Experiments on layout generalizaion and reward shaping in Cooperative Multi-Agent Overcooked-AI environment

Gabriele Ceccolini
Università di Bologna
gabriele.ceccolini@studio.unibo.it

Abstract

We investigate if policies trained via self-play with PPO can generalize across multiple layouts in Overcooked-AI, evaluating the role of reward shaping and entropy regularization. Our experiments show that while strong cooperation can be learned on individual layouts, generalization remains a challenge due to high performance variance across environments.

1 Introduction

Reinforcement learning (RL) has made significant progress in recent years, especially in competitive games such as Go [5] and in complex video games like StarCraft (a single-agent 1vs1 game). However, cooperative multi-agent RL is still a difficult research area, especially when agents need to learn to coordinate in environments with sparse rewards and complex dynamics.

The Overcooked-AI environment [1] is now a standard benchmark for evaluating cooperative policies and generalization in these scenarios.

In this work, we first study the importance of reward shaping. Then, we investigate how well PPO agents trained with self-play can play in multiple layouts and also in unseen layouts.

2 Problem

The Overcooked-AI environment [1] is a benchmark for cooperative reinforcement learning, where two agents must coordinate in a kitchen to prepare and deliver onion soups. Agents follow a sequence of actions—gathering ingredients, cooking, and delivering dishes—to maximize the number of deliveries within a fixed time limit, receiving shared rewards for each successful delivery.

Due to the sparse nature of the main reward, the environment allows the use of reward shaping: intermediate rewards for useful sub-tasks (e.g., placing onions in the pot) that occur more frequently. In our experiments, reward shaping was crucial in the early training stages to guide the agents toward effective cooperative behaviors.

Event	**Shaped Reward**
Place ingredient in pot	$+3$
Pick up dish	$+3$
Pick up soup	$+5$
Deliver soup (sparse)	$+20$

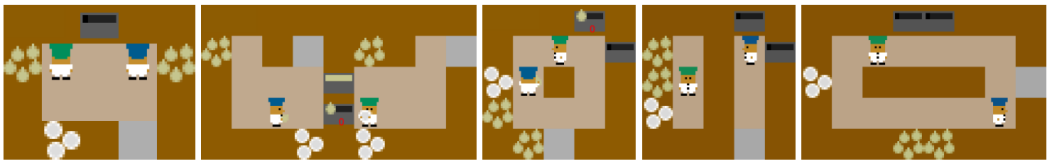

Figure 1: Example of kitchen layouts in Overcooked-AI

The creators of the benchmark designed Overcooked-AI to support a wide range of generalization experiments. The environment features multiple scenarios, known as "layouts" representing different kitchen configurations in which the positions of key objects and the walkable areas can vary a lot.

Some layouts are intrinsically more complex than others. In certain scenarios, cooperation is not only encouraged but enforced: here, agents do not have free access to all areas or all objects, and must necessarily rely on their partner to accomplish specific steps of the recipe.

In this report we adopt the simplest training setup called self-play, in which both agents use the same policy and play with themselves during training. The main objective is to investigate whether the learned policy can generalize to previously unseen layouts and perform well across a variety of scenarios.

3 Proximal Policy Optimization (PPO)

As mentioned before, Overcooked is a highly challenging cooperative benchmark where achieving good performance requires close collaboration between the two agents and adaptation to non stationary strategies, since actions must be taken not only based on the environment but also on what the other agent is doing.

A preliminary decision involves choosing between an on-policy or off-policy algorithm. In particular, off-policy methods such as DQN [3] which have shown excellent results in stationary strategy games like Atari are not suitable in this scenario. This is because off-policy updates use data collected with strategies potentially very different from the current one. In Overcooked, even a small change in the partner agent's strategy can lead to a radical shift in the game's dynamics, making it natural to prefer on-policy algorithms.

For these reasons, we selected PPO. Compared to previous policy gradient methods such as REINFORCE and A2C [2], PPO's clipping mechanism offers the important advantage of maintaining stable, gradual, and cautious policy updates. This helps to avoid breaking the often delicate balance and cooperative logic with overly drastic updates.

The idea and goal behind PPO are very similar to those of its direct predecessor, TRPO: both aim to leverage the available data to achieve the best possible policy update without risking overly drastic changes that could cause a collapse in performance.

TRPO framed this as a second-order optimization problem, which is mathematically complex and challenging to implement in practice. In contrast, the success of PPO is largely due to its implementation simplicity: it formulates the update as a first-order problem, relying only on the gradient.

3.1 PPO-CLIP

PPO-CLIP is the main implementation of PPO and is based on a "clipping" mechanism that automatically limits the maximum policy change in a given update to a value defined by the hyperparameter ϵ.

First, we define the policy ratio:

$$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \tag{1}$$

which represents the ratio between the probability of taking action a_t in state s_t under the new policy π_θ and under the previous policy π_{θ_k}. If this value is high in absolute value, without the clipping mechanism, the policy update could be too drastic. so clipping will be used.

The PPO updates are defined as:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{2}$$

where tipically multiple steps of (mini-batch) stochastic gradient ascent are performed to maximize the following objective function:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\ \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},\ 1 - \epsilon,\ 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right) \tag{3}$$

where,

$$A^{\pi_{\theta_k}}(s_t, a_t) = Q^{\pi_{\theta_k}}(s_t, a_t) - V^{\pi_{\theta_k}}(s_t) \tag{4}$$

is called the advantage, and it measures how much better or worse a particular action is compared to the average of all possible actions in the same state.

In general, therefore, in all cases we have a maximum update shift in absolute value:

$$\text{Maximum update} = \begin{cases} (1 + \epsilon)\, A^{\pi_{\theta_k}}(s, a) & \text{if } A^{\pi_{\theta_k}}(s, a) \geq 0 \\ (1 - \epsilon)\, A^{\pi_{\theta_k}}(s, a) & \text{if } A^{\pi_{\theta_k}}(s, a) < 0 \end{cases} \tag{5}$$

In practice, the total objective optimized at each update step also includes a value function loss term and an entropy regularization term, as in the official PPO paper [4]:

$$L_t^{\text{PPO}}(\theta) = \mathbb{E}_t \Big[L_t^{\text{CLIP}}(\theta) - c_1 (V_\theta(s_t) - R_t)^2 + c_2 \mathcal{H}(\pi_\theta(\cdot|s_t)) \Big] \tag{6}$$

where c_1 and c_2 are coefficients weighting the value loss and entropy bonus, respectively.

This clipping mechanism ensures that, regardless of the magnitude of the advantage, the policy update is bounded within a controlled range. As a result, it prevents excessively large changes to the policy in a single update step, so improving training stability and avoiding the risk of destroying previously learned cooperative behaviors due to too big policy shifts.

In the following sections, we describe the experimental setup using a simple implementation of PPO-CLIP, and discuss the results in the Overcooked multi-agent environment.

4 Training Setup and Experiments

In this section, we describe the training setup used throughout our experiments, starting from solving the $cramped\_room$ layout and moving towards generalization tests on unseen layouts.

4.1 Solving `cramped_room`

As a first goal, we aimed to solve the $cramped\_room$ layout, setting a target of achieving an average greedy score above 50 (every sparse reward is worth 20 points) in 400-step episodes. This initial objective was usefull in developing a solid training infrastructure, a first working PPO implementation, and a set of hyperparameters for stable learning.

For our minimal PPO implementation, both the policy and value networks are simple feedforward networks: each is a 3-layer fully connected neural network with 256 and 128 hidden units and ReLU activations. The only difference is the output layer size, which is the action space size for the policy (6 possible actions) and 1 for the values function (the value of the state).

The main hyperparameters used are:

- Discount factor: $\gamma = 0.99$
- Clipping ratio: 0.15 (PPO objective)
- Learning rate: 5×10^{-4} (Adam optimizer)
- Entropy bonus coefficient: β = 0.01

The policy network outputs logits over the action space, which are converted to probabilities using the softmax function. Actions are sampled from this distribution during training (sampling selection) and selected greedily during evaluation (greedy selection). The value network estimates the expected return for each state.

In our PPO implementation, the total loss minimized at each update step is:

$$\mathcal{L} = -\mathbb{E}_t \left[\min \left(r_t(\theta) A_t,\, \text{clip}\big(r_t(\theta),\, 1-\epsilon,\, 1+\epsilon\big) A_t \right) \right] + 0.5\, \mathbb{E}_t \left[\big(V_\theta(s_t) - R_t \big)^2 \right] - \beta\, \mathbb{E}_t \left[\mathcal{H}\big(\pi_\theta(\cdot|s_t)\big) \right] \tag{7}$$

where:

$$\begin{aligned} r_t(\theta) &= \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \\ A_t &= R_t - V_\theta(s_t) \\ \mathcal{H}\big(\pi_\theta(\cdot|s_t)\big) &= -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \end{aligned}$$

and β is the entropy coefficient (set to 0.01 in our main experiments).

For the advantage function, we use the standard formulation: the advantage is computed as the difference between the empirical return and the value estimate, $A_t = R_t - V_\theta(s_t)$.

The batch size of the update is limited at the 400 steps of one episode of training in this project.

4.1.1 Reward shaping

During the initial training runs with the previously described setup, I struggled significantly to achieve convergence. This was due to the fact that, with only sparse rewards, a model which initially performs random actions, it is almost impossible to execute the correct sequence of actions needed to obtain its first deliveries and therefore its first rewards. It quickly became clear that reward shaping was necessary during training. I therefore enabled the default reward shaping provided by the environment, which specifically rewards actions considered useful and sensible, such as filling the pot with onions or picking up a plate. These intermediate rewards proved to be essential for achieving convergence in this setup.

Reward shaping is especially helpful in the initial and intermediate phases of training, when the agent struggles to achieve the first sparse rewards. For this reason, it is necessary to modulate and gradually decrease the weight of these intermediate rewards over time. Ideally, in the final stages of training, once the model has learned how to obtain the sparse rewards, the intermediate rewards can even be dangerous, as they may encourage behaviors that are not always directly aimed at achieving the true goal (the sparse rewards).

For this reason, a stepwise modulation was adopted. Every a certain number of epochs, the multiplicative coefficient is decreased. Additionally, there is a linear decay factor from 1 in the first epoch to 0 in the final epoch, to make the modulation function slightly smoother.

4.2 Layout generalization

The main idea behind this project was to evaluate the agent's ability to generalize across multiple layouts. The experiment therefore consists of a new training session, analogous to the previous setup on `cramped_room`, where at each episode one of the three chosen training scenarios is randomly sampled.

The only architectural difference in the training setup was setting a much longer training horizon, 20,000 episodes and accordingly adjusting the reward shaping scaling. The ultimate goal was to evaluate the model's performance in greedy mode on all three training layouts, and subsequently on a previously unseen layout.

Another interesting experiment was to test the agent on a completely new and unseen layout, such as `coordination_ring`, to evaluate whether the model could successfully transfer its learned strategies to a novel scenario.

(a) *cramped_room* (b) *asymmetric_advantages* (c) *forced_coordination*

Figure 2: Training layouts

5 Results

The results obtained during training on the single layout were satisfactory: after about 6,500 episodes, the agent reached a greedy score of approximately 10 (200 points) deliveries per episode. A graphical analysis of the games clearly shows the emergence of a convincing cooperative strategy, with an effective division of roles and a routine of actions that is efficient and comparable to, or even superior to, that of skilled human players.

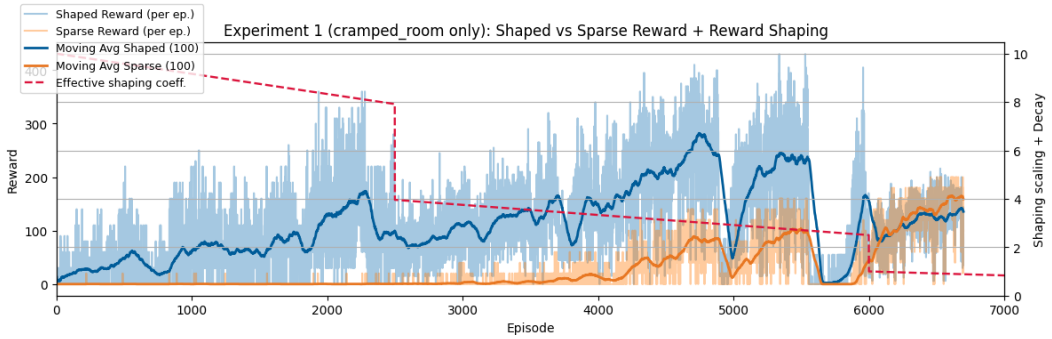

Figure 3: Experiment 1: Training log in *cramped_room*

As for the attempt to generalize across multiple layouts, the success was only partial. Training on the three selected layouts was stopped after approximately 17,000 episodes. The model was able to achieve good scores on `cramped_room` and `asymmetric_advantages`, but on the more challenging `forced_coordination` layout it was not possible to reach satisfactory results (only a few sparse rewars in sampling mode).

It was also observed that the agent's performance showed high variance throughout training: performance on the different layouts increased and decreased independently, and the overall metrics were rather noisy. I attribute this effect to the fact that I am using a single episode of 400 timesteps (sampled randomly from the three layouts) as the update batch. As a consequence, learning may suffer in terms of consistency and stability of the strategies acquired for the different layouts.

Analyzing the gameplay GIFs generated during evaluation, we observed that the agents developed an effective cooperative strategy in the `cramped_room` layout. However, in `asymmetric_advantages`, the learned policy was sub-optimal, as one of the agents did not move or cooperate for most of the episode. In the `forced_coordination` layout, we noticed a peculiar behavior: under greedy evaluation, the agents enter a deadlock situation early in the episode, while this issue did not occur when using stochastic (sampling-based) evaluation.

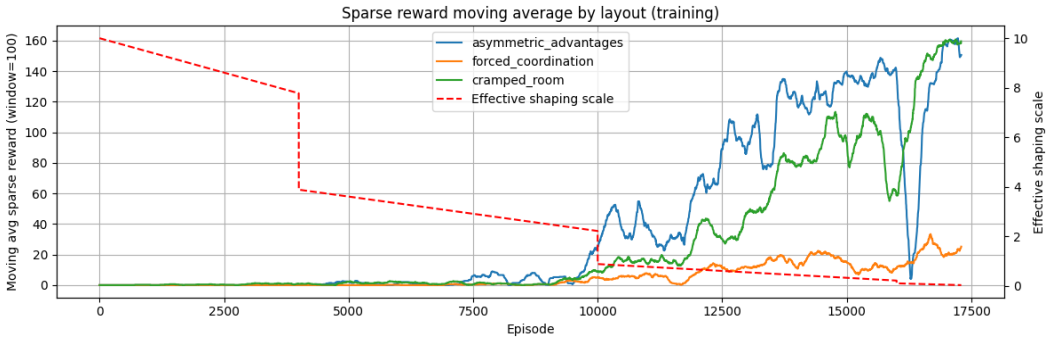

Figure 4: Experiment 2: Training log in multilayout learning

6 Conclusions

In this work, we have explored cooperative multi-agent reinforcement learning in the Overcooked-AI environment, focusing both on single-layout mastery and on the challenge of generalization across multiple kitchen layouts.

Our experiments confirm that reward shaping and entropy regularization are crucial components for enabling agents to discover cooperative behaviors in sparse-reward scenarios. When training on a single layout, PPO agents can reach high performance and develop convincing division-of-labor strategies, comparable to or even surpassing strong human play.

However, the results also highlight the limitations of straightforward self-play in multi-layout generalization. While the agent was able to achieve satisfactory performance on some layouts, its success did not extend uniformly to more challenging scenarios, such as `forced_coordination`. Furthermore, training in the multi-layout setting exhibited high variance and instability, with the agent's proficiency oscillating independently across different layouts. This suggests that the single-episode update scheme, where each update is based on an episode from a randomly selected layout—can make it harder for the agent to learn robust, consistent strategies that transfer across all environments.

A promising direction for future work is to experiment with larger batch sizes during training, which may help stabilize learning and reduce variance across different layouts. Additionally, incorporating Generalized Advantage Estimation (GAE) could further improve the stability and efficiency of policy updates, potentially leading to more robust generalization across diverse environments.

References

[1] Michael Carroll, Rohin Shah, Mark K Ho, Thomas L Griffiths, Sanjit A Seshia, Pieter Abbeel, and Anca D Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

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

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[5] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.