Overcooked environment for cooperative reinforcement learning strategy

Autonomous System Course Project

Gianluca Di Nenno

September 19, 2025

Abstract

This report presents a deep reinforcement learning approach to address the challenge of cooperative multi-agent control in complex environments. In particular the tests are made on the simulated kitchen domain of Overcooked-AI, a benchmark environment based on the video game Overcooked, where two agents must collaboratively prepare and deliver meals. Proximal Policy Optimization (PPO) has been investigated with a shared policy architecture and with one policy for each agent. At first, the training of the two agents has been made in a simple kitchen layout, cramped room, which led to pretty good results. After that, the focus has gone to enable agents to generalize their learned cooperative behaviors across diverse and previously unseen kitchen layouts. The results demonstrate that the trained agents successfully acquire sophisticated coordination strategies, achieving a decent average sparse reward across a set of more difficult layouts.

1 Introduction

In the context of agent-based AI, reinforcement learning (RL) provides a powerful paradigm for training agents to make decisions by interacting with an environment and receiving feedback in the form of rewards. While single-agent RL is already challenging, many real-world problems involve multiple agents that must coordinate and collaborate.

To address these challenges, function approximation and policy gradient methods are preferred over tabular methods, which become infeasible when the state space is large and continuous. In this work, proximal policy optimization (PPO), has been used, together with a value function approximator and Generalized Advantage Estimation (GAE). PPO stabilizes training by preventing too large policy updates, while the value function provides a baseline for estimating expected returns, reducing

variance in gradient updates. GAE also helps with the estimation of advantages, striking a balance between bias and variance, and enabling more efficient learning from limited rollouts.
Finally, to encourage generalization and improve cooperation, the agents were trained under slightly different strategies and with shaped rewards. Although this led to slower convergence, it promoted richer behaviors and more adaptive policies compared to training with sparse rewards alone.

2 System description

2.1 Overcooked-ai

This project is built upon the Open Source Overcooked-AI repository, which provides the implementation of the environment, including the logic of the game, the representation of the state, and the basic interactions of the agents, such as actions and rewards.
The goal of the game is to deliver soups as fast as possible by putting ingredients in a pot, waiting the ingredient to cook, pickup the soup and delivering it in a max total time.
Reward returned by the system for delivering a soup has a value of 20 while other shaped rewards are returned for each timestep in which a good action has been done.
Observation space is a full-observable MDP represented by a vector of 96 elements and the action space is composed of six different actions: right, left, up, down, interact, stay.

2.2 Kitchen layouts

Overcooked environment offers different kitchen layouts to test the algorithm. In this work the focus went at first in solving the easier layout 'cramped room', and then the focus went on training the agents to perform also on more difficult layouts.

2.3 Generalized-Overcooked

To solve the problem of generalization, a Generalized-Overcooked class has been implemented that simply collects a list of overcooked environments from a list of layouts name. Every time reset() function is called one of the environments in the list is returned either randomly or sequentially. This class also has a function to return a one-hot vector that represents the current layout in order to feed the neural network also with this information.

2.4 Deep learning framework

In order to handle the training process and the creation of the neural network for the policy and value function, Keras Tensorflow framework has been used.

3 Background

Reinforcement Learning (RL) involves an agent learning optimal actions in an environment to maximize cumulative rewards. Early RL algorithms like Q-learning and SARSA were successful in tabular settings but struggled with large or continuous state spaces. Deep RL integrates neural networks, enabling learning from complex inputs, in particular policy gradient methods directly parameterize the policy with a neural network and optimize it via gradient ascent on expected returns. These methods are particularly well-suited for multi-agent environments with discrete and structured action spaces, such as Overcooked

My implementation of PPO involves a value function approximator trained to minimize the mean squared error between the actual returns and its predictions:

$$L_V(\phi) = \left(R_t - V_\phi(s_t)\right)^2 \tag{1}$$

The actor's weights are updated with the ppo clip:

$$\theta \leftarrow \theta + \alpha_\pi \nabla_\theta L^{\text{PPO}}(\theta) \tag{2}$$

with:

$$L^{\text{PPO}}(\theta) = \min\left(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right) \tag{3}$$

With the advantage function being calculated with GAE :

$$A_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l\, \delta_{t+l} \tag{4}$$

4 Project workflow and methodology

The methodology followed an iterative approach, beginning with a training of the model on a simple layout and progressively advancing towards a generalized policy effective across multiple, unseen environments. The core algorithm used was Proximal Policy Optimization (PPO)

4.1 Model architecture

The agent's decisions are possible thanks to two neural networks: a Policy Network (Actor) which outputs a probability distribution over the actions, and a Value Network (Critic) that estimates the quality of a given state.

Both networks share a common feature extraction backbone consisting of three fully connected dense layers respectively made of 64, 128 and 64 units, with the first layer that acts as a feature extractor, the second layer, the largest, is responsible for learning higher level abstractions and relationships between the features extracted by the first layer and the third that refines the high-level abstractions into a more compact representation.

The two networks differs on the output, with the policy networks that terminates in two separate output heads, each with a softmax activation function, producing an action probability distribution for each agent, while the policy network terminates in a single linear output unit, predicting the scalar state-value.

Another difference is between the network used for training a single simple layout and the one for training a general layout with the second one that takes in addition a one-hot encoding vector of the current layout among the list of layouts, so that the networks have an additional information about what layout it's in.

4.2 Initial training on single layout

The initial phase focused on training the agents on the simplest layout, cramped room. The challenge in this environment is the sparse reward which is only given upon successful soup delivery. This makes the credit assignment problem particularly difficult, as it's unclear which of the hundreds of preceding actions were responsible for the final reward.
To address this, Generalized Advantage Estimation (GAE) was employed. GAE provides a good solution to the credit assignment problem by using the critic's value estimates to look many steps into the future. It calculates a better advantage signal for each action, indicating whether it led to a better or worse-than-expected outcome in the long run. This was crucial for making the agents learn the long sequence of sub-tasks required to successfully make and deliver a soup.

4.3 Generalization and Training Strategies

After achieving a good baseline, the focus shifted to training a single, generalized policy that could perform well across a diverse set of five layouts, including those

requiring intricate coordination. This introduced some challenges, which were addressed with the following strategies:

- **Curriculm learning** : Instead of training on all layouts at once, agents were trained first on the two simplest layouts (crampedroom and asymmetric advantages) until they achieved good performance. Then the weights from the first training stage were loaded, and training was resumed on a set of layouts that included the original two plus more complex ones. This process was repeated until the last stage that had all the five different layouts.
- **Reward shaping**: The sparse reward signal from soup delivery was augmented with an event-based reward function. By extracting the detailed end-of-episode game statistics, we could go back and apply rewards or penalties to the exact timesteps where key events occurred. Doing that rewards were given to good actions like 'useful-onion-pickup' or 'useful-dish-pickup' and penalties were given to timesteps where bad actions like 'useless-onion-potting' occurred.
- **Entropy bonus**: When agents find new difficult layout, it's common that they perform safe strategies and stop exploring.Using an entropy bonus the policy is encouraged to maintain a degree of randomness, preventing it from becoming too deterministic too early. This forces the agent to continue exploring the action space, making it more likely to discover a more coordinated strategies required for the harder layouts.

5 Experimental setup and results

To arrive at the final successful training methodology, a series of experiments were conducted. Each experiment utilized a different training script and strategy to isolate and address specific challenges in multi-agent learning and generalization. All experiments were built upon the PPO algorithm with GAE, but varied in their approach to policy structure and environment sampling.

5.1 Baseline(training_gpu.py)

The initial goal was to establish a performance baseline and validate the core model architecture. This experiment focused on training agents to master a single, controlled environment.
A single, shared policy was used for both agents, and training was conducted exclusively on the cramped room layout, training for 1000 episodes with 4 epochs per episode and a batch size of 128.

- Results: This experiment was successful. The agents learned to perform effectively in cramped room, confirming that the model architecture and GAE implementation were capable of solving the fundamental task in a simplified setting.

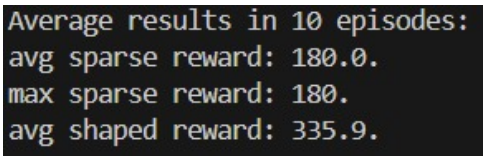

Figure 1: Results after evaluation of agents on cramped room

5.2 Standard Generalization (train_gen.py)

This experiment tested the model's ability to generalize by training on all five layouts simultaneously from scratch.
Since the high variance in data from five different environments, the number of training episodes was increased to 7000, and a larger batch size of 256 was used. A shared policy was maintained.
This approach proved to be ineffective. The agents failed to learn a successful strategy for any of the layouts. The conflicting demands of the different environments prevented the single policy from learning a generalizable skill set.

5.3 Different policies (train_two_agents.py)

In this script i tried to investigate if encouraging role specialization could lead to better cooperative strategies. In this experiment two separate actor networks were initialized, one for each agent, while maintaining a single shared critic to provide a consistent value estimate. Training was attempted across all layouts.
This strategy also resulted in poor performance. The two independent policies struggled to co-adapt in a meaningful way without a structured learning process. The agents failed to discover the complementary behaviors needed for coordination. This experiment suggests that it requires a more guided training approach, such as curriculum learning, to be more effective.

5.4 Curriculum Learning (training_incr.py)

This was the most successful approach. This setup implemented tall the strategies described in the methodology. It began by training a shared policy on the simplest layouts, then progressively introduced more difficult ones in successive training

stages. Crucially, it always used the previously mastered layouts in the training set to prevent catastrophic forgetting. This particular experiment was augmented with the entropy bonus and the event-based reward shaping to accelerate learning on the more challenging coordination-based layouts, together with an augmentation of the batch size and reducing the learning rate as more difficult layouts were introduced.

- Results: This was the most successful experimental setup . The agents not only mastered the initial layouts but were also able to decently transfer and adapt their skills to the harder environments. Unfortunately the performace decreased as the difficulty improved with the hardest layout "counter_circuit" performing really bad.

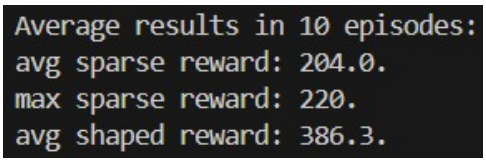

Figure 2: Results after evaluation of agents on cramped room

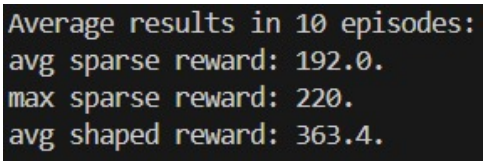

Figure 3: Results after evaluation of agents on asymmetric advantages

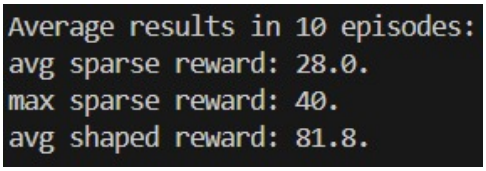

Figure 4: Results after evaluation of agents on coordination ring

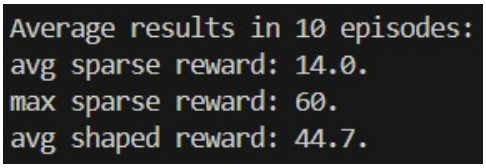

Figure 5: Results after evaluation of agents on forced coordination

6 Conclusion

The experimental results demonstrate that a structured curriculum is the most effective strategy for training cooperative agents in complex, multi-layout environments like Overcooked. By combining new unseen layouts with simpler layouts to prevent catastrophic forgetting with dense, event-based reward shaping and an entropy bonus for exploration, the developed agents successfully learned to generalize their performance from simple to moderately complex layouts.
However, the current shared-policy approach still faces limitations on the most difficult layouts, such as counter_circuit, where good performance requires a high role specialization and more strategic coordination.

Future work should therefore explore more advanced multi-agent training experiments. Implementing asymmetric self-play with separate policies for each agent would be a promising next step to encourage the emergence of specialized roles.