UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

TECHNICAL REPORT

Multi-Agent Reinforcement Learning for Energy-Aware and Fault-Resilient Scheduling in Industrial IoT Manufacturing Systems

Author	Aloka Seneviratne
Supervisor	Sumudu Samarakoon

August 2025

Seneviratne P. (2025) Multi-Agent Reinforcement Learning for Energy-Aware and Fault-Resilient Scheduling in Industrial IoT Manufacturing Systems. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering 22 p.

ABSTRACT

This report investigates the application of multi-agent reinforcement learning (MARL) in industrial Internet of Things (IIoT) manufacturing systems, with a specific focus on energy efficiency and fault resilience. Traditional single-agent reinforcement learning models often fail to adapt when the operating environment changes, leading to reduced autonomy and performance. To address this limitation, this study explores a distributed approach where each machine or group of machines is managed by an individual agent. These agents communicate via fixed communication protocols and learn through decentralized or centralized training frameworks. Key parameters evaluated include the number of agents and their observability scope, model training strategies such as centralized training with decentralized execution (CTDE), the optimization of communication bitrates, and the development of scalable, resilient communication protocols. The goal is to establish a system capable of making real-time scheduling decisions under dynamic and constrained industrial environments. The findings and comparisons presented aim to guide the development of efficient, robust, and scalable MARL architectures for future industrial automation solutions.

Keywords: MARL, Multi-Agen Reinforcement Learning, Job Scheduling System

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

IIOT	Industrial Internet of Things
ML	Machine Learning
AI	Artificial Intelligence
RL	Reinforcement Learning
MARL	Multi Agent Reinforcement Learning
TMC	Temporal Message Control
CACOM	Context Aware Communication
Dec-POMDP	Decentralized Partially Observable Markov Decision Process

1 INTRODUCTION

1.1 Introduction to Industrial Automation

Manufacturing organizations rely on different types of machinery that work seamlessly with each other throughout the process of converting raw materials into industrial-grade products. In order for these machines to work seamlessly with each other as well as automate the process, communication needs to occur between various types of machines and systems. By this inter-communication between different components, the activities carried out by them, which are also known as jobs, can be planned, assigned, and monitored to ensure that the whole process works as efficiently as possible.

Initially, manufacturing plants connected systems using wires to ensure communication. However, with the development of wireless technology and IIOT, along with the impractical nature of connecting scalable systems using wires, the manufacturing industry started using wireless communication protocols to connect various machines and systems within the manufacturing process. This transition allowed manufacturing automation to scale exponentially while keeping its accompanying complexity relatively low. Although various types of topologies were used to connect systems, the common factor for all automated systems that were built at the time was the need for human intervention or supervision to ensure everything operated smoothly and as expected.

1.2 Integration of ML and AI with Industrial Automation

With the invention of Machine Learning (ML) and Artificial Intelligence (AI), and its growing popularity and adoption, it redefined what it meant to automate a system. Before the use of ML or AI, automation meant the configuration of systems to run based on a predefined protocol and would be subjected to any changes as required by humans. However, after adopting AI, the definition of automation transformed into training a model to make decisions to ensure the efficiency of a system is maintained by taking into account the situations that it is facing and thereby reducing or removing the influence and oversight that humans have in the operation of the manufacturing system. The development of AI and wireless communication technologies it led to the creation of AI agents capable of running entire operations smoothly and as intended with the help of the data inputs that it gets from all the machinery and the sensors that are connected to it.

1.3 Use of AI agents in Industrial Automation

For the purposes of controlling energy expenditure of manufacturing systems and controlling jobs to minimize production delays due to faults, the industry has leaned into single-agent reinforcement learning models that have a global view of the entire manufacturing process and follow a centralized communication topology for the transferring of data which includes the status of each machine, faults with any machine, dependencies on manufacturing process, usage level of machine resources, etc and the controlling of the system and manufacturing operation. Even though this single-agent model works well in trained environments, it begins to fall apart when the operating environment is changed, and therefore cannot maintain the system's autonomy and be retrained. To combat this issue, multi-agent reinforcement models can be introduced into these changing industrial environments, where each machine, or cluster of machines, can be handled by one agent and through the communication of these agents with one another based on fixed protocols, automation can be achieved in a manner capable of withstanding changes to the manufacturing process due to the autonomy of each component and protocols set in place. The communication language that is established among agents based on the protocols they need to follow is defined as protocol learning, which serves as the foundation for this report.

1.4 Problem Statement

Modern industrial manufacturing environments are increasingly distributed and dynamic, involving heterogeneous machines, unpredictable workloads, and strict efficiency requirements. Traditional centralized scheduling and control systems face limitations in scalability, adaptability, and robustness, particularly when reacting to local faults or changes in operational conditions. These limitations hinder the system's ability to make timely, energy-efficient, and fault-tolerant decisions.

This project investigates how decision-making can be decentralized using a multi-agent reinforcement learning (MARL) framework, where each agent represents an individual machine or a logical unit within the industrial system. The goal is to enable agents to make autonomous, context-aware decisions while cooperating through limited-bandwidth communication protocols. Key challenges include ensuring global coordination under partial observability, maintaining communication efficiency, minimizing energy consumption, and improving system-wide fault resilience.

By modelling these decision-making entities as reinforcement learning agents, the system aims to dynamically adapt to changing conditions, infer behaviour patterns, and reduce operational costs through learned scheduling policies. This study focuses on identifying the critical MARL parameters that impact performance in such settings and evaluating communication-aware strategies to improve overall system behaviour.

2 System Model Problem Formulation

2.1 Overview of System Model

The system model that is being developed is developed considering a distributed industrial environment which are made up of numerous machines and/or processing units. These machines are capable of carrying out tasks such as manufacturing, transporting and maintenance autonomously. These machines are equipped with sensors, actuators and communication capabilities and therefore could be considered as cogs in a cyber-physical system. Therefore, these machines are to act independently instead of relying on a central controller, with the end goal being the collaboration between these machines to achieve the overall system objectives.

2.2 Decision Makers of the System and their roles

The most notable decision-makers of this system are the individual machines or the cluster of machines which act as autonomous agents. They are responsible for deciding when to execute, delay or skip tasks based on their current tasks or workload. They are also responsible for communicating critical system information with nearby agents. Finally, they are also responsible for adapting to varying environmental and manufacturing conditions.

2.3 Importance of Communication

This communication between the agents is important because it helps the system as a whole meet its objectives by helping agents avoid bottlenecks in the process, share local information with nearby agents and help agents to cooperate and balance the workload when facing real-time constraints.

2.4 Main problems that need to be addressed

In creating such a system, there are several problems that need to be addressed and subsequently resolved.

- Decentralize the scheduling process among the agents that are independent but cooperative.
- Maintain overall system stability even while local faults and breakdowns may occur.
- Minimize energy consumption
- Minimize delays, idle time and thereby the operational cost of the system
- Ensure reliable communication is established between the agents even though the communication protocol may be limited
- Allow agents to identify slow-occurring changes in the machinery, such as depreciation and take them into account.

2.5 Role of MARL in solving problems

To address the above objectives, we formulate the system as a Multi-Agent Reinforcement Learning (MARL) environment:

- **Agents:** Each decision-making unit (machine or subsystem).
- **States:** Local observations from sensors, including workload status, machine health, and environmental inputs.
- **Actions:** Decisions such as task execution, message transmission, and scheduling adjustments.
- **Environment Dynamics:** The system evolves based on task completions, energy use, and external disturbances.
- **Rewards:** Defined to encourage energy efficiency, minimal latency, fault resilience, and communication economy.

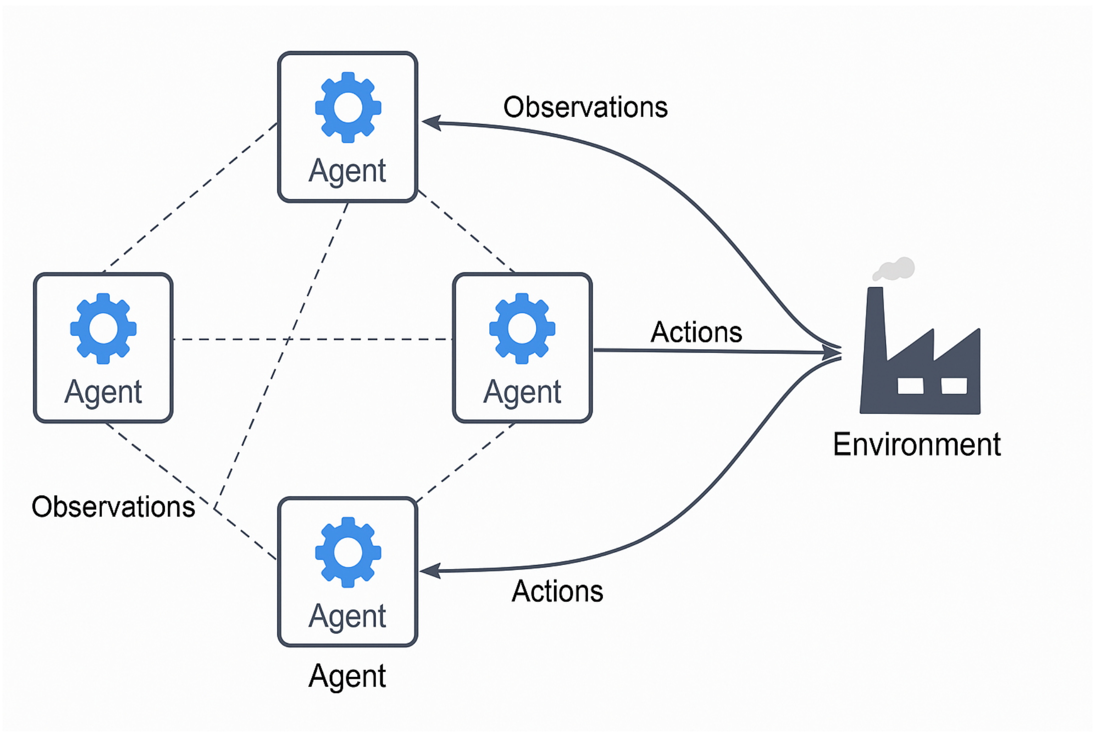

Figure 1. Conceptual system model illustrating agent interactions and communication flow.

3 Parameters of the Multi-Agent RL Model

For this report, the following criteria were studied and based on the current research done in this space, the most optimal parameters were subsequently selected.

3.1 Number of Agents and scope of observability

When implementing a multi-agent RL model, the first hurdle that needs to be addressed is how many agents are needed to handle the complexities of the system. This includes defining the limitations of each agent's observation of the environment, which allows it to develop its model based only on the information that it has received.

3.2 Model Training Strategy

In order to build an AI agent, they need to be trained based on existing or simulated datasets. While this is relatively straightforward for single-agent based RL models, there are different approaches and strategies when it comes to multi-agent based RL models. These approaches will be compared to choose the most effective strategy

3.3 Number of bits in data transfer

Every agent in a multi-agent RL system needs to communicate with each other constantly and rely on vital information from the other agents to make decisions. This information is transferred via bits. It is important to choose how many bits would be transferred in each message so that all the relevant information is passed through, but at the same time, ensuring that excess bits aren't used, which can slow down the rate of communication.

3.4 Communication Protocol

The connection protocol is one of the most important aspects, as it is what allows agents to communicate with each other, as well as allows for the scalability of the system. There are currently different short-range wireless protocols used in autonomous systems, which would also be compared to ensure the most efficient data transfer with minimal data loss levels.

4 Literature Review

4.1 Number of Agents and Observability

Industrial scheduling problems naturally involve many agents (machines, factories, robots) that each observe only a portion of the global state. For example, smart factories are highly distributed, and single-agent approaches struggle with parallel decision-making [1]. A common formalism is the Decentralized POMDP: each agent receives a local observation and takes actions to optimize a shared objective [2, 3]. In this setting, agents must operate independently on partial information while coordinating with others to maximize cumulative reward [2].

Such partial observability and large agent counts introduce tractability challenges (e.g., non-stationarity, exponential joint action spaces). Recent works exploit problem structure to scale MARL. For instance, [4] shows that in many homogenous populations containing domains, only the *counts* of the agents taking each action matter due to the implementation of the count vector(action anonymity), allowing planning complexity to drop from exponential to polynomial in agent number [2]. Other approaches explicitly adjust the number of agents during training: [5] treat agent count as a curriculum variable, gradually increasing team size by tracking learning progress [5].

In practice, the state representation is often carefully designed. For example, [6] build a compact DRL environment for job-shop scheduling by encoding job and machine status into a small state vector, with a dense reward approximating makespan minimization [6].

4.2 Model Training Strategies

A key design choice in MARL is how agents are trained: fully centralized, fully independent, or hybrid. Centralized training treats the joint system as one MDP, allowing a single agent (or critic) to access global state and coordinate all decisions. Fully decentralized training (each agent learns independently) scales easily but suffers from non-stationarity. The centralized training with decentralized execution (CTDE) paradigm balances these extremes [2, 3].

Under CTDE, agents are trained using shared information (e.g., a joint critic or mixing network) but execute using only local observations. For example, in QMIX each agent has its own Q-network but a mixing network enforces monotonicity to produce a global Q-value [2]. [7] incorporates CTDE in AGV scheduling, providing both local and global rewards during training [2].

Curriculum learning in MARL can adapt agent count or environment complexity over time. [5] automatically adjust the number of agents from an easier initial distribution to the full-scale scenario, using TD-error as a learning progress signal [5].

Underlying these strategies are the learning algorithms themselves. Policy-gradient methods (actor–critic, REINFORCE) remain fundamental. [8] introduced the REINFORCE family, proving that the expected weight update is aligned with the gradient of the expected reward [8]. These ideas underpin MADDPG and similar modern CTDE approaches.

4.3 Communication Efficiency

Most industrial tasks require explicit information exchange to compensate for partial observability. However, real channels have bandwidth limits. Recent work addresses this by jointly learning what, when, and how to communicate.

Concretely, recent work demonstrates why protocol design must be driven by efficiency constraints. [9] shows that a bandwidth cap can be formalised as an information bottleneck and that efficient protocols and schedulers should produce low-entropy, task-relevant messages rather than full broadcasts. [10] proposes a two-stage, context-aware protocol (coarse broadcast followed by personalised follow-ups) and applies learned step-size quantisation so messages are discrete and suitable for low-bit channels [10]. [11]

Temporal Message Control (TMC) demonstrates that temporal smoothing and reuse of recent messages can drastically reduce transmitted information while improving robustness to packet loss [**?**]. Finally, [4] shows that when system dynamics depend on action counts (action anonymity), low-bit aggregated summaries or counts can be sufficient message content, further motivating compact encodings [4].

4.4 Communication Methods, Topologies and the Need for Robustness

In this context, network topology refers specifically to the *communication structure among agents*; therefore, it describes how agents are connected for exchanging messages. Different topologies determine which agents can directly share information, how quickly knowledge spreads across the team, and how scalable the system remains as the number of agents grows.

[12] introduces ExpoComm, which employs an exponential communication graph to accelerate information dissemination across all agents. Sparse but well-structured topologies, especially when combined with memory processors, enable scalability and even zero-shot transfer across different team sizes. Other approaches explore attention-based message routing, where each agent dynamically selects which peers to communicate with, and graph neural networks that learn structured communication patterns optimised for multi-agent coordination.

Robustness is critical for real-world systems because communication channels are subject to noise, packet drops, and latency. Here, robustness refers to the general capacity of a communication method to handle such imperfections, while resilience captures the specific ability to maintain functionality under disruptions. [13] demonstrates that Temporal Message Control (TMC) improves resilience to packet loss by reusing prior messages and smoothing updates over time. Likewise, [10] shows that quantised, discrete protocols are less sensitive to channel noise than continuous signals.

In summary, communication methods must not only define effective topologies for agent interaction but also ensure robustness and resilience so that multi-agent systems can operate reliably under practical constraints.

5 Experimental Application

In this chapter, we will investigate an application of the MARL framework with an experimental problem setting. This experiment investigates how MARL frameworks address decision-making tasks under structured settings. By constraining agents with predefined rules, we can observe how their individual actions align through the learning process, leading to coordinated strategies that approximate or achieve the optimal system outcome. The results provide insight into the mechanisms through which rule-guided agent interactions shape overall performance.

5.1 Problem Statement

In this scenario, the problem that we are attempting to solve is the classic hat riddle, and it can be defined as follows.

"An executioner lines up 100 prisoners in a single file and puts a red or a blue hat on each prisoner's head. Every prisoner can see the hats of the people in front of him in the line - but not his own hat, nor those of anyone behind him. The executioner starts at the end (back) and asks the last prisoner the colour of his hat. He must answer "red" or "blue." If he answers correctly, he is allowed to live. If he gives the wrong answer, he is killed instantly and silently. So that everyone hears the answer, yet no one knows whether the answer was right or wrong. On the night before the line-up, the prisoners confer on a strategy to help them. What should they do?"

5.2 MARL Parameters

As per the definitions in the earlier chapters, the parameters of the MARL model were defined as follows.

- **Agents:** Each prisoner was considered to be an agent. Therefore, for a prison with 100 prisoners, there would be 100 agents.
- **States:** Each prisoner can see the hat colours of prisoners only in front of them, and none of the hat colours of the prisoners behind them. Each prisoner also knows their place in the lineup, and can hear the responses of the prisoners before them. In this problem, a *training step* refers to a single interaction between the learning algorithm and the environment, where one agent observes its state (visible hats and past announcements), selects an action, and receives a reward signal. The collection of such steps across agents and episodes constitutes the learning process.
- **Actions:** Each prisoner can choose one of the 2 potential hat colours.
- **Environment Dynamics:** Hat colours don't evolve based on the responses(static). Only belief updates occur through sequential announcements. Therefore, it's not a standard Markov Decision Process with changing states, but rather a sequential information transfer.
- **Rewards:** In this problem, the true hat configuration does not provide a natural notion of reward. Therefore, the reward function must be defined as a modelling choice to guide the learning process. We consider the following formulations:

(1) Individual Reward.

Each agent i receives a binary reward depending on whether its guess matches its actual hat color:

$$r_i = \begin{cases} 1, & \text{if } a_i = h_i, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $a_i \in \{0, 1\}$ is the action (announced color) and $h_i \in \{0, 1\}$ is the true hat color.

(2) Team Reward.

Alternatively, we can define a cooperative reward shared among all agents, given by the total number of correct guesses:

$$R = \sum_{i=1}^{N} r_i, \tag{2}$$

This encourages strategies that maximize the group survival rate.

In practice, we adopt the *team reward* formulation, as it directly reflects the cooperative nature of the 100-prisoner hats puzzle. The individual reward, while intuitive, can lead to misaligned incentives where some agents optimize only their own survival without contributing useful information to others. Reward shaping can be useful in early training to improve stability, but it introduces bias and may encourage agents to overfit to auxiliary signals. Thus, using the team reward strikes a balance between clarity, simplicity, and alignment with the overall group objective.

5.3 MARL algorithms considered

To stabilise training, we employ a curriculum learning strategy where the number of prisoners N is gradually increased. Formally, let

$$C = \{5, 7, 10, 15, 20, 50, 100\}$$

denote the curriculum schedule. At stage k, the environment size is set to $N = C_k$, and training proceeds for a fixed number of episodes before moving to the next stage.

An *episode* is defined as one complete round of the hat-guessing game, where all N prisoners sequentially decide whether to guess or pass, and the episode terminates once every prisoner has acted. The cumulative reward is then computed based on the number of survivors in that round.

For training, a curriculum learning schedule is adopted, where the number of prisoners N is gradually increased. At each curriculum stage, the model is trained for a fixed number of episodes before advancing to the next stage. In this scenario, the fixed number of episodes is 10,000.

To solve this riddle, the following algorithms were considered and tested on the same riddle, but with just 5 prisoners instead of 100 to reduce the complexity and processing time of the model. Therefore, the number of agents as per the earlier definition is N = 5 (Only 5 agent curriculum will be trained), while the states, actions, environment dynamics and rewards remain the same.

The advantages and disadvantages of using each method were illustrated in the table given below, along with the prisoner success rate for each algorithm. Here, the success rate refers to, on average, what percentage of prisoners survive at the end of all the episodes.

- **PPO (Proximal Policy Optimization)**: This method learns by making small, careful updates to the strategy. It is very stable and easy to use, but it needs a lot of training data to work well.
- **DDRQN (Deep Distributed Recurrent Q-Network)**: This method gives each agent a memory, so they can remember what was said before. All agents share one common brain, which helps them learn to cooperate faster. It is powerful for this puzzle but sometimes unstable if strategies change quickly.
- **MADDPG (Multi-Agent Deep Deterministic Policy Gradient)**: This method lets agents train with full information during learning, but then act only with their local view. It is good for teamwork but was originally made for continuous actions, so it is more complicated to apply here.

5.3.1 Comparison Table

Table 1. Simplified comparison of PPO, DDRQN, and MADDPG.

Algorithm	**Strengths**	**Weaknesses**
PPO	Stable learning, easy to use, makes safe updates.	Needs a lot of training data, slower to improve.
DDRQN	Agents can "remember" past hints, share one brain, strong at cooperation.	Can become unstable if strategies keep changing, only works for simple (yes/no) choices.
MADDPG	Uses full information during training, good for teamwork coordination.	More complex to set up, originally made for continuous choices, less stable in this puzzle.

By comparing the advantages and disadvantages, it is clear that the DDRQN algorithm works best with this riddle as the strategy remains constant. Therefore, this algorithm was used to train for the problem with 100 prisoners.

5.4 Hyperparameters in DDRQN

In training the Deep Distributed Recurrent Q-Network (DDRQN), three key hyperparameters play an essential role in determining the stability and efficiency of learning: the discount factor γ, the learning rate α, and the soft target update rate τ.

Discount Factor (γ).

The discount factor determines how much weight is given to future rewards compared to immediate ones. The return G_t at time t is defined as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots . \tag{3}$$

A small γ (close to 0) makes the agent short-sighted, focusing on immediate survival only. A larger γ (close to 1) encourages long-term cooperation, where early prisoners may sacrifice information to benefit those later in the line. In our experiments, we set $\gamma = 0.95$, which balances present and future rewards.

Learning Rate (α).

The learning rate controls the step size used in updating network parameters. At each gradient descent step, the parameters θ are updated as

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta), \tag{4}$$

where $L(\theta)$ is the loss function. If α is too large, updates become unstable and oscillatory; if too small, training converges very slowly. We adopt $\alpha = 0.001$, a common choice that ensures both convergence and stability.

Soft Target Update Rate (τ).

To stabilize Q-learning, DDRQN maintains a separate *target network* with parameters θ^-. Instead of copying weights directly, a soft update is applied:

$$\theta^- \leftarrow (1 - \tau)\, \theta^- + \tau\, \theta. \tag{5}$$

Here τ controls how quickly the target network follows the main Q-network. A very high τ leads to instability, while a very low τ slows learning. In our setup, $\tau = 0.01$ provides a good balance by ensuring smooth and stable target updates.

Exploration Policy: ε-greedy.

To balance exploration and exploitation, we adopt the ε-greedy strategy. At each step, the agent selects the action that maximizes its current Q-value with probability $1 - \varepsilon$, and chooses a random action with probability ε. Formally,

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|}, & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|\mathcal{A}|}, & \text{otherwise,} \end{cases}$$

where $\mathcal{A}$ is the action set $\{\text{red}, \text{blue}\}$. We use a decaying schedule, defined as $\varepsilon = 1 - 0.5^{1/n}$, where n denotes the training step. This ensures that the agents explore extensively at the start, and gradually shift towards exploitation as training progresses.

5.5 Problem and Solution Model

5.5.1 Problem Setup

We consider the sequential hats puzzle with $N \in \mathbb{N}$ prisoners (agents), indexed $i \in \{0, 1, \dots, N-1\}$, where $i = 0$ is at the front of the line and $i = N - 1$ is at the back. Each agent i is assigned a binary hat color $h_i \in \{0, 1\}$, and let $h = (h_0, \dots, h_{N-1}) \in \{0, 1\}^N$ denote the hat vector. The hats are drawn independently and uniformly at random:

$$h_i \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}\left(\tfrac{1}{2}\right).$$

Agents act from back to front. At time $t \in \{0, \dots, N-1\}$ the acting agent is

$$i_t \triangleq N - 1 - t,$$

who must announce $a_{i_t,t} \in \{0, 1\}$ (its guess for its own hat). All announcements are audible to every agent, but each agent only sees the hats of those in front.

5.5.2 Information Structure and Beliefs

At time t, the observation available to agent i_t is

$$o_{i_t,t} = \left(h_{<i_t},\ a_{>i_t}\right),$$

where $h_{<i} = (h_0, \dots, h_{i-1})$ are the hats in front of agent i, and $a_{>i} = (a_{i+1}, \dots, a_{N-1})$ are the previous announcements. The environment state h is static; what evolves are agents' *beliefs* about their own hats, updated upon hearing new announcements.

5.5.3 Reward Model

The reward at time t for the acting agent i_t is

$$r_{i_t}(s_t, a_{i_t,t}) = 1\{a_{i_t,t} = h_{i_t}\}.$$

This sparse reward directly encodes the survival objective: 1 if the guess is correct, 0 otherwise. The total episodic return is

$$G(\tau) = \sum_{i=0}^{N-1} 1\{a_i = h_i\}.$$

5.5.4 Optimization Objective

The learning goal is to maximize the expected number of correct announcements:

$$\max_{\pi} J(\pi) = \mathbb{E}_{h,\tau}\left[\sum_{t=0}^{N-1} \gamma^t\, r_{i_t}(s_t, a_{i_t,t})\right],$$

with discount factor $\gamma = 0.95$. For evaluation, we use the *success rate*

$$\text{SuccessRate}(N) = \frac{1}{N}\,\mathbb{E}[G(\tau)],$$

which is the fraction of correct guesses per episode.

5.5.5 DDRQN Solution

To solve this Decentralised Partially Observable Markov Decision Process(Dec-POMDP), we employ a **Deep Distributed Recurrent Q-Network (DDRQN)**:

- Each agent shares parameters of a recurrent Q-network, which maps observation histories to Q-values.
- Training is off-policy with experience replay and double-Q learning to reduce overestimation bias.
- An ε-greedy exploration policy is used, with

  $$\varepsilon = 1 - 0.5^{\frac{1}{N}},$$

  where N is the number of agents in the current curriculum stage.
- Learning rate is $\alpha = 0.001$, and target update rate $\alpha^{-} = 0.01$.

The recurrent structure allows DDRQN to track information over time and update beliefs based on past announcements. By maximising $J(\pi)$ under this setup, DDRQN learns communication strategies that approach the performance of the optimal parity-based protocol.

5.6 Results and Discussion

When tested at $N = 5$, PPO achieved 66% accuracy, DDRQN 93%, and MADDPG 51%. These results are discussed in detail below.

The model was trained on the ascending curriculum to 5, 7, 10, 15, 20, 50 and 100. This allowed the algorithm to solve the riddle to begin with fewer agents and solve for it more easily due to a lower level of complexity, learn from that step and implement it on the following step and thereby work its way up to the level expected in the problem statement. Each step in the curriculum consisted of 10,000 episodes. The final success rates for each step of the curriculum are as follows.

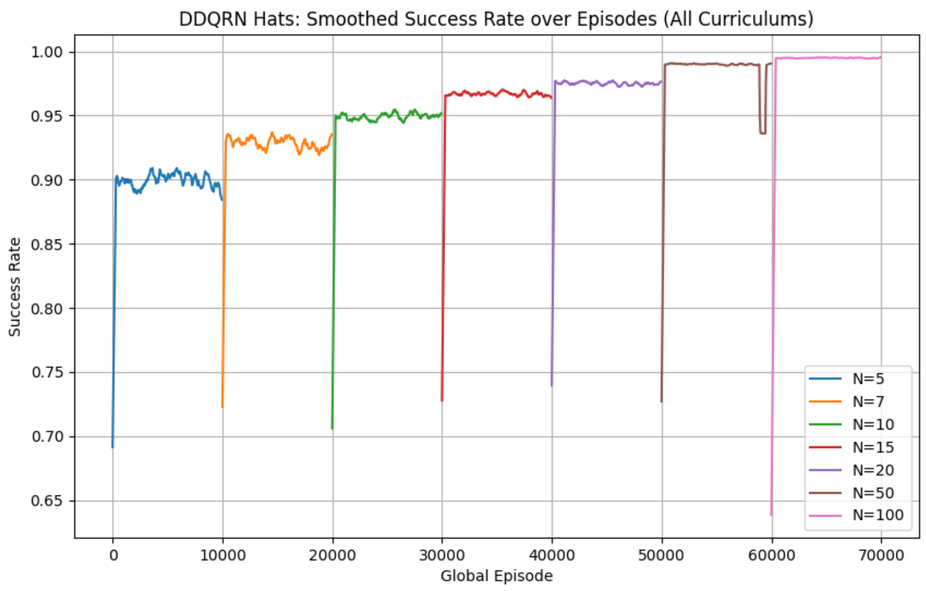

Figure 2. Success rates at each step of the curriculum.

It can be seen that as the number of prisoners increased, so did the accuracy of the experiment, with the success rate at 100 prisoners being **99.5**%.

5.7 Conclusion

5.7.1 Policy and Optimality

In this problem, the **policy** $\pi(a_t \mid o_t; \theta)$ represents the rule that determines each prisoner's guess based on what they see (hats in front) and hear (past announcements). In DDRQN, this policy is implemented with a recurrent network, which allows prisoners to "remember" previous information when making their decisions.

5.7.2 Sensitivity to Parameters

The performance of reinforcement learning algorithms depends strongly on hyperparameters:

- **Discount factor** (γ): Controls how much agents value future rewards. A value of 0.95 balances immediate and long-term coordination.
- **Learning rate** (α): Determines how fast the model updates. If too high, learning becomes unstable; if too low, progress is very slow.
- **Exploration** (ϵ**-greedy**): Ensures agents try different strategies. Too fast decay causes premature convergence; too slow decay prevents stability.

5.7.3 Why DDRQN is Optimal

DDRQN was found to be the most suitable algorithm for this puzzle because:

1. It naturally handles **partial observability** through its recurrent memory.
2. It models the **sequential nature** of the puzzle, where each agent acts in turn.
3. It fits the **binary action space** of the problem (red/blue).
4. Sharing a common network across agents supports effective **coordination**.
5. Experimentally, DDRQN achieved the highest success rate (93%), outperforming PPO and MADDPG.

With the results of this experiment, we can summarise that MARL systems can be used to solve complex problems with independent agents that have limited observability and fixed parameters. Real-life problems in manufacturing organisations can also be addressed using similar methods to increase efficiency, reduce energy emissions and for early fault detection

6 REFERENCES

[1] F. Bahrpeyma and D. Reichelt, “A review of the applications of multi-agent reinforcement learning in smart factories,” *Frontiers in Robotics and AI*, vol. 9, 2022.

[2] W. Xu, J. Li, and W. Yu, “Multi-agent reinforcement learning for flexible shop scheduling problem: A survey,” *Frontiers in Industrial Engineering*, vol. 31, 2025.

[3] X. Sun, W. Shen, J. Fan, B. Vogel-Heuser, F. Bi, and C. Zhang, “Deep Reinforcement Learning-based Multi-Objective Scheduling for Distributed Heterogeneous Hybrid Flow Shops with Blocking Constraints,” *Engineering*, vol. 46, pp. 278–291, 2025.

[4] K. He, P. Doshi, and B. Banerjee, “Reinforcement learning in many-agent settings under partial observability,” in *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 780–789, 2022.

[5] W. Zhao, Z. Li, and J. Pajarinen, “Learning Progress Driven Multi-Agent Curriculum,” in *Proc. 42nd International Conference on Machine Learning (ICML)*, 2025.

[6] P. Tassel, M. Gebser, and K. Schekotihin, “A Reinforcement Learning Environment for Job-Shop Scheduling,” *arXiv preprint arXiv:2104.03760*, 2020.

[7] H.-B. Choi, J.-B. Kim, Y.-H. Han, S.-W. Oh, and K.-H. Kim, “Marl-based cooperative multi-agv control in warehouse systems,” *IEEE Access*, vol. 10, pp. 8768–8781, 2022.

[8] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[9] R. Wang, X. He, R. Yu, W. Qiu, B. An, and Z. Rabinovich, “Learning Efficient Multi-agent Communication: An Information Bottleneck Approach,” in *Proc. 37th International Conference on Machine Learning (ICML)*, pp. 9460–9471, 2020.

[10] X. Li and J. Zhang, “Context-aware Communication for Multi-agent Reinforcement Learning,” in *Proc. 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2024.

[11] S. Q. Zhang, Q. Zhang, and J. Lin, “Efficient communication in multi-agent reinforcement learning via variance based control,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[12] X. Li, X. Wang, C. Bai, and J. Zhang, “Exponential Topology-enabled Scalable Communication in Multi-agent Reinforcement Learning,” in *International Conference on Learning Representations (ICLR) 2025*, 2025.

[13] S. Q. Zhang, J. Lin, and Q. Zhang, “Succinct and Robust Multi-Agent Communication With Temporal Message Control,” in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.