Learning Cooperative Policies Across Layouts in Overcooked

Mattia Maranzana
University of Bologna
mattia.maranzana@studio.unibo.it

Abstract

This project investigates the problem of coordinated policy learning in the Overcooked multi-agent environment, where two agents must collaborate across diverse kitchen layouts. An algorithm inspired by Multi-Agent Proximal Policy Optimization (MAPPO) is proposed to address the challenges of generalization and adaptability in cooperative tasks. To promote generalization, we train a single joint policy across a diverse set of kitchen layouts. Crucially, we introduce an adaptive curriculum sampling strategy that dynamically focuses training on layouts where the agents are performing poorly. Our experiments demonstrate that this combined approach produces a policy that achieves strong performance across multiple layouts.

1 Introduction

Overcooked Carroll et al. [2020] is a popular benchmark in multi-agent reinforcement learning (MARL) that focuses on teamwork and coordination. It requires two agents to work together in a kitchen to cook and serve soups while managing limited space and time. Different layouts highlight distinct coordination challenges, ranging from avoiding collisions in tight spaces to adopting complementary strategies for task completion.

In this project, we focus on four representative layouts—Cramped Room, Asymmetric Advantages, Coordination Ring, and Forced Coordination—as they capture both low-level and high-level aspects of teamwork. The cooking task is simplified to include only onions, dishes, and soups, requiring agents to complete a sequence of actions before a dish can be served for reward.

To address these challenges, we use Multi-Agent Proximal Policy Optimization (MAPPO) Yu et al. [2022], a widely used extension of PPO Schulman et al. [2017] for cooperative settings. MAPPO leverages centralized training with decentralized execution, allowing agents to learn coordinated behaviors while still acting independently at test time.

2 Methodology

2.1 Network Architecture

For this task, both agents share a single policy network for action selection and a separate value network for state evaluation, rather than having individual networks. Sharing these networks reduces the total number of learnable parameters, speeds up training, and encourages coordinated behavior, as both agents learn symmetric policies and update their strategies from the same learning signal.

Each network is a multilayer perceptron (MLP) with two hidden layers, each consisting of 128 and 64 units, respectively, and uses ReLU activations.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Policy network: takes an agent's local observation and outputs logits over the six discrete actions {`up`, `down`, `left`, `right`, `noop`, `interact`}.

Value network: receives the joint state representation, where inputs are the concatenated observations of all agents, and outputs a scalar state-value estimate.

Both networks are optimized independently with Adam, using separate learning rates for the policy and value functions.

2.2 Reward Shaping

The task requires agents to complete a sequence of actions: place three onions in a pot, take the resulting soup onto a plate, and deliver it. Since the environment provides a reward only after completing all these steps, learning without guidance is extremely difficult. To address this, we employ intermediate reward shaping, assigning additional rewards for partial progress toward completing a dish.

The shaped reward is computed as:

$$r_{total} = r_{env} + \alpha \cdot r_{shaped}$$

where r_{env} is the standard environment reward, r_{shaped} is the intermediate reward derived from agent progress (e.g., placing onion), and α is a shaping coefficient. The shaping coefficient can optionally decay over time, allowing the agents to gradually rely more on the environment reward as training progresses. This mechanism significantly improves learning, enabling agents to solve the task even in the early stages of training.

2.3 Environment Sampling

Since the agents are trained on multiple layouts, we implement a weighted environment sampling strategy to select which layout to use at each rollout. The idea is to prioritize layouts where the agents have lower recent performance, encouraging more learning on challenging tasks.

For each layout, we track the average reward over the last 50 episodes. Sampling weights are computed as the inverse of these averages, so layouts with lower rewards are more likely to be selected. To ensure stability and exploration, each layout is assigned a minimum probability ϵ, and the weights are normalized to sum to 1:

$$P(L) \propto \frac{1}{mean(rewards_L) + \epsilon}$$

During training, a layout is randomly chosen according to this weighted distribution, and the environment is reset after each episode.

This approach enables the agents to balance learning across multiple environments, prioritizing layouts that remain challenging while still occasionally sampling easier ones to prevent forgetting previously learned environments.

2.4 Batch Synchronization

In our centralized training with decentralized execution (CTDE) framework, the actor and critic networks are trained on different types of data: the critic learns from global states, while the actor learns from individual agents' observations. This requires a strategy for constructing minibatches for stochastic gradient updates. We implemented and analyzed two approaches: Synchronized Timestep Sampling and Decoupled Agent-State Sampling.

Synchronized Timestep Sampling: This conventional approach preserves the correspondence between actor and critic updates. Minibatches are formed by first sampling a random set of timesteps from the rollout buffer. The critic is trained on the global states from these timesteps, while the actor is trained on the corresponding agent-level experiences from the same timesteps. This ensures that the policy update for a given state is informed by the value function's estimate of that state within the same gradient step, maintaining theoretical consistency with the PPO formulation.

Decoupled Agent-State Sampling: In this variant, the state data for the critic and agent-level observations for the actor are shuffled independently. Gradient updates for the critic are performed over minibatches of randomly sampled states, and updates for the actor are performed over independently sampled agent-level experiences. This breaks the direct correspondence between actor and critic updates at the minibatch level, introducing variance but increasing minibatch diversity and allowing for more thorough shuffling of agent experiences across timesteps.

2.5 Advantage Estimation

To train the agents, we compute advantages using Generalized Advantage Estimation (GAE) Schulman et al. [2018]. For each agent, the temporal-difference residuals (deltas) are computed based on the immediate rewards and value predictions. These deltas are then exponentially weighted with a discount factor γ and a smoothing parameter λ to produce the advantage estimates:

$$\hat{A}_t = \delta_t + \gamma\lambda(1 - d_t)\hat{A}_{t+1}$$

where d_t indicates whether the episode ended at timestep t. Advantages are normalized before training to stabilize learning.

2.6 Policy and Value Updates

Policy updates are performed using the PPO clipped objective to constrain large updates:

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}\Big[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\Big],$$

where $r_t(\theta) = \pi_\theta(a_t \mid o_t)/\pi_{\theta_{\text{old}}}(a_t \mid o_t)$ is the probability ratio and ϵ is the clipping parameter. An entropy regularization term is added to encourage exploration:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{policy}} - \beta\,\mathcal{H}[\pi_\theta],$$

with β the entropy coefficient and $\mathcal{H}[\pi_\theta]$ the Shannon entropy of the policy.

The value network is trained independently to minimize the mean-squared error between predicted and target returns.

3 Experiments

3.1 Hyperparameters

We summarize the main hyperparameters used for training in Table 1. The table lists the selected values for our experiments along with the ranges tested during preliminary tuning.

Hyperparameter	Selected Value	Values Tested
Learning rate (policy)	3e-4	3e-4, 1e-4, 1e-3
Learning rate (value)	1e-3	3e-4, 1e-4, 1e-3
Discount factor γ	0.99	0.99
GAE lambda λ	0.95	0.95, 0.99
PPO clip ratio ϵ	0.3	0.1, 0.2, 0.3
Entropy coefficient	0.01	0.01, 0.05
Value coefficient	0.01	0.01, 0.1
Batch size	256	128, 256, 512
Epochs per update	7	4, 7, 10
Buffer size	4000	2000, 4000, 8000
Linear decay	full run	half run, full run, no decay
Network activation	relu	relu, tanh
Batch Synchronization	True	True, False

Table 1: Hyperparameters used for training.

We tested multiple hyperparameter configurations to assess their influence in the Overcooked layouts (see `main.ipynb` for full runs). The learning rate proved to be the critical parameter, as unsuitable values quickly prevented learning. The choice of buffer size and batch size also played a decisive role: too small a buffer hindered training entirely, while larger batch sizes consistently improved performance. The number of epochs per update showed a narrow effective range—too few epochs limited progress, whereas excessive epochs led to overfitting and stalled learning multiple layouts.

Reward-related hyperparameters were similarly important. Setting the value coefficient or entropy coefficient too high prevented meaningful learning, highlighting the need for careful tuning. The GAE parameter λ was also sensitive, with performance varying noticeably between 0.95 and 0.99.

Other hyperparameters had more moderate effects. Using ReLU activation gave a slight advantage over tanh, and synchronized batch updates were marginally better than asynchronous ones. Linear decay of the shaping coefficient could be considered negligible.

3.2 Training Performance

After an initial hyperparameter search, the most effective configuration was selected and trained for 2 million environment steps. This allowed us to evaluate their stability and long-term performance across the considered Overcooked layouts.

Figure 1 shows the evolution of the total team reward during training, averaged across multiple runs. Figure 2 reports the reward for each layout separately.

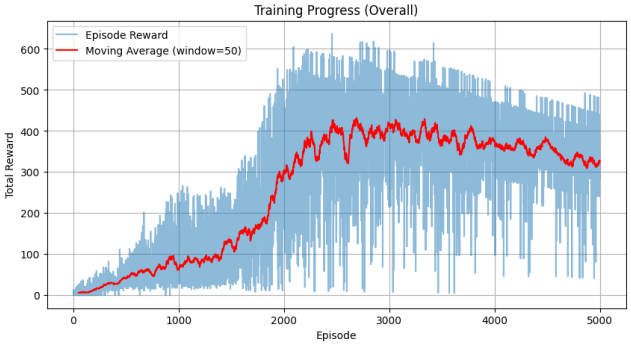

Figure 1: Total team reward over training.

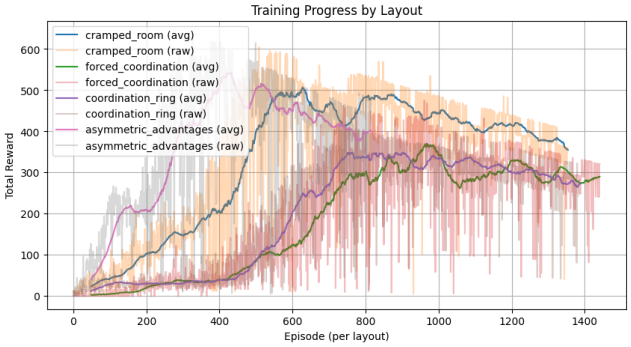

Figure 2: Reward evolution for each layout individually.

The learning curves across layouts show coherent trends, with rewards increasing steadily during training. More demanding environments, such as Forced Coordination and Coordination Ring, required a larger number of episodes to stabilize compared to simpler layouts like Cramped Room and Asymmetric Advantages. Nonetheless, all layouts reached stable performance well before the

end of training, suggesting that the full 2 million steps were more than necessary. This indicates that shorter training runs could be sufficient once suitable hyperparameters are chosen.

3.3 Evaluation

Table 2 summarizes the test-time performance of the trained agents across 10 runs per layout.

Layout	Mean Reward	Standard Deviation
Cramped Room	164.00	28.00
Asymmetric Advantages	214.00	29.73
Coordination Ring	150.00	13.42
Forced Coordination	128.00	24.00

Table 2: Test-time performance over 10 runs per layout.

Beyond the quantitative results, video analysis of the trained agents reveals interesting behavioral patterns. In the Asymmetric Advantages layout, the agents often adopt complementary roles: for example, one agent waits with a plate while the other collects onions and fills the pot. Such role specialization suggests that the shared policy can still produce coordinated division of labor when the environment structure makes it beneficial. In contrast, in the Cramped Room layout, this behavior did not emerge, and the agents instead operated in a more symmetric manner, likely due to the tighter spatial constraints.

Another observation is that the agents frequently converged on a suboptimal strategy, where only a single pot was used for cooking despite two being available. While this strategy is sufficient to achieve consistent rewards, it indicates that the learned policies prioritize reliability over maximizing throughput.

For reference, the folder `evaluation_videos` contains one recorded episode for each layout, illustrating these behaviors.

4 Conclusion

We demonstrate that a single Multi-Agent PPO policy can be trained to learn diverse Overcooked layouts. Our key finding is that an adaptive curriculum, which prioritizes training on layouts with lower performance, yields a more robust policy than both single-layout specialists and agents trained with uniform sampling. This shows that intelligently guiding an agent's exposure to environmental diversity is crucial for effective generalization.

Despite these promising results, we acknowledge the inherent limitations of our approach. A primary practical barrier is the computational demand of on-policy MARL. The significant number of environment steps required to see meaningful learning progress makes extensive hyperparameter tuning and ablation studies impractical without access to large-scale compute resources. This hardware constraint necessarily limits the breadth of experimentation.

Furthermore, our experiments confirmed that the policy fails at zero-shot transfer to entirely unseen layouts, indicating it still overfits to the features of the training maps rather than learning a truly abstract cooperative strategy. Future work could address this by exploring architectures that better capture spatial invariances. For instance, while the environment provides a feature-based state, a pre-processing step could render this state into an image-like tensor. This would allow a Convolutional Neural Network (CNN) to learn transferable spatial patterns, potentially leading to the more generalizable policies needed for real-world application.

References

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination, 2020. URL `https://arxiv.org/abs/1910.05789`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022. URL https://arxiv.org/abs/2103.01955.