Partner and Layout Generalization in Overcooked via Multi-Agent Reinforcement Learning

Cosimo E. Russo
Computer Science and Engineering - DISI
Università di Bologna
cosimoemanuele.russo@studio.unibo.it

Abstract

Generalization in cooperative multi-agent reinforcement learning remains a critical challenge, particularly when agents must adapt to novel partners and unseen environments. In this work, I present a purely deep RL framework for the Overcooked benchmark which is the result of a systematic ablation study over key architectural and algorithmic components - such as feature representations, and training schedules and underlying neural architecture - to identify the combinations that most significantly accelerate convergence and improve zero-shot generalization.

1 Introduction

Overcooked-AI has become the go-to benchmark for cooperative multi-agent reinforcement learning because it packs the key challenges of real-world teamwork—task allocation, spatial coordination, communication, and zero-shot partner generalisation into a fast, grid-world simulator that runs thousands of episodes per second. Its simple graphics keep computation cheap, yet the need for dynamic collaboration makes it far more discriminative than classic environments. Furthermore, the Overcooked environment anchors public challenges that define state-of-the-art cooperative performance.

This work investigates how neural-network architecture and feature representation influence both final performance and convergence speed, and examines ways to mitigate two key training challenges—high variance and sample inefficiency.

My approach builds on **Proximal Policy Optimization**(**PPO**)[1]: at every timestep the environment supplies a structured feature vector that is used to feed both policy (actor) and value (critic) networks. PPO's clipped surrogate objective allows to train the policy network with short, truncated trajectories and minibatch SGD, avoiding the high-variance full-episode Monte-Carlo updates that a purely returns-based method would require.

The best approach generalises smoothly to new partners, handling both naïve "dummy" agents and high-skill experts. Yet, environment generalisation proves harder: only the more refined variants deliver encouraging gains, with the best model achieving [future data] on unseen layouts.

The remainder of this paper is organized as follows. In Section 2, we formalize the Overcooked MDP and define our partner/layout generalization evaluation protocol. Section 3 describes the two network architectures, the input features, and the RL training procedures. Finally, Section **??** presents empirical results and a detailed comparison of the approaches.

Block	#Dims	Feature groups (with dimensionality)
Self–centric	46	Orientation (4); Held object (4); Distances to six classes (12); Nearest-soup composition (2); Two reachable pots ($2\times10 = 20$); Adjacent walls (4)
Teammate (same template)	46	Identical 46-dimensional encoding computed for the teammate
Relative position	2	$(\Delta x,\ \Delta y)$ of teammate relative to self
Absolute position	2	Player's grid coordinates $(x,\ y)$
Total	**96**	

Table 1: Breakdown of the 96-dimensional feature vector returned by `featurize_state` in OVERCOOKED-AI.

2 Problem Formalization

2.1 State Representation in Overcooked

In the Overcooked environment, one *could* treat each state $s \in S$ as a raw pixel-matrix, similar to how DQN was applied to Atari games [2]. However, the overcooked environment supplies a compact, featurized vector that can be used as the state input. Concretely, this vector encodes the informations described in Table 1. Two natural network back-ends are possible: **(i)** a feedforward neural network that consumes the flat 1-D vector, and **(ii)** a convolutional model that reshapes it into a multi-channel 3-D grid, where each binary channel flags the spatial presence of a specific feature.

By using this structured, symbolic representation, the networks can focus on high-level task information rather than low-level pixel observations.

Each agent's observation already embeds its partner's state, so the policy network doesn't need a separate partner input, which would also force us to tag the acting role (player 1 vs. player 2) and would bloat computation over time. Instead, we supply partner information only to the critic, because it evaluates the joint state and it naturally captures collaboration without any extra role-specific encoding.

2.2 Generalization Problem

I evaluate generalization along two axes: *partner generalization* and *layout generalization*.

Partner generalization. To test how well a trained agent cooperates with unseen partners, I fix a single kitchen layout and maintain an *evaluation pool* of pre-trained policies. Specifically: **(i)** During training, every n epochs a snapshot of the current policy is saved and added to the evaluation pool. These policy configurations are never used during training. **(ii)** At evaluation time, the PPO agent is paired with several older PPO-policy snapshots from the evaluation pool (i.e., partners with different weights), and an unseen "greedy" human-model agent (which follows a hand-designed, greedy heuristic).**(iii)** We measure the average joint reward achieved over a set of fixed episodes with each partner. This quantifies how well the agent generalizes to partners whose behavior was never encountered during training.

Layout generalization. To evaluate how well an agent generalizes to unseen kitchen layouts, we proceed as follows: **(i)** train the PPO agent on a specific set of layouts using self-play; **(ii)** At test time, place the trained agent back into novel layouts that it never saw during training. We let it self-play (i.e., both players use the same trained policy) and record the average joint reward. **(iii)** Assume the set of ingredients, objects, and basic game rules remain unchanged across these new layouts, only the spatial arrangement of counters, ovens, and delivery stations differs.

By comparing the average joint rewards in these two evaluation protocols, I can isolate the agent's ability to generalize to new partners versus new environments. The generalization of both is left as future work.

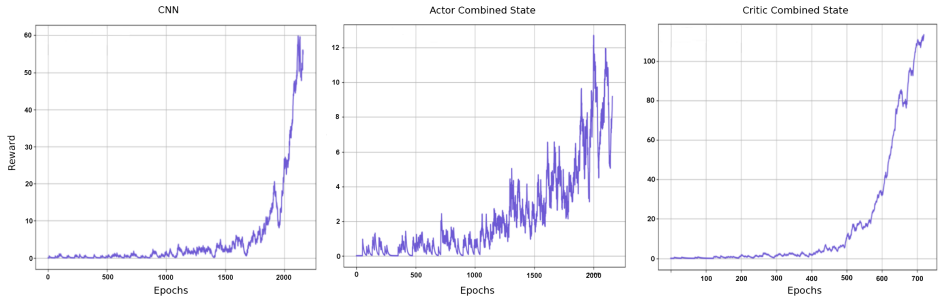


Figure 1: Architectures comparison.

3 Exploited Architectures

The evaluated architectures split into two categories based on the input encoding: feed-forward networks that ingest the flat 1-D feature vector, and convolutional networks that process the spatial tensor. Early experiments revealed that the CNNs were both slower and less accurate, so the study centres on the feed-forward variants.

3.1 Architectural Details

All four agent variants use one of two compact networks ($\approx$ 30k parameters), a size chosen empirically as a sweet-spot between capacity and overfitting.

1. **Feed-Forward Policy.** A flat observation vector enters two fully–connected layers of 128 and 64 units with ReLu activation function, followed by a six-way soft-max that matches the Overcooked action set.
2. **CNN Policy.** The lossless state encoding of 26-channel provided by the environment is zero-padded to the maximum layout spatial dimension, processed by two 3×3 convolutions (32 and 64 filters, stride 1, *same* padding, ReLU), globally averaged, passed through a 128-unit dense layer, and finished with the same six-way soft-max head as in 1).

For each policy back-bone I build a matching critic that keeps the *identical* feature extractor (FF or CNN) but swaps the soft-max head for a single linear unit, yielding a scalar state-value estimate $\bar{v}(s)$.

The only thing that changes across variants is what we feed into the networks.

Feed-forward (FF) backbone. Try three useful setups: **(i)** actor gets the joint observation, critic gets its own; **(ii)** critic gets the joint observation, actor gets its own; **(iii)** both use their individual observation;

Giving both actor and critic the joint observation was dropped early—extra features made the actor stumble and slowed learning.

CNN backbone. Stick to a single layout where actor and critic each process one 26-channel tensor since even this version proves slower and no more accurate than the FF alternatives. Doubling the channels to build a joint-observation CNN would only increase computations without any clear upside.

3.2 Comparison

Figure 1 compares learning curves for the three architectural variants:

1. **Joint-observation policy.** Feeding the policy the concatenated observations of both agents yields the noisiest curve and the poorest final return. The concatenation introduces redundant features and forces the network to decode its own role from a prefix tag, adding unnecessary complexity.

2. **CNN variant.** Although the convolutional model climbs more smoothly, it trains far slower: every layout must be zero-padded to the maximum grid size, inflating the input tensor and lengthening each update. Given the long convergence time, I drop this variant from further study.
3. **Feed-Forward with joint-state critic.** Supplying the policy with the single-agent feature vector and the critic with the joint state produces the fastest and most stable convergence. The flat 96-dimensional vector evidently holds enough information for action selection, while the critic benefits from partner data without the role-identification burden.

These observations motivate our focus on the third configuration in the subsequent experiments.

4 Training Procedure

The training loop follows a four-step cycle for each epoch: Select a partner, pick a layout, roll out a trajectory with the current policy–partner pair and update actor and critic by running minibatch SGD on the collected rollout.

4.1 Partner Selection

At the start of every epoch I pick the teammate for the learning agent. There are two possibilities:

- **Self-play.** The agent is paired with a clone of itself, but I keep training decentralised: only the learning agent's action and return feed into the PPO update. This single-objective update proved noticeably more stable than updating two agents at once.
- **Partner-generalisation.** When we want robustness to new partners, we sample from a policy pool: every n epochs the current network is snap-shotted and added to the pool (the same trick DeepMind used in the original AlphaGo [3]). Drawing partners from this growing set gives a steady supply of diverse behaviours and boosts zero-shot coordination.

4.2 Layout Selection

To achieve layout generalisation, the agent must be trained on three progressively harder scenarios: *CrampedRoom* (to master the game's basic mechanics), *AsymmetricAdvantages* (to develop cooperative behaviour), and *CoordinationRing* (to learn path-sharing in tight spaces without blocking the partner). A naïve regime that picks a random layout each epoch fails, because the agent cannot coordinate before it even understands how to play. Instead, a curriculum is required: start with *CrampedRoom* alone; once the agent exceeds a tunable score threshold, add *AsymmetricAdvantages* to the pool and adjust the sampling probabilities; after the next threshold, introduce *CoordinationRing* with another probability shift. This resampling step limits catastrophic forgetting by ensuring previously seen layouts remain in the mix. For the curriculum to work, three conditions proved essential:

- **Adaptive sampling**: update the layout distribution whenever a new map is introduced;
- **Per-layout reward normalisation**: scale returns by statistics of the current layout;
- **Shaped rewards**: retain dense shaping until the agent consistently earns sparse rewards on the new map.

When the final threshold on *CoordinationRing* is met, the sampling is reset to uniform and the model is fine-tuned.

4.3 Reward Definition

Delivering a full three-ingredient dish by pure luck—and thus with no reward signal—is highly improbable, so I employ reward shaping. Overcooked supplies a dense "shaped" score for partial actions; I use this as a stepping-stone to accelerate learning and prevent the policy from stalling. At every timestep the training reward is the larger of the learner's shaped reward and the joint sparse reward earned by both agents. Because distributed training limits me to the learner's own

shaped signal, I rely on the partner's sparse return to foster cooperation: the learner should act to boost the team's reward even when its own shaped payoff is low, which is essential on layouts like *CoordinationRing* and *AsymmetricAdvantages*.

The reward function is:

$$R = \max(r_{sparse}, w_r \cdot r^{(i)}_{shaped}) \tag{1}$$

Where: r_{sparse} is the teams's cumulative sparse reward, $r^{(i)}_{shaped}$ is the learner's shaped reward and w_r weights the shaping term.

Whenever the team reaches 20 sparse points (one full order), a decay factor r_{decay} is subtracted from w_r. It then declines linearly with each epoch, letting the true sparse reward take over once regular dish delivery is achieved.

Return scaling has to be layout-specific. Without it, the large rewards already common on well-practised maps generate huge critic losses that swamp the modest gains earned on a newly added layout. To prevent this, I track each layout's return mean and variance on every rollout and use those running statistics to normalise the returns whenever that layout is sampled.

4.4 Actor-Critic Updates

Each training epoch consists of four episodes, and every episode is capped at 400 time-steps. The 400-step horizon is long enough for an agent to complete substantial work but short enough to abort unproductive runs (e.g., when an agent wanders around clutching an onion it cannot place) so it strikes a balance between meaningful experience and wasted wall-clock time.

Updates are carried out with stochastic gradient descent using a minibatch of 256 trajectories. Although 256 is large, it is necessary because rewards are extremely sparse: a return is registered only in the instant a dish is delivered. Smaller batches would too often contain no positive signal at all, undermining the update.

For the policy (actor) I employ Proximal Policy Optimization with Generalised Advantage Estimation (GAE) to reduce update variance, and I add an entropy bonus to the objective so the policy continues to explore rather than collapsing prematurely. The value function (critic) is trained with mean-squared error against returns that have been normalised on a per-layout basis.

Before applying any parameter update, both actor and critic gradients are clipped by their global L2 norm. Global-norm clipping dampens occasional gradient spikes that can arise when advantages are large, preventing any single layer with huge activations from dominating the update, and keeping the effective step size roughly consistent across batches, altogether improving the stability of PPO's optimisation process.

5 Results

Figure 2 plots the learning curves when layout-specific returns are not normalised. As soon as the agent starts earning large rewards on *CoordinationRing*, its score on *AsymmetricAdvantages* collapses—a clear case of catastrophic forgetting. The likely cause is scale mismatch: the higher raw returns from *CoordinationRing* generate larger advantages and, in turn, larger policy and value-network gradients; these outsized updates shift shared weights toward the new layout and overwrite features that used to support *AsymmetricAdvantages*.

Figure 3 shows the effect of per-layout return normalisation. With each layout's returns rescaled to zero mean and unit variance, gradient magnitudes stay comparable across tasks, the learning curve becomes noticeably smoother, and no forgetting is observed. In fact, *CoordinationRing* continues to improve beyond the peak reached in the unnormalised run, demonstrating that the stabilised updates help rather than hinder optimisation.

Figure 4a ranks performance against partners of increasing skill—dummy on the far left, expert on the far right. The gap is obvious: dummy partners can't manage even the basic step of dropping an onion into the pot, yet our agent still posts a solid score, highlighting its strong generalisation.

Finally, Figure 4b plots the rewards from ten runs per layout, showing consistently low episode-to-episode variance, a strong overall score, and headroom for further gains.

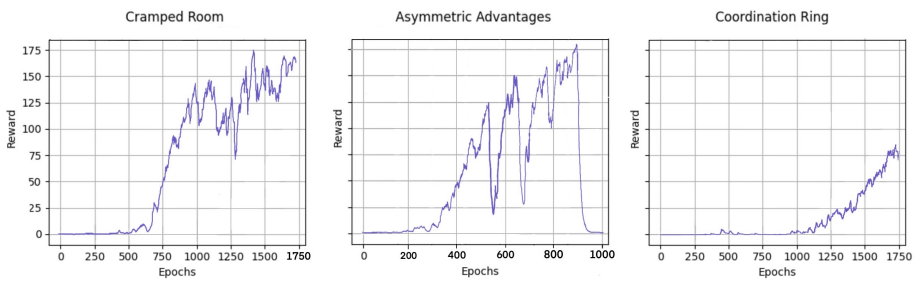


Figure 2: Reward signal on the three Overcooked layouts without per-layout return normalisation.

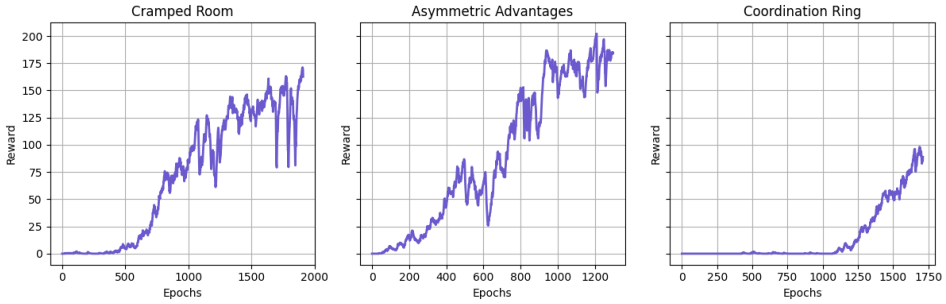
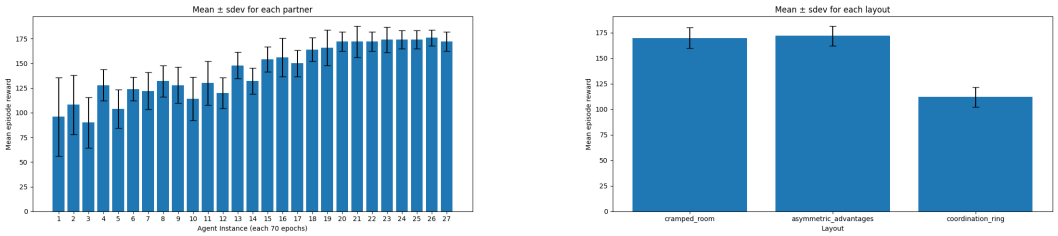


Figure 3: Reward signal on the three Overcooked layouts after per-layout return normalisation.

6 Conclusions

The agent does not yet generalise to completely unseen layouts, but it adapts quickly to layouts that are merely new to its experience—a promising sign that broader environment coverage is within reach. Extending training to tougher maps such as *ForcedCoordination* and *CounterCircuit* is an immediate next step. Within the layouts it saw during training, the agent has converged on near-optimal tactics, maximising reward consistently.

Future work falls into three directions: **Batch-level layout mixing**: Injecting multiple layouts and different agent configurations into each update batch could curb catastrophic forgetting and speed convergence. **Joint partner-and-layout generalisation**: Training for both axes simultaneously is appealing but computationally heavy, because pairing with low-skill partners slows learning; smarter sampling or curriculum strategies will be needed. **Richer environment set**: Adding the aforementioned expert-level layouts would push the agent's adaptability further and serve as a stronger test of its coordination skills.



(a) Partner Generalization.

(b) Layout Test.

Figure 4: (a) Generalization results across different partners; (b) Test results across different layouts.

References

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

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. In *Proceedings of the Deep Learning Workshop at Neural Information Processing Systems*, 2013.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, and Julian Schrittwieser. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.