Multi-Agent Reinforcement Learning

Sze-Hang, Wong
Georgia Institute of Technology

Abstract

In this paper, I discuss a multi-agent reinforcement learning (MARL) approach to solving the Overcooked environment. The first part of the paper outlines the key challenges of MARL compared to the single-agent setting (Section 2), and presents major MARL frameworks along with the rationale behind the algorithm selection (Section 3). The second part of the paper focuses on the implementation of MAPPO (Multi-Agent Proximal Policy Optimization) (Section 4), detailing important design decisions and highlighting the key challenges encountered during model training.

1 Overcooked Environment

Overcooked is a multi-agent environment simulating a cooperative cooking task in a constrained kitchen layout. Each agent receives observations consisting of surrounding grid cells, the total dimension of observation received from the environment is 96 (46 observed by target agent, another 46 observations by other agent the last 4 is the position of target agent), including positions of ingredients and cooking tools. The action space includes six discrete commands: stay, up, down, left, right, and interact. Agents must coordinate to pick up ingredients, place them into pots, cook soups, and deliver them to a service window. Episodes last a fixed at 400 time steps.

2 Core Challenges of Multi-Agent Reinforcement Learning

2.1 Non-Stationarity

Non-stationarity arised from multiple agents simultaneously learning and updating its policy. From the perspective of individual agent i, the environment appears non-stationary since the transition function depend not only on its own action a_i, but also on the joint actions of other agents $\mathbf{a}_{-i}$. The environment's transition function can be expressed as

$$T_i(s^{t+1} \mid s^t, a_i^t) \propto \sum_{a_{-i} \in \mathcal{A}_{-i}} T(s^{t+1} \mid s^t, \langle a_i^t, a_{-i} \rangle) \prod_{j \neq i} \pi_j(a_j \mid s^t)$$

- $T_i(s^{t+1} \mid s^t, a_i^t)$: the effective transition probability for agent i
- $a_{-i} \in \mathcal{A}_{-i}$: joint action of all agents except i, drawn from their joint action space $\mathcal{A}_{-i}$

2.2 Credit Assignment

Credit assignment is difficult where agents receive a shared team reward $R(s, \mathbf{a})$. It is unclear how to assign this reward to agents' contributions. A principled approach to this is the *difference reward*, defined as $D_i = G(z) - G(z_{-i})$, where $G(z)$ is the global reward under full joint action and $G(z_{-i})$ is the reward when agent i's action is replaced by a baseline or null action. Another strategy is value decomposition, such as in QMIX, which assumes that the joint action-value function $Q_{\text{tot}}(\mathbf{s}, \mathbf{a})$ can be decomposed into monotonic individual value functions Q_i such that enabling centralized training while preserving decentralized execution.

2.3 Dimensionality Curse of Explorations

Exploration is particularly challenging due to the exponential growth of the joint action space: for n agents with m actions each, the joint space has size $|\mathcal{A}| = m^n$. This makes random exploration ineffective.

3 Algorithm Selection

3.1 Centralized Learning

Centralized learning in MARL trains agents using global state, action, and reward information, effectively reducing the problem to a single-agent RL setting. While this improves coordination and learning stability, Reward design becomes challenging, especially in zero-sum or general-sum games where agent objectives may conflict. The joint action space grows exponentially with the number of agents. Additionally, real-world constraints often require agents to act independently due to limited communication or policy restrictions, reducing the practicality of centralized training in deployment.

3.2 Independent Learning

Independent learning in MARL treats each agent as a separate learner, using only its local observations and rewards without access to other agents' states or actions. This simplifies implementation and scales well, as each agent solves its own RL problem. However, the environment becomes non-stationary due to other agents' evolving policies. Reward assignment can also be ambiguous, especially in cooperative settings where individual contributions to team success are hard to disentangle.

3.3 Centralized Training with Decentralized Execution (CTDE)

Centralized Training with Decentralized Execution (CTDE) is a framework that leverages global information during training while allowing each agent to act independently based on its own local observations at execution time. This approach combines the strengths of centralized and independent learning: it enables coordinated learning and efficient credit assignment during training, while remaining scalable and practical for real-world deployment. However, CTDE still faces challenges, the discrepancy between centralized training and decentralized execution can degrade performance if agents become overly reliant on global context during training.

3.4 Conclusion

A comprehensive comparison of training strategies is summarized in Table 1. Among them, Centralized Training with Decentralized Execution (CTDE) emerges as the most balanced approach. While independent learning offers advantages such as low implementation complexity, smaller action space, and agent independence, it suffers from non-stationarity, which is detrimental to convergence. Therefore, CTDE is chosen as the training strategy for the Overcooked problem.

	Independent Learning	**Centralized Learning**	**CTDE**
Algorithm complexity	low	low	high
Action space complexity	low	high	low
Agent independence	high	no	high
Non-Stationarity	high	low	medium
Training stability	low	high	medium

Table 1: Comparison of algorithms

4 Multi-Agent Proximal Policy Optimization (MAPPO)

Proximal Policy Optimization (PPO) have most of the desired properties of reinforcement learning, namely excellent training stability, good sample efficiency, and the ability to model policy directly. The detailed analysis and discussion of PPO can be found at the previous assignment paper (Sze Hag (2025). Here we discuss the properties specific for multiple agent learning. The multi-agent policy gradient for agent i is given by equation 1

$$\nabla_{\phi_i} J(\phi_i) = \mathbb{E}_{a^t \sim \pi} \left[Q\left(h_i^t, z^t, \langle a_i^t, a_{-i}^t \rangle; \theta_i\right) \cdot \nabla_{\phi_i} \log \pi_i \left(a_i^t \mid h_i^t; \phi_i\right) \right] \tag{1}$$

- h_i^t: Agent i's local observation history at time t
- z^t: Global context or encoded state shared among agents (may include s^t)

The key difference between the single-agent and multi-agent versions lies in the use of joint actions and observation histories in the computation. In MAPPO, the Q-function (Critic) considers the actions of all agents, while the policy function π only depends on the action of the agent being optimized. These two characteristics form the core of the MAPPO framework. The model architecture is shown in the Fig 4.1, and the implementation flow is shown in Figure 4.2.

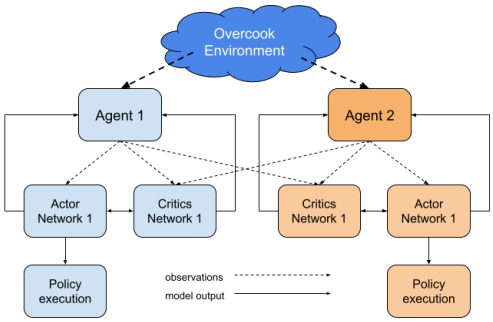

Fig 4.1 - Model Architecture of MAPPO

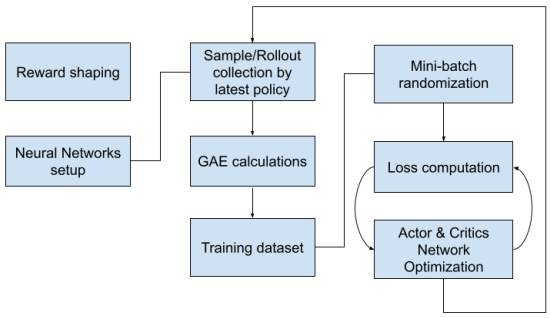

Fig 4.2 - implementation algorithm of MAPPO

MAPPO adopt a **Actor-Critic Architecture** where each agent has an **actor network** (π_θ) that outputs a categorical distribution over discrete actions. And a **critic network** V_ϕ that estimates the value of the global state. The agents select actions by sampling from their respective policy distributions. During training, each agent's critic receives full state information. Actor policies only use local observations. During execution, each agent acts based solely on its local observation.

5 Major Design Decision of MAPPO

5.1 Neural Network Design

The Overcooked environment is fully observable, a plain vanilla neural network is sufficient. There is no need to incorporate recurrent structures such as RNNs to infer hidden state information from historical observations.

5.2 Centralized Critic Network

To address the non-stationarity problem described in Section 2.1, a centralized critic trained using global information gathered from both agents is adopted. The decentralized actor networks use

feedback from the critic network to evaluate the actions quality. The architecture is illustrated in Figure 4.1, where the critic networks receive observations from both agents.

5.3 Independent Actor Network

To address the explosion of the joint action space described in Section 2.3, and to ensure scalability, each agent's actor network is trained independently. This approach reduces implementation complexity. This keeps the model modular and maintainable. The 'Policy Execution' block in Figure 4.1 illustrates this design.

6 Aligning Reward Functions with Solution Concepts

The Overcooked task is inherently cooperative to delivery the maximum number soups. However, there are competitive aspects, particularly around credit assignment described in section 2.2, where agents must be rewarded based on their individual contributions. Overall, the solution concept should be **Social Welfare Maximization**. Reward functions experiments in the following 2 sections is based on the `cramped_room` layout.

6.1 Reward Sharing scheme

To incentivize cooperation among agents, an appropriate reward-sharing scheme is crucial. Fully shared rewards **(common reward game)** encourage cooperation but may lead to free-riding behavior. Individually assigned rewards **(general sum game)** promote self-interest which can trap the system in suboptimal Nash equilibria. Partially shared reward schemes aim to strike a balance between these extremes. Two schemes are tested in the Cramped Room layout: one with no reward sharing and another with partial sharing of sparse rewards.

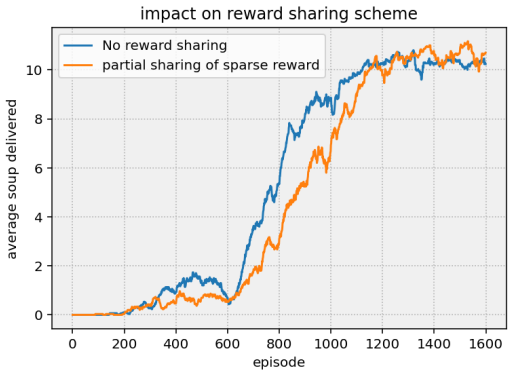

Fig 6.1

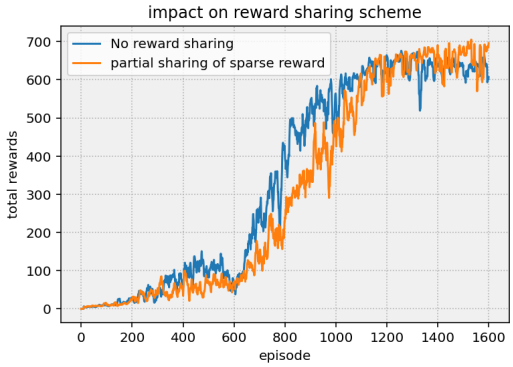

Fig 6.2

Fig 6.1 shows that no reward sharing works better. This result is attributed to small layout size and resources (e.g., ingredients, cooking stations) are abundant, there is limited necessity for tight cooperation between agents. Each can perform their tasks independently without significant interference or dependency on others.

6.2 Reward Shaping

The default reward structure is highly sparse. This makes it difficult for exploration. Reward shaping is necessary to provide guides to agents toward the goal of soup delivery. A small rewards is assigned to intermediate sub-goals that contribute to the final outcome. Those 4 sub-goals are shown in the following table. From trial runs, I observed **agents often begin cooking with insufficient onions**. Based on this, I experimented with the following reward scheme:

	scheme 1 - pickup soup over onion	**scheme 2 - pickup onion over soup**
ONION_PICKUP_REWARD	1	1
PLACEMENT_IN_POT_REW	1	3
DISH_PICKUP_REWARD	1	1
SOUP_PICKUP_REWARD	3	1

Figure 6.3 shows that performance improves significantly when more reward is assigned to picking onions than to picking up soup from the pot—arguably the most important action in solving the Overcooked problem. This result highlights that a major bottleneck in learning stems from agents attempting to cook soup prematurely. Proper reward design is therefore critical for guiding agents toward the right incentives and achieving the overall goal efficiently.

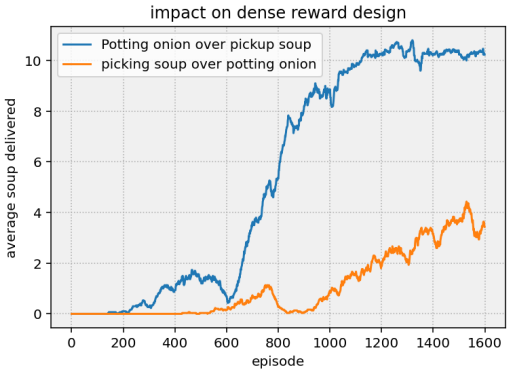

Fig 6.3

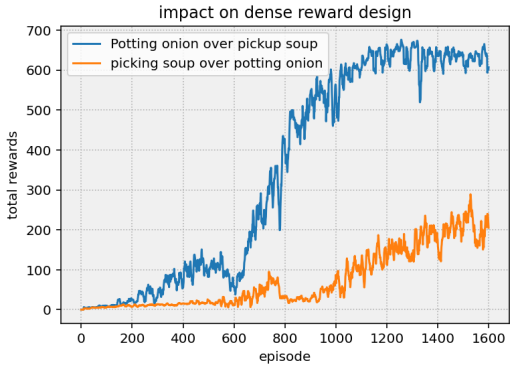

Fig 6.4

7 Implementation

Following table is the summary of design decision discussed (section 5, section 6.1, section 6.2) that will be implemented in the MAPPO Python implementations:

Component	**Design decision & relevance to MRAL**
Neural Networks Architecture	Deploy as vanilla Neural Networks to take advantage of full environment visibility
Critic network	shared global observation as input to reduce Non-stationarity
Actor network	agent's own observation as input to reduce explosive growth of joint-action space and enable scalable independent execution
Reward shaping	assiging higher reward to onion picking than soup picking to better align agent's incentive to the goal of soup delivery
Reward sharing scheme	no reward sharing to reduce overhead

Follow the Githash published at this paper for the full Python source code. Below are major functions implemented in the MAPPO, detailed pseudo-code are described at Figure 4.2:

ActorNN()	2 Policy Neural Network of 96x96x64 for 2 agents
CriticNN()	2 Value Neural Network of 192x256x64 for 2 agents
optimize_step()	This is where MAPPO optimization takes place. Each rollout (4000 timesteps) is fed into this function and processed over 5 passes (epochs). The rollout is divided into mini-batches of 128 timesteps, resulting in a total of 160 updates per rollout (calculated as 4096x5/128)
calculate_GAE()	GAE generating function to facilitate debugging and hyperparameter tuning (γ and λ)
main training loop	this is the entry point where sampling will be collected according to the latest policy. Each rollout contains 4096 time steps with a variable number of episodes.

8 Training Result

Figure 8.1 presents the training results across the three required layouts. The shaded lines in the background represent the number of soups delivered per episode, while the solid lines show the running average to aid visualization. The results demonstrate that this MAPPO implementation solves the `cramped_room` layout in approximately ~1000 training episodes. It takes around ~2500 episodes to solve `coordination_ring`, and about ~4000 episodes to solve the most challenging layout, `counter_circuit0_1order`

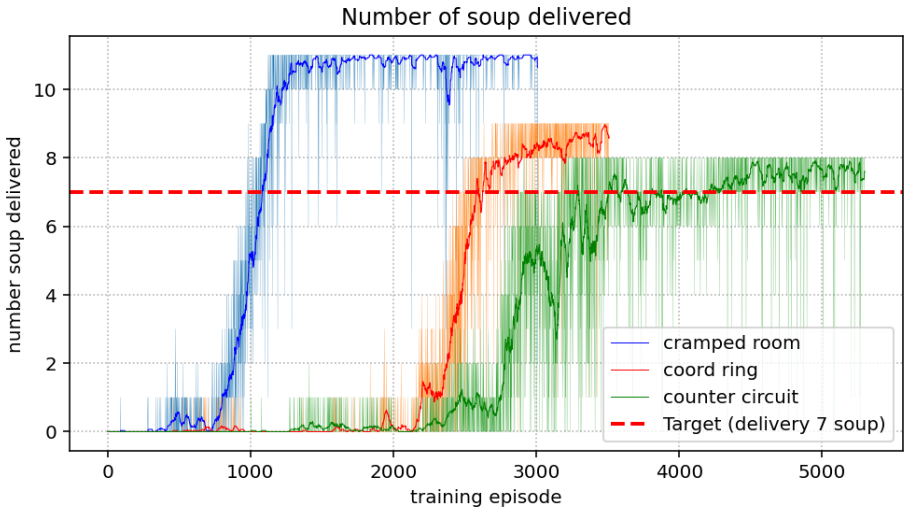

Fig 8.1: final training results on all 3 layouts

ALL 3 layouts are trained with the same training code, therefore shares the same set of hyper-parameters. Below tables show the hyper-parameters used in this training, Details of tuning process can be referenced to previous assignment paper Sze Hag (2025)

Hyper-parameters	
Lambda (λ)	0.5
Gamma (γ)	0.995
PPO Clipping Parameter	0.1

Training Batch Arrangements	
Time-steps sampled per Rollout	4000
Training Epoch per rollout	5
Mini batch size	128

9 Testing Results over 100 Consecutive Episodes

Below Table shows the four performance metrics used to evaluate the performance of MAPPO

Performance Metrics	**Definition**
Number of Soup delivered	it is the key performance metric to show Number of Soup delivered
Potting Efficiency (%)	defined as potting_onion/onion_pickup, it shows the % of onion successfully delivery to the pot
Pot Utilization (%)	defined as soup_pickup/(potting_onion/3), it shows the % of times the pot is cooking with 3 onions
Soup Deliver Efficiency (%)	defined as soup_delivery/soup_pickup, it shows the % soup successfully delivered to the counter

9.1 Final Testing Results

The model meets the target of delivering over 7 soups within 400 steps in all three layouts over 150 testing episodes as shown in Fig 9.1.

Fig 9.1: final testing results over 150 episode on all 3 layouts

9.2 Performance Metrics Evaluation

Fig 9.2 shows that `coordination_ring` achieves the highest potting efficiency, while surprisingly, `cramped_room` has the lowest. Still, all three layouts reach over 90% potting efficiency. Fig 9.3 shows that `counter_circuit0_1Order` has the lowest pot utilization, indicating agents initiate cooking prematurely, as discussed in Section 6.2. Fig 9.4 shows that all three layouts exhibit excellent soup delivery efficiency, meaning agents effectively find paths from pot to counters. Overall, all 3 layouts shows top % in all performance metrics that contributed to the overall success of the MAPPO.

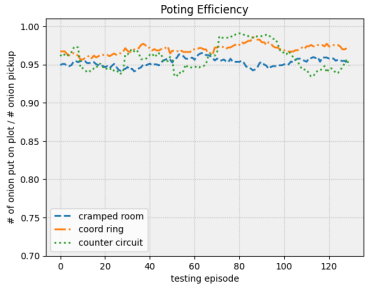

Fig 9.2 - potting efficiency

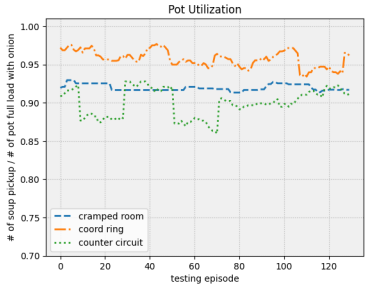

Fig 9.3 - pot utilizations

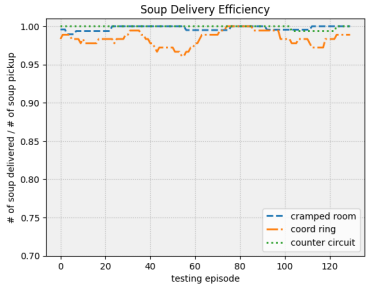

Fig 9.4 - soup delivery efficiency

10 Ablation Study

The success of MAPPO is largely attributed to its CTDE (Centralized Training with Decentralized Execution) architecture. To verify this claim, an ablation study is conducted by comparing the training results on the `cramped_room` layout using the CTDE model against alternative architectures:

1C2A – The original CTDE model with one centralized Critic and two decentralized Actors.
1C1A – A simplified architecture with one shared Actor and one shared Critic for both agents.
2C2A – An independent architecture with a separate Actor and Critic for each agent

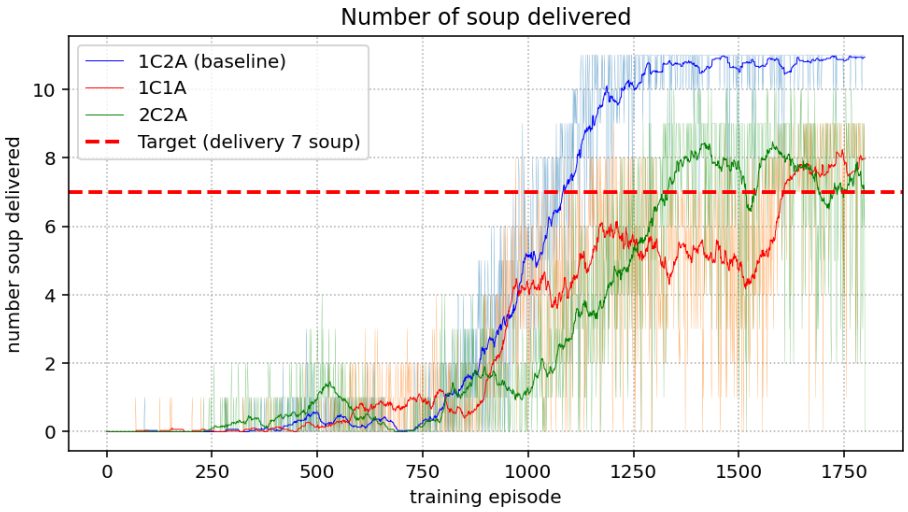

Fig 10.1

Figure 10.1 highlights the clear dominance of the CTDE model (blue) in both learning speed and final performance. it also demonstrating high stability in learning. However, Independent Learning (1C1A & 2C2A) lags far behind—its, marked by high variance and prolonged stagnation, suggesting that without centralized coordination, the agents struggle to develop coherent cooperative behavior. This stark disparity underscores the importance of CTDE in solving complex, coordination-heavy tasks like Overcooked.

11 Result Discussion - What Worked and What Didn't?

I began with A2C, which solved the cramped_room, but it failed to solve Coordination_Ring. Consequently, I upgraded to MAPPO, which solved Coordination_Ring but it failed with Counter_circuitO_1order. I discovered that only one agent performed mostly meaningless actions, so I applied reward-sharing (section 6.1) to better align with the solution concept of the problem but still fell short of meeting the problem's success criteria. Further analysis revealed that agents would start cooking even insufficient ingredients were in the pot. Adjusting the reward structure (section 6.2) proved effective, and the final version successfully solved all 3 layouts.

12 Retrospect - What will I try if I had more time?

I will try out Value Decomposition based models like QMIX. Since it belongs to a radically different design pattern, comparing them with policy-based algorithms will give me more insight into the structure of MARL problems. Also, I will investigate the possibility of integrating solution concepts like Nash Equilibrium into the algorithm to enhance learning efficiency. In addition, I would explore the impact of different coordination mechanisms in team-based tasks, such as communication channels. I am also interested in studying the robustness of the learned policies under distributional shift or adversarial settings.

References

Wong Sze Hag. Cs 7642 lunar lander project paper. *CS 7642 course project*, 2025. https://github.gatech.edu/gt-omscs-rldm/7642RLDMSummer2025swong308/blob/main/project_2/CS_7642_RLCM_Project_2.pdf.