchain-assisted Demonstration Cloning for Multi-Agent Deep Reinforcement Learning

5.1 Introduction

In this chapter, we build upon the Demonstration Cloning (DC) method introduced in the previous chapter by integrating a blockchain-assisted framework to facilitate the sharing of pre-trained MDRL models across multiple users. As environments become more complex, the need for faster learning and robust collaboration between agents becomes even more critical. However, training agents from scratch in such scenarios is time-consuming and resource-intensive. To address this, we propose a novel Multi-Expert Demonstration Cloning (MEDC) method that leverages models from multiple experts to assist in training MDRL solutions. The blockchain is used for decentralized sharing of pre-trained models. The proposed methods are tested on the target search and localization problem, as well as other similar problems such as maze cleaning and fleet coordination.

Despite its great achievements, DRL comes with several challenges that hinder its usability. One challenge is sample efficiency, which refers to the difficulty of learning from insufficient interactions with the environment, due to the high cost of collecting such interactions. This problem has been recently tackled by several works using Federated Reinforcement Learning (FRL) [57, 82, 83], in which multiple agents learn from their local experiences and periodically share their models to be aggregated in a global model, which is then shared to all agents. Another challenge associated with DRL, and amplified in MDRL, is the curse of dimensionality, which refers to the increasing size and complexity of the state and action spaces with the increasing number of agents [22]. This challenge is further amplified with the use of sparse rewards, which are rewards that are rarely present in the state space of the environment. One common sparse reward function is to assign a single reward value only when the task is successfully completed, such as delivering a checkmate in a game of chess or destroying the opponent team's Ancient in a game of Dota. This increases the difficulty of the learning, as the agents are rarely exposed to the reward, especially in the early stages of the learning where the agents act randomly in the environment to collect experiences. This has been addressed through reward shaping, in which specific reward functions are designed for each problem with the aim of exposing the agents to the reward frequently in the environment [46, 84, 85].

The existing methods in the literature suffer from several drawbacks when tackling the aforementioned challenges. While FRL helps in addressing the sample efficiency issue, faulty and inaccurate shared models could adversely impact the aggregated model (backdoor poisoning attacks), leading to severe difficulty in learning [86]. Additionally, in applications where the models are represented as deep neural networks (DNN), the architecture of the DNN might vary from one user to the other. In FRL, the global model is obtained by averaging the shared models, which cannot be simply achieved if the models are of different architectures, and would require additional steps (such as knowledge distillation)

that increase the computational overhead. This constrains all the FRL nodes to use the same DNN architecture, which is inefficient as some nodes might have capabilities to train more complex architectures than other nodes. In knowledge distillation, a smaller model (student) is trained to mimic the behavior of a larger and more complex model (teacher). While this allows different DNN architectures to learn from each other, knowledge distillation algorithms usually train both the student and teacher models using the same data (i.e. same application), since they are only concerned with reducing the complexity of the model [87]. Therefore, applying the aforementioned methods in RL environments of inherently different dynamics leads to struggles in learning convergence [56]. This is mainly because these methods operate directly on the model weights by averaging models (FRL) or altering weights according to external models (student-teacher), without verifying if the source models are suitable for the current environment dynamics. On the other hand, while reward shaping helps in providing frequent feedback to the agents during training, shaped reward functions require considerable engineering and experimentation, and could frequently lead to unstable learning or convergence to local optima [48, 49].

In summary, existing works suffer from the following drawbacks:

(1) FRL methods can be severely impacted by faulty or malicious models, which affects the performance of the aggregated model.

(2) Typical FRL methods do not allow for different DNN architectures to be used in the same process.

(3) FRL and knowledge distillation methods cannot handle models from environments of different dynamics.

(4) Methods based on reward shaping require considerable engineering and could lead to local optima.

To tackle the aforementioned issues, we propose a Blockchain-assisted Multi-Expert

Demonstration Cloning (MEDC) for MDRL. Inspired by Imitation Learning (IL), MEDC is a novel method that uses expert demonstrations in guiding the learning of new MDRL agents. Rather than averaging MDRL models, the proposed method utilizes experiences from previously trained models (experts) to suggest actions for new MDRL agents to follow during the training. This enhances the quality of the new collected experiences and helps the agents get more exposed to sparse rewards. The proposed method utilizes the suggested actions from the expert models in guiding the learning, without the need of aggregating or averaging, which allows for models of different architectures to be utilized in the same learning process. Additionally, the proposed method is resilient to faulty and malicious expert models, since the new agents are still learning based on their own experiences and rewards, which would reflect bad actions if suggested by experts. To allow for model sharing across different users, a framework based on a Consortium Blockchain is proposed, in which users share or request expert models. Model sharing on centralized servers, as in the typical FRL systems, introduces several issues including single point-point-of-failure, the need of a single trusted server, and the vulnerability to security bottlenecks such as the modification of the information shared by FRL nodes [88]. Unlike centralized cloud computing, Blockchain helps in providing a decentralized, transparent, and autonomous platform for model sharing with no repudiation. A Consortium Blockchain, specifically, provides better privacy, scalability, and efficiency when compared to public Blockchains [89], and better allowance for collaboration and data sharing between entities when compared to private Blockchains, making them suitable for model sharing. The InterPlanetary File System (IPFS) is used to manage the storage of models, and Smart Contracts (SCs) on the Blockchain are designed to manage the model allocation across users, while considering the models' contexts and performances, as well as the users' reputations. In summary, the contributions of this work are as follows:

(1) The design of a novel multi-expert Demonstration Cloning (MEDC), a method that

utilizes experiences from multiple trained models to guide the learning of new MDRL agents.

(2) A Consortium Blockchain-based framework for model sharing across MDRL users, with the assistance of IPFS for storage.

(3) A model allocation mechanism, implemented through smart contracts, which considers the models' quality and users' reputations when assigning models to users.

Collectively, the entire proposed framework is responsible for managing model sharing on the Blockchain, and training through MEDC locally, and is generally described in Fig. 5.1. Users locally train their MDRL models for their respective problems and share models via IPFS to the Blockchain, with the hope of receiving incentives. A user can request a set of models from the Blockchain for a certain pre-determined price. The requested models can be used locally with MEDC. The smart contracts on the Blockchain manage users' registrations and model submissions, in addition to allocating suitable models for users based on their requirements. The owners of the shared expert models are then paid accordingly, which incentivizes users to share their models.

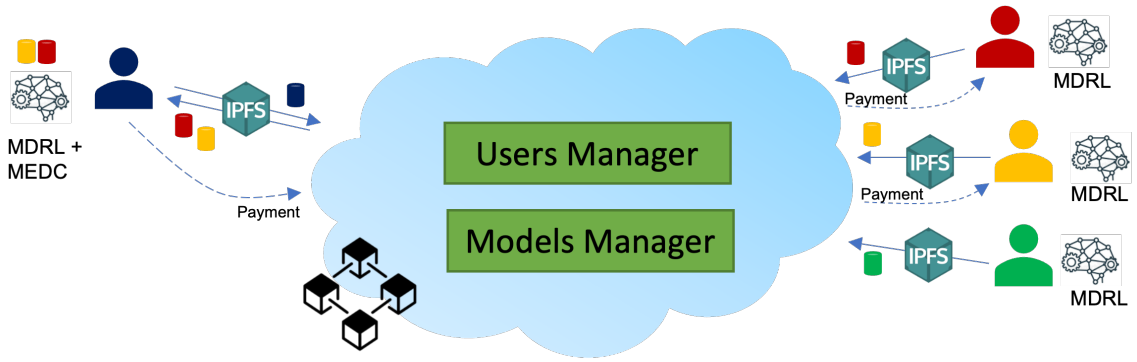


Figure 5.1: A general overview of the proposed framework.

The proposed methods are evaluated for the problem of target localization, where sensing agents are to be trained using MDRL to learn how to localize a radioactive target by progressively searching the area of interest. The environment of the problem could be of

many different complexities, i.e. with varying number of agents or environment obstacles, which represents a suitable scenario for model and knowledge sharing across the different environments of the same problem. The adaptability of the proposed methods is tested on other applications, such as fleet coordination and maze cleaning, which are typical environments used to test MDRL algorithms. The proposed methods are tested and analyzed in terms of learning performance, showing scalability to different environments and resilience towards faulty models. The proposed methods are also benchmarks against works in FRL, Reward Shaping, and IL-assisted RL, showing dominance in terms of performance.

5.2 Multi-Expert Demonstration Cloning (MEDC)

The main idea of the proposed MEDC method is to use demonstrations from experts to guide the MDRL agents into collecting better experiences. In MDRL environments with sparse rewards, better experiences are defined as ones where the agents are more frequently exposed to the sparse reward. In this work, and for a given problem of interest, an expert is defined as a previously trained model with expertise in tackling a similar environment. On the other hand, a semi-expert is one with expertise in tackling a similar environment that slightly differs in complexity. For example, in the problem of target localization with a team of 5 agents in an area that has 3 obstacles, an expert model is one that has been previously trained on a similar environment, while a semi-expert model is one that has been trained on a target localization environment with only a single agent. The semi-expert is not fully capable of tackling the current environment (5 agents), but can still provide some guidance that would be better than acting randomly in the environment. One of the main issues faced in MDRL with sparse rewards is the rare occurrence of reward states, which is dominant in the early stages of the learning where the agents' policies produce randomized actions. The proposed method aims at utilizing demonstrations from experts and/or semi-experts to assist a new MDRL process by providing better experiences with more rewards during the

learning stage.

In this work, the proposed MEDC method utilizes the experiences from multiple experts (or semi-experts) to guide the learning of the new agents, while maintaining the “learning from rewards" concept in RL. In the proposed method, the learning alternates between MDRL and MEDC. During MDRL, the agents take actions in the environment based on their policy networks, and collect rewards accordingly. During MEDC, the experts suggest actions to be followed by the MDRL agents to explore the environment. Here, even though the actions are suggested by the experts, the agents still learn from their own execution of those actions, i.e. based on the collected rewards. As a result, experiences with more frequent occurrence of reward states are expected. This method is resilient to faulty-, malicious-, and semi-experts, i.e. models that might occasionally suggest bad actions, as such actions could be identified through the collected rewards. This is unlike the methods using IL, in which bad- or semi-experts would significantly deteriorate the performance of the new agents who have no input in deciding whether the suggested actions are good or bad.

The learning process of MDRL with MEDC is described in Fig. 5.2, which is an extension to the previously proposed DC method in Section 4.6.2, Chapter 4. At the beginning of each episode, an expert probability R_E determines whether the episode will follow MDRL or MEDC. During MDRL (i.e. the switch is closed), a typical process is followed where agents act based on their copies of the actor network. During MEDC, i.e. when the switch is on, one of the experts is selected, which takes the observations seen by the agents and suggests actions. The selection of the expert, out of the available experts, is done using a roulette wheel based on the attribute R_S, which is a common way used for selection [90, 91]. In this method, each candidate is associated with a probability which is proportional to R_S, where experts with higher R_S have higher probability of being selected. After arranging the experts on a roulette wheel according to their probabilities, the wheel

is spun to randomly select an expert. In this work, the attribute R_S reflects a similarity measure between the environment of the expert and the current environment of interest. The method of computing R_S depends heavily on the application. Some applications favor the number of agents as a measure of similarities, where environments with similar number of agents are favored, while other applications have problem-specific attributes related to the environment dynamics. In this work, we use a QoS (Quality of Service) metric to obtain the value of R_S, which is later discussed in Section 5.4. Experts trained on environments/problems similar to the one of interest are more probable to be selected than ones trained on slightly different environments. Experts with lower similarities are still desired, as they introduce some variance in the experiences collected, which could be beneficial to the learning process. The actions suggested by the expert are then followed by the agents and the corresponding experiences are stored. This method is resilient to faulty-, malicious-, and semi-experts, i.e. models that might occasionally suggest bad actions, as such actions could be identified through the collected rewards.

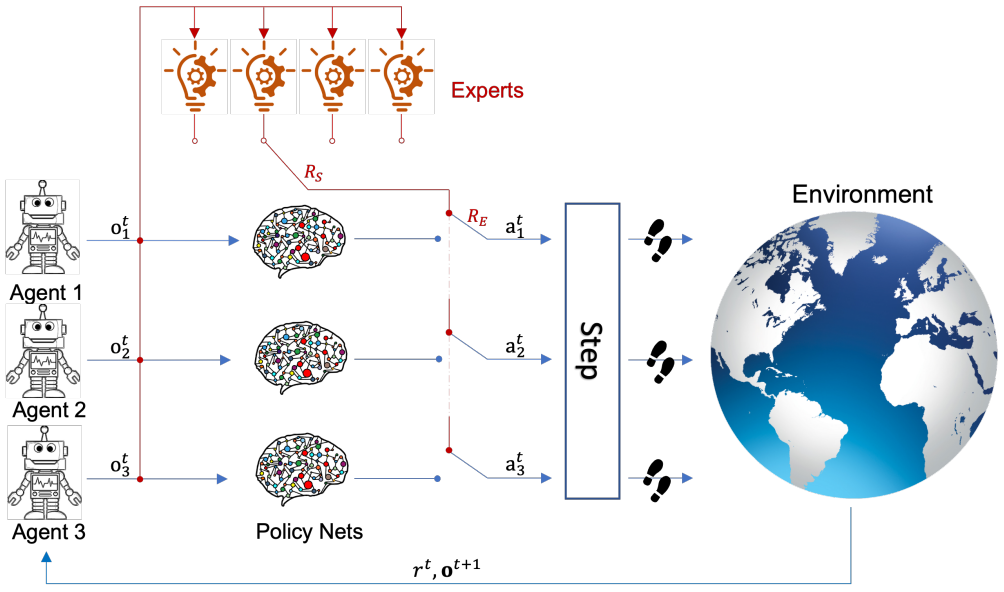


Figure 5.2: The proposed Multi-Expert Demonstration Cloning method.

Since PPO is an on-policy RL algorithm, the sampled actions taken by an agent should

follow the agent's latest policy. If an agent follows actions that are not probable under their own policies, the learning would be unstable. To tackle this in the proposed MEDC method, the actions suggested by an expert are followed by the agents *only if* they meet a probability condition l. Given a probability distribution P generated by the agent's policy over the actions, an action a suggested by an expert is followed by the agent only if $P(a) > l$. During MEDC, the expert ranks the possible actions, and the agent picks the best action that satisfies the l condition. This helps utilize demonstrations to collect better experiences, while not violating the on-policy learning. The value of l should be selected in a way that balances between the expert involvement and the on-policy learning. A very high value of l might limit the benefit of using an expert, as most of the suggested actions would not meet the threshold. On the other hand, a very low value of l results in most of the suggested actions being followed by the agents, even those that are far from the agents' own policy, which could lead to unstable learning when using the on-policy PPO. At later stages of the learning, the agents become more confident of their decisions, and the involvement of experts gets reduced since actions suggested by them will rarely meet the l threshold. This is because the probability distribution P is concentrated in one action, and all other actions would have very low probabilities.

Algorithm 5.1 describes the learning process of MDRL with MEDC. The process is similar a typical MDRL process (Algorithm 3.3), with the addition of MEDC (lines 3 and 7-14) that only affects the action selection process. Before the beginning of each episode, a randomly generated probability (ExpertCheck) determines if the episode follows MDRL or MEDC. If the value does not meet the Expert Rate (i.e. ExpertCheck $\geq R_E$), then the agents follow a typical MDRL process. If the Expert Rate is met (i.e. ExpertCheck $< R_E$), the agents follow MEDC. Here, a roulette wheel selects which expert model to use according to their R_S values. The agents' observations are fed into the chosen expert model,

which returns the suggested expert actions (ExpActions). For each agent k, the expert actions that do not meet the threshold l under the agent's policy distribution P_k are eliminated. The agents then follow the most valued expert actions in the environment to collect experiences. The process repeats throughout the episode, and the collected experiences are then used to update the actor and critic networks.

Algorithm 5.1: Training MDRL with MEDC

Input: Environment dynamics, actor/critic networks, Expert Models, Similarity Measures (R_S), Expert Rate (R_E)

for i = 0,1,...,TotalSteps:
  $\mathbf{o}^0$ = reset()
  ExpertCheck = rand() #random probability
  for j = 0,1,. . . ,EpisodeLength:
    for k = 1,2,. . . ,TeamSize:
      P_k = actor(o_k^j) #probability distribution under agent k's actor
      if ExpertCheck $< R_E$ **then**: #MEDC
        ExpertModel = RouletteWheel(ExpertModels, R_S)
        ExpActions = ExpertModel(o_k^j) #expert suggested actions
        for x = 0,1,. . . ,*len*(ExpActions):
          if P_k[x]$< l$:
            ExpActions[x] = 0 #exclude improbable actions under P_k
        end for
        a_k^j = argmax(ExpActions) #take most probable action
      else:
        a_k^j = sample(P_k) #sample an action from P_k
      end if
    end for
    $\mathbf{a}^j = [a_1^j, a_2^j, \dots]$
    $\mathbf{o}^{j+1}, r^j, f^j$ = Step($\mathbf{a}^j$)
    Store $\mathbf{o}^{j+1}$, $\mathbf{a}^j$, and r^j
    if i % H == 0 **then**: #PPO update
      Compute $\hat{A}$ using the collected experiences
      Use $L^{CLIP}(\theta)$ to update the actor and critic networks
    end if
    if $f^j == 1$ **then** break #if the episode is done
  end for
end for

5.3 Blockchain-based model sharing for Demonstration Cloning

This work proposes a Blockchain-based framework that complements the MEDC method. The framework is responsible for managing the users' registration, model submission, and appropriate model allocation to requesting users. A Consortium Blockchain is used due to its ability to offer increased privacy, shared control, efficiency, cost savings, and trust for multiple organizations or entities collaborating on a project or sharing data [92]. A Consortium Blockchain is operated by a group of entities, which introduces increased privacy and trust when compared to public Blockchains, and more collaboration allowance when compared to private Blockchains. This makes Consortium Blockchains suitable for data sharing across entities, and hence suitable for the purpose of model sharing in the proposed framework.

A high-level view of the proposed framework is shown in Fig. 5.3. The framework is built using two smart contracts: 1) Users Manager Contract (UMC) and 2) Models Manager Contract (MMC). The users interact with the UMC to register in the system and add their information. Users could also share their trained models by interacting with the MMC, with the hope of receiving incentives if the models are requested and used by other users. A user could share their trained model on IPFS, which returns a unique Content Identifier (CID) that can be used to access the file. The IPFS is a protocol designed to create a content-addressable Peer-to-Peer (P2P) decentralized file system [93]. Users share the CID with the MMC when submitting a trained model, along with information describing the problem and the environment details. In the proposed framework, the user indicates the problem of interest from a set of pre-defined problems. Additionally, the environment details are given as a tuple of pre-defined features that the user needs to specify. For example, in the field of target localization, the problem is defined as "target localization",

while the environment details could be given as (Number of agents = 3, Number of targets = 1, Number of obstacles = 3). A user requests a set of models by interacting with the MMC and specifying the requirements. The MMC allocates suitable models after the requester submits the required payment, which is shared with the model owners. This incentivizes users to share their models with the hope of receiving payments. To further incentivize users to share efficient models, extra payments could be given if the shared model meets performance requirements. Alternatively, instead of paying fixed amounts for the shared models, model owners could have different valuations for their shared models based on their complexities and performances. Specific cost-efficient incentive mechanisms that motivate model owners to be truthful about their valuations are outside the scope of this work, but some potential works include auction-based incentive mechanisms [94, 95]. For example, one possible way is to use variations of reverse auctions, where model requestors post their requirements on the blockchain, and model owners bid lower costs progressively. Another example is using a variation of second-price auctions, where model owners give valuations for their models and the lowest bid wins and gets picked by the requestor but gets paid the second lowest bid [96].

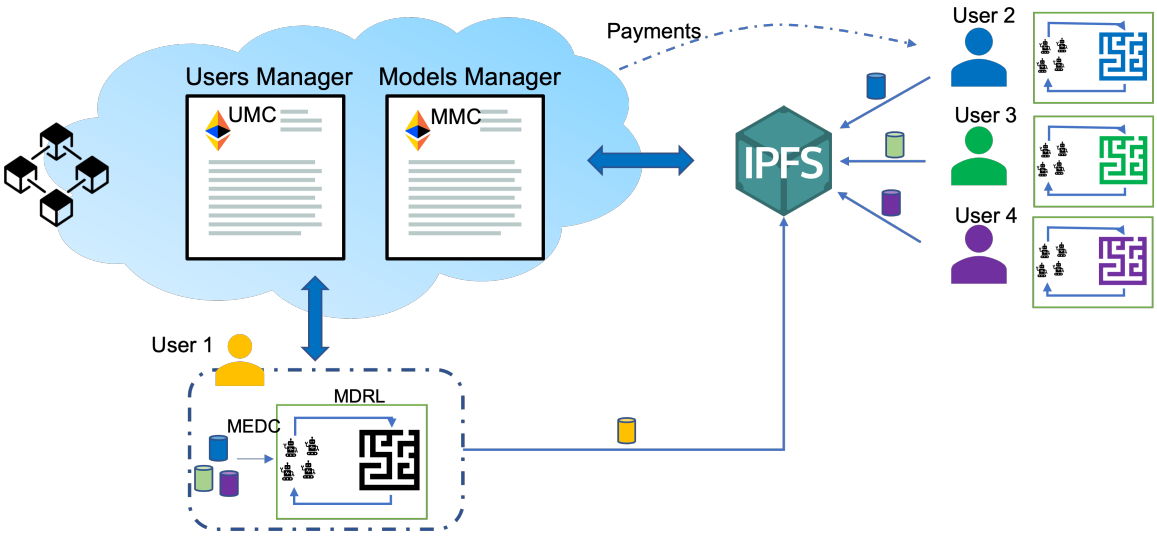


Figure 5.3: The proposed Blockchain-assisted model sharing framework for Demonstration Cloning.

5.3.1 Smart Contract Implementation

The details of the User Manager Contract (UMC) are shown in Table 5.1. The *User* data structure is designed to hold user information, including their Ethereum address and reputation. The *Models Alloc. Count* reflects the number of times the user's models have been allocated to other users/requesters, while the *Total Review* reflects the numerical sum of all the reviews submitted by requesters upon using the user's previously submitted models. Using these information, user i's reputation is computed as

$$Rep_i = \frac{Total\ Review}{Models\ Alloc.\ Count}. \tag{10}$$

The UMC keeps users' information in the *Users List* mapping, which maps a user's address to their *User* object. The *addUser()* function is responsible for registering users by creating a *User* object and mapping it in the *Users List*. The *updateInfo()* function is invoked when one or more of the user's shared models are allocated to requesters, in order to update the user's information (*Models Alloc. Count)*. The *updateReputation()* function is responsible for updating the user's reputation as per Eq. 10.

Table 5.1: Users Manager Contract (UMC)

Data Structure		
User		
User Address (address)	*Reputation (uint)*	
Models Alloc. Count (uint)	*Total Review (uint)*	
Variables		
Users List (*address* $\rightarrow$ *User*)		
Function	**Parameters**	**Return**
addUser()	*address*	-
updateInfo()	*No. Models*	-
updateReputation()	*Review*	-

The details of the Models Manager Contract (MMC) are shown in Table 5.2. The *Model* data structure is designed to hold the information of a submitted MDRL model. This

includes the *Owner*'s Ethereum address, the *CID* generated by IPFS, the *Model Reputation*, the *Allocation Count* of the model indicating the number of times the model has been allocated to requesters, the *Total Model Review* based on previous requesters, a general *Description* of the model indicating its architecture and input requirements, the *Application* for which the model has been trained, and a tuple/array of *Environment Details* giving values to each of the specific environment details. The MMC keeps the details of the shared models in a *Models List* mapping, which maps an application to all the available models shared by users. The *addModel()* function is responsible for registering the details of a newly submitted model. The *allocateModel()* function is responsible for allocating models to a requester based on their requirements. The *updateModelRep()* function is responsible for updating the model's reputation following a submitted review by a requester, which is computed for model m as

$$Rep_m = \frac{Total\ Model\ Review}{Allocation\ Count}. \tag{11}$$

Table 5.2: Models Manager Contract (MMC)

Data Structure		
Model		
Owner (address)	*CID (string)*	
Model Reputation (uint)	*Allocation Count (uint)*	
Total Model Review (uint)	*Description (string)*	
Application (string)	*Environment Details (uint[])*	
Variables		
Models List (string → Model[])		
Function	**Parameters**	**Return**
addModel()	*Model Info*	-
allocateModels()	*Model Requirements*	*Model[]*
updateModelRep()	*Review*	-

The *allocateModel()* function employs a Greedy method to allocate models to the requester. A requester specifies a set of requirements including the application, the minimum

model reputation, the minimum owner reputation, the desired environment details, and the number of models desired. The owner's reputation Rep_i is important to consider as it reflects the owner's historical performance, which is essential especially when the current shared model is new and has not been reviewed yet. On the other hand, the model's reputation Rep_m reflects the effectiveness of the shared model in guiding the learning, as per previous requesters. It is assumed that a requester pays the same amount of incentives for each requested model, and hence the number of models requested reflects the available budget. Models within the same application that do not meet the reputation requirements are filtered out. The rest of the models are ranked based on the following Quality of Service (QoS) metric:

$$QoS_m = \frac{Rep_i \times Rep_m}{D_m}, \tag{12}$$

where QoS_m is the QoS of model m, Rep_i is the reputation of m's owner, Rep_m is the reputation of model m, and D_m is a similarity measure between the environment in which model m has been trained and the requester's environment. Given a set of p environment attributes $E = [E_1, E_2, \ldots, E_p]$, the value of D_m is computed as

$$D_m = \sum_{i}^{p} w_i \times \left| E_i^m - E_i^r \right|, \quad \sum_{i}^{p} w_i = 1, \tag{13}$$

where E^m is the set of environment attributes associated with the model, E^r is the set of attributes associated with the requester's environment, and w_i are weighting parameters. The weights w_i in Eq. 13 reflect the importance of each attribute. Higher values give more significance to the difference between the environment of the model and that of the requester, which lowers the QoS if that difference increases. All environment attributes are given equal weights by default, unless specified otherwise by the requester in the *Model Requirements* when invoking the *allocateModel()* function.

5.3.2 Framework Time Sequence

Fig. 5.4 shows a time sequence diagram for a scenario under the proposed Blockchain-based framework for MEDC. It presents the interactions between the users and the smart contracts constituting the framework, which are given as follows:

- **User Registration**: Users register to the UMC by invoking the *addUser()* function. The function creates a *User* object and appends it to the *User List*. Each user is assumed to have a single Ethereum address linked to their account. The reputations are initialized with a value of 0.5, while *Models Alloc.* *Count* and *Total Review* are initialized with a value of 0.

- **Model Training (MDRL)**: Users locally train MDRL models for their respective problems/applications. This step could occur at any time (before or after registration). Previously trained models could be stored by the users and shared later.

- **Model Sharing**: Users who wish to share their models upload the model files to IPFS, which returns a CID for each submitted model. The users then submit model details to the MMC by invoking the *addModel()* function and passing the required information. The MMC creates a *Model* object and appends it to the *Models List*.

- **Model Allocation**: A user who wishes to request a set of expert models communicates with the MMC by invoking the *allocateModel()* function and passing the requirements. The MMC runs the allocation mechanism and returns the models and their details to the user. The user accesses the models files on IPFS using the provided CIDs.

- **Model Training (MDRL + MEDC) and feedback**: The user utilizes the proposed MEDC with the allocated expert models to help in training MDRL agents for the problem of interest. The user then submits feedback reviewing the obtained models, which are used to update the models' and users' reputations.

- **Payments**: The owner of each requested model is paid a pre-determined amount in return for the shared model.

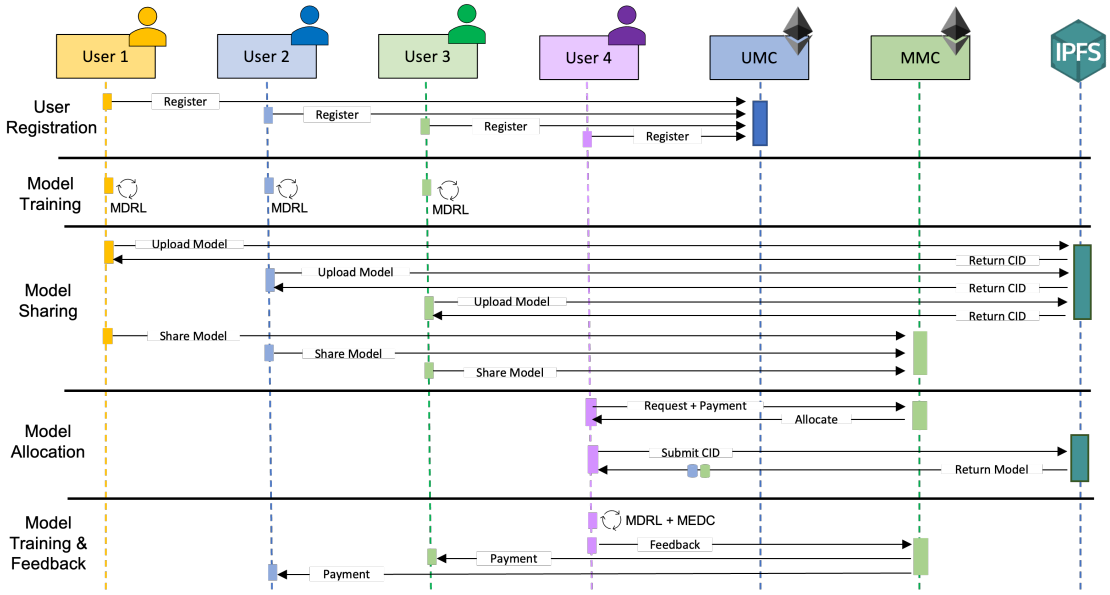


Figure 5.4: The interactions between the users and smart contracts as part of the proposed framework.

5.4 Simulation and Evaluation

This section presents and discusses different experiments conducted to validate the proposed methods. The experiments are first conducted in a custom environment for a task of Target Localization, which is a complex multi-agent problem requiring agents to cooperate in finding the target location. Subsequently, the adaptability of the proposed method is tested across two additional typical Multi-Agent applications, which are Fleet Coordination for Autonomous Vehicles [97, 98] and Multi-Agent Maze Cleaning [99]. All the simulations have been conducted using an Intel E5-2650 v4 Broadwell workstation equipped with 128 GB RAM, 800 GB SSD, and NVIDIA P100 Pascal GPU (16 GB HBM2 memory).

Target localization is chosen as a real-world problem to test the proposed methods due to the different possible variations the environment, which would be a good fit to test the

proposed MEDC method and compare it with FRL. The environment could vary in terms of the number of agents or the number of walls, where each combination of agents and walls gives a different learning problem with its own complexity. In the following experiments, we first conduct analysis on the proposed MEDC and its effectiveness in achieving good learning, in addition to its resilience to faulty or malicious models, then compare it against existing benchmarks. Table 5.3 shows the training hyperparameters used with PPO and MEDC. Here, the expert similarity measure R_S that is used to determine the usage rate of each expert model is proportional to the QoS of that model. We later extend the applicability of the proposed methods to other applications such as fleet coordination and maze cleaning.

Table 5.3: Hyperparameters used for PPO and MEDC.

PPO Hyperparameters	Value
Learning rate	3×10^{-4}
PPO clipping parameter ε	0.2
Entropy coefficient c_2	0.01
Discount factor γ	0.99
Discount factor λ	0.95
Timesteps per update (Horizon H)	4000
Number of epochs per update	20
MEDC Hyperparameters	**Value**
Expert Rate R_E	0.1
Expert Similarity R_S	$\frac{QoS_i}{\sum QoS_j}$
Action Probability Threshold Q	0.05

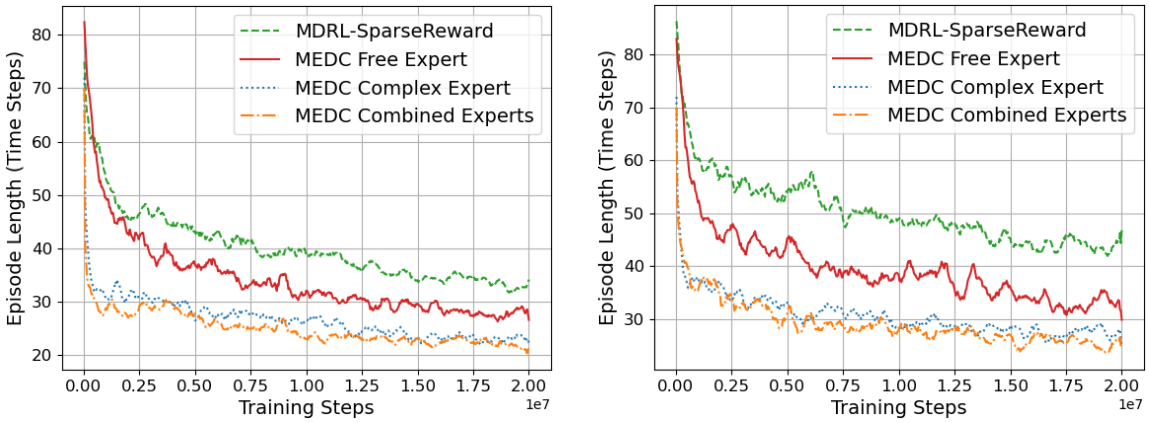
5.4.1 Performance of MEDC

This section analyzes the performance of the proposed MEDC method and its effectiveness in guiding the learning for new agents in an MDRL system. During training, a localization episode has a maximum length of 100 steps, and the total number of training steps is given as 2×10^7. An episode terminates when the target is found, or when the

maximum episode length is reached. After every 40,000 training steps, the average results of 4,000 testing steps, where the agents act greedily based on the latest policy update, are recorded and plotted. In this work, the target localization problem is characterized by two features: the number of agents and the number of walls. We report the results in terms of episodic length throughout the learning, as it reflects the time needed by the agents to localize the target, which is the main aim of target localization. We use the notation $AyWz$ to denote an environment with y agents and z walls.

To show the effectiveness of MEDC in enhancing the learning and tackling issues with sparse rewards, Fig. 5.5 compares variations of an environment trained using MDRL and MEDC with one that is trained using MDRL with sparse rewards. Here, the problem of interest is one with 3 agents aiming to localize the target in an environment of 2 walls (A3W2) or 3 walls (A3W3). Three types of experts are used to guide the learning: a "Free Expert" is a model previously trained to tackle an environment with a single agent and no walls (A1W0), a "Complex Expert" is a model previously trained to tackle an environment with a single agent and 3 walls (A1W3), and "Combined Experts" refers to using both aforementioned experts at the same time in MEDC to guide the learning. It is evident that the used experts come from environment that are not exactly the same as the current environments of interest. Here, a better learning is indicated by a smaller episode length, as it reflects faster localization. As shown in Fig. 5.5, using MEDC with experts from similar environments to guide the MDRL agents helps in achieving better and faster learning, when compared to training MDRL independently using a sparse reward. Even though both experts are not familiar with a multi-agent environment, and can only tackle a single agent environment (hence they are not familiar with cooperation), they are partially beneficial in guiding the new agents into collecting better experiences, especially in the initial stages of the learning. Additionally, it can be seen that using the complex expert (A1W3) gives better results than using the free expert (A1W0), because it is trained in an environment

closer to the current environment of interest. This reflects the importance of considering the QoS (Eq. 12) when allocating expert models. In this scenario, and assuming that Rep_m and Rep_i are constant across both experts, the complex expert would have a higher QoS than the free expert since it has smaller D_m values (assuming the features have the same importance).



(a) Number of walls = 2

(b) Number of walls = 3

Figure 5.5: The episodic length throughout the learning for an environment of 3 agents and (a) 2 walls and (b) 3 walls.

To study the resiliency of the proposed MEDC method, the following experiment uses faulty and malicious experts. Fig. 5.6 shows the learning performance in an environment of 3 agents and 2 walls, while using the following faulty experts:

- Random Expert: a model that suggests random actions in the environment.
- Biased Expert: a model that always suggests the same action regardless of the collected observations.
- Malicious Expert: a model that is aware of the right action, but suggests the opposite action (opposite direction).

For each MEDC scenario, 5 of the mentioned expert types are used. For example, in "MEDC Random Experts", the agents are being trained using MDRL and MEDC with

5 Random Experts. "MEDC Proper Experts" is a scenario where 5 good experts are used from different environments, while "MDRL-Sparse" shows a scenario of training the MDRL agents independently using a sparse reward. As seen in the figure, the proposed MEDC is resilient to faulty and malicious expert models. At worst, the performance of MEDC drops to match that of "MDRL-Sparse". This means that, the use of experts in MEDC could either help the learning in MDRL or cause no harm. This is mainly because the experts are used for a small portion of the learning ($R_E = 0.1$). Additionally, the main purpose of the experts is to expose the agents to the sparse reward more frequently. If this does not occur, then the agents are still learning based on their own experience, even if these experiences are induced by the experts, which explains why the performance at worse is similar to the MDRL with sparse reward and no experts.

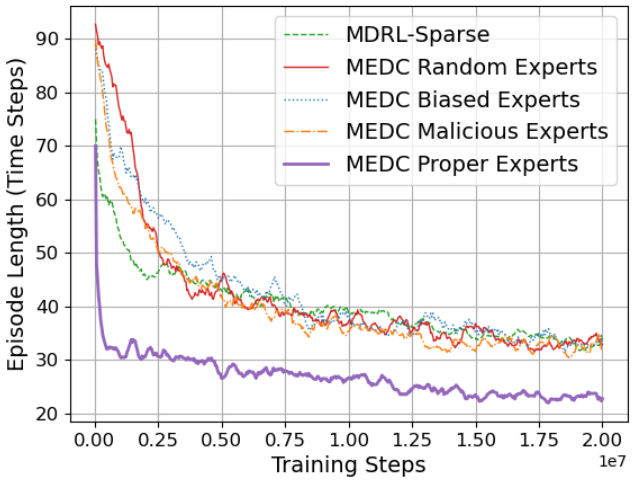


Figure 5.6: The episodic length throughout the learning for an environment of 3 agents and 2 walls, while using different faulty and malicious experts.

5.4.2 MEDC vs Benchmarks

This section discusses and highlights the main advantages of MEDC when compared frameworks in FRL, Reward Shaping (RS), and IL-assisted RL. The benchmarks are summarized as follows:

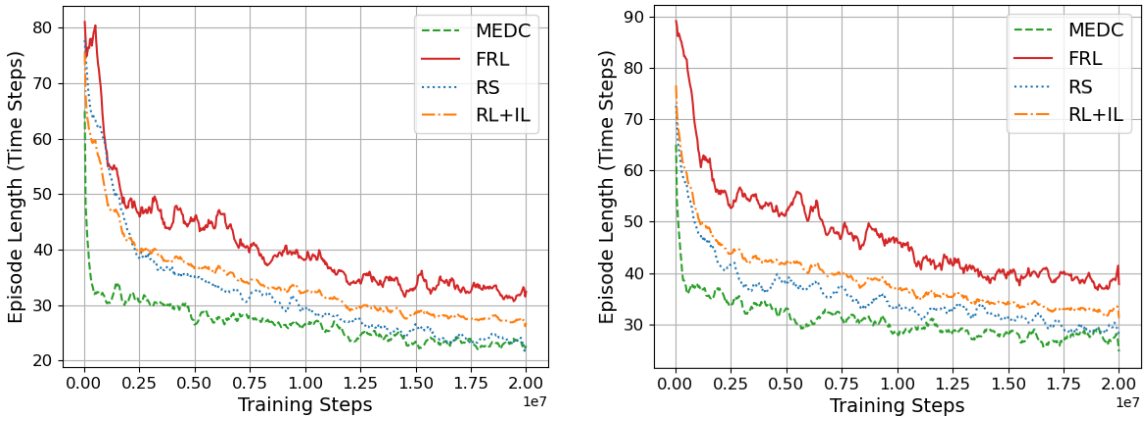
- In FRL, models from different users are averaged frequently in a global model, which is shared back to them. We benchmark with the works in [54, 55], where FRL is used to combine DRL models across different users.

- In Reward Shaping (RS), a shaped reward function is used to guide the learning. Here, we use a distance-based reward function, where the agents receive a positive reward in each step if they move closer to the target during the training, which is similar to the works in [2, 44]. In each episode, the distances are computed using Breadth-First Search (BFS). BFS is used due to the existence of walls, which makes simpler distance measures (such as Euclidean distance) inapplicable.

- In IL-assisted RL, the learning alternates between RL and IL, similar to the works in [17, 50]. For fairness, We extrapolate this work to use the same structure of MEDC with multiple experts. The main difference is, when IL is switched on, the MDRL agents use Behavioral Cloning (BC) to mimic the behavior of the expert without any reward input.

Figure 5.7 compares Blockchain-assisted MEDC with the 3 benchmarks. For MEDC and FRL, the learning involves 8 users of varying environments. The 8 users are training on different environments, given as A1W0, A3W0, A2W1, A2W2, A2W3, A3W1, A3W2, A3W3. For FRL, during the training, the models are shared and a global model is returned to each user to carry on the training. In MEDC, the first 6 users share their models to the Blockchain, which are then used by A3W2 and A3W3 in MEDC to guide the learning. For IL-assisted RL, the models of the first 6 users are used as experts, from which the MDRL agents in A3W2 and A3W3 choose to imitate. We show the training results from the perspective of the A3W2 and A3W3 users. As seen in the figure, the performance of the models trained with MEDC is significantly better than that of FRL. As discussed in Section 5.1, if the models are trained in environments that inherit different dynamics, the

learning convergence of the aggregated model could face issues [56], which is seen in the obtained results. This is unlike MEDC, in which the expert models only suggest actions, and the MDRL agents still learn based on their experience. Similarly, MEDC outperforms IL-assisted RL, whose performance is negatively affected by experts being from different variations of the environment. This is because the agents in IL-assisted RL blindly clone the behavior of the expert into their models. Additionally, the performance of MEDC when compared to the model trained on a shaped reward shows slight outperformance. Even though RS gives a similar performance towards the end of the training, it is worth noting that the model trained with the BFS shaped reward required *double* the wall time to train for 2×10^7 steps. This due to the computational power wasted on executing BFS in each episode, which reflects the complexity of most shaped rewards. MEDC, on the other hand, is able to achieve similar, if not better, results, with a simple sparse reward.



(a) Environment: A3W2 **(b)** Environment: A3W3

Figure 5.7: The episodic length throughout the learning for an environment of 3 agents and (a) 2 walls or (b) 3 walls, while comparing MEDC with the benchmarks.

To evaluate the resiliency of MEDC in comparison with FRL and IL-assisted RL, Fig. 5.8 compares the methods in the same scenario of 8 users, out of which 3 random users have the faulty experts discussed in section 5.4.1. It is evident that the proposed MEDC method is more resilient and is able to maintain faster and better learning than FRL and IL-assisted RL, which are negatively affected by the bad actions suggested by the faulty

and malicious experts.

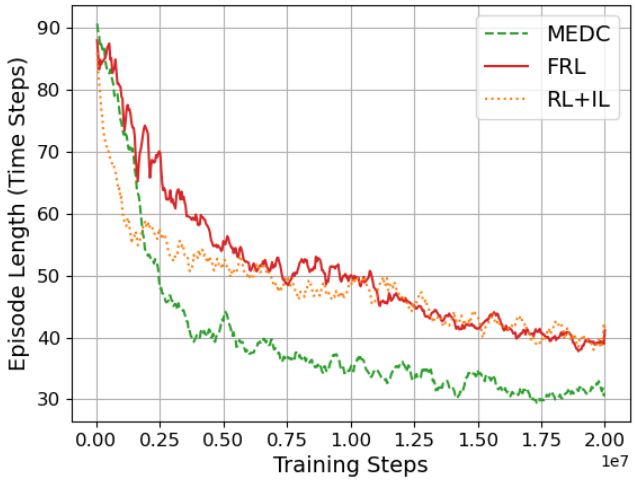


Figure 5.8: The episodic length throughout the learning for an environment of 3 agents and 2 walls, while using faulty and malicious experts.

5.4.3 Adaptability to Other Applications

This section analyzes the adaptability of the MEDC method to the following applications:

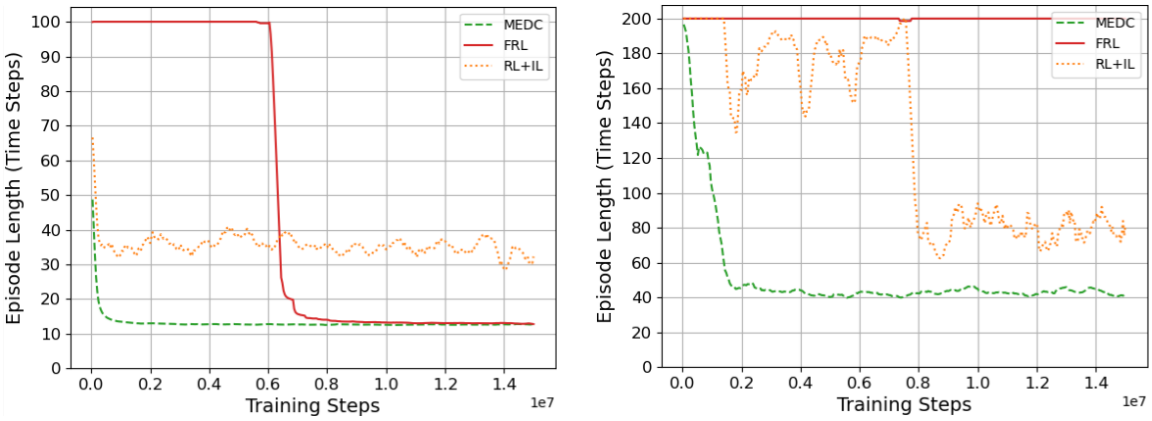
- Fleet Coordination for Autonomous Vehicles [97, 98]: in this problem, a team of autonomous vehicles is tasked with picking up and dropping costumers at specific locations. Each vehicle has a limit in terms of the number of costumers it can accommodate simultaneously. At the beginning of each episode, all agents (vehicles) and costumers are placed randomly in the environment, with each costumer having a desired destination. The vehicles are tasked with cooperation and coordination, such that the time needed to drop all costumers at their destinations is minimized. This requires the agents to learn how to divide the costumers according to their locations. The agents receive a small reward for picking up a costumer, a large reward for dropping all costumers correctly, and a small negative reward for each time step to

account for time cost. The agents observe their locations, the costumers' locations, and the desired destinations in a 2D format similar to the target localization problem.

- Multi-Agent Maze Cleaning [99]: in this problem, the agents are placed in a maze with the task of cleaning the maze as fast as possible. At the beginning of the episode, a maze is randomly generated, where the maze is considered entirely dirty initially, and each spot covered by a cleaning agent is considered to have been cleaned. The agents are required to coordinate and distribute in the maze in a way that minimizes the time needed for the maze to be fully cleaned. Each agent observes its own location as well as other agents' locations, in addition to the maze cleaning status in 2D format. At each step, the team gets a positive reward for each spot cleaned, and a negative reward as time cost.

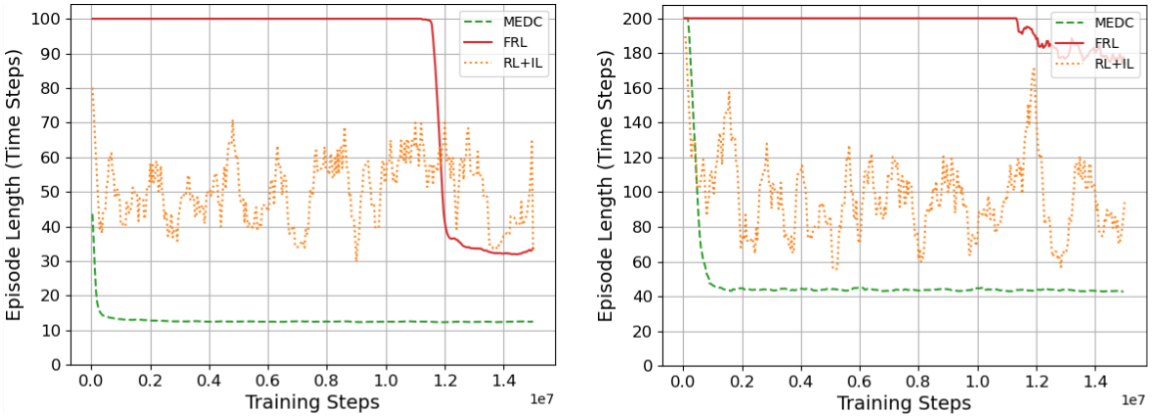
The two environments are simulated using MEDC against the FRL and IL-assisted RL benchmarks. Both environments are simulated for a 10×10 grid and 3 agents, with the fleet coordination environment having 8 costumers. The episode length is used as a metric to assess the learning process, since both applications require the tasks to be finished as fast as possible. Similar to process describe in Section 5.4.2, MEDC and IL-assisted RL use 6 previously trained single- and multi-agent experts from environments of varying complexities (i.e. different number of agents and costumers, and different maze complexities) to assess in training. For FRL, 7 users of varying environments are used in each application, where users share their models to update a global model that is then returned to the users. We report the performance from the perspective of a single user with an environment of 3 agents. Fig. 5.9 and Fig. 5.10 compare MEDC with benchmarks for the cases of truthful and faulty experts, respectively. Similarly to the results obtained for the target localization environment, it can be seen that MEDC outperforms the benchmarks for both environments, and under the two scenarios of truthful and faulty experts. MEDC achieves faster convergence and better performance, in terms of episodic length, even with the existence of faulty and

malicious experts, showing the resiliency of the proposed method.



(a) Fleet Coordination **(b)** Maze Cleaning

Figure 5.9: The episodic length throughout the learning for the (a) fleet coordination and (b) maze cleaning environments.



(a) Fleet Coordination **(b)** Maze Cleaning

Figure 5.10: The episodic length throughout the learning for the (a) fleet coordination and (b) maze cleaning environments, while using 3 faulty experts.

5.4.4 Smart Contracts Complexity Analysis

This section analyzes the complexity of the developed smart contracts in terms of gas cost. Since a Consortium Blockchain is proposed, the deployment and execution of the smart contracts do not require any payments by the users. However, the gas cost is a good measure of the complexity of the smart contracts to indicate their feasibility. Table 5.4

presents the gas cost of the deployment and execution of the smart contracts and their functions. As seen in the table, the gas costs are low, reflecting the feasibility of the proposed contracts. For reference, we present a benchmark of the gas cost of deploying and executing a similar UMC smart contract as discussed in [92].

Table 5.4: Blockchain gas cost.

Contract	Function	gas cost
UMC	deployment	433933
	addUser()	35857
	updateInfo()	67582
	updateReputation()	77286
MMC	deployment	1557311
	addModel()	287448
	allocateModels()	43842
	updateModelRep()	79374
UMC - Benchmark	deployment	1228566
	addUser()	352352

5.5 Conclusion

In this chapter, the problems of sample efficiency and reward sparsity in Multi-Agent Deep Reinforcement Learning systems is tackled. A novel Blockchain-assisted Multi-Expert Demonstration Cloning (MEDC) framework is proposed, in which users share trained models to be used as expert models by other users in the MEDC method. The proposed MEDC method utilizes expert models to suggest actions to new MDRL agents, aiming to provide better experiences where the sparse reward is more frequently obtained, which speeds up the learning. Unlike methods such as FRL, the proposed MEDC method is more resilient to faulty and malicious shared models, and allows for models of different architectures to be used together. On a Consortium Blockchain, smart contracts are designed

to manage the model sharing and allocation process using a Greedy method, in which attributes about the model and the users are used in the assignment process. Experiments in the target localization environment show that the proposed MEDC method speeds up the learning noticeably when compared to models trained only with sparse rewards. Additionally, it was shown that the MEDC is resilient to faulty and malicious expert models, where the performance of MEDC could be the same as MDRL with sparse reward at worse. When compared to FRL, RL with shaped rewards, and IL-assisted RL, MEDC showed dominance in terms of utilizing experiences from different environments in guiding the learning, and in being resilient against bad expert models. Moreover, the adaptability of the proposed methods is tested on two other multi-agent environments, namely Fleet Coordination and Maze Cleaning, showing dominance in learning convergence and resiliency to faulty models, when compared to FRL and IL-assisted RL. Finally, the developed smart contracts that manage users' and models' allocations are analyzed in terms of gas cost and compared to benchmarks, showing low cost that reflects their feasibility.

Chapter 6

Adaptive Target Localization under Uncertainty using Multi-Agent Deep Reinforcement Learning with Knowledge Transfer

6.1 Introduction

In this chapter, we extend the challenge of target search and localization to cover scenarios involving uncertainties, such as false alarms and unreachable targets due to obstacles. In complex environments, agents often face situations where the target may be physically blocked by obstacles, or the search process might be complicated by false signals indicating targets where none exist. These uncertainties significantly complicate the decision-making process, requiring agents to not only navigate the environment but also discern whether a target is reachable or even present.

To tackle these challenges, we propose novel MDRL methods that incorporate multi-dimensional decision-making. Agents must assess target reachability, flag false alarms, and modify their search strategies accordingly. A key innovation in this approach is the use of Transfer Learning (TL) to reuse the encoder from pre-trained MDRL models. This allows us to create a final model with multiple action heads, each responsible for different action dimensionalities (e.g., mobility, flagging, and localization), while sharing a common encoder. This design not only reduces the computational burden but also enables efficient reuse of learned representations, ensuring that the model is capable of handling complex, multi-dimensional tasks with minimal redundancy. The key contributions of this work are summarized as:

(1) The formulation of the target localization problem in uncertain environments using MDRL.

(2) The design of a reward function and decision-making actions over three dimensionalities that determine agents' mobility, target detection, and target reachability.

(3) The development of a learning framework based on TL that leverages the knowledge of the trained MDRL model in training a DL model for target estimation, allowing for reduced computational overhead.

The proposed method is assessed within the context of radiation localization, in which a team of sensing agents is tasked with finding the location of a radioactive material by intelligently searching the area. The proposed work is compared with multiple target localization benchmarks [11, 15, 17, 40, 44, 45] in terms of the learning performance and target localization efficiency.

6.2 Proposed System

This section presents the MDRL method used to intelligently produce sensing agents capable of tackling the different complex scenarios of target localization. An overview of the final proposed model is presented in Fig. 6.1. At each timestep, and for a given agent, the MDRL model translates its observations into one of three possible action dimensionalities: Movement, Detection, and Reachability. Movement actions are responsible for controlling the mobility of the sensing agent in the environment. A detection action is concerned with flagging the existence of a target in the AoI, with the aim of conserving resources if the target does not exist. A reachability action determines if a target exists but is unreachable due to environment complexity. Given that the target is unreachable, an estimation process is triggered, which is responsible for estimating the location of the unreachable target. The key challenge here is to produce all actions, as well as the estimation process, in one AI model capable of translating an agent's observations, while ensuring its cooperation with other sensing agents. This section formulates the MDRL problem and discusses the modeling of the MDRL methods used to obtain the final model.

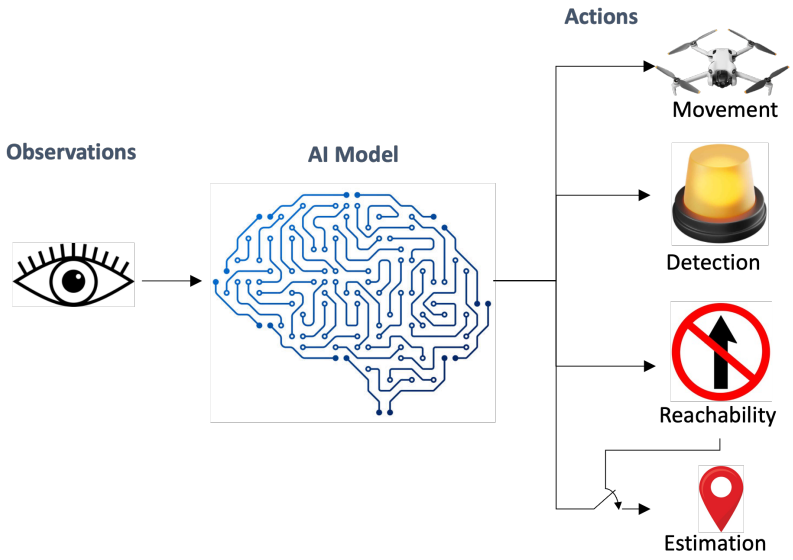


Figure 6.1: An overview of the model proposed, which is to be deployed on each sensing agent.

6.2.1 Observation and Action Spaces

In this work, we re-use the same method of modeling the observation space presented in Section 4.3 and Fig. 4.2. The original observations, i.e. the location map, team locations map, readings map, visit counts map, and walls map are processed into local and global observations, which are fed into the policy for decision-making. The global observation of the walls map is obtained using a Convolutional AutoEncoder (CAE) with the structure shown previously in Fig. 4.3.

To address the complex scenarios of target localization, the action space at a given step for an agent is divided into mobility, detection, and reachability actions. In terms of mobility, the agent has a fixed speed and decides on the direction of movement. Given B possible discrete directions $\{1, 2, ..., b_i, ..., B\}$, the movement angle is given as:

$$\theta = 2\pi \frac{b_i}{B} \tag{14}$$

where the hyperparameter B determines how detailed the movement is. In this work, we use $B = 8$, which allows the agent to move in one of the cardinal or ordinal directions. Assuming a fixed speed, the agent can move a fixed distance in one of these directions with the aim of contributing to the localization task. It has been found in this work that discretizing the direction of movement into 8 possible values is sufficient. On the other hand, choosing to stay idle, if needed, helps in preserving resources.

The detection and reachability actions are each represented by a binary value to flag the existence of the target and its reachability, respectively. At a given time step t, an agent could determine that the target does not exist and hence set the existence flag to 1. Similarly, an agent could determine that a target is unreachable by setting the reachability flag to 1.

The decision-making process is discretized, where at each step, an agent can choose only one of the 11 actions. If the majority of the agents declare that the target does not exist,

the search process stops. Similarly, if the majority declare that the target is unreachable, a target estimation process is triggered, which estimates the target location (to be explained later in Section 6.2.3). In certain time steps, some actions may not be possible, such as moving outside the boundaries of the AoI or into a wall. Such invalid actions are masked out during the decision-making process, where the agent only chooses from the available actions.

6.2.2 Policy Networks and Learning Process

In this work, we use PPO to train the MDRL agents, with CNNs representing the actor and critic since the observations of each agent are represented as 2D maps. CNNs are crucial for the target localization problem as they effectively correlate spatial features within the input maps.

Fig. 6.2 shows the architecture used for the actor (upper part of the figure), which is similar to the LeNet-5 architecture [74]. The network takes the first 9 reduced observations as input, which are then processed in convolution and max pooling layers for feature extraction. The embedding for the environment layout observation is concatenated with the processed and flattened observations, before being fed into the fully connected layers. The fully connected layers produce 11 outputs, which are fed into a softmax function that produces a probability distribution for the possible actions. During the decision-making process, the agent samples an action from this distribution, which is then executed in the environment.

The proposed MDRL method uses Centralized Learning and Distributed Execution (CLDE) [41]. Here, a copy of the actor is given to each agent, where agents act independently based on their observations. During the training stage, a centralized critic is used to assess the experiences collected by the agents. The architecture of the critic is similar to that of the actor, but with 1 output representing the value function. Using a centralized

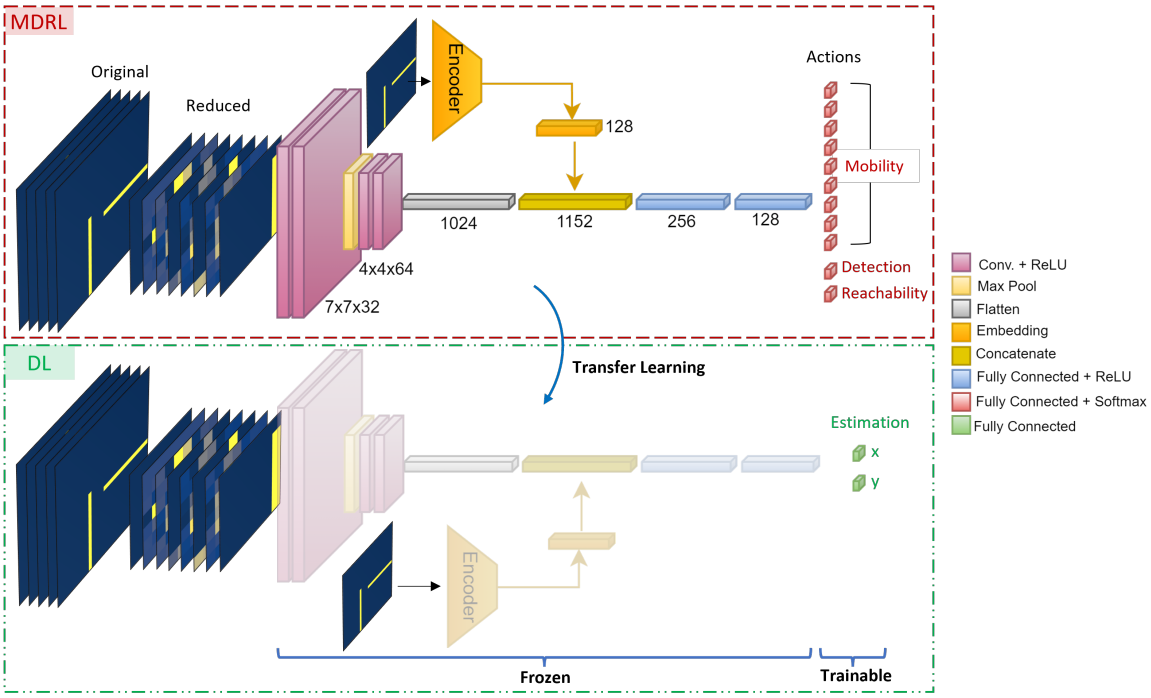


Figure 6.2: The actor architecture trained through MDRL (top), and the architecture of the estimation model trained through DL and TL (bottom).

critic is essential in addressing the non-stationarity problem that is common in MDRL due to the influence agents have on each other's view of the environment [73]. The value function, which is the critic's output, is used during the PPO process to update the actor and the critic.

A *team-based shaped* reward function is proposed in this work to guide the learning. A team-based (joint) reward function gives equal rewards to the agents according to the collective behavior. As a result, the agents would be motivated to act in a way that benefits the entire team. A shaped reward gives more frequent feedback to the agents during the task, which helps speed up the learning process. Following the joint action taken by the agents in a given time step, the environment returns to all the agents a similar reward. To achieve all the desired behaviors, the reward function after step t is given as:

$$R_t = \begin{cases} -Q & \text{if flags are incorrect} \\ -v+1 & \text{if } \min(D_t) < \min(D_{t-1}) \\ -v-1 & \text{otherwise} \end{cases} \tag{15}$$

where Q is a large penalty, v is the total number of mobility actions taken, and D_t represents the set of distances between the agents and the target. In the function, the first condition gives the team a high negative reward (in this work $Q = 500$) if a target is incorrectly declared non-existent or unreachable. Alternatively, in the second and third conditions, the agents are rewarded according to their proximity to the target and their resource consumption. At each timestep, the agents are penalized by $(-v)$, where v is the number of agents who moved in the environment. This indicates resource consumption, which is essential in pushing the agents towards finishing the task as fast as possible. It also motivates the agents to only move in the environment if needed and stay idle otherwise. The agents receive an additional positive or negative reward (± 1) based on their proximity to the target. The reward would be positive if they have moved closer to the target from step $t-1$ to t, otherwise they receive a negative reward. The team is considered to have moved closer to the target if the nearest agent(s) at step $t-1$ took a mobility action towards the target at step t, meaning $\min(D_t) < \min(D_{t-1})$. Due to the environment complexity and the existence of obstacles, it is inaccurate to compute the travel distance between an agent and the target using generic methods like Manhattan and Euclidean distance. Instead, Breadth First Search (BFS) is used, which determines reward calculations. BFS explores different paths starting from an initial node until the goal is reached, aiming to find the shortest path. It is worth mentioning that the agents have no knowledge of the location of the target, and act only based on their observations. The reward function serves as a feedback mechanism, used only during the training stage, to improve the policies of the agents using PPO.

The training process is done using a Centralized-Learning & Decentralized-Execution

(CLDE) method [41], which is described previously in Algorithm 3.3. In this method, the agents act in a distributed manner in the environment according to their actor network copies. The experiences collected by the agents are then gathered and used centrally for updating the actor and critic. Once the training is over (which usually occurs in simulations), the final model is deployed on the sensing agents which act independently based on their observations.

6.2.3 Target Estimation with Transfer Learning

The MDRL model discussed in Section 6.2.2 is responsible for regulating the continuous decision-making process for each agent throughout the localization task. However, if a target is determined unreachable, it is essential to provide an estimate for its location. To reduce the complexities of the MDRL training process and the final model, the target estimation is not performed at each time step, but only when target unreachability is determined. To achieve this, a separate model is trained using a typical Deep Learning (DL) process with Transfer Learning (TL). Here, based on the observations collected until unreachability is determined, the aim is to provide an estimate for the (x,y) coordinates of the target. A dataset is pre-built using sets of observations and the corresponding target locations to be estimated, which are used to train the DL model to estimate the location based on the observations. To reduce the training and deployment complexities, we use TL to utilize knowledge from the previously obtained MDRL model, as shown in Fig. 6.2. Here, rather than training a new DL model from scratch for target estimation, the weights of the MDRL actor network are initially copied into the new DL model. During the training process, these layers are frozen, i.e. not trained, and only the last fully connected layer is trained. This is viable because the initial layers, especially the convolution layers, have already been trained to extract features related to target localization in the MDRL process. Such features would still be similar in the target estimation model, and hence only a final

classification layer would be sufficient. Following the training process, and since both the MDRL and DL models have common initial layers, they can be combined into one model with two output heads; one for continuous decision making (mobility, detection, and reachability) and one for target estimation, as shown in Fig 6.3. Both action heads take the same set of extracted features from the initial layers, but only differ in the weights of final layer, with the target estimation output head only getting triggered when unreachability is determined.

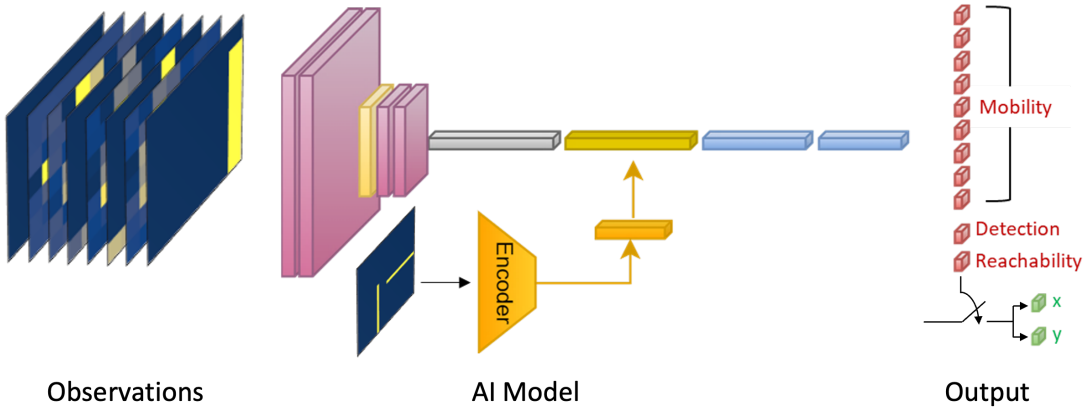


Figure 6.3: The final model deployed on each of the sensing agents.

6.3 Experiments and Evaluation

Extensive experiments are conducted in this section to validate the efficiency of the proposed method, as well as benchmark it against existing works in the literature. All the simulations have been conducted using an Intel E5-2650 v4 Broadwell workstation equipped with a 128 GB RAM, an 800 GB SSD, and an NVIDIA P100 Pascal GPU (16 GB HBM2 memory). To validate the proposed methods, a sample environment of radioactive source localization is used as discussed in Section 3.7.1.

For all the following experiments, each model is trained for 30 million environment steps using MDRL. At the beginning of each episode, the target location, the agents' initial

distribution, and the environment layout (with varying number of obstacles) are randomized. Hence, the model is trained on variations of random combinations of the possible states. Additionally, each episode is randomly set to one of three possible scenarios: 1) target exists and reachable, 2) target exists and unreachable, and 3) target does not exist, where each scenario has an equal chance of occurring. An episode terminates if the agents make the right decisions based on the scenario, or if a 100-timestep limit is reached. Following each 40k training steps, the agents are placed in a testing environment for 4k steps where they follow the most probable actions based on their policy (instead of sampling). The average performance of the testing steps is then recorded and plotted. The set of hyperparameters used in the PPO and CAE methods is shown in Table 6.1. The PPO hyperparameters are based on the original work in [31].

Table 6.1: PPO and CAE Hyperparameters

PPO	Value
ε	0.2
γ	0.99
λ	0.95
H	4000
Epochs per update	20
Learning rate	3×10^{-4}
CAE	**Value**
Learning rate	1×10^{-3}
Embedding Size d	128
Dataset Size	500000

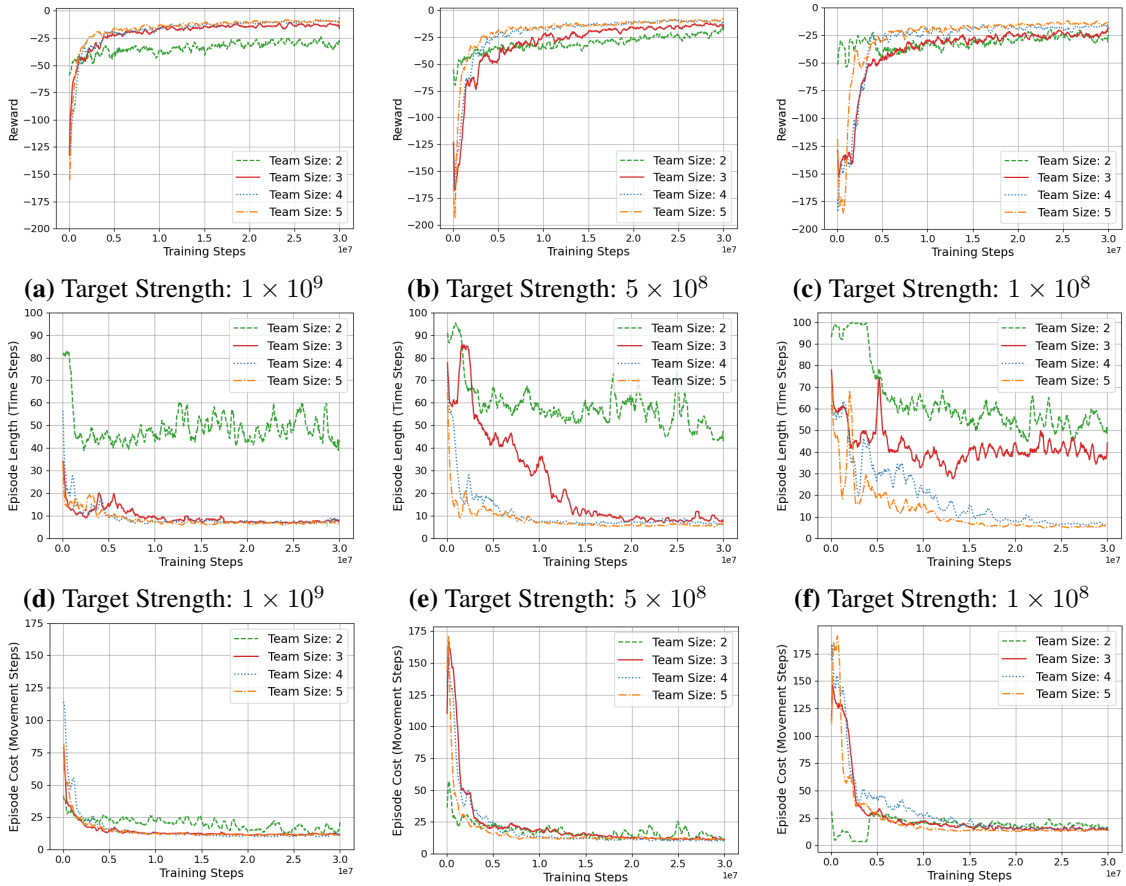
6.3.1 MDRL Performance Analysis

The performance of the proposed MDRL model is analyzed in terms of the collected reward, episode length, and cost. The episodic reward reflects the total accumulated reward collected in an episode, on average. The episodic length represents the number of steps until

an episode is terminated, which reflects how quick the agents are in finishing the task. The episodic cost reflects the total number of movement actions taken during an episode, which reflects resources consumption. The lowest movement cost in a given time step is 0 if none of the agents moved, while the highest is N if all the agents have moved.

Figure 6.4 summarizes the MDRL training results of the proposed method under different scenarios. Each sub-figure shows the results for varying team sizes to validate the scalability of the proposed approach. The figure also shows different target strengths, where S varies from 1×10^9 (strong) to 1×10^8 (weak). A strong target can be detected from further away, and hence is easier to detect and localize when compared to weaker target. It is worth mentioning that target estimation is not considered here yet since it is a different training process.

As seen in the figure, the training converges in all scenarios, indicating the feasibility of the proposed methods. When comparing the reward plot across different team sizes in Fig. 6.4a, it can be seen that the 2-agent team performs the worst, while the other three team sizes perform better. This reflects the challenging nature of the problem for a team of only 2 agents, as it would require them more time to search the area. On the other hand, the similar performance across the other team sizes indicates that the agents efficiently learn to cooperate to finish the task in a timely manner with reduced resource consumption. The reason behind converging to negative rewards is because the agents take exploration steps in the early stages of an episode, which are necessary for data collection. In scenarios where a target exists, the agents initially do not have sufficient readings to take proper decisions and move towards the target, and hence could collaborate to move in the environment to collect data, which could initially give negative rewards if agents move away from the target. In cases where the target is behind obstacles, the agents might need to move more to collect data, resulting in more negative rewards. Nonetheless, once data are gathered, the agents immediately learn to make the right decisions, which explains



(a) Target Strength: 1×10^9 **(b)** Target Strength: 5×10^8 **(c)** Target Strength: 1×10^8

(d) Target Strength: 1×10^9 **(e)** Target Strength: 5×10^8 **(f)** Target Strength: 1×10^8

(g) Target Strength: 1×10^9 **(h)** Target Strength: 5×10^8 **(i)** Target Strength: 1×10^8

Figure 6.4: A summary of the training plots for different team sizes and for varying target strengths. The episodic reward is shown in (a)-(c), the episodic length is shown in (d)-(f), and the episodic cost is shown in (g)-(i).

the low magnitude of negative rewards (around -17) that the models converge to. This efficient performance, in terms of time and cost, can be seen in Fig. 6.4d and Fig. 6.4g. In terms of episode length, it can be noticed that bigger team sizes give lower search time, as expected. However, proving the efficacy of the proposed team-based reward function, it can be noticed that nearly all the team sizes (aside from team size = 2) converge to a similar episodic cost. This indicates that, while larger teams have higher expected cost per step, the efficient coordination and the timely search result in lower episodic cost. The agents make cost efficient decisions, such as staying idle if found non-beneficial to the team, resulting in lower resource consumption.

Studying the performance across different target strength shows the increasing difficulty of the problem with lower target strengths. It can be seen that, as the target strength increases, the difference in performance between the different team sizes increases, as smaller team sizes need to put more effort to finish the task. It is evident that all teams achieve a lower reward when going from higher target strength to a lower one, i.e. from Fig. 6.4a to Fig. 6.4b and from Fig. 6.4b to Fig. 6.4c. The two-agent and three-agent teams collect the lowest rewards for the low target strengths, indicating the need for bigger teams. This is evident when analyzing the episode length (Fig. 6.4d-6.4f) and the episode cost (Fig. 6.4g-6.4i). In terms of episodic length, more time is needed to search for weaker targets. This is mainly because weaker targets can be detected within a shorter range, hence requiring the agents to get closer. It can be also seen that the weakest target shows the need for larger teams for faster execution, since more search can be done in less time. When looking at the episodic cost, it is noticed that it increases when going from strong to weak targets, since more search is needed and hence more resource consumption. However, it is evident that regardless of the team size, the cost is nearly similar, with a slight advantage for bigger teams.

To further study the model's behaviors, Fig. 6.5 analyzes the agents' performance under different scenarios, namely target search, target non-existence, and target unreachability, given a team of 4 agents under different target strengths. To obtain these results, the trained agents are deployed in the corresponding environment for 40,000 steps and the average performance is collected. As evident, in terms of time, the agents are equally able to reach the correct decision regardless of the scenario or the target strength. However, as evident in Fig. 6.5b, more cost is spent as the target strength gets weaker. This indicates that due to the increased difficulty of the process with weaker targets, more agents need to be involved to maintain a timely decision-making, resulting in higher costs. This behavior reflects the desired cooperation between the agents to achieve quick decision-making. Additionally, It

can be noticed that the difference between time and cost is insignificant for stronger targets. For example, in the case of the strongest target, the search process on average takes 7.7 time steps and costs 12.4 movement steps in total. This means that, the agents quickly determine their informativeness to the task, and only informative agents carry the tasks while others maintain idle status to preserve resources.

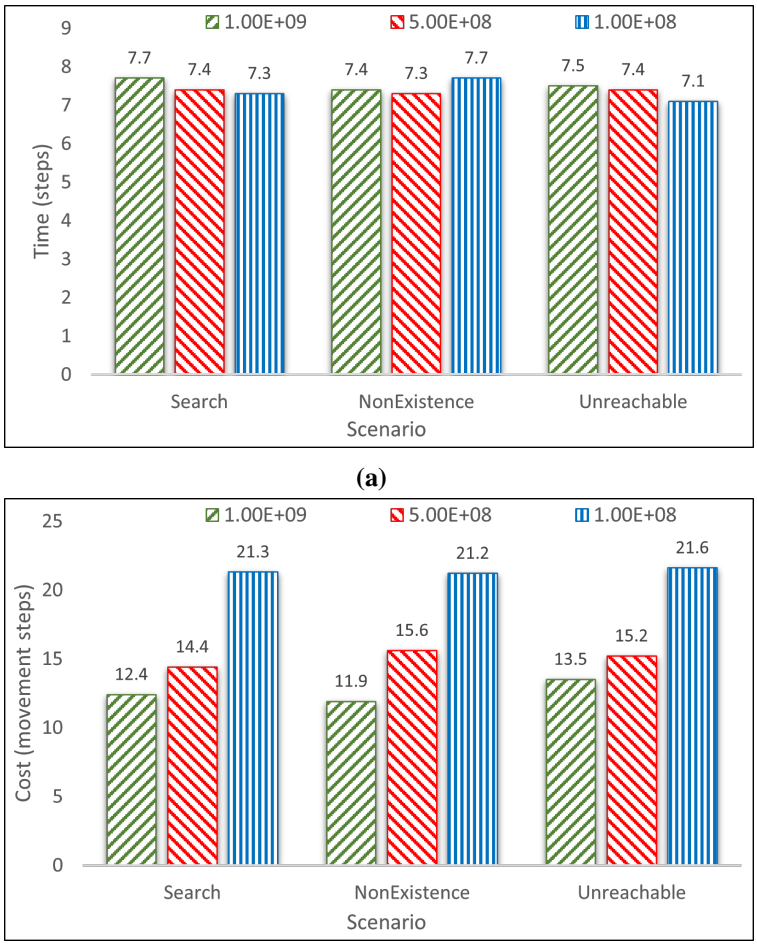


Figure 6.5: Agents' performance under the different scenarios in terms of (a) episodic time and (b) episodic cost, for a team of 4 agents and varying target strengths.

6.3.2 Target Estimation

The aforementioned results analyze the performance of the MDRL model, which is responsible for the decision-making related to target search and flagging its non-existence or unreachability. Following the unreachability flag, a DL model is used to estimate the target locations based on the collected observations. Fig. 6.6 shows the training loss and validation loss curves obtained while training a DL model, using TL, for target estimation. Here, a dataset is built combining the observations collected by the MDRL agents until the unreachability flag is produced, and then used to train the model to estimate the target. The dataset is split into training and validation sets, where the training set is used to update the model and the validation set is used to assess the model on unseen data. The training is done to reduce the Mean Squared Error (MSE) loss between the predicted (x,y) coordinates and the true coordinates. As can be seen in the figure, upon the completion of the training, both the training and validation loss values converge to 0, indicating that the model is able to accurately estimate the target location. One factor leading to these good results is the use of TL to transfer knowledge from the MDRL model to the DL model. Additionally, the observations collected by the MDRL agents and used to determine the unreachability of the target prove efficient in estimating its location.

6.3.3 Benchmarks

This section compares the performance of the proposed MDRL method with some existing benchmarks in the literature. Existing methods struggle when addressing the realistic scenarios of target localization. To reflect on this, we compare the proposed approach with three existing benchmarks covering traditional target search techniques, single-agent DRL methods, and MDRL-based methods with no considerations to the uncertainties about the target. For the subsequent results, 4 agents are placed in a simulation environment with varying target strengths to analyze the episode length (time) and cost. After training, the

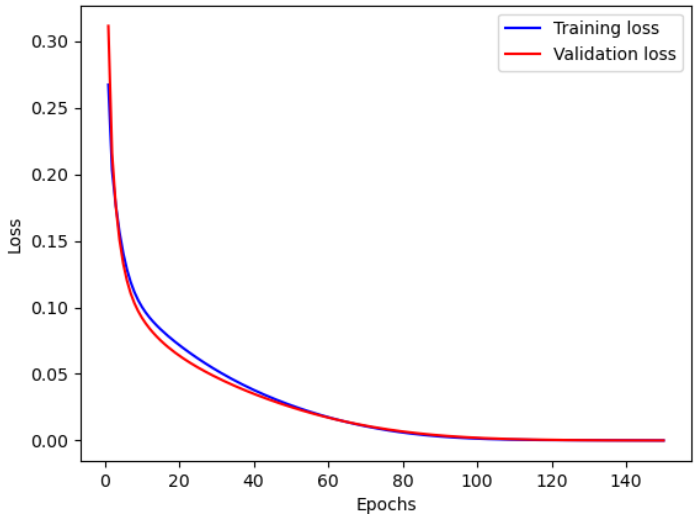


Figure 6.6: The training and validation loss for the target estimation model using transfer learning.

agents are placed in different environments for inference, where the episode length and cost are averaged throughout 40k time steps. Agents obtained following these methods are placed in the environment, where the aim is to make decisions regarding the target location, its existence, and its reachability. Each episode has a limit of 100 steps, within which the agents are to finish the task. For fair comparison, the DRL- and MDRL-based solutions have been trained for the same number of steps as the proposed approach, i.e. for 30 million environment steps. The benchmarks are summarized as the following:

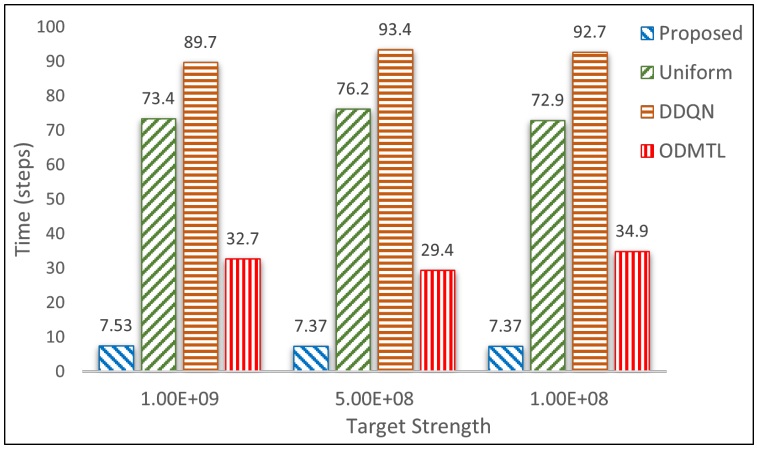
- Uniform: A traditional method where agents follow a search path that is pre-defined to cover the area uniformly[40].
- DDQN: A single-agent DRL approach using Double Deep Q-learning for target search. The model is extrapolated into multi-agent settings by having a single model control all the agents in a centralized manner. The policy takes all the observations of the agents at once and produces a joint action vector [2].
- ODMTL: An optimized deep multi-agent target localization using MDRL [44], as presented in Chapter 4. Here, the methods follow a CLDE approach, but they do not

consider scenarios of target unreachability or target non-existence.

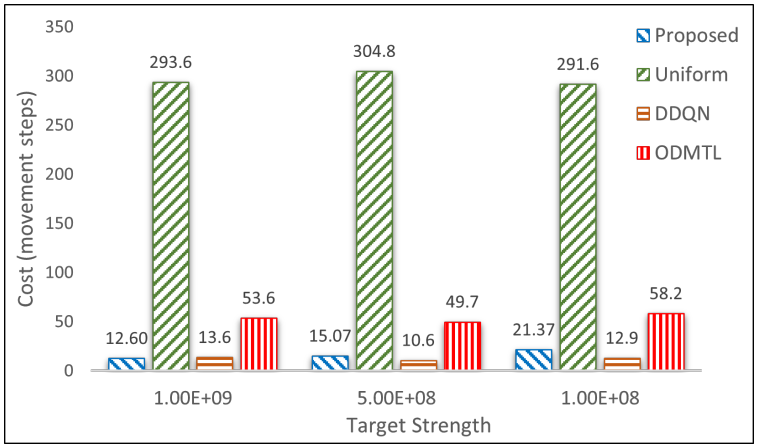
Figure 6.7 summarizes the performances of the different benchmarks under varying target strengths in terms of time and cost. As can be seen, the proposed work outperforms all the benchmarks by achieving faster and less costly localization tasks. Traditional uniform search methods do guarantee finding the target, but cannot handle cases where the target is unreachable or does not exist. In such scenarios, the agents keep searching the environment and waste resources. Additionally, due to the lack of cooperation between agents, there is an increased resource consumption. When comparing the proposed work with single-agent DDQN methods, it can be seen that these methods struggle to learn to perform the task. Here, within the same amount of learning experience, the agents fail to learn any desired behaviors and resort to staying idle most of the time, hence the high localization time and low cost. This is a form of local optima, which is a result of the scalability issues faced in single-agent DRL methods when extrapolated into multi-agent settings. The ODMTL method generally performs better than the other two methods, due to the existence of intelligent decision-making and cooperation between agents. However, the lack of consideration of the uncertainties in the environment, such as the non-existence of the target or its unreachability, hinder their ability.

6.4 Conclusion

In this chapter, the target localization problem is addressed using MDRL methods, while considering realistic scenarios of target non-existence and unreachability. Based on the collected observations, which are modeled as 2D maps representing the environment, the agents learn efficient decision-making through MDRL, encapsulating actions related to the mobility in the environment and determining the existence and reachability of the target. The MDRL policy is represented by a Convolutional Neural Network (CNN) which



(a)



(b)

Figure 6.7: Comparison between the performance of the proposed method and the benchmarks in terms of (a) episode length and (b) episode cost.

is optimized using Proximal Policy Optimization (PPO) and a team-based shaped reward function. Using Transfer Learning (TL), the same model is expanded to also cover target estimation, where the target coordinates are approximated if it is unreachable. The proposed MDRL method was tested on different scenarios, covering varying target strengths (from weak to strong) and varying number of agents. The scalability and adaptability of the proposed method was verified through quick and efficient target localization tasks, with very accurate target estimation. Compared to traditional and DRL-based benchmarks, the proposed work shows outperformance, especially in tackling scenarios with uncertainties such as target non-existence and unreachability.

Chapter 7

Blockchain-assisted Demonstration Cloning for Multi-Agent Deep Reinforcement Learning

7.1 Introduction

Despite its significant success, designing and training DRL solutions has been, and still is, a challenging task. From one side, designing DRL solutions for real-world problems requires tremendous expertise and domain knowledge. Designers often have to go through intricate steps which include the modeling of the environment and its interactions, the modeling of the agents' observations, the choice of the policy optimization algorithm and the tuning of its hyperparameters, and the design of the reward function, to name a few [100]. Each of these steps could significantly affect one another as well as the outcome of the learning process, and require customized solutions tailored to each application. These obstacles are further amplified with the typical challenges faced in DRL, which include sample inefficiency, curse of dimensionality, reward sparsity, and the exploration-exploitation

dilemma [101]. On the other side, training DRL requires a significant amount of computational resources that might not be available for certain users or businesses. This is mainly due to the complexity of DNNs and DRL algorithms, and the high dimensionality of typical DRL environments. For example, the authors in [23] used nearly 51 thousand CPUs and 512 GPUs to train a DRL system for the game of Dota, and such resources are not accessible to the majority of users and researchers.

Over the past few years, the paradigm of Machine Learning as a Service (MLaaS) came as a way to promote greater accessibility to Machine Learning (ML) solutions for developers, business, and the general public. MLaaS is an offering that provides ML capabilities and infrastructure as a service to users in exchange for money [102]. Generally, MLaaS providers target a wide spectrum of users, ranging from those with no experience in ML, to those who have the expertise but lack computational resources. In the first case, users interact with ML APIs by providing training data, which is used by the service to produce a trained model [103]. The automated process usually involves data pre-processing, model training, and evaluation. Such services often cover a range of ML applications, including but not limited to classification, regression, clustering, and natural language processing. Service providers include Google AutoML and Vertex AI, which enable developers with limited ML expertise to train high-quality models. In the second case, MLaaS provides infrastructure and tools for experienced users to build and develop models, as well as computational resources, if needed. Such services include Amazon SageMaker, Microsoft Azure, and Google Colab.

Extending MLaaS directly into the realm of DRL is infeasible with the existing services. Firstly, DRL commonly requires much longer time and more computational resources to train when compared to typical ML problems. This would induce significant overhead on existing MLaaS providers, which usually operate via a shared platform between users that is allocated on demand [58]. Secondly, the environment variability in DRL results in a

highly dynamic, and often unpredictable, interaction between the agent and the environment during the training process. This contrasts with most ML scenarios in which the environment is static or well-defined. Thirdly, having few MLaaS providers entails higher costs on users, especially those with problems requiring high computational resources and long training times. Finally, and most importantly, the design and training processes in DRL are much more complex when compared to ML, hence requiring more human expertise and interventions. Since ML problems generally rely on static and available datasets, the learning process can be automated. This is unlike DRL, which relies on dynamic and sequential interactions with the environment that need to be modeled and optimized by an expert. Recent works in the literature propose crowdsourcing systems for ML tasks, with the aim of using the scalability, diversity, and expertise of the crowd [66, 68, 70, 104, 105]. Crowdsourcing refers to the practice of obtaining services (to requesters) by soliciting contributions from a group of people (workers) [13, 106, 107]. However, all the existing works in ML only crowdsource tasks related to data collection, data annotation, and model validation. To our knowledge, none of the existing methods discuss the crowdsourcing of ML or DRL training and model sharing tasks, which require expertise and computational capabilities in addition to DRL-related attributes. In summary, the drawbacks in existing methods are:

(1) Existing MLaaS systems are usually centralized and automated services, making them infeasible to DRL tasks that have higher computational overhead and require more distributed solutions and long training times.

(2) Having few MLaaS platforms entails higher costs and low accessibility to solutions.

(3) The lack of diverse expertise in existing MLaaS solutions hinders them incapable of addressing the complex and diverse DRL tasks that require intricate modeling and optimization by experts.

(4) Existing crowdsourcing systems for ML focus on data gathering and model validation tasks, with no consideration for training and model sharing tasks that require computational capabilities and expertise.

To circumvent the aforementioned issues, this chapter introduces a novel blockchain-based crowdsourced DRL as a service (DRLaaS) framework. In this work, the proposed framework addresses the lack of diverse expertise and computational capabilities in MLaaS by crowdsourcing DRL training tasks instead of relying solely on centralized platforms. It also introduces DRL-related worker recruitment for crowdsourcing, which existing ML crowdsourcing systems do not consider. The proposed framework covers two types of tasks, namely DRL training and model sharing. In DRL training tasks, users have the ability to request the full design and training of DRL solutions based on the specified requirements to be executed by the recruited workers. Such a task depends heavily on the computational capabilities and expertise of the worker. In model sharing tasks, users can benefit from pre-trained and available models by expert workers, which can be used to assist in training new DRL solutions using methods in knowledge transfer, such as Imitation Learning-based DRL and Demonstration Cloning [44, 48]. Here, the reputation of a worker and the suitability of the models they provide are instrumental when allocating pre-trained models to requesters. For both types of tasks, the proposed framework is responsible for managing the allocation of training tasks and models to suitable workers. For DRL training tasks, specific DRL-related metrics are designed to assess candidate workers according to the task requirements and based on their capabilities. Specifically, attributes like expertise in designing and training DRL solutions, reputation in accepting and finishing DRL tasks, and computational capabilities are considered in a Quality of Service (QoS) metric that is optimized using a greedy algorithm. The computational capability parameters are designed to assess critical worker attributes like GPU availability, RAM capacity, and CPU parallelization, which are pivotal for effective DRL training. For model sharing

tasks, in addition to the worker's expertise and reputation in managing this type of tasks, a DRL model similarity metric is designed to assess the available models, which considers the DRL environment of the model and how suitable it is based on the requester's requirements. The framework is deployed on a Consortium Blockchain, which manages the interaction between requesters and workers, the task and resource allocation, and the model sharing processes. Blockchain is used, instead of centralized cloud-servers, to mitigate issues related to single-point of failure and the need of a single trusted server, by providing a decentralized, transparent, and autonomous platform with no repudiation. A Consortium Blockchain, specifically, provides better privacy, scalability, and efficiency when compared to public blockchains [89], and better allowance for collaboration and data sharing between entities when compared to private Blockchains. Simple and efficient Smart Contracts (SCs) are designed to manage the task and model allocation processes, and the InterPlanetary File System (IPFS) is used to manage the storage of models. In summary, the contributions of this chapter are as follows:

(1) The design of a comprehensive framework for crowdsourced DRLaaS that utilizes the expertise and computational capabilities of expert workers in answering tasks related to the design, training, and sharing of DRL solutions, aiming to assist inexperienced and experienced users in return for incentives.

(2) The design of task and model allocation processes using greedy methods that consider DRL-related and workers-related attributes, such as computational capabilities, task requirements, DRL environment details, model quality, and workers expertise and reputation.

(3) The design of DRL-related computational capability and model similarity attributes. Computational capability attributes consider the effect of RAM, CPU, and GPU specifically on the DRL training task. Model similarity attributes consider features

in the DRL environment to assess how existing models can be beneficial to the requesting user.

(4) The design of simple and efficient smart contracts that fully manage the aforementioned processes on the blockchain, with the assistance of IPFS to store trained models.

A general overview of the proposed DRLaaS framework is shown in Fig. 7.1. Requesters submit tasks by interacting with the smart contracts on the blockchain. The smart contracts manage users' registrations, tasks allocations, and model sharing processes. Once tasks are allocated and executed by workers, the returned outcomes (trained models) are shared via IPFS. The tasks' outcomes are then shared back to the requesters, and workers are paid accordingly.

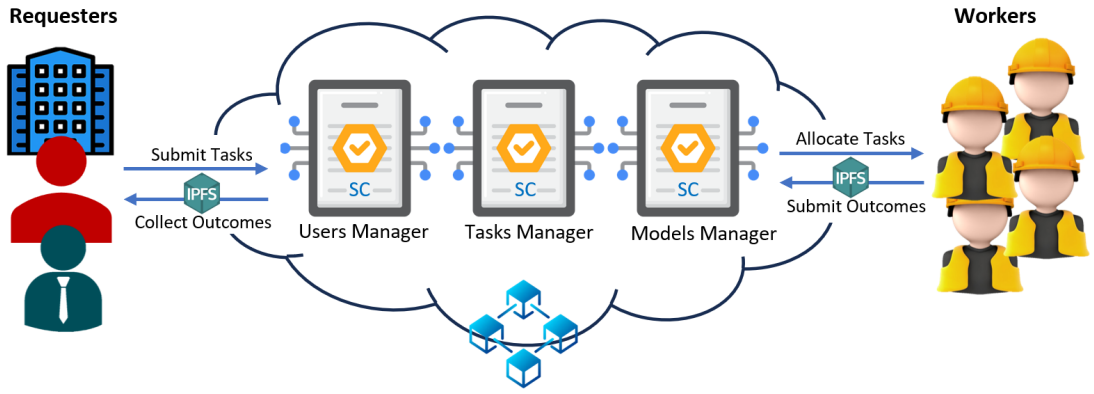


Figure 7.1: A general overview of the proposed framework.

The proposed framework is evaluated for several complex DRL applications, including Target Localization [44], Autonomous Vehicles Fleet Coordination [97], and Multi-Agent Maze Cleaning [99].

7.2 DRL Design and Training Requirements

The design and training of DRL models are complex tasks, requiring specialized *expertise*, *computational capabilities*, and often *compatible pre-trained models*. This section explains these needs, which are to be met by the proposed crowdsourcing framework.

7.2.1 Expertise: Environment, Reward, and Optimization

The process of designing and training DRL solutions requires expertise in several areas. This process can be summarized in three main steps, namely 1) Environment Design, 2) Reward Engineering, and 3) Policy Optimization. The *Environment Design* step is concerned with modeling the problem of interest in an environment that follows the DRL formulation. This requires the designer to have expertise in developing environments that mimic real-world dynamics and encapsulate the possible interactions (actions) between the agent(s) and the environment. For example, in a simple environment for the game of chess, the designer needs to fully understand the rules that govern the game, which are needed to define the possible actions for the chess agent. Additionally, the state of the chess environment could be modeled as a grid image of the board, or by numerical features representing the locations of the pieces. Such a choice is critical to be made by the designer, as it affects the training process. The *Reward Engineering* step is critical since the behavior of the agents in DRL is directly influenced by the reward function. Reward functions require considerable engineering, which if done poorly could lead to local optima [48]. This requires the designer to have knowledge in the domain of interest, as well as expertise in reward shaping [46, 108]. For the game of chess, for example, a simple reward function is to assign a positive value only for a checkmate. However, this complicates the learning since the agent does not get regular feedback, and hence calls for the need for more complex reward functions that require expertise in this domain. In the *Policy Optimization* step, the aim is to design policies that translate the agent's observations into actions, which are then

optimized using the collected rewards. In DRL, policies are represented as DNNs, and the choice of architecture depends on several attributes, such as the type and dimensionality of the input and the problem complexity, which require expertise in DL. On the other hand, the optimization algorithm determines how the collected experiences (observations and rewards) update and optimize the agent's policy. Designing and choosing the appropriate optimization algorithm requires expertise in the domain of the application, as well as in hyperparameter tuning, which is essential in finding optimal solutions. For example, in the aforementioned chess environment, convolutional neural networks (CNNs) could be used as policy architectures if the state is represented as an image, while feed forward networks (FFNs) could be used if the state is represented in numerical features. These could be optimized using methods such as Proximal Policy Optimization (PPO), Deep Q-Networks (DQN), or Deep Deterministic Policy Gradients (DDPG). All of the aforementioned are decisions to be made by the designer.

7.2.2 Computational Capabilities

Due to the complexity of DRL, the training process usually requires heavy computational resources. While the choice of optimization algorithm has a significant effect on the learning speed and convergence, the availability of computational resources holds equal significance. For example, the availability of GPUs could significantly speed up the execution of DNNs, which in turn speeds up the learning, especially for image-based applications (2D data with Convolutional Neural Networks). Additionally, many policy optimization algorithms for DRL can harness parallel processing on multiple processing units (CPU cores or GPUs) to speed up the learning process by parallelizing the experience collection process. Moreover, most DRL optimization methods come with many several hyperparameters, which should be fine-tuned for each application. This process requires heavy computational resources, in addition to expertise in this domain.

7.2.3 Model Availability and Compatibility

The DRL training process has been recently approached using methods from imitation learning. Recent works [45, 48, 50] proposed using demonstrations from an expert in guiding the learning of DRL agents. An expert model is a pre-trained model that is already familiar with the environment (or with a similar environment), and can help the agents collect better experiences, or can be used partially in Behavioral Cloning (BC) to enhance the DRL policy. Here, the availability of such expert models becomes an important issue to address. Additionally, the compatibility of the expert models with the current environment of interest is another crucial challenge. If the demonstrations suggested by the expert introduce variances to the current DRL training problem, this introduces difficulties in the learning convergence. This necessitates the availability of models that are similar to the current environment.

All the aforementioned needs are to be met by the proposed crowdsourcing framework in this work. The complicated design processes call for the need of expertise, which can be obtained through crowdsourcing. Additionally, crowdsourcing could help in providing computational resources through workers that have access to multiple CPUs and GPUs. The recent methods enhancing DRL through expert demonstrations call for the need of compatible expert models, which can also be obtained through crowdsourcing.

7.3 Overview: Blockchain-based DRLaaS Framework

The proposed framework aims to target two types of DRLaaS tasks that can be requested by users, namely DRL training and model sharing. In DRL training tasks, users can request the design and training of DRL solutions to be done by expert workers, i.e. workers with experience in tackling problems in the domain of interest using DRL. This

task is highly dependent on the worker's expertise and computational capabilities, as discussed in Section 7.2. On the other hand, in model sharing tasks, users can request pre-trained models in certain domains, which can be shared by other expert workers. This task depends on the efficiency and compatibility of the available models with the requester's environment. The flow of the proposed framework is shown in Fig. 7.2, which is to be detailed in the next sections by discussing the different DRLaaS tasks, the recruitment metrics and processes, and the design of the smart contracts. In this framework, users register in the system by submitting information about their attributes. Users can then submit task requests by indicating their requirements. For DRL training tasks, workers are selected following the designed recruitment process. The recruitment is done based on the workers' attributes and their expected Quality of Service (QoS), which considers metrics related to the expertise and computational capabilities of the candidate workers. A recruited worker then performs the training task, which includes steps such as environment design, reward engineering, policy optimization, and model tuning, before submitting the resultant model. For DRL model sharing, workers initially declare the availability of DRL models by submitting information about their models to the platform. Once a model request is submitted, a selection mechanism uses these information to compute the QoS for the candidate workers based on their attributes, as well as their model's attributes, to assess their efficiency and compatibility. When a worker is selected, they are tasked with sharing their model through the IPFS. In all the tasks, once the workers finish execution, the platform returns the outcomes to the requesters and forwards payments to workers. Subsequently, requesters rate the tasks by submitting reviews, which are used in the future to compute the QoS of a worker, as will be discussed in this section.

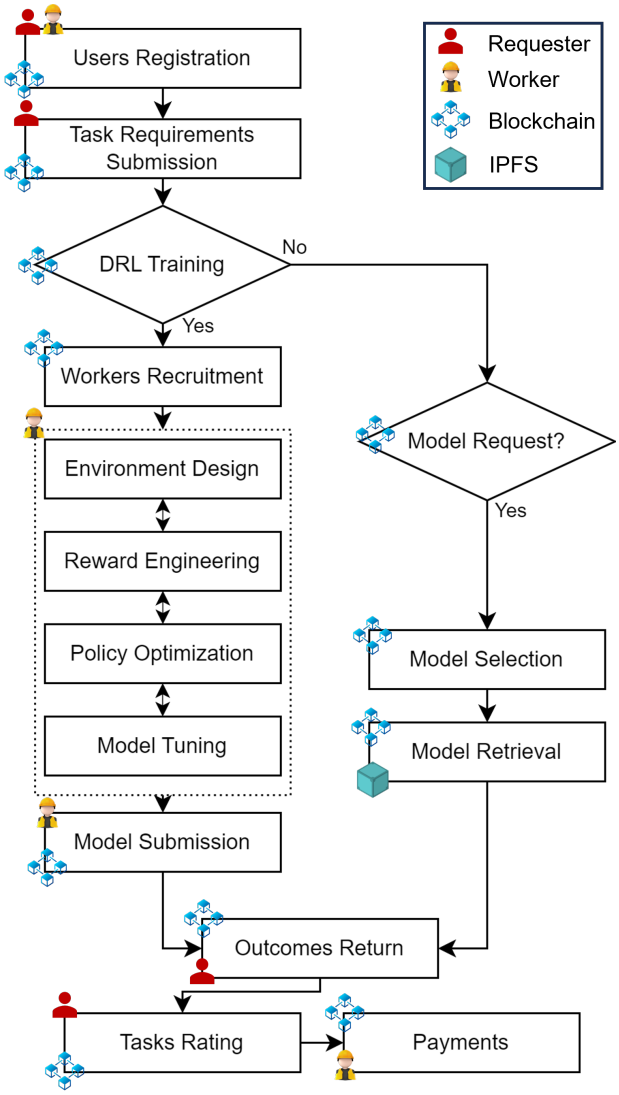


Figure 7.2: The proposed Blockchain-assisted DRLaaS framework. The different steps could involve a single entity, or an interaction between two entities. Entities include requesters, workers, the blockchain, and IPFS.

7.4 Problem Formulation

Each task type, i.e. DRL training and model sharing, has its own requirements and recruitment metrics to be used when selecting workers. Generally, given a set of tasks $T = \{T_1, T_2, T_3, ...\}$ submitted by requesters and a pool of candidate workers $W = \{W_1, W_2, W_3, ...\}$, the aim is to select a set of workers $W^{T_i} = \{W_1^{T_i}, W_2^{T_i}, W_3^{T_i}, ...\}$ for each task T_i. Each task T_i is defined as a tuple in the form of $T_i = (ID^{T_i}, NW^{T_i}, K^{T_i}, D^{T_i}$

$, Rep_{min}^{T_i}, R_{min}^{T_i}, Dom^{T_i}, TC^{T_i}, CR^{T_i})$, where ID^{T_i} is the task ID, NW^{T_i} is the number of workers desired for task T_i , K^{T_i} is the task type (training or sharing), D^{T_i} is a detailed description of the problem of interest, $Rep_{min}^{T_i}$ and $R_{min}^{T_i}$ are the minimum reputation and rating required for a worker to perform the task, Dom^{T_i} is the task domain, TC^{T_i} is the time constraint specified for the task to be completed within, and CR^{T_i} is the computational requirements for the task. On the other hand, a worker W_j is defined as a tuple in the form $W_j = (ID^{W_j}, Rep_{Tr}^{W_j}, Rep_{MS}^{W_j}.R_{Tr}^{W_j}, R_{MS}^{W_j}, Dom^{W_j}, Exp_{Tr}^{W_j}, Exp_{MS}^{W_j}, CC^{W_j})$, where ID^{W_j} is the worker's ID, $Rep_{Tr-Dom^{T_i}}^{W_j}$ and $Rep_{MS-Dom^{T_i}}^{W_j}$ are the worker's reputations based on historical performances in the platform for the two task types in domain Dom^{T_j}, $R_{Tr-Dom^{T_i}}^{W_j}$ and $R_{MS-Dom^{T_i}}^{W_j}$ are the worker's ratings for the two task types in domain Dom^{T_j}, Dom^{W_j} is the set of domains that the worker is familiar with, $Exp_{Tr-Dom^{T_i}}^{W_j}$ and $Exp_{MS-Dom^{T_i}}^{W_j}$ are the worker's expertise in the two task types in domain Dom^{T_j}, and CC^{W_j} is the worker's computational capabilities. It is worth mentioning that a worker's reputation is mainly based on their commitment to accepting and finishing tasks before the specified deadlines. On the other hand, a worker's rating is based on their performance, obtained through feedback on the trained models from users that have already used the service. Table 7.1 summarizes the task and worker attributes and their definitions.

The aforementioned attributes are utilized differently based on the task type. For each of the tasks, the following section describes the task requirements and worker recruitment process.

7.5 Worker Recruitment Parameters

Each of the two task types requires a dedicated worker recruitment process to select the most suitable set of workers to execute the task. To address this, we propose two different Quality of Service (QoS) metrics which are used to assess candidate workers and select the most suitable for each type of tasks. The proposed QoS metrics take into consideration

the task requirements and worker attributes, and ensure that the selected workers meet the desired expectations.

7.5.1 DRL Training Tasks

A worker here is asked to design a suitable environment for the task and engineer an efficient reward function. Additionally, the worker is required to choose a suitable DRL optimization algorithm, train a policy for the task at hand, and fine-tune the trained model (policy). The worker's expertise is essential for designing the environment and choosing a suitable policy architecture (type of neural network) and optimization method based on the provided environment. The availability of computational resources is also crucial for training and fine-tuning the model in a reasonable time.

To quantify the computational capabilities of a worker, we consider three important computational resources that are essential for DRL training: Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Random Access Memory (RAM). The availability of multiple CPU cores and a powerful GPU is crucial for DRL training, as discussed in Section 7.2. Since the series/brand of the GPU is what matters the most, it is assumed that the proposed crowdsourcing platform accepts a pre-defined list of GPU series, which are to be checked against the one available for the worker in the constraints. Finally, during the process of DRL training, huge amounts of data in the form of experiences (observations and rewards) need to be stored, which requires high RAM storage.

For a given DRL training task T_i, the requester specifies a set of requirements and constraints, given as:

- Problem Description (D^{T_i}): this requirement describes the detailed nature of the problem for task T_i, which is needed by the worker to design the DRL environment. This includes descriptions of the type of data available to collect, the type of interactions between the agent and the environment, the type of interactions between the

Table 7.1: List of attributes and their definitions.

Attribute	Definition
W^{T_i}	Set of workers for task T_i
ID^{T_i}	ID of task T_i
NW^{T_i}	Number of workers desired for task T_i
K^{T_i}	Type of task T_i
D^{T_i}	Description of task T_i
$Rep_{min}^{T_i}$	Minimum reputation requirement for task T_i
$R_{min}^{T_i}$	Minimum rating requirement for task T_i
Dom^{T_i}	Domain of task T_i (DRL training or model sharing)
TC^{T_i}	Time constraint of task T_i
CR^{T_i}	Computational requirements for task T_i
ID^{W_j}	ID of worker W_j
Dom^{W_j}	Set of domains covered by worker W_j
$Rep_{Tr-Dom^{T_i}}^{W_j}$	Reputation of worker W_j in DRL training for Dom^{T_i}
$Rep_{MS-Dom^{T_i}}^{W_j}$	Reputation of worker W_j in model sharing for Dom^{T_i}
$R_{Tr-Dom^{T_i}}^{W_j}$	Rating of worker W_j in DRL training for Dom^{T_i}
$R_{MS-Dom^{T_i}}^{W_j}$	Rating of worker W_j in model sharing for Dom^{T_i}
$Exp_{Tr-Dom^{T_i}}^{W_j}$	Expertise of worker W_j in DRL training for Dom^{T_i}
$Exp_{MS-Dom^{T_i}}^{W_j}$	Expertise of worker W_j in model sharing for Dom^{T_i}
CC^{W_j}	Computational capabilities of worker W_j

agents themselves (in multi-agent systems), etc. This attribute comes in the form of a textual description. For example, in the problem of target search, where a group of UAVs try to find a certain target, the problem description contains information about the number of sensing agents, the number of targets (single/multi), the dynamicity of the environment (with or without obstacles), the type of data that can be collected by

the agents (images or sensor readings), etc.

- Problem Domain (Dom^{T_i}): this requirement describes the general domain of the desired task T_i, which is essential to focus on candidate workers with experience in similar tasks within the same domain. For example, in the problem of target search, the problem domain could be "robot swarms", which encompasses a range of problems that demand similar expertise.

- Reputation Requirement ($Rep^{T_i}_{min}$): this requirement describes the minimum reputation required by the requester, to be met by the candidate workers for task T_i. Reputation is a measurement of the historical commitment of the worker, which is mainly based on acceptance and completion of previous tasks.

- Time Constraint (TC^{T_i}): this requirement specifies the time window within which task T_i is to be completed.

- Number of Workers (NW^{T_i}): in some problems, a requester might desire the task to be performed by multiple independent workers, to increase the likeliness of efficient results.

- Minimum number of CPU cores $CR^{T_i}_{CPU}$.

- Minimum RAM capacity $CR^{T_i}_{RAM}$.

It is worth mentioning that, in most cases, the number of GPU cores does not significantly affect the learning process, hence only the availability of a suitable GPU will be checked, regardless of the number of cores. To assess the expertise of a candidate worker for a task of DRL training, assume $N^{W_j}_{Tr-Dom^{T_i}}$ is the number of DRL training tasks completed by worker W_j in domain Dom^{T_i}, then their expertise $Exp^{W_j}_{Tr-Dom^{T_i}}$ for DRL training in that domain is computed as:

$$Exp^{W_j}_{Tr-Dom^{T_i}} = \frac{N^{W_j}_{Tr-Dom^{T_i}}}{N^{max}_{Tr-Dom^{T_i}}} \tag{16}$$

where $N^{max}_{Tr-Dom^{T_i}}$ is the maximum number of training tasks completed by one of the candidate workers in the same domain, which is used for normalization. This equation measures the expertise of a candidate worker relative to the pool of available workers, normalized between 0 and 1, where 1 is given to the candidate worker with the greatest number of tasks completed. On the other hand, the worker's reputation $Rep^{W_j}_{Tr-Dom^{T_i}}$ for DRL training in domain Dom^{T_i} is characterized by two attributes, namely commitment rate $CM^{W_j}_{Tr-Dom^{T_i}}$ and completion rate $CP^{W_j}_{Tr-Dom^{T_i}}$. Assume $A^{W_j}_{Tr-Dom^{T_i}}$ is the total number of tasks assigned to worker W_j, $Ac^{W_j}_{Tr-Dom^{T_i}}$ is the total number of tasks accepted by worker W_j, and $E^{W_j}_{Tr-Dom^{T_i}}$ is the total number of tasks completed by worker W_j, all for DRL training in domain Dom^{T_i}, then $CM^{W_j}_{Tr-Dom^{T_i}}$ and $CP^{W_j}_{Tr-Dom^{T_i}}$ are given as:

$$CM^{W_j}_{Tr-Dom^{T_i}} = \frac{Ac^{W_j}_{Tr-Dom^{T_i}}}{A^{W_j}_{Tr-Dom^{T_i}}} \tag{17}$$

$$CP^{W_j}_{Tr-Dom^{T_i}} = \frac{E^{W_j}_{Tr-Dom^{T_i}}}{Ac^{W_j}_{Tr-Dom^{T_i}}} \tag{18}$$

In these equations, $CM^{W_j}_{Tr-Dom^{T_i}}$ reflects the confidence the platform has in worker W_j to accept the assigned task, while $CP^{W_j}_{Tr-Dom^{T_i}}$ reflects the confidence in worker W_j to complete the tasks they accepted to perform. The reputation $Rep^{W_j}_{Tr-Dom^{T_i}}$ of the worker is computed using the geometric mean, which is given as:

$$Rep^{W_j}_{Tr-Dom^{T_i}} = \sqrt{CM^{W_j}_{Tr-Dom^{T_i}} \times CP^{W_j}_{Tr-Dom^{T_i}}} \tag{19}$$

Another metric to be considered in the QoS is the worker's performance rating $R^{W_j}_{Tr-Dom^{T_i}}$ for DRL training tasks in the desired domain, which is based on reviews given to the worker by previous requesters. Here, reviews could be in the form of a numerical rating in a given

scale, which are averaged to give the worker a score between 0 and 1. Finally, to quantify the computational capabilities of a worker CC^{W_j}, a metric is proposed based on the number of CPU cores. Assume that the numbers of CPU cores available for worker W_j are given as $N_{CPU}^{W_j}$, then the worker's computational capabilities CC^{W_j} is quantified as:

$$CC^{W_j} = \frac{2}{\pi} tan^{-1}(w_1 N_{CPU}^{W_j}) \tag{20}$$

The $tan^{-1}(x)$ function is used because its output increases rapidly initially with x then slows down as x gets higher. This is essential to capture the effect of CPU cores on DRL training. The parallelization process in the training process significantly speeds up the learning. However, after a certain number of parallel environments (depending on the problem), further parallelization does not bring more benefit. Generally, more complex DRL tasks could benefit from more parallelization. The weight w_1 controls the stretch (the rapid increase) of the function. High values of these weights help differentiating between workers with low number of CPU cores, but workers with high number of cores would nearly get the same score (which is sufficient for less complex DRL tasks). On the other hand, low values help differentiating between workers with high number of cores (which is desirable for more complex DRL tasks). The $tan^{-1}(x)$ function is multiplied by $\frac{2}{\pi}$ to normalize the output between 0 and 1.

Considering all the aforementioned metrics, the QoS of worker W_j for the DRL training task in domain Dom^{T_i} is then given as:

$$QoS_{Tr-Dom^{T_i}}^{W_j} = Exp_{Tr-Dom^{T_i}}^{W_j} \times Rep_{Tr-Dom^{T_i}}^{W_j} \times R_{Tr-Dom^{T_i}}^{W_j} \times CC^{W_j} \tag{21}$$

In this QoS function, all the attributes have values between 0 and 1. The format of Eq. 21 is common in crowdsourcing literature [39, 92], as it provides normalized metrics that are used to assess and compare candidate workers. The aforementioned attributes

are updated regularly following task assignments and completion by the workers. The proposed QoS is to be maximized while meeting the constraints for each candidate worker, given as:

- $Rep^{W_j}_{Tr-Dom^{T_i}} \geq Rep^{T_i}_{min}$
- $R^{W_j}_{Tr-Dom^{T_i}} \geq R^{T_i}_{min}$
- $Dom^{T_i} \in Dom^{W_j}$
- $N^{W_j}_{CPU} \geq CR^{T_i}_{CPU}$
- $N^{W_j}_{RAM} \geq CR^{T_i}_{RAM}$
- The brand and series of W_j's GPU is in the accepted list.

The QoS, along with the task constraints, are then used in an optimization method to select most suitable workers, which is described later in Section 7.6.

7.5.2 DRL Model Sharing Tasks

In model sharing tasks, a user shares information about their pre-trained model with the platform, with the hope of receiving incentives if the model is requested by other users. For a given model sharing task T_i, a requester specifies the Problem Description (D^{T_i}), Problem Domain (Dom^{T_i}), Reputation Requirement ($Rep^{T_i}_{min}$), Time Constraint (TC^{T_i}), and Number of Workers (NW^{T_i}). In D^{T_i}, the requester specifies information about the environment they hope to train, which are essential to find suitable shared models. The number of workers indicates the desired number of models to be shared with the requester, as some DRL methods require multiple models to assist in the learning [109].

For model sharing tasks in the framework, the worker's expertise $Exp^{W_j}_{MS-Dom^{T_i}}$, reputation $Rep^{W_j}_{MS-Dom^{T_i}}$, and performance rating $R^{W_j}_{MS-Dom^{T_i}}$ are defined similar to the attributes for the DRL training tasks in Section 7.5.1 (as per Eqs. 16-19). Additionally, it

is essential to assess the similarity between the environment within which the worker's model has been trained and the environment of the requester. It is assumed that DRL problems in the same domain Dom^{T_i} have a set of d pre-defined environment attributes $A = [F_1, F_2, ..., F_d]$ that characterize the environment. Let $m_k^{W_j}$ be the k^{th} model shared by worker W_j (assuming a worker can share multiple models). If $m_k^{W_j}$ is within the task domain Dom^{T_i}, then the similarity S between model m and the requirements of task T_i is given as:

$$S(m_k^{W_j}, T_i) = \sum_{n=1}^{d} w_n \times |F_n^m - F_n^{T_i}| \,, \quad \sum_{n=1}^{d} w_n = 1 \tag{22}$$

where F^m is the set of environment attributes associated with shared model, and F^{T_i} is the set of environment attributes required by the task. The main intuition behind this metric is that models from similar environments could be of more benefit to the requester. The nature of the environment attributes depends on the task, but some examples include the number of agents, number of obstacles (in obstacle avoidance tasks), number of destinations (in autonomous vehicles applications), etc. Depending on the problem, some attributes might have more importance than others, and hence we use a weighted some with a weight w_n for each attribute.

Considering all the aforementioned metrics, the QoS of worker W_j for the DRL model sharing task is then given as:

$$QoS_{MS-Dom^{T_i}}^{W_j} = \frac{Exp_{MS-Dom^{T_i}}^{W_j} \times Rep_{MS-Dom^{T_i}}^{W_j} \times R_{MS-Dom^{T_i}}^{W_j}}{1 + S(m_k^{W_j}, T_i)} \tag{23}$$

where the +1 in the denominator is used for smoothing to avoid issues when $S(m_k^{W_j}, T_i)$ is zero, indicating that the two environments are exactly the same. The QoS is subject to the following constraints:

- $Rep^{W_j}_{MS-Dom^{T_i}} \geq Rep^{T_i}_{min}$
- $R^{W_j}_{MS-Dom^{T_i}} \geq R^{T_i}_{min}$
- $Dom^{T_i} \in Dom^{W_j}$

7.6 Recruitment Optimization Process

Given a task T_i and a pool of candidate workers, the recruitment optimization process aims to recruit a group of NW^{T_i} workers that maximize the expected QoS, while meeting the task requirements and constraints. To do so, a Greedy algorithm for worker recruitment is used, where the optimization process is treated as a knapsack problem. Greedy algorithms are the common in crowdsourcing recruitment works [89, 91], due to their simplicity and scalability. While other algorithms, such as Genetic algorithm and Particle Swarm Optimization, could be used, the simplicity of Greedy algorithm makes it more suitable for deployment on the blockchain. In this problem, the aim is to maximize the weight of items filled in the knapsacks without violating its maximum capacity. In the context of worker recruitment, the maximum capacity represents the specified group size NW^{T_i}, while the weights represent the individual QoS values of the workers. Regardless of the task type (training or model sharing), the recruitment process is the same, with the QoS evaluation and task requirements being the difference between the task types. Once a task is pushed to a worker, they are given a time limit to accept, after which the task request is retracted and given to the next best worker. The recruitment process used in this work is described in Fig. 7.3.

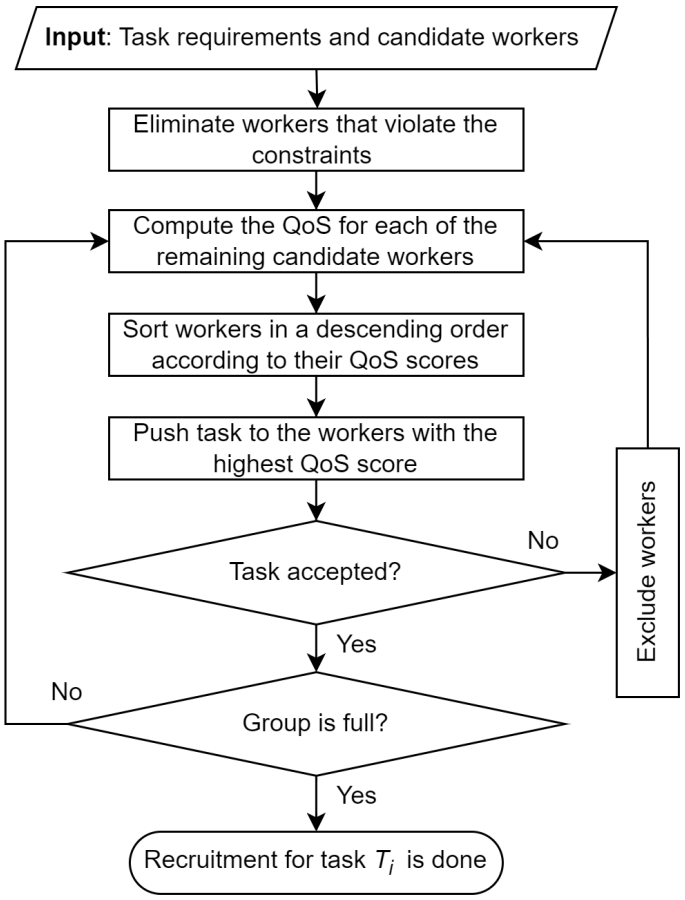


Figure 7.3: Flowchart of the recruitment optimization process.

7.7 Smart Contracts Implementation

In this work, the crowdsourcing platform is built on top of a Consortium Blockchain. The blockchain is responsible for managing users' registration, task requests, task allocation, and feedback through smart contracts. A blockchain is used instead of a centralized management system to provide a decentralized, transparent, and autonomous platform for crowdsourcing with no repudiation. A Consortium Blockchain, specifically, is used due to its ability to offer increased privacy, shared control, efficiency, cost savings, and trust for multiple organizations or entities collaborating on a project or sharing data [92]. A Consortium Blockchain is operated by a group of entities, which introduces increased privacy and trust when compared to public Blockchains, and more collaboration allowance when

compared to private Blockchains, making them suitable for crowdsourcing. Recent works explored the utilization of blockchain with DRL [109, 110], where agents cooperate to train models on the chain, but none provided solutions or services for users.

To execute the proposed crowdsourcing framework on the blockchain, three smart contracts are designed: 1) Users Manager Contract (UMC), Tasks Manager Contract (TMC), and Models Manager Contract (MMC). The users interact with UMC to register in the system by providing information. Task requesters interact with the TMC to submit tasks and provide requirements. The TMC is responsible for the worker recruitment, task allocation, task submission, and feedback processes. If users decide to share their trained models with the platform, they interact with the MMC, which manages and keeps track of the available DRL models. In all tasks, a worker could share the task outcomes, including trained models, through the InterPlanetary File System (IPFS). IPFS returns a unique Content Identifier (CID) that can be used to access the file. The IPFS is a protocol designed to create a content-addressable Peer-to-Peer (P2P) decentralized file system [93]. Workers share the CID with the smart contracts when returning the task outcomes.

The details of the **Users Manager Contract (UMC)** are shown in Table 7.2. The *Worker* data structure holds the worker's information. The information in this structure include the *Worker Address* (Ethereum address) and their *Reputation*. The *Tasks Assigned* and *Tasks Accepted* attributes reflect the total number of tasks assigned to and accepted by the worker. The *Domains* attribute lists the domains covered by the worker, where each domain is represented by an index. The $Status$ indicates whether the worker is active or idle. The *Expertise* indicates the number of tasks completed by the worker per each of the 2 task types. The *Total Ratings* indicates the sum of ratings received for the tasks performed by the worker, for each of the 2 task types. The *Comp. Capabilities* attribute represents the number of CPU cores, the RAM capacity, and the GPU type available for the worker. Finally, the *Requester* data structure holds the *Requester Address*.

Table 7.2: Users Manager Contract (UMC)

Data Structure		
Worker		
Worker Address (address)	*Reputation* (uint)	
Tasks Assigned (uint)	*Domains* (uint[])	
Tasks Accepted (uint)	*Status* (uint)	
Expertise (uint[2])	*Total Ratings* (uint[2])	
Comp. Capabilities (uint[3])		
Requester		
Requester Address (address)		
Variables		
Workers List (address → Worker)		
Domain Workers (uint → address[])		
Function	**Parameters**	**Return**
addWorker()	Worker Information	-
addRequester()	Requester Address	-
updateStatus()	Status	-
updateInfo()	Performance Details	-
getWorkers()	Domain	*Worker*[]

The UMC stores workers' information in the *Workers List* mapping, which maps a worker's address to their *Worker* object. Workers are grouped into domains in the *Domain Workers* mapping, which maps a domain code to an array of Ethereum addresses for workers in that domain. The *addWorker()* and *addRequester()* functions allow users to register in the platform by providing necessary information to create *Worker* or *Requester* objects. The *updateStatus()* and *updateInfo()* functions are responsible for updating the worker's attributes frequently following events in the platform, like performing a task or receiving a review. The *getWorkers()* function is responsible for retrieving all worker objects in given a domain code.

The **Task Manager Contract (TMC)** is shown in Table 7.3. The *Task* data structure holds information about the task, including the *Requester*'s Ethereum address, the *Duration* limit within which the task is to be finished, and the *Type* of the task. The *No. Workers* attribute reflects the number of workers required by the requester, *Min. Reputation* and *Min.*

Rating describe the reputation and rating requirements for the task, *Domain* indicates the domain code of the task, *Problem Description* contains a textual description of the problem details, the *Status* indicates whether the task is pending or completed, and *Computational Reqs.* indicates the minimum requirements for CPU, GPU, and RAM.

Table 7.3: Tasks Manager Contract (TMC)

Data Structure		
Task		
Requester (address)	*Duration* (uint)	
Type (uint)	*No. Workers* (uint)	
Min. Reputation (uint)	*Min. Rating* (uint)	
Problem Description (string)	*Status* (uint)	
Computational Reqs. (uint[3])	*Domain* (uint)	
Variables		
Domain Tasks (uint → Task[])		
Function	**Parameters**	**Return**
addTask()	Task Information	-
allocateTask()	Task Requirements	-
updateTaskStatus()	Status	-
submitOutcome()	Task Outcomes	-

The TMC stores the tasks information in a *Domain Tasks* mapping, which maps a domain code to an array of task objects in that domain. The requester interacts with the *addTask()* function by providing necessary information about the task to create a *Task* object. The *allocateTask()* function is responsible for worker recruitment and forwarding the task to the selected workers. the task status throughout the process is updated through the *updateTaskStatus()* function, while the *submitOutcome()* functions is called to submit the outcomes of a given task.

The **Model Manager Contract (MMC)** is shown in Table 7.4. The MMC is responsible for storing information about shared models. The *Owner* attribute stores the address of the owner, the *CID* stores the IPFS identifier for the shared files, *Description* stores a textual description of the model and its application, *Domain* indicates the domain code of the model's application, and *Environment Details* stores attributes that identify the model's

environment.

Table 7.4: Models Manager Contract (MMC)

Data Structure		
Model		
Owner (address)	*CID* (string)	
Description (string)	*Domain* (uint)	
Environment Details (uint[])		
Variables		
Domain Models (uint → Model[])		
Function	**Parameters**	**Return**
addModel()	Model Info	-
allocateModel()	Model Requirements	Model[]

The MMC stores models' information in the *Domain Models* mapping, which maps a domain code to the available models for that domain. A worker calls the *addModel()* function to add details about the shared model, while the *allocateModel()* function is responsible for finding the most suitable models amongst the available models based on the requester's requirements.

In terms of time complexity, most of the functions in the three smart contracts have a complexity of O(N) or O(1). The *addWorker()*, *addRequester()*, *addTask()*, and *addModel()* functions have a complexity of O(N), where N is the number of existing elements (workers, requesters, etc.), since the functions check for duplicates before adding. Aside from the allocation functions, all the remaining functions have a simple complexity of O(1). The *allocateTask()* and *allocateModel()* functions employ a greedy algorithm that uses a sorting mechanism to find the best workers, which hence has a time complexity of O(N logN).

7.8 Framework Time Sequence

Figure 7.4 shows a time sequence diagram for scenarios under the proposed blockchain-based crowdsourced DRLaaS. It discusses the interactions between the users and the smart

contracts constituting the framework. For DRL training tasks, these interactions are given as follows:

- **User Registration**: Users register to the UMC by invoking the *addWorker()* and *addRequester()* functions. Each user, worker or requester, has a unique Ethereum address linked to their account. Workers provide information related to their domains and computational capabilities. The rest of the attributes are initialized and updated following task executions.

- **Task Request and Allocation**: A request creates a task by interacting with the TMC through the *addTask()* function and providing the necessary information and the required payment. The TMC allocates the task to suitable workers through the *allocateTask()* function, and workers perform the task and return their outcomes through IPFS to TMC through the *submitOutcome()* function. The outcomes are then forwarded to the task requester.

- **Feedback**: Requesters rate the provided outcomes, and the ratings are used in the UMC to update workers' attributes. Workers then get paid for the tasks performed.

As for the DRL model sharing tasks, following the user registration, the process is as follows:

- **Model Upload**: Workers who wish to share their trained models upload their files to the IPFS and interact with the MMC through the *addModel()* function by sharing the model details.

- **Task Request and Allocation**: A requester creates a model sharing task by interacting with the TMC and specifying the task type and the attributes of the desired environment. The TMC interacts with the MMC by invoking the *allocateModel()* function, which returns the most suitable models, which are then forwarded to the requester.

- **Feedback**: The requester rates the shared model, and the UMC updates the workers' attributes accordingly and provides the payments

7.9 Experiments and Evaluation

This section presents and discusses several experiments conducted to validate the proposed methods. The experiments are conducted on several Multi-agent DRL environments, namely Target Localization [44, 45], Fleet Coordination for Autonomous Vehicles [97], and Multi-Agent Maze Cleaning [99]. We opted to employ Multi-agent DRL instead of single-agent DRL due to its increased complexity, which allows for a more rigorous examination of the robustness and adaptability of our proposed methods. All the simulations have been conducted using an Intel E5-2650 v4 Broadwell workstation equipped with 128 GB RAM, 800 GB SSD, and NVIDIA P100 Pascal GPU (16 GB HBM2 memory). To validate the proposed framework, the key attributes in the worker selection metrics are evaluated, including agent's computational capabilities CC^{W_j} and model similarity $S(m_k^{W_j}, T_i)$. The learning convergence in the following experiments is evaluated in terms of Episode Length, which is the time it takes for the agents to finish the task. The episode length is tracked throughout the learning, to reflect how fast the agents learn to efficiently perform the task. It is also essential to track how long the training process takes, in wall time, in order to verify the importance of the proposed selection metrics. For all the experiments, the conditions of the simulations are fixed, and only the variables under examination are varied.

7.9.1 DRL Application Environments

The DRL environments used in to validate the proposed methods are:

- Target Localization [44, 45]: this is a multi-agent problem in which the location of

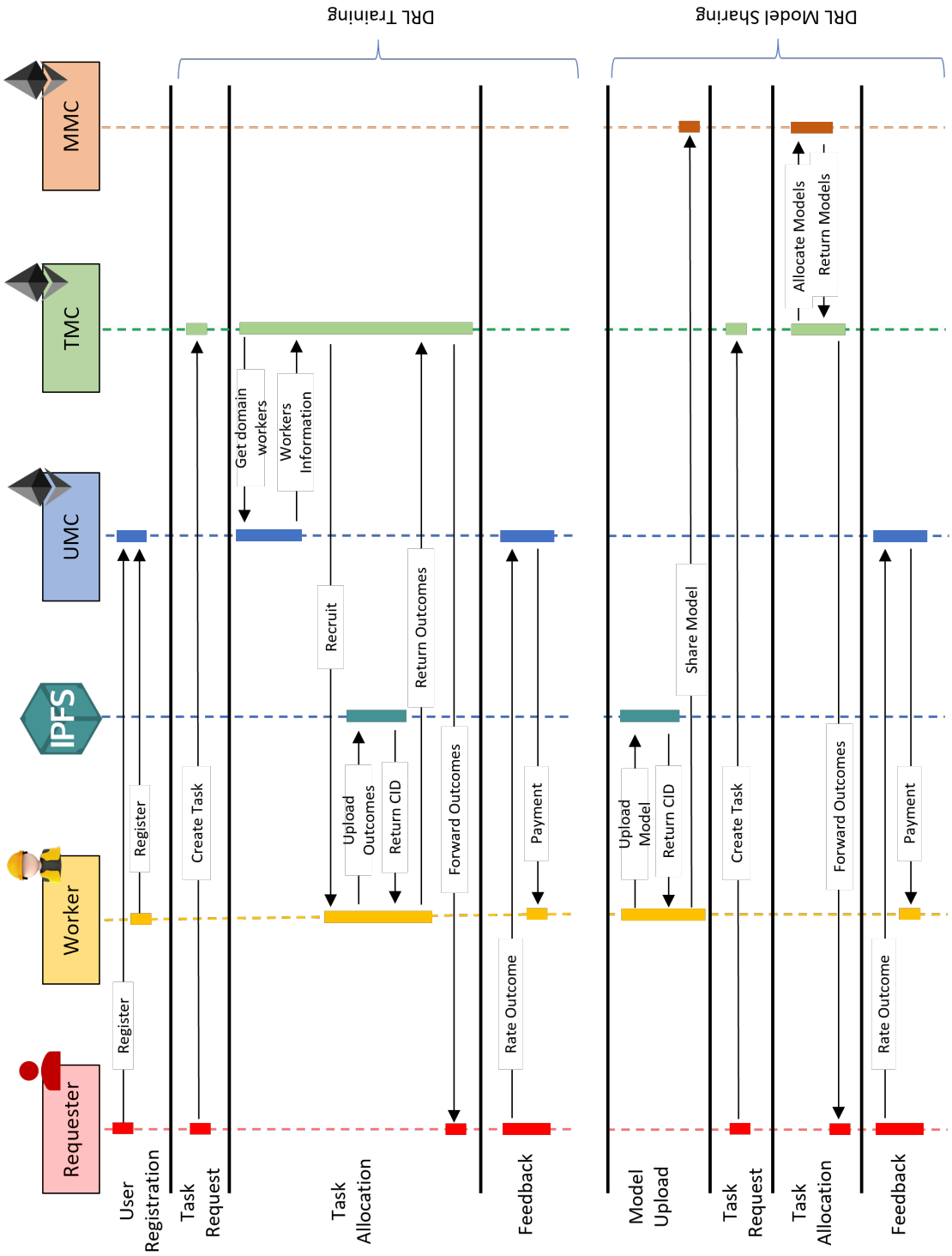


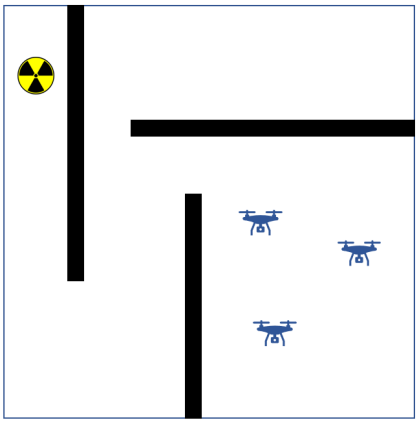
Figure 7.4: The interactions between the users and smart contracts as part of the proposed framework.

a certain target is to be identified, based on sensory readings collected by the sensing agents. This is common in applications related to radiation monitoring, search and rescue, and path-finding. In this problem, the sensing agents (robots or UAVs) observe the environment, collect data readings, and communicate with each other in order to cooperate and locate the unknown target. The learning problem is complicated since the agents need to learn how to communicate and coordinate, in addition to how to take the best set of actions in order to reach the target as fast as possible. We model the DRL environment as discussed and presented in [44]. The agents' observations are modeled as 2D heatmaps and fed to a Convolutional Neural Network (CNN) that acts as an actor network in a Proximal Policy Optimization (PPO) algorithm. A sample scenario of the target localization problem is shown in Fig. 7.5a.

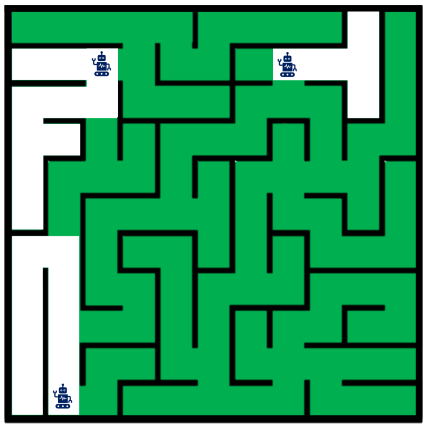
- Multi-Agent Maze Cleaning [99]: in this problem, a group of agents is placed in a maze with the task of cleaning it as fast as possible. Initially, the maze is entirely dirty, and each spot covered by an agent is considered to have been cleaned. This problem requires coordination between the agents to allocate tasks, and for them to learn how to quickly clean the maze. Each agent observes its own location, the location of the other agents, as well as the status of the map, in 2D format. The observations are fed to a CNN as the actor network in a PPO algorithm. A sample scenario of the maze cleaning problem is shown in Fig. 7.5b.

- Fleet Coordination for Autonomous Vehicles [97]: in this problem, a team of autonomous vehicles is tasked with picking up and dropping customers at specific locations. Customers are randomly distributed in a certain area of interest, with each customer having a certain desired destination. Vehicles have a limit in terms of the number of customers accommodated simultaneously. The team's goal is to minimize

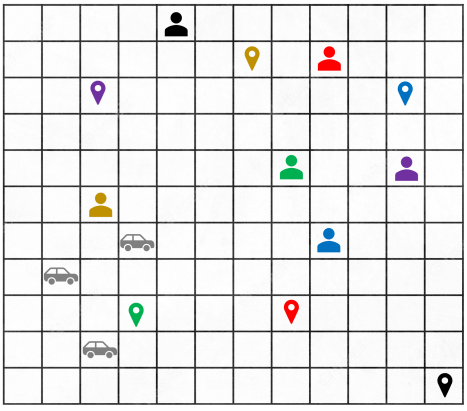
the time needed to pick up and drop off customers off at their destinations. The complexity of the learning comes from the fact that the agents (i.e. the vehicles) need to cooperate to properly allocate tasks, such as the time is minimized. This is seen as a multi-objective DRL problem, since each agent is required to find the shortest path that goes through customers towards their destinations, while coordinating with other agents. The agents observe their locations, the customers' locations, as well as the desired destinations, and take actions accordingly. The observations are modeled as 2D heatmaps and fed to CNNs with PPO as the DRL optimization algorithm. A sample scenario of the fleet coordination problem is shown in Fig. 7.5c.



(a) Target Localization



(b) Maze Cleaning



(c) Fleet Coordination

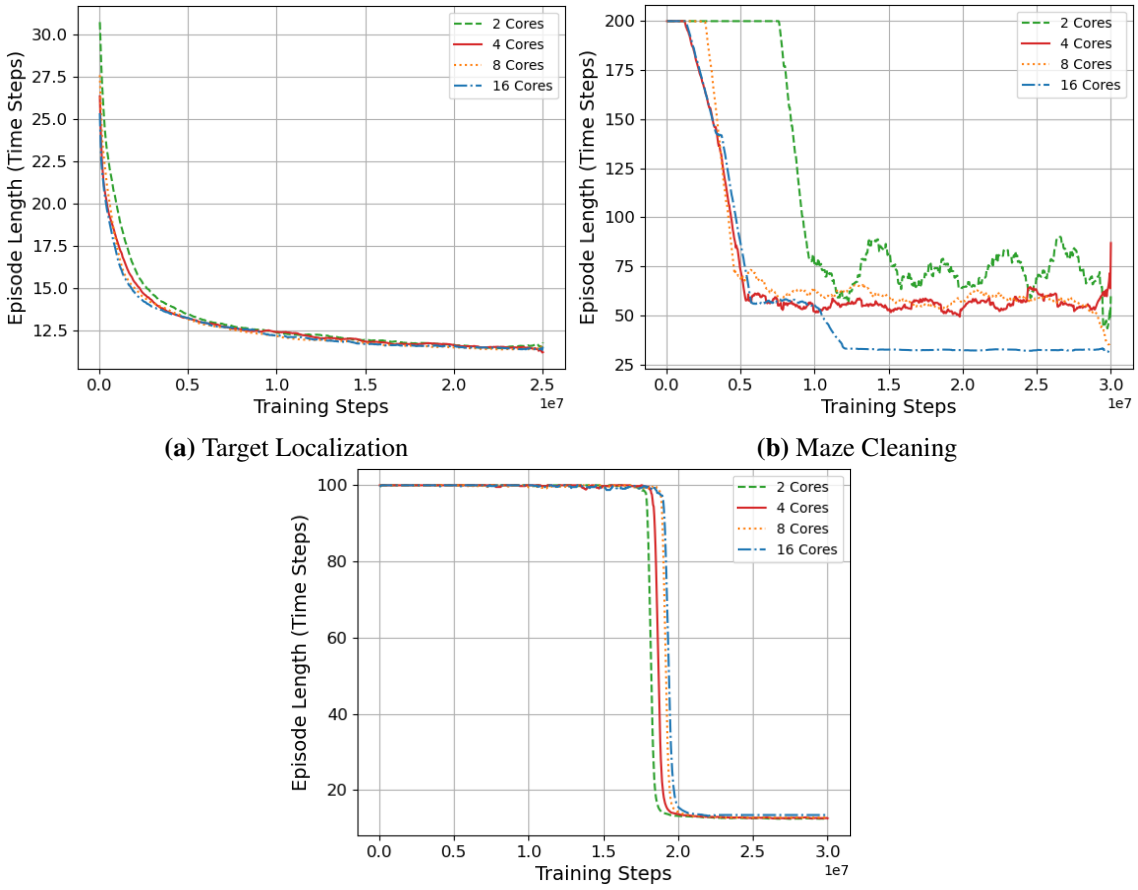
Figure 7.5: Use-case scenarios of the DRL application environments used to validate the proposed methods.

These environments are examples of complex DRL problems that require expertise and computational resources. In the following sections, the proposed methods will be validated using these environments, in which users rely on the proposed DRLaaS framework to push tasks to workers and get outcomes.

7.9.2 DRL Training Tasks

As discussed in Section 7.5.1, most common DRL algorithms utilize parallelized processing to run copies of the environment for experience collection during training. To verify the importance of considering the worker's CPU capabilities $CC_{CPU}^{W_j}$ during the recruitment stage for DRL training tasks, Fig. 7.6 presents the effect of varying the number of CPU cores on the training convergence of DRL for the 3 applications. For many applications, parallelizing the data collection in DRL does not significantly affect the learning convergence, which is the case for Target Localization (Fig. 7.6a) and Fleet Coordination (Fig. 7.6c). However, in some applications where the task has a long horizon (takes generally longer time steps to finish), increasing the number of parallel processes brings benefits, as seen in Fig. 7.6b for the Maze Cleaning environment. This is because parallelizing the data collection enhances the exploration process, since more unique experiences in different parallel environments are being collected, resulting in better training.

While the training convergence across different number of cores is similar for most applications, and slightly better with more cores for some other applications, the case is different in terms of wall time. Fig. 7.7a elaborates on the number of training steps obtained in each of the DRL applications within 12 hours of wall time, for varying number of CPU cores. It can be seen that, despite the similar training convergence in Target Localization and Fleet Convergence environments (previously shown in Fig. 7.6), the training time is significantly different. Specifically, a training method using 16 cores can execute 2.5, 2.4, and 2.2 times the training steps with 2 cores, for the Target Localization, Maze Cleaning,



(a) Target Localization

(b) Maze Cleaning

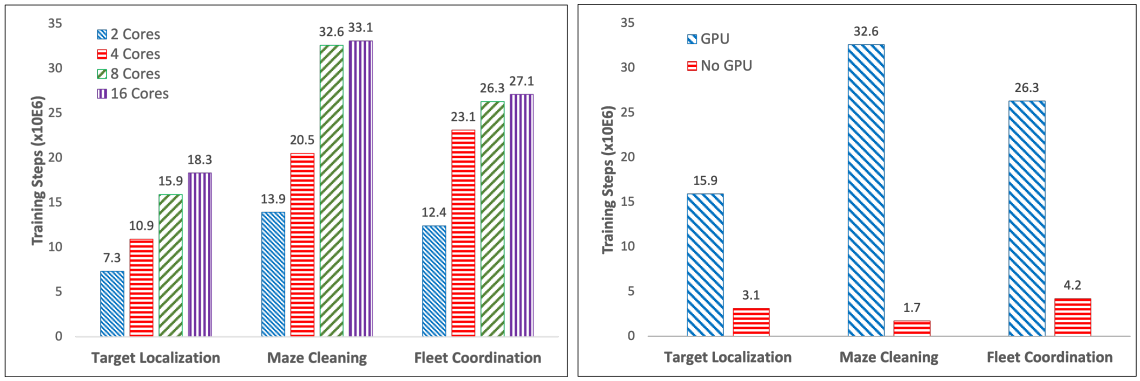
(c) Fleet Coordination

Figure 7.6: The effect of parallelizing the DRL process over a varying number of CPU cores on the learning convergence, for different DRL environments.

and Fleet Coordination environments, respectively. This means that, in the case of Target Localization for instance, the algorithm with 16 CPU cores needed less than half the wall time to converge when compared to the algorithm with 2 CPU cores. This shows the importance of the CPU capabilities attribute used to assess candidate workers. It is also essential to point out that the effect of CPU cores on the wall time begins to saturate as the CPU cores increase, which can be shown when going from 8 to 16. This verifies the discussion in Section 7.5.1, which states that at a certain point, further parallelization does not bring more benefit, and justifies the use of the tan^{-1} function in the assessment process.

In terms of GPU, Fig. 7.7b presents the effect of training with and without GPU, for

the 3 different applications. GPUs are essential when training DNNs, especially when dealing with CNNs. As can be seen in the figure, training with GPU for a duration of 12 hours executes up to 19 times the training steps executed without GPU, which validates the consideration of GPU capabilities in the worker assessment process.



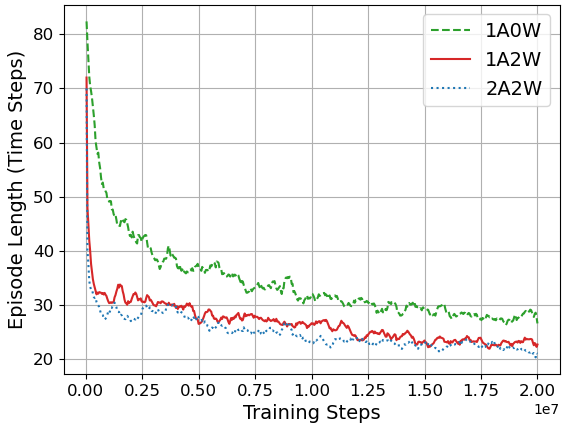
(a) Training Steps vs CPU Cores **(b)** Training Steps vs GPU

Figure 7.7: The total number of training steps in a 12h duration, while (a) varying the number of CPU cores (parallelized DRL) and (a) varying the use of GPU.

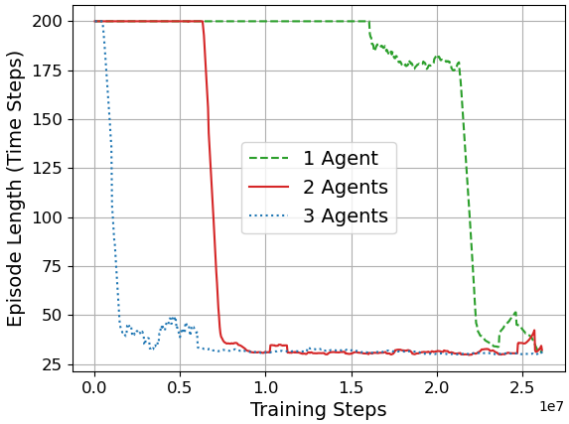
7.9.3 DRL Model Sharing

To study the importance of the similarity metric proposed in Eq. 22 for the DRL model sharing tasks, Fig. 7.8 shows the learning performance when using different expert models in assisting DRL, using Demonstration Cloning (DC) [45]. In DC, expert models help current agents collect better experiences with more exposure to reward values, resulting in better learning. For the target localization problem (Fig. 7.8a), a team of agents is to be trained on environment with 3 agents and 3 walls (3A3W), with the help of 3 expert models that have been previously trained on different environments, including 1A0W, 1A2W, and 2A2W. It can be seen that the closer the expert model is to the current environment of interest (3A3W), the better the learning and the faster the convergence. Specifically, the expert model from the 2A2W environment assists the training the best, since it is the closest to the current environment. Similarly, for the maze cleaning problem, an expert model

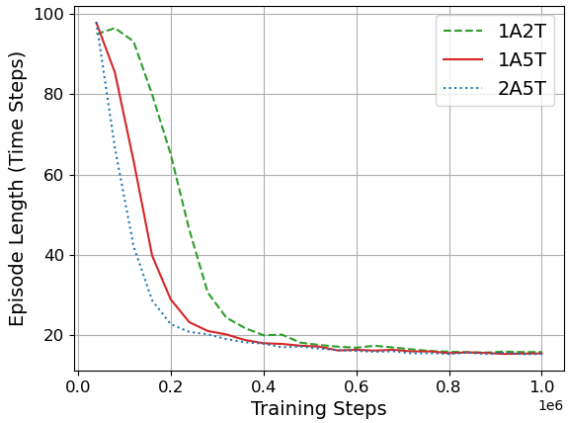
trained on a 3-agent environment provides the best assistant to train a 5-agent environment, when compared to expert models trained on single- and two-agent environments. For fleet coordination, an expert model trained on an environment of 2 agents and 5 targets (2A5T) gives the best assistant when training an environment of 3 agents and 10 targets (3A10T), when compared to expert models trained on 1A2T and 1A5T environments.



(a) Target Localization (3A3W)



(b) Maze Cleaning (5-agent)



(c) Fleet Coordination (3A10T)

Figure 7.8: The effect of model similarity on the learning performance, when training a 3-agent 3-wall target localization problem (3A3W), a 5-agent maze cleaning environment, and a 3-agent 10-target fleet coordination problem (3A10T).

7.9.4 Recruitment Optimization

To validate the choice of the greedy algorithm for the optimization of the recruitment process, the proposed method is benchmarked against common methods in crowdsourcing works, such as Genetic Algorithm (GA) [1], Particle Swarm Optimization (PSO) [95], and Ant Colony Optimization (ACO) [111]. In GA-based methods, a population of candidate solutions (i.e. possible sets of workers) is created and iteratively modified through genetic operators such as crossover, mutation, and selection, aiming to converge toward an optimal solution eventually. In PSO-based methods, a swarm of candidate solutions (particles) explores potential worker selections based on their QoS values. Here, the positions and velocities of the particles are iteratively updated based on their own experience (personal best) and the experience of the swarm (global best). In ACO-based methods, the recruitment problem is modeled as a graph where nodes represent workers and edges represent possible selections. Ants placed on the graph deploy pheromones on the paths they take, which is proportional to the quality of the solution. Fig. 7.9 compares the performance of the greedy-based recruitment method with the benchmarks in terms of group average QoS for different group sizes, where a group size is the number of workers recruited. The results are obtained using a synthetic dataset of 600 candidate workers, where the workers' attributes are generated randomly following a uniform distribution for fair comparison. It can be seen in the figure that, regardless of the group size, a greedy-based method always outperforms the other benchmarks. This is mainly because the selection of workers occurs independently for each worker in a greedy method. Hence, the size of its search space is simply the available pool of workers, making the search process simple. On the other hand, GA-, PSO-, and ACO-based methods operate on a broader search space by considering the different combinations of groups of workers, making the search problem harder. It is worth mentioning that for the current results in Fig. 7.9, on average, the greedy algorithm is nearly 200 times faster than GA, 73 times faster than PSO, and 132 times faster than

ACO. This is significant for the deployment of the proposed framework on the blockchain.

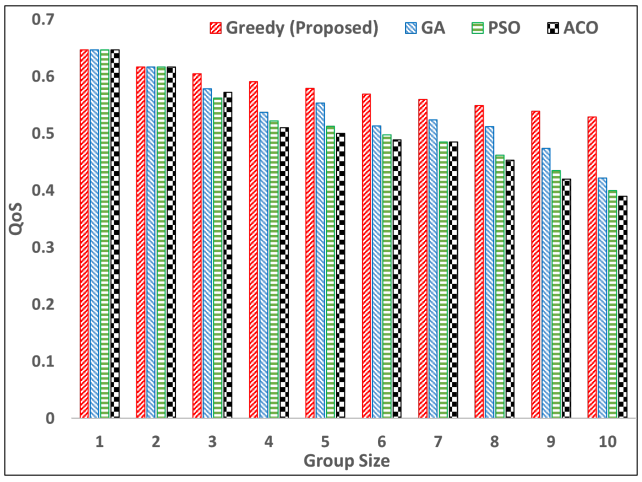


Figure 7.9: Comparison between the proposed greedy-based recruitment and the benchmarks for different group sizes.

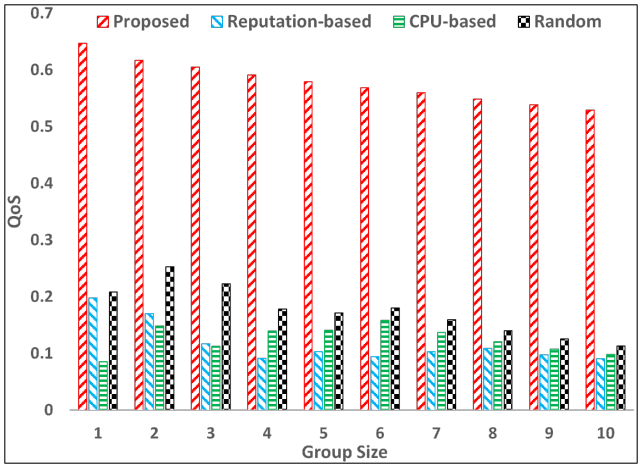
To analyze the effect of the recruited workers on the QoS and the DRL training results, Fig. 7.10 compares the proposed work based on the QoS in Eq. 21 with several recruitment benchmarks. Since no works in the literature address the crowdsourcing of DRL training tasks, we choose benchmarks from similar domains in crowdsourcing and federated learning. Reputation-based recruitment methods [65] focus on reputation-related attributes to assess the workers, with the aim of selecting the most reputed ones. CPU-based recruitment methods [112, 113] focus on the computational capabilities of the workers in terms of CPU and RAM. We also consider a random recruitment method as a baseline. As shown in Fig. 7.10a, the three benchmarks fall short in terms of QoS when compared to the proposed work. This is mainly because each of the benchmarks considers only a subset of the DRL-related parameters considered in Eq. 21. To demonstrate the effect of the recruitment of the DRL task, Fig. 7.10b shows the training results (in terms of episode length to be minimized) using the workers selected by each benchmark. Here, the aforementioned 600-candidate workers dataset is used, and selected workers are given the task of training a model to solve a maze-cleaning environment of 4 agents. For a fair comparison, we modify our method to match each of the benchmarks by keeping only the attributes considered by

them. As can be seen in the figure, reputation-based and random recruitment benchmarks struggle with convergence. This is due to the lack of DRL-related attributes such as computational capabilities. On the other hand, the CPU-based recruitment benchmark performs nearly as well as the proposed work, with the latter showing a slight performance advantage in terms of episode length. It is also worth noting that the workers chosen by the proposed work are nearly 20 times faster than the CPU-based benchmark in training the $30M$ steps, which is due to the use of GPU that is missing from the benchmark.

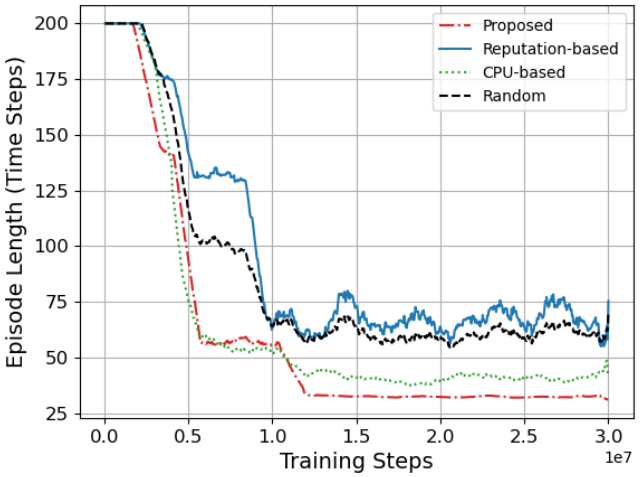
7.9.5 Blockchain and Smart Contracts Complexity Analysis

To analyze the complexity and feasibility of the proposed smart contracts, Table 7.5 presents the gas cost of the deployment and execution of the smart contracts and their functions. Gas cost is defined as the computational effort required to execute operations such as smart contracts. In public blockchains, the gas cost can be used to determine the fees to be paid for executing transactions or running smart contracts based on the gas price. Since a Consortium Blockchain is proposed, the deployment and execution of the smart contracts do not require any payments by the users since the gas price is zero. However, the gas cost is a good measure of the complexity of the smart contracts and its functions to indicate their feasibility. As seen in the table, the gas costs are low, reflecting the feasibility of the proposed contracts. For reference, we present a benchmark of the gas cost of deploying and executing a similar UMC as discussed in [92].

It is worth mentioning that consortium blockchains are generally known to have a significantly lower latency and a higher throughput when compared to public blockchains. This is mainly because they have a predetermined set of nodes that are known to each other, which allows faster transaction processing due to optimized consensus mechanisms with a small number of trusted validators. The exact computations of latency and throughput depend on several factors such as the number of nodes, security requirements, and the



(a)



(b)

Figure 7.10: Comparison between the proposed method and different benchmarks in terms of (a) QoS for different group sizes and (b) DRL training results for a group size of 4, using the maze cleaning environment.

consensus mechanisms, which are out of the scope of this work. However, since DRL task allocation in the proposed crowdsourcing system is not time-sensitive, studying the latency and throughput is not significant to this work. Nonetheless, in common consortium blockchains such as Quorum, the blockchain can handle hundreds to thousands of transactions per second (throughput) with latency as low as few milliseconds to 2 seconds.

Table 7.5: Blockchain gas cost.

Contract	Function	gas cost
UMC	deployment	735736
	addWorker()	85455
	addRequester()	74569
	updateStatus()	37286
	updateInfo)	67486
	getWorkers()	95287
TMC	deployment	1477451
	addTask()	487452
	allocateTask()	1053842
	updateTaskStatus()	99374
	submitOutcome()	29374
MMC	deployment	1357425
	addModel()	634341
	allocateModel()	903842
UMC - Benchmark [92]	deployment	1228566
	addUser()	352352

Chapter 8

Conclusion and Future Direction

8.1 Conclusion

This thesis focused on designing scalable Multi-Agent Deep Reinforcement Learning (MDRL) methods to address the complex problem of cooperative target search and localization in dynamic and uncertain environments. It began by addressing the fundamental problem of cooperative target search and localization using Multi-Agent Deep Reinforcement Learning MDRL in simple environments. The initial focus was on developing scalable methods that could handle increasing numbers of agents while ensuring efficient coordination and collaboration between them. In these simpler scenarios, the primary challenge was to ensure that the agents could work together effectively, translate their observations into cooperative actions, and achieve quick and cost-efficient localization. This early work laid the groundwork for exploring more realistic and challenging environments.

As the research progressed, we shifted our focus to more realistic and challenging environments, where obstacles introduced significant complexities. These obstacles impacted both agent mobility and the accuracy of sensor data, making the task far more demanding.

The complexity of the problem is further amplified when considering scenarios with uncertainties, where targets could be unreachable or non-existing. We proposed initial MDRL-based methods while using methods such as Convolutional AutoEncoders and Breadth-First Search. While these initial methods proved successful in such complex environments, they faced long training times to reach convergence. To overcome these challenges, we introduced a novel Demonstration Cloning (DC) method, which integrates ideas from Imitation Learning (IL). This solution enabled agents to learn from the expertise of others, in addition to their own experiences, which significantly reduces training times. Building on this, we proposed the Multi-Expert Demonstration Cloning (MEDC) method, leveraging blockchain technology to allow for decentralized sharing of pre-trained models. Realizing the need for knowledge sharing and the lack of available pre-trained MDRL models, we proposed a framework where trained models could be easily shared between users to assist in training new MDRL solutions. The blockchain-assisted MEDC framework provided the foundation for this, ensuring that new models could benefit from the collective intelligence of previous agents, thereby significantly reducing the time required to develop new MDRL solutions.

Finally, recognizing the challenges that users face when attempting to train DRL models, we proposed a novel platform: Deep Reinforcement Learning as a Service (DRLaaS). DRLaaS offers a crowdsourced solution to the difficulties of accessing computational resources and expertise, allowing users to outsource DRL design and training tasks to a community of experts rather than relying on expensive, centralized services. This service-oriented approach democratizes access to DRL development, providing a scalable and cost-effective way for users to develop complex multi-agent systems for problems like target search and localization.

8.2 Future Directions

Throughout this thesis, we have showcased the effectiveness of our proposed solutions through rigorous experiments and evaluations on real-world applications and scenarios of target search and localization. However, the target localization problem remains sensitive, demanding, and continually evolving. Real-world applications, such as search and rescue, environmental monitoring, and surveillance, are characterized by dynamic conditions that constantly push the boundaries of existing MDRL solutions. The unpredictability of environments, the presence of obstacles, and the uncertainty in target behavior all contribute to the complexity of the problem. As these applications become more critical, there is a continuous need to design new, innovative solutions to keep up with the evolving demands of target localization tasks.

One key direction for future work lies in the **sustainability** of multi-agent systems in target localization applications, particularly in long-term deployments where agents need to operate for extended periods. Introducing energy management solutions, such as integrating charging stations within the environment, could ensure that the agents maintain their effectiveness over longer missions. This would require designing policies, based on MDRL, that consider charging stations in the decision-making process, and address the increased complexity in the coordination between agents.

Another promising direction is to extend the current MDRL methods to support **target tracking**, where the target is mobile and potentially evasive. In such scenarios, the target may actively avoid detection, requiring agents to adapt their strategies dynamically and work collaboratively to track and intercept the moving target. This can be tackled using *self-play*, a technique in DRL where two teams of agents, acting as the trackers and the targets, take turns building their policies. This allows the agents to learn the evasive behaviors and develop policies that counter those of the evasive target, ultimately enhancing their ability to handle complex, dynamic interactions with mobile targets.

Lastly, the issue of explainability is particularly critical in sensitive target search and localization applications, such as military operations, disaster response, and intrusion detection. In these contexts, the decisions made by autonomous agents must be interpretable and transparent, as incorrect decisions can have severe consequences. Future work should focus on developing explainable MDRL (XMDRL) systems that provide clear interpretations for their decision-making processes, allowing human operators to understand the rationale behind actions taken by the agents during the target search.

Bibliography

[1] Ahmed Alagha, Shakti Singh, Rabeb Mizouni, Anis Ouali, and Hadi Otrok. Data-driven dynamic active node selection for event localization in IoT applications-a case study of radiation localization. *IEEE Access*, 7:16168–16183, 2019.

[2] Zheng Liu and Shiva Abbaszadeh. Double Q-learning for radiation source detection. *Sensors*, 19(4):960, 2019.

[3] Tomas Lazna, Petr Gabrlik, Tomas Jilek, and Ludek Zalud. Cooperation between an unmanned aerial vehicle and an unmanned ground vehicle in highly accurate localization of gamma radiation hotspots. *Int. Journal of Advanced Robotic Systems*, 15(1):1729881417750787, 2018.

[4] Mohsen Sadi, Youmin Zhang, Wen-Fang Xie, and FM Anim Hossain. Forest fire detection and localization using thermal and visual cameras. In *2021 Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, pages 744–749. IEEE, 2021.

[5] Saleh O Al-Jazzar, Sami Ahmed Aldalahmeh, Des McLernon, and Syed Ali Raza Zaidi. Intruder localization and tracking using two pyroelectric infrared sensors. *IEEE Sensors Journal*, 20(11):6075–6082, 2020.

[6] Raghuram Bharadwaj Diddigi, KJ Prabuchandran, and Shalabh Bhatnagar. Novel sensor scheduling scheme for intruder tracking in energy efficient sensor networks. *IEEE Wireless Communications Letters*, 7(5):712–715, 2018.

[7] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.

[8] Ebtehal Turki Alotaibi, Shahad Saleh Alqefari, and Anis Koubaa. Lsar: Multi-UAV collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019.

[9] Er-wei Bai, Kidane Yosief, Soura Dasgupta, and Raghuraman Mudumbai. The maximum likelihood estimate for radiation source localization: Initializing an iterative search. In *2014 53rd IEEE Conference on Decision and Control*, pages 277–282. IEEE, 2014.

[10] Annie Liu and et al. Design tradeoffs for radiation detection sensor networks. *Preprint, available at http://www. cs. caltech. edu/˜ aliu/documents/IPSN_final. pdf*, 2009.

[11] Annie H Liu, Julian J Bunn, and K Mani Chandy. An analysis of data fusion for radiation detection and localization. In *2010 13th International Conference on Information Fusion*, pages 1–8. IEEE, 2010.

[12] Ashok Sundaresan, Pramod K Varshney, and Nageswara SV Rao. Distributed detection of a nuclear radioactive source using fusion of correlated decisions. In *2007 10th Int. Conf. on Information Fusion*, pages 1–7. IEEE, 2007.

[13] Ahmed Alagha, Shakti Singh, Hadi Otrok, and Rabeb Mizouni. RFLS-resilient fault-proof localization system in IoT and crowd-based sensing applications. *Journal of Network and Computer Applications*, 170, 2020.

[14] Alexei V Klimenko, William C Priedhorsky, Nicolas W Hengartner, and Konstantin N Borozdin. Efficient strategies for low-statistics nuclear searches. *IEEE Transactions on Nuclear Science*, 53(3):1435–1442, 2006.

[15] Hu Xiao, Rongxin Cui, and Demin Xu. A sampling-based bayesian approach for cooperative multiagent online search with resource constraints. *IEEE Transactions on Cybernetics*, 48(6):1773–1785, 2017.

[16] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[17] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[19] Rashid Ali, Imran Ashraf, Ali Kashif Bashir, and Yousaf Bin Zikria. Reinforcement-learning-enabled massive internet of things for 6g wireless communications. *IEEE Communications Standards Magazine*, 5(2):126–131, 2021.

[20] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Thanh Thi Nguyen et al. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[23] Christopher Berner et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[24] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[25] Jingjing Cui, Yuanwei Liu, and Arumugam Nallanathan. Multi-agent reinforcement learning-based resource allocation for UAV networks. *IEEE Transactions on Wireless Communications*, 19(2):729–743, 2019.

[26] Junchen Jin and Xiaoliang Ma. Hierarchical multi-agent control of traffic lights based on collective learning. *Engineering applications of artificial intelligence*, 68: 236–248, 2018.

[27] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[29] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[30] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pages 1–49, 2021.

[31] John Schulman et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[32] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *2016 Proc. Int. Conf. on Learning Representations (ICLR)*, 2016.

[33] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd Int. Conf. on Machine Learning (ICML)*, volume 37, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[34] Saurabh K Pandey and Mukesh A Zaveri. Event localization in the internet of things environment. *Procedia computer science*, 115, 2017.

[35] François Grondin et al. Sound event localization and detection using crnn on pairs of microphones. In *2019 Proc. Detection and Classification of Acoustic Scenes Events Workshop*, 2019.

[36] Jren-Chit Chin, David KY Yau, Nageswara SV Rao, Yong Yang, Chris YT Ma, and Mallikarjun Shankar. Accurate localization of low-level radioactive source under noise and measurement errors. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 183–196. ACM, 2008.

[37] Zhenyu Liu, Wenhan Dai, and Moe Z Win. Node placement for localization networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[38] AH Mohamed and KH Marzouk. Optimizing the energy consumption of wireless sensor networks. *International Journal of Applied Information Systems (IJAIS) Volume*, 10, 2015.

[39] Ahmed Alagha, Rabeb Mizouni, Shakti Singh, Hadi Otrok, and Anis Ouali. SDRS:

A stable data-based recruitment system in IoT crowdsensing for localization tasks. *Journal of Network and Computer Applications*, 177:102968, 2021.

[40] KP Ziock and WH Goldstein. The lost source, varying backgrounds and why bigger may not be better. In *AIP Conference Proceedings*, volume 632, pages 60–70. American Institute of Physics, 2002.

[41] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *2020 Proc. Int. Conf. on Learning Representations (ICLR)*, 2020.

[42] Deepak A Vidhate and Parag Kulkarni. Cooperative multi-agent reinforcement learning models (cmrlm) for intelligent traffic control. In *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*, pages 325–331. IEEE, 2017.

[43] Junjia Liu, Huimin Zhang, Zhuang Fu, and Yao Wang. Learning scalable multi-agent coordination by spatial differentiation for traffic signal control. *Engineering Applications of Artificial Intelligence*, 100:104165, 2021.

[44] Ahmed Alagha, Shakti Singh, Rabeb Mizouni, Jamal Bentahar, and Hadi Otrok. Target localization using multi-agent deep reinforcement learning with proximal policy optimization. *Future Generation Computer Systems*, 136:342–357, 2022.

[45] Ahmed Alagha, Rabeb Mizouni, Jamal Bentahar, Hadi Otrok, and Shakti Singh. Multi-agent deep reinforcement learning with demonstration cloning for target localization. *IEEE Internet of Things Journal*, 2023.

[46] Hani Sami, Jamal Bentahar, Azzam Mourad, Hadi Otrok, and Ernesto Damiani. Graph convolutional recurrent networks for reward shaping in reinforcement learning. *Information Sciences*, 608:63–80, 2022.

[47] Yunlong Dong, Xiuchuan Tang, and Ye Yuan. Principled reward shaping for reinforcement learning via lyapunov stability theory. *Neurocomputing*, 393:83–90, 2020.

[48] Ashvin Nair et al. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299, 2018.

[49] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[50] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[51] Zheqi Zhu, Shuo Wan, Pingyi Fan, and Khaled B Letaief. Federated multiagent actor–critic learning for age sensitive mobile-edge computing. *IEEE Internet of Things Journal*, 9(2):1053–1067, 2021.

[52] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *Ieee Network*, 33(5):156–165, 2019.

[53] Shuai Yu, Xu Chen, Zhi Zhou, Xiaowen Gong, and Di Wu. When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network. *IEEE Internet of Things Journal*, 8(4):2238–2251, 2020.

[54] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems. *IEEE Robotics and Automation Letters*, 4(4):4555–4562, 2019.

[55] Xinle Liang, Yang Liu, Tianjian Chen, Ming Liu, and Qiang Yang. Federated transfer reinforcement learning for autonomous driving. In *Federated and Transfer Learning*, pages 357–371. Springer, 2022.

[56] Jiaju Qi, Qihao Zhou, Lei Lei, and Kan Zheng. Federated reinforcement learning: Techniques, applications, and open challenges. *arXiv preprint arXiv:2108.11887*, 2021.

[57] Xiaofeng Fan, Yining Ma, Zhongxiang Dai, Wei Jing, Cheston Tan, and Bryan Kian Hsiang Low. Fault-tolerant federated reinforcement learning with theoretical guarantee. *Advances in Neural Information Processing Systems*, 34:1007–1021, 2021.

[58] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*, pages 896–902. IEEE, 2015.

[59] Shuai Zhao, Manoop Talasila, Guy Jacobson, Cristian Borcea, Syed Anwar Aftab, and John F Murray. Packaging and sharing machine learning models via the acumos ai open platform. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 841–846. IEEE, 2018.

[60] Abhinav Kumar, Reza Tourani, Mona Vij, and Srikathyayani Srikanteswara. Sclera: A framework for privacy-preserving mlaas at the pervasive edge. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 175–180. IEEE, 2022.

[61] Sukhdeep Singh, Joseph Thaliath, Isma Farah Siddiqui, Ashish Jain, Seungil Yoon, Mohammad Attique, and Nawab Muhammad Faseeh Qureshi. Machine learning as a service for beyond 5g networks. In *2022 IEEE Globecom Workshops (GC Wkshps)*, pages 455–460. IEEE, 2022.

[62] Dan Graur, Damien Aymon, Dan Kluser, Tanguy Albrici, Chandramohan A Thekkath, and Ana Klimovic. Cachew: Machine learning input data processing as a service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 689–706, 2022.

[63] Menatalla Abououf, Hadi Otrok, Rabeb Mizouni, Shakti Singh, and Ernesto Damiani. How artificial intelligence and mobile crowd sourcing are inextricably intertwined. *IEEE Network*, 35(3):252–258, 2020.

[64] Yingying Ren, Wei Liu, Anfeng Liu, Tian Wang, and Ang Li. A privacy-protected intelligent crowdsourcing application of iot based on the reinforcement learning. *Future generation computer systems*, 127:56–69, 2022.

[65] Menatalla Abououf, Shakti Singh, Hadi Otrok, Rabeb Mizouni, and Ernesto Damiani. Machine learning in mobile crowd sourcing: A behavior-based recruitment model. *ACM Transactions on Internet Technology (TOIT)*, 22(1):1–28, 2021.

[66] Mahmoud Aly, Kamel H Rahouma, and Safwat M Ramzy. Pay attention to the speech: Covid-19 diagnosis using machine learning and crowdsourced respiratory and speech recordings. *Alexandria Engineering Journal*, 61(5):3487–3500, 2022.

[67] Supattra Puttinaovarat and Paramate Horkaew. Flood forecasting system based on integrated big and crowdsource data by using machine learning techniques. *IEEE Access*, 8:5885–5905, 2020.

[68] Peter Washington, Emilie Leblanc, Kaitlyn Dunlap, Yordan Penev, Aaron Kline, Kelley Paskov, Min Woo Sun, Brianna Chrisman, Nathaniel Stockham, Maya Varma, et al. Precision telemedicine through crowdsourced machine learning: testing variability of crowd workers for video-based autism feature recognition. *Journal of personalized medicine*, 10(3):86, 2020.

[69] Joseph Chee Chang, Saleema Amershi, and Ece Kamar. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2334–2346, 2017.

[70] Irene Martín-Morató and Annamaria Mesaros. Strong labeling of sound events using crowdsourced weak labels and annotator competence estimation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:902–914, 2023.

[71] Guillaume Sartoretti, Yunfei Shi, William Paivine, Matthew Travers, and Howie Choset. Distributed learning for the decentralized control of articulated mobile robots. In *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3789–3794. IEEE, 2018.

[72] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[73] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In *Proc. of the 2021 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 844–852, 2021.

[74] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[75] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[76] Nikhil Barhate. Minimal pytorch implementation of proximal policy optimization. https://github.com/nikhilbarhate99/PPO-PyTorch, 2021.

[77] Glenn F Knoll. *Radiation detection and measurement*. John Wiley & Sons, 2010.

[78] Don Davis and Eugene Patronis. *Sound system engineering*. CRC Press, 2014.

[79] Rashid Ali, Yousaf Bin Zikria, Sahil Garg, Ali Kashif Bashir, Mohammad S Obaidat, and Hyung Seok Kim. A federated reinforcement learning framework for incumbent technologies in beyond 5g networks. *IEEE Network*, 35(4):152–159, 2021.

[80] Jorge Sola and Joaquin Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3):1464–1468, 1997.

[81] Philippe Proctor, Christof Teuscher, Adam Hecht, and Marek Osiński. Proximal policy optimization for radiation source search. *Journal of Nuclear Engineering*, 2 (4):368–397, 2021.

[82] Zhu Tianqing, Wei Zhou, Dayong Ye, Zishuo Cheng, and Jin Li. Resource allocation in iot edge computing via concurrent federated reinforcement learning. *IEEE Internet of Things Journal*, 9(2):1414–1426, 2021.

[83] Chetan Nadiger, Anil Kumar, and Sherine Abdelhak. Federated reinforcement learning for fast personalization. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 123–127. IEEE, 2019.

[84] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

[85] Hani Sami, Hadi Otrok, Jamal Bentahar, Azzam Mourad, and Ernesto Damiani. Reward shaping using convolutional neural network. *arXiv preprint arXiv:2210.16956*, 2022.

[86] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[87] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[88] Dinh C Nguyen, Ming Ding, Quoc-Viet Pham, Pubudu N Pathirana, Long Bao Le, Aruna Seneviratne, Jun Li, Dusit Niyato, and H Vincent Poor. Federated learning meets blockchain in edge computing: Opportunities and challenges. *IEEE Internet of Things Journal*, 8(16):12806–12825, 2021.

[89] Maha Kadadha, Shakti Singh, Rabeb Mizouni, and Hadi Otrok. A context-aware blockchain-based crowdsourcing framework: Open challenges and opportunities. *IEEE Access*, 2022.

[90] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[91] Ahmed Alagha, Shakti Singh, Hadi Otrok, and Rabeb Mizouni. Influence-and interest-based worker recruitment in crowdsourcing using online social networks. *IEEE Transactions on Network and Service Management*, 2022.

[92] Maha Kadadha, Hadi Otrok, Rabeb Mizouni, Shakti Singh, and Anis Ouali. On-chain behavior prediction machine learning model for blockchain-based crowdsourcing. *Future Generation Computer Systems*, 136:170–181, 2022.

[93] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[94] Yingjie Wang, Zhipeng Cai, Zhi-Hui Zhan, Yue-Jiao Gong, and Xiangrong Tong. An optimization and auction-based incentive mechanism to maximize social welfare for mobile crowdsourcing. *IEEE Transactions on Computational Social Systems*, 6 (3):414–429, 2019.

[95] Yingjie Wang, Yang Gao, Yingshu Li, and Xiangrong Tong. A worker-selection incentive mechanism for optimizing platform-centric mobile crowdsourcing systems. *Computer Networks*, 171:107144, 2020.

[96] Karl-Martin Ehrhart, Marion Ott, and Susanne Abele. Auction fever: Rising revenue in second-price auction formats. *Games and Economic Behavior*, 92:206–227, 2015.

[97] Elias Xidias, Paraskevi Zacharia, and Andreas Nearchou. Path planning and scheduling for a fleet of autonomous vehicles. *Robotica*, 34(10):2257–2273, 2016.

[98] Cane Punma. Autonomous vehicle fleet coordination with deep reinforcement learning. 2018.

[99] Shuo Jiang and Christopher Amato. Multi-agent reinforcement learning with directed exploration and selective memory reuse. In *Proceedings of the 36th annual ACM symposium on applied computing*, pages 777–784, 2021.

[100] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[101] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pages 1–49, 2022.

[102] Qizheng Wang, Wenping Ma, and Weiwei Wang. B-lnn: Inference-time linear model for secure neural network inference. *Information Sciences*, 638:118966, 2023.

[103] Zijie Pan, Jiajin Zeng, Riqiang Cheng, Hongyang Yan, and Jin Li. Pnas: A privacy preserving framework for neural architecture search services. *Information Sciences*, 573:370–381, 2021.

[104] Yu Dong, Liangxiao Jiang, and Chaoqun Li. Improving data and model quality in crowdsourcing using co-training-based noise correction. *Information Sciences*, 583: 174–188, 2022.

[105] Yao Zhang, Liangxiao Jiang, and Chaoqun Li. Instance redistribution-based label integration for crowdsourcing. *Information Sciences*, 674:120702, 2024.

[106] Nada Elsokkary, Hadi Otrok, Shakti Singh, Rabeb Mizouni, Hassan Barada, and Mohammed Omar. Crowdsourced last mile delivery: Collaborative workforce assignment. *Internet of Things*, 22:100692, 2023.

[107] Decui Liang, Wen Cao, Zeshui Xu, and Mingwei Wang. A novel approach of two-stage three-way co-opetition decision for crowdsourcing task allocation scheme. *Information Sciences*, 559:191–211, 2021.

[108] Hani Sami, Hadi Otrok, Jamal Bentahar, Azzam Mourad, and Ernesto Damiani. Reward shaping using convolutional neural network. *Information Sciences*, 648: 119481, 2023.

[109] Ahmed Alagha, Jamal Bentahar, Hadi Otrok, Shakti Singh, and Rabeb Mizouni. Blockchain-assisted demonstration cloning for multi-agent deep reinforcement learning. *IEEE Internet of Things Journal*, 2023.

[110] Hani Sami, Rabeb Mizouni, Hadi Otrok, Shakti Singh, Jamal Bentahar, and Azzam Mourad. Learnchain: Transparent and cooperative reinforcement learning on blockchain. *Future Generation Computer Systems*, 150:255–271, 2024.

[111] Yang Wang, Chenxi Zhao, and Shanshan Xu. Method for spatial crowdsourcing task assignment based on integrating of genetic algorithm and ant colony optimization. *IEEE Access*, 8:68311–68319, 2020.

[112] Osama Wehbi, Sarhad Arisdakessian, Omar Abdel Wahab, Hadi Otrok, Safa Otoum, Azzam Mourad, and Mohsen Guizani. Fedmint: Intelligent bilateral client selection in federated learning with newcomer iot devices. *IEEE Internet of Things Journal*, 2023.

[113] Mario Chahoud, Hani Sami, Azzam Mourad, Safa Otoum, Hadi Otrok, Jamal Bentahar, and Mohsen Guizani. On-demand-fl: A dynamic and efficient multi-criteria federated learning client deployment scheme. *IEEE Internet of Things Journal*, 2023.