A Case for Decentralized Model-Based Multi-Agent Reinforcement Learning

Rafał Niedziółka-Domański[0009−0004−0405−5508]

Maria Curie-Sklodowska University, 5 M. Curie-Skłodowskiej Square, Poland
rafal.niedziolka-domanski@mail.umcs.pl

Abstract. Decentralized Training and Execution (DTE) is an appealing paradigm in multi-agent reinforcement learning due to its scalability and autonomy, yet it suffers from severe *non-stationarity* arising from simultaneously learning agents. This work shows how a model-based approach can mitigate this issue without relying on centralized training or access to global information and achieve performance comparable to the state-of-the-art method from the currently dominant Centralized Training for Decentralized Execution (CTDE) paradigm.
For this, a model-based algorithm is proposed, where each agent independently learns an internal model of the environment dynamics from its own experience and integrates this model into the decision-making process. By acting according to its internal model rather than directly reacting to the evolving environment, an agent's policy becomes more stable, reducing the impact of *non-stationarity* created by other learning agents. The proposed approach is evaluated and compared with independent Deep Q-Networks and MADDPG, a CTDE algorithm. Experimental results demonstrate that the model-based method achieves performance comparable to that of MADDPG, despite operating in a fully decentralized manner. These results indicate that model-based, decentralized approach can serve as an effective alternative to centralized training for cooperative multi-agent reinforcement learning.

Keywords: Reinforcement learning · Multi-Agent · Model-Based.

1 Introduction

Multi-Agent Reinforcement Learning has shown potential to solve complex problems involving multiple autonomous agents. However, training agents in fully decentralized manner remains challenging due to the inherent non-stationarity of the environment: as each agent learns and updates its policy, the environment effectively changes from the perspective of the others. This makes it difficult for traditional reinforcement learning algorithms to converge, when agents cannot rely on global information or inter-agent communication.

The goal of this paper is to explore whether a model-based approach can address these challenges by enabling each agent to learn an internal model of the environment and use it in its decision-making process. By reasoning about the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

consequences of actions through its own internal environment model rather than reacting solely to the evolving environment, an agent could stabilize its policy and achieve performance comparable to methods that rely on centralized training. A fully decentralized method that mitigates the effects of *non-stationarity* could be useful in scenarios where centralized or global information is unavailable even for the learning phase (e.g., when training real-world robots without any communication channels, relying only on their sensors).

1.1 The Problem Definition

Reinforcement Learning (RL) stems from the idea of learning through interaction with the environment. Good actions should be rewarded and bad ones penalized, thus reinforcing the behavior leading to the most positive outcomes in the long run.

Markov Decision Processes From the computational point of view, the problem for RL algorithms can be defined as solving a Markov Decision Process (MDP) [10], where *the agent* makes sequential decisions in some *environment* to achieve some goal. (Such a definition is at least sufficient for single-agent Reinforcement Learning.) The goal is specified in some way by the *reward function* of the MDP and the agent learns to maximize expected *return* — discounted sum of the gained rewards (positive and/or negative scalars). The agent learns a *policy* $\pi(\mathbf{s})$ that selects *action* based on the current *state* $\mathbf{s}$ in a way that maximizes the expected return given that state and action. The agent's objective therefore, is to find policy π that maximizes expected discounted sum of rewards [10]:

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R_{a_t}(s_t, s_{t+1}) \,\middle|\, a_t = \pi(s_t),\ s_{t+1} \sim P_{a_t}(s_t, s_{t+1})\right] \tag{1}$$

where $a_t = \pi(s_t)$ means that the action a_t in time step t is chosen from the policy π for state s_t, and the next state, s_{t+1}, is sampled from the probability transition function of the MDP for the chosen action a_t (P_{a_t}), given state s_t. γ is the discount factor for future time steps — the more steps into the future, the less important the reward is. The expected sum of rewards over an infinite horizon is calculated using the reward function $R_{a_t}(s_t, s_{t+1})$, which specifies the scalar reward for transitioning from state s_t to state s_{t+1} when performing the action a_t.

Multi-Agent Reinforcement Learning Problem In Multi-Agent Reinforcement Learning (MARL), the underlying problem is more complex to define. The probability transition function is not only dependent on the action of the given agent, but also on the choices of all other agents. Also, the objective of the agent can change depending on what we want to achieve. Using game theory studies, we can distinguish three basic scenarios of interaction between agents [1]. In *cooperative* scenario, we want agents to work together to maximize their total

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

returns, so each agent needs to consider other agents behavior and adjust accordingly. In *competitive* scenarios, each agent wants to maximize it's own returns against other agents' policies. There are also *mixed competitive and cooperative* scenarios. Most competitive cases can be solved efficiently using standard RL algorithms in multi-agent setup, where each agent just simply tries to maximize it's own returns, which in turn forces other agents to improve as well — leading to improved performance of all agents over time. Thus, we want to focus mainly on the cooperative case.

For the purposes of this paper, we will define the MARL problem as a Stochastic Game (Markov Game) [1, 7]. For each agent there is a separate set of actions and separate reward function. State transition probability function takes into consideration not just one action, but actions of all agents. Similar to MDP, Stochastic Game has the Markov property, where the next state and reward depend only on the current state and joint action, and are conditionally independent of all previous states and joint actions.

1.2 Common Approaches and Challenges

How does one adapt reinforcement learning to work in multi-agent environments? There is probably no single perfect solution for all cases, but current MARL methods can be broadly divided into three approaches.

Centralized Training and Execution Centralized Training and Execution (CTE) effectively reduces a multi-agent problem into a single-agent formulation [1]. There is one policy that chooses actions for all agents at once. It requires as input some representation of a global state. If it is not directly available, the easy way to represent a global state is simply to concatenate observations of all agents. Effectively, we have one controlling agent with huge action space (as it needs to output action for all agents) to control multiple entities. As one can quickly observe, the main drawback of the CTE approach is poor scalability, as the action space (and possibly the observation space) grows exponentially with the number of agents. It would also require communication between the controlling agent and the controlled entities during execution. On the positive side, the availability of global information about the environment mitigates the problem of *non-stationarity* [13] from the agent's perspective (meaning that, the environment seems to change over time, when in reality only the policies of other agents change).

Decentralized Training and Execution Using standard RL methods in multi-agent environment leads to Decentralized Training and Execution (DTE) approach [1]. Each agent trains and executes using only information available to that agent. This makes it fully decentralized and easily scalable. In this case, other agents are effectively part of the environment. However, this also creates the main challenge of the approach: their policies change during training, rendering the environment *non-stationary* from the agent's perspective. A simple

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

example of this approach is independent Q-learning (IQL), in which each agent applies standard single-agent Q-learning [12] (or Deep Q-Learning, DQN [4]). In recent years, at least one method has been proposed to address non-stationarity in IQL: *multi-agent alternate Q-learning* (MA2QL) [8], where agents take turns updating their policies via Q-learning. Learning one agent at a time is a simple solution, but certainly not perfect, especially if we want agents to be truly decentralized and independent.

The goal of this article is to address the problem of non-stationarity in DTE by modeling the environment from the perspective of the agent and using this model in its decision making process. Since the agent will act according to its internal environment model, changes in the environment (other agents' policy updates) should not immediately impact its policy; the agent must first adjust its internal model to account for the change.

Centralized Training for Decentralized Execution The currently dominant paradigm in multi-agent reinforcement learning (MARL) is Centralized Training for Decentralized Execution (CTDE) [1], as it combines advantages of centralized and decentralized approaches. Algorithms such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [3] require a separate training phase in which any available information can be used. This may include the global state of the environment, observations from other agents, or even their policies. Contrary to CTE, CTDE methods do not require communication during execution. They execute in decentralized manner, using only observations available to a given agent during execution, similarly to DTE. Although CTDE methods scale better than fully centralized approaches, they can still suffer from scalability issues — particularly when global state representations are formed by concatenating all agents' observations. Moreover, access to other agents' actions or policies during training might not be feasible in some use cases.

2 Model-Based Approach

This paper explores whether using an internal environment model in decision making of an agent can mitigate non-stationarity effects of the Decentralized Training and Execution approach and help it achieve performance comparable to CTDE methods.

2.1 Model-Based Reinforcement Learning

What makes a reinforcement learning algorithm model-based? An algorithm is model-based if it explicitly learns or is given a model of the environment and then uses that model to plan [6]. The model approximates transition dynamics and reward function of Markov Decision Process (or Markov Game in multi-agent scenarios).

The environment model can be used two ways [6]: a) to simulate possible trajectories ("mental rollouts") for additional experience to train policy or value

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

function without making decisions in real environment, e.g., *Dyna* algorithm [9]; b) to plan ahead for selecting the action.

2.2 Integration of Environment Model in Proposed Algorithm

Starting with the standard DQN algorithm [4], we want to modify it and integrate an internal environment model that each agent will learn during training, alongside with it's state value function. We want agents to make decisions not strictly on the basis of the current state but also based on what they believe will happen when they choose each action according to their internal model. Introduction of the environment model in decision making should stabilize policy of the agent and limit the effects of non-stationarity, as it does not instantly reflect changes in the environment (i.e. changes in other agents policies).

For each state, the agent predicts the next states and rewards one step ahead (we possibly could predict more than one step ahead, but model inaccuracy will have a compounding effect on predictions) and uses its state-value function, $V^{\pi}(s)$, to approximate the future expected return for each next state. When the predicted value for each action is calculated, the agent chooses the action using the ϵ-greedy policy (the policy that picks the best action most of the time and random action with probability given by small ϵ value [10]), just like in DQN (and standard Q-Learning). The state-value function for the current policy is trained using real experience gathered by the policy, which itself depends on the current environment model. In contrast to the action-value function in DQN, we cannot use *off-policy* experience (old, outdated, or generated by a different policy [10]) to train the state-value function, as it can only be calculated for the current policy π.

The environment model is trained on the agent's own experiences (from replay buffer [5]), without access to any additional external or global information, and represents the agent's belief about how the environment behaves from its perspective. The model can be trained asynchronously, provided that it eventually converges to accurately reflect the dynamics of the real environment (based also on the current policies of other agents). The agent is capable of making decisions using this imperfect model—which is necessary during early learning—and optimizes its policy with respect to the current model. As the model progressively converges to the real environment, the learned policy is expected to converge to the optimal policy for the real environment as well. The above process is also visualized in the figure 1.

State-value function. The state-value function is approximated using a simple feed-forward neural network with 3 hidden layers that use the ReLU activation function. It takes as input the current state and outputs one scalar — representing how valuable being in that state is (meaning, the expected future return starting from that state and following the policy π, as described in equation 1).

Environment Model. The environment model is represented as a small feed-forward neural network with 3 hidden layers that use ReLU activation function.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

Algorithm 1 Model-Based RL Algorithm

(Algorithm controls agent i)

1: Initialize predictive model $\hat{\mathcal{T}}_i$ and state-value function V^{π_i}
2: For every episode:
3: **for** $t = 0, 1, 2, \ldots$ **do**
4: Observe current state s^t
5: **if** explore with probability ε **then**
6: Choose random action a_i^t
7: **else**
8: Using model $\hat{\mathcal{T}}_i$, obtain predicted next states $\hat{s}_j^{t+1}$ and predicted rewards $\hat{r}_{ij}^t$ for each action j given the current state s^t
9: Calculate expected returns for each action j in current state $ER(a_{ij}, s_t) \leftarrow \hat{r}_{ij}^t + \gamma V^{\pi_i}(\hat{s}_j^{t+1})$
10: Choose action $a_i^t = arg\max_{a_{ij}} ER(a_{ij}, s_t)$
11: **end if**
12: (meanwhile, other agents $j \neq i$ choose their actions a_j^t)
13: Observe real reward r_i^t and next state s^{t+1}
14: Add experience $(s_t, a_i^t, r_i^t, s_{t+1})$ to the agent's Replay Buffer
15: Update state-value function using the gathered experience $V^{\pi_i}(s^t) \leftarrow V^{\pi_i}(s^t) + \alpha[r_i^t + \gamma V^{\pi_i}(s^{t+1}) - V^{\pi_i}(s^t)]$
16: Sample a batch of experiences B from Replay Buffer
17: Use batch B to update environment model $\hat{\mathcal{T}}_i$
18: **end for**

(steps 16 and 17 can be done asynchronously)

It predicts the next states and rewards for each action given the current state. So, it's final goal is to approximate the transition and the reward function of the environment. The predicted rewards and next states are then used to evaluate which action is the best in the current state.

3 Experiments and Results

3.1 Testing Environment

As we are mainly interested in cooperative environments, a good choice for testing environment is *Cooperative Navigation* from *Multi-Agent Particle Environments* (MPE) introduced in [3] (see Fig. 2). It was used to test MADDPG in the original paper and should serve as a good base for comparison with the aforementioned CTDE algorithm. In this environment, agents need to spread to cover a set of landmarks. Agents are forced to do this as fast as possible without any communication and are penalized for colliding with each other. Each agent has complete information about the positions of the landmarks and the positions of other agents. The implementation of this environment used in experiments comes from the *PettingZoo* [11] python library and is known there as "Simple Spread."

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

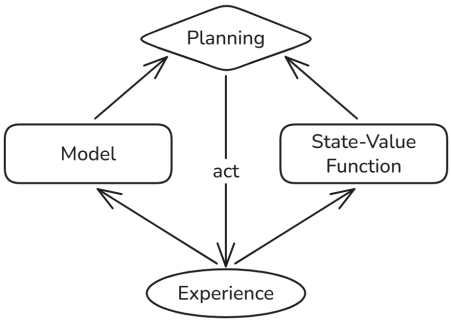

Fig. 1. Integration of the environment model into reinforcement learning routine: model is trained using experience gathered in real environment and used in selecting action alongside with state-value function of the current policy.

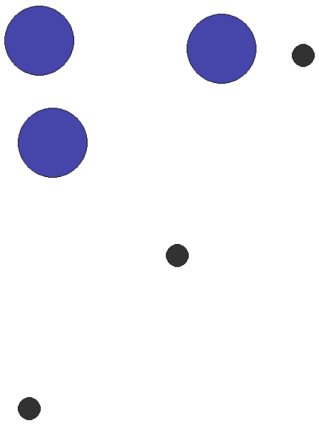

Fig. 2. *Cooperative Navigation* environment. Black dots are the landmarks that blue agents need to cover.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

3.2 Results

The created model-based algorithm was evaluated in the *Cooperative Navigation* environment, and its training results were compared with those of the independent Deep Q-Networks (IQL algorithm) to assess whether integrating an environment model effectively mitigates the non-stationarity problem. It was also compared with the results of a straightforward implementation of the MADDPG algorithm presented in [3] to evaluate whether a model-based approach to DTE can achieve performance comparable to CTDE methods. Among CTDE approaches, MADDPG is one of the most popular algorithms and is also the most closely related to both the independent DQN and the implemented model-based method.

The parameters of the environment (i.e., maximum number of steps, penalty for colliding, etc.) were set to default (same as in the MADDPG paper that introduced this environment).

3.3 Scalability

To verify that proposed approach performs well as the number of agents increases, the tests included: the default test with 3 agents (as in the original MADDPG paper), test with 4 agents, and with 8 agents in the environment. The results of these tests and comparison of the performance with other algorithms mentioned above are presented below.

3 agents. The model-based (MB) algorithm, independent DQNs, and MADDPG were each trained for 200,000 episodes (5 million steps). Fig. 3 presents the comparison of their training performance in terms of the total episodic reward summed across all agents. To reduce variance in the per-episode rewards, the results are shown as a moving average over the last 5,000 episodes.

In this test, independent DQNs showed no sign of improvement, while the model-based method managed to surpass MADDPG, which is explicitly designed to leverage global information in multi-agent environments. The 3 agents scenario serves as the baseline, because that is how the MADDPG algorithm was tested in the original paper.

4 agents. All three algorithms were trained in the environment with 4 agents for one million episodes (25 million steps). The results are presented in Fig. 4. This scenario served as the primary test case and clearly demonstrates the inability of independent DQNs to solve the task due to agents' perceived non-stationarity of the environment. The model-based algorithm mitigates this issue and achieves performance comparable to MADDPG, while doing so without providing the agents with any additional external or global information about the environment.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

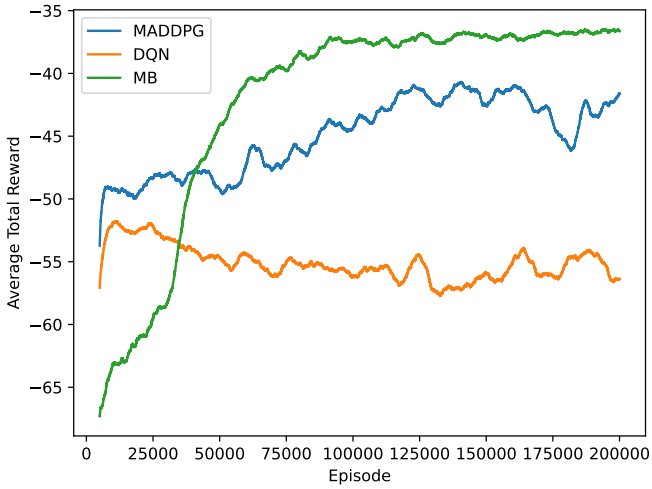

Fig. 3. Comparison of training results for *Cooperative Navigation* with 3 agents. Compared algorithms: independent DQNs, Multi-Agent Deep Deterministic Policy Gradient (MADDPG), and proposed Model-Based algorithm (MB).

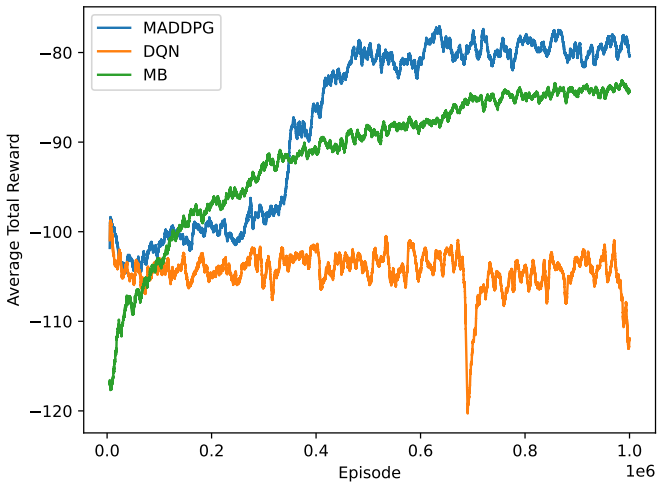

Fig. 4. Comparison of training results for *Cooperative Navigation* with 4 agents.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

8 agents. The model-based algorithm and MADDPG were also evaluated with 8 agents over 300,000 episodes (7.5 million steps). Their training performance is shown in Fig. 5. While both methods could possibly benefit from further training, the current results show that they achieve comparable performance.

Eight agents were used to evaluate the scalability of the Model-Based approach. This represents a twofold increase over the four-agent setting, while larger numbers of agents were not considered due to significantly higher computational cost and runtime.

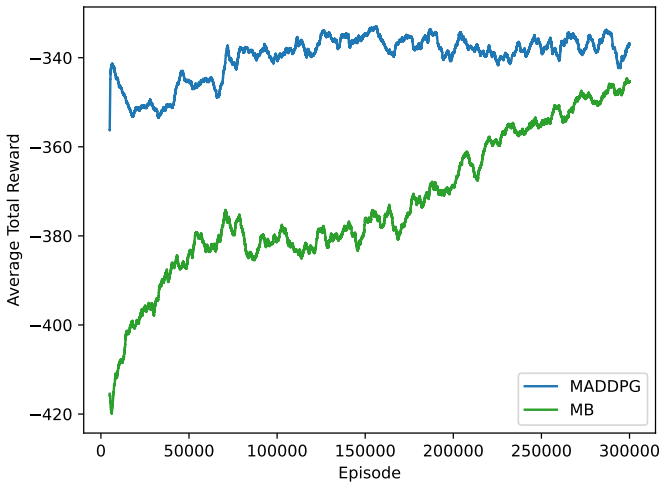

Fig. 5. Comparison of training results for *Cooperative Navigation* with 8 agents.

3.4 Target network

The Deep Q-Network (DQN) algorithm employs a special target network [4], which is a copy of the primary state-action value network (Q-network) but is updated less frequently, remaining several steps "behind" the main network. This target network is used to compute the target Q-values during training. By separating the target calculation from the main Q-network, DQN avoids the instability that arises when the same network is used to estimate both the current Q-value and the target. This design stabilizes learning by providing a slowly moving target, reducing oscillations and divergence in the Q-value updates, and enabling the network to converge more reliably.

Since the proposed model-based algorithm is inspired by DQN (just using a state-value function instead of a state-action value function), the effect of incorporating a target network for the state-value network was also investigated in

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

a 4-agent *Cooperative Navigation* environment. The results, presented in Fig. 6, indicate that using a target network improves the stability and performance of the algorithm, although it may slow down initial learning.

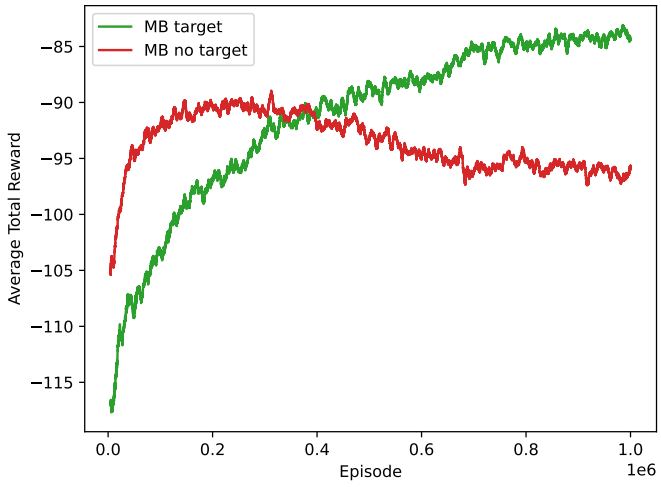

Fig. 6. Comparison of training results for *Cooperative Navigation* with 4 agents between the proposed model-based algorithm without a target state-value network and with a target state-value network.

3.5 Partially Observable Stochastic Games

Training an agent in partially observable environments is especially challenging because a single observation may correspond to multiple underlying states. To act effectively, the agent must "remember" past events and base its decisions not only on the current observation, but also on previous observations [1]. Such scenarios can be formalized as Partially Observable Stochastic Games (POSGs) [1].

The only difference between Partially Observable Stochastic Game and a standard Stochastic Game is that an agent does not have direct access to the full environment state; it can only perceive a partial observation of it — for example, a limited area surrounding itself. Formally, it means that for each agent there is additionally defined a finite set of *observations* and an *observation function* [1], which defines a probability distribution over the agent's possible observations given the current environment state and the last joint action of all agents.

This small change prevents agents from making fully informed decisions based solely on their current observation. In a POSG, the policy of agent i, π_i, is conditioned on the agent's observation history $h_i^t = (o_i^0, o_i^1, ..., o_i^t)$which includes all

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

past observations up to and including the most recent one [1]. Consequently, the agent's internal environment model and state-value function must take as input the entire history of observations up to the current timestep. When implementing this with deep neural networks, this necessitates the use of recurrent neural networks (RNNs) or another mechanism capable of encoding a sequence of observations. This will also slow down learning, as each observation history represents a different "state" for the agent, making the number of "states" quickly grow.

Preliminary results for POSG. The model-based algorithm was adapted for the POSG framework by incorporating a simple history encoder-decoder architecture using small recurrent neural networks. The encoder recurrently processes each new observation together with the hidden state from the previous step, effectively encoding the full history of observations from the beginning of the episode. The decoder then recurrently reconstructs the encoded history to retrieve the most recent observation. The encoder and decoder are trained in an autoencoder fashion with recurrent properties by minimizing a reconstruction loss over the observations across timesteps. This ensures that the encoded history remains meaningful and can be accurately decoded back into the sequence of observations that generated it. During training of the environment model, the decoder is also used to ensure that the model predicts the correct next "state," that is, the correct next encoded history of observations.

Although there is no certainty that this architecture represents the optimal approach for this model-based method, preliminary experiments suggest that incorporating the full observation history in this way can improve agent performance, where other tested methods have failed. However, these results are currently limited to short tests in a single environment, and further evaluation is required to assess the generality and effectiveness of this approach. During training, rapid updates to the encoder–decoder network can cause the encoded observation histories stored in the replay buffer to become outdated, as they were generated using earlier versions of the encoder. Until the encoder–decoder converges, this mismatch between stored representations and the current encoding function may introduce additional instability into the training of the environment model. At the same time, maintaining a larger amount of historical experience for training the environment model may be beneficial in later stages of learning, as it can help stabilize the learned policy once the encoder–decoder representations have largely converged.

For testing the architecture described above, the environment called *Pursuit* introduced in [2] was used (its implementation comes from the *PettingZoo* library). Pursuer agents (red) are placed on the 16 x 16 grid with blue evaders (evaders move randomly) and an obstacle shown in white in the middle (see fig. 7). The goal of the pursuers is to catch all the evaders, but they need to work together as 2 pursuers are required to catch an evader. Each pursuer receives a range-limited observation of its surroundings (marked as 7 x 7 orange rectangle around an agent) and can move up, down, left, right, or stay.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

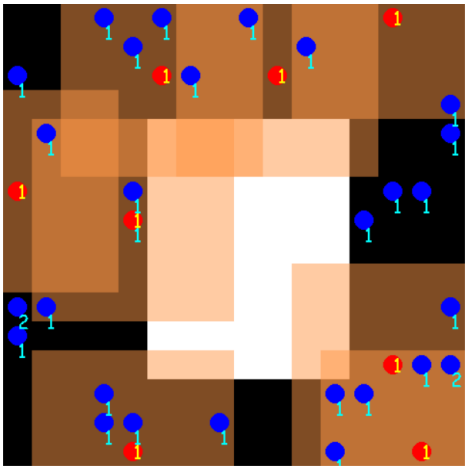

Fig. 7. *Pursuit* environment. Pursuer agents are marked as red dots, evaders are blue.

The maximum number of steps was reduced from 500 to 64, and the rest of the parameters stayed default as per *PettingZoo* implementation. 500 steps is excessive for this environment, and in practice it only slows down learning the policy based on the full observation histories. The preliminary results of training the model-based algorithm in *Pursuit* environment with 8 agents and 30 evaders are shown in fig. 8. The tested architecture shows signs of improvement, but there are no baseline comparisons yet, since both MADDPG and independent DQNs would require similar modifications for POSG as well.

4 Conclusions

This work investigated whether a model-based reinforcement learning approach can mitigate the non-stationarity inherent in Decentralized Training and Execution. By equipping each agent with an internal model of the environment and integrating this model into action selection, the proposed method stabilizes learning without relying on centralized training, global state information, or inter-agent communication.

Experimental results in the *Cooperative Navigation* environment demonstrate that the proposed algorithm outperforms independent Deep Q-Networks and achieves performance comparable to MADDPG, a representative algorithm of the currently dominant Centralized Training for Decentralized Execution paradigm. Importantly, these results hold across different numbers of agents, indicating that the method scales well while preserving full decentralization.

The findings suggest that internal environment model is an effective mechanism for addressing non-stationarity in cooperative multi-agent settings (however, some quantitative indicators could — and likely should — be provided to support this claim, such as policy change rates or other measures of stability.)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

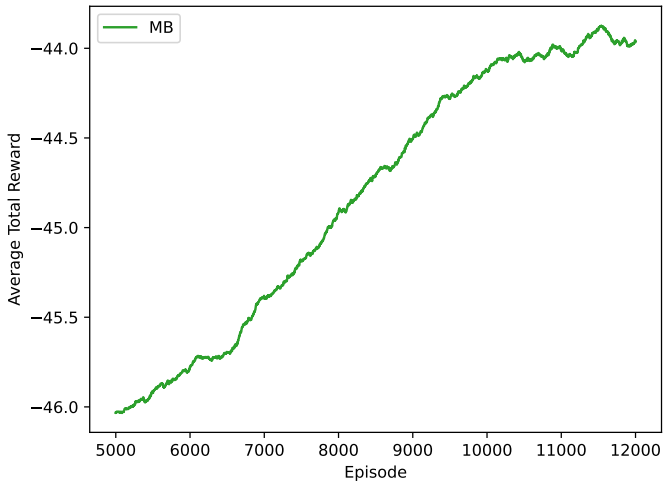

Fig. 8. Results of training 8 agents in *Pursuit* environment using Model-Based (MB) algorithm.

and can serve as a viable alternative to centralized training. The code used to produce above results is also available at:
https://github.com/rniedziolkad/MultiAgentRL/tree/f-spread_env

4.1 Future work

There is a need to check how the size of the replay buffer affects learning the environment model and, thus, how it affects performance. Additionally, the model-based approach should be evaluated in a wider range of environments, particularly those with partial observability. As stated above, some stability metrics could be used to provide empirical evidence that the proposed model alleviates the non-stationarity problem.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Albrecht, S.V., Christianos, F., Schäfer, L.: Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. MIT Press (2024), https://www.marl-book.com
2. Gupta, J.K., Egorov, M., Kochenderfer, M.: Cooperative multi-agent control using deep reinforcement learning. In: Sukthankar, G., Rodriguez-Aguilar, J.A. (eds.) Autonomous Agents and Multiagent Systems. pp. 66–83. Springer International Publishing, Cham (2017)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5

3. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments (2020), https://arxiv.org/abs/1706.02275
4. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013), https://arxiv.org/abs/1312.5602
5. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (Feb 2015). https://doi.org/10.1038/nature14236, https://doi.org/10.1038/nature14236
6. Moerland, T.M., Broekens, J., Plaat, A., Jonker, C.M.: Model-based reinforcement learning: A survey (2022), https://arxiv.org/abs/2006.16712
7. Shapley, L.S.: Stochastic games*. Proceedings of the National Academy of Sciences **39**(10), 1095–1100 (1953). https://doi.org/10.1073/pnas.39.10.1095, https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095
8. Su, K., Zhou, S., Jiang, J., Gan, C., Wang, X., Lu, Z.: Ma2ql: A minimalist approach to fully decentralized multi-agent reinforcement learning (2023), https://arxiv.org/abs/2209.08244
9. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Porter, B., Mooney, R. (eds.) Machine Learning Proceedings 1990, pp. 216–224. Morgan Kaufmann, San Francisco (CA) (1990). https://doi.org/10.1016/B978-1-55860-141-3.50030-4, https://www.sciencedirect.com/science/article/pii/B9781558601413500304
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning. Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, 2 edn. (2018), http://incompleteideas.net/book/the-book.html
11. Terry, J.K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N.L., Lokesh, Y., Ravi, P.: Pettingzoo: Gym for multi-agent reinforcement learning (2021). https://doi.org/10.48550/arXiv.2009.14471
12. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3), 279–292 (May 1992). https://doi.org/10.1007/BF00992698, https://doi.org/10.1007/BF00992698
13. Wong, A., Bäck, T., Kononova, A.V., Plaat, A.: Deep multiagent reinforcement learning: challenges and directions. Artificial Intelligence Review **56**(6), 5023–5056 (Jun 2023). https://doi.org/10.1007/s10462-022-10299-x, https://doi.org/10.1007/s10462-022-10299-x

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29909-3_5