Learning to Incentivize Other Learning Agents

Jiachen Yang[*]
Georgia Institute of Technology
jiachen.yang@gatech.edu

Ang Li
DeepMind
anglili@google.com

Mehrdad Farajtabar
DeepMind
farajtabar@google.com

Peter Sunehag
DeepMind
sunehag@google.com

Edward Hughes
DeepMind
edwardhughes@google.com

Hongyuan Zha
Georgia Institute of Technology
zha@cc.gatech.edu

Abstract

The challenge of developing powerful and general Reinforcement Learning (RL) agents has received increasing attention in recent years. Much of this effort has focused on the single-agent setting, in which an agent maximizes a predefined extrinsic reward function. However, a long-term question inevitably arises: how will such independent agents cooperate when they are continually learning and acting in a shared multi-agent environment? Observing that humans often provide incentives to influence others' behavior, we propose to equip each RL agent in a multi-agent environment with the ability to give rewards directly to other agents, using a learned incentive function. Each agent learns its own incentive function by explicitly accounting for its impact on the learning of recipients and, through them, the impact on its own extrinsic objective. We demonstrate in experiments that such agents significantly outperform standard RL and opponent-shaping agents in challenging general-sum Markov games, often by finding a near-optimal division of labor. Our work points toward more opportunities and challenges along the path to ensure the common good in a multi-agent future.

1 Introduction

Reinforcement Learning (RL) [34] agents are achieving increasing success on an expanding set of tasks [25, 18, 29, 38, 5]. While much effort is devoted to single-agent environments and fully-cooperative games, there is a possible future in which large numbers of RL agents with imperfectly-aligned objectives must interact and continually learn in a shared multi-agent environment. Excluding the option of centralized training with a global reward, which faces difficulty in scaling to large populations and may not be adopted by self-interested parties, it is difficult for groups of agents to achieve high individual and collective return [26]. Moreover, agents in many real world situations with mixed motives, such as settings with nonexcludable and subtractive common-pool resources, may face a social dilemma wherein mutual selfish behavior leads to low individual and total utility, due to fear of being exploited or greed to exploit others [27, 20, 21]. Whether, and how, independent learning and acting agents can cooperate while optimizing their own objectives is an open question.

The conundrum of attaining multi-agent cooperation with decentralized[2] training of agents, who may have misaligned individual objectives, requires us to go beyond the restrictive mindset that the collection of predefined individual rewards cannot be changed by the agents themselves. We draw inspiration from the observation that this fundamental multi-agent problem arises at multiple

[*]Work done during internship at DeepMind.

[2]We take decentralized to mean that no agent is designed with an objective to maximize collective performance, and that agents optimize separate sets of policy parameters.

Preprint. Under review.

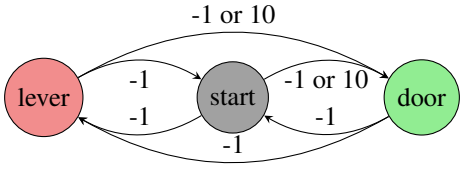


Figure 1: The N-player *Escape Room* game ER(N, M). For $M < N$, if fewer than M agents pull the lever, which incurs a cost of -1, then all agents receive -1 for changing positions. Otherwise, the agent(s) who is not pulling the lever can get $+10$ at the door and end the episode.

scales of human activity and, crucially, that it can be successfully resolved when agents give the right incentives to *alter* the objective of other agents, in such a way that the recipients' behavior changes for everyone's advantage. Indeed, a significant amount of individual, group, and international effort is expended on creating effective incentives or sanctions to shape the behavior of other individuals, social groups, and nations [36, 7, 8]. The rich body of work on game-theoretic side payments [17, 15, 13] further attests to the importance of inter-agent incentivization in society.

Translated to the framework of Markov games for multi-agent reinforcement learning (MARL) [23], the key insight is to remove the constraints of an immutable reward function. Instead, we allow agents to *learn* an incentive function that gives rewards to other learning agents and thereby shape their behavior. The new learning problem for an agent becomes two-fold: learn a policy that optimizes the total extrinsic rewards and incentives it receives, and learn an incentive function that alters other agents' behavior so as to optimize its own extrinsic objective. While the emergence of incentives in nature may have an evolutionary explanation [14], human societies contain ubiquitous examples of learned incentivization and we focus on the learning viewpoint in this work.

The Escape Room game. We may illustrate the benefits and necessity of incentivization with a simple example. The *Escape Room* game ER(N, M) is a discrete N-player Markov game with individual extrinsic rewards and parameter $M < N$, as shown in Figure 1. An agent gets +10 extrinsic reward for exiting a door and ending the game, but the door can only be opened when M other agents cooperate to pull the lever. However, an extrinsic penalty of -1 for any movement discourages all agents from taking the cooperative action. If agents optimize their own rewards with standard independent RL, no agent can attain positive reward, as we show in Section 5.

This game may be solved by equipping agents with the ability to incentivize other agents to pull the lever. However, we hypothesize—and confirm in experiments—that merely augmenting an agent's action space with a "give-reward" action and applying standard RL faces significant learning difficulties. Consider the case of ER$(2, 1)$: suppose we allow agent A1 an additional action that sends +2 reward to agent A2, and let it observe A2's chosen action prior to taking its own action. Assuming that A2 conducts sufficient exploration, an intelligent reward-giver should learn to use the give-reward action to incentivize A2 to pull the lever. However, RL optimizes the expected cumulative reward within *one* episode, but the effect of a give-reward action manifests in the recipient's behavior only after many learning updates that generally span *multiple* episodes. Hence, a reward-giver may not receive any feedback within an episode, much less an immediate feedback, on whether the give-reward action benefited its own extrinsic objective. Instead, we need an agent that explicitly accounts for the impact of incentives on the recipient's learning and, thereby, on its own future performance.

As a first step toward addressing these new challenges, we make the following conceptual, algorithmic, and experimental contributions. (1) We create an agent that learns an incentive function to reward other learning agents, by explicitly accounting for the impact of incentives on its own performance, through the learning of recipients. (2) Working with agents who conduct policy optimization, we derive the gradient of an agent's extrinsic objective with respect to the parameters of its incentive function. We propose an effective training procedure based on online cross-validation to update the incentive function and policy on the same time scale. (3) We show convergence to mutual cooperation in a matrix game, and experiment on a new deceptively simple *Escape Room* game, which poses significant difficulties for standard RL and action-based opponent-shaping agents, but on which our agent consistently attains the global optimum. (4) Finally, our agents discover near-optimal division of labor in the challenging and high-dimensional social dilemma problem of *Cleanup* [16]. Taken together, we believe this is a promising step toward a cooperative multi-agent future.

2 Related works

Learning to incentivize other learning agents is motivated by the problem of cooperation among independent learning agents in intertemporal social dilemmas (ISDs) [20], in which defection is preferable to individuals in the short term but mutual defection leads to low collective performance in the long term. Algorithms for fully-cooperative MARL [12, 28, 32] may not be applied as ISDs have mixed motives and cannot canonically be reduced to fully cooperative problems. Previous work showed that collective performance can be improved by independent agents with *intrinsic* rewards [9, 16, 39], which are either hand-crafted or slowly evolved and modulate each agent's own extrinsic reward. In contrast, a reward-giver's incentive function in our work is *learned* on the same timescale as policy learning and is given to, and maximized by, *other* agents. Empirical research shows that augmenting an agent's action space with a "give-reward" *action* can improve cooperation during certain training phases in ISDs [24].

Learning to incentivize is a form of opponent shaping, whereby an agent learns to influence the learning update of other agents for its own benefit. While LOLA [11] and SOS [22] exert influence via actions taken by its policy, whose effects manifest through the Markov game state transition, our proposed agent exerts direct influence via an incentive function, which is distinct from its policy and which explicitly affects the recipient agent's learning update. Hence the need to influence other agents does not restrict a reward-giver's policy, potentially allowing for more flexible and stable shaping. We describe the mathematical differences between our method and LOLA in Section 3.1, and experimentally compare with LOLA agents augmented with reward-giving actions.

Our work is related to a growing collection of work on modifying or learning a reward function that is in turn maximized by another learning algorithm [4, 31, 41]. Previous work investigate the evolution of the prisoner's dilemma payoff matrix when altered by a "mutant" player who gives a fixed incentive for opponent cooperation [2]; employ a centralized operator on utilities in 2-player games with side payments [31]; and directly optimize collective performance by centralized rewarding in 2-player matrix games [4]. In contrast, we work with N-player Markov games with self-interested agents who must individually learn to incentivize other agents and cannot optimize collective performance directly. Our technical approach is inspired by online cross validation [33], which is used to optimize hyperparameters in meta-gradient RL [40], and by the optimal reward framework [30], in which a single agent learns an intrinsic reward by ascending the gradient of its own extrinsic objective [41].

3 Learning to incentivize others

We design Learning to Incentivize Others (LIO), an agent that learns an incentive function by explicitly accounting for its impact on recipients' behavior, and through them, the impact on its own extrinsic objective. For clarity, we describe the ideal case where agents have a perfect model of other agents' parameters and gradients; afterwards, we remove this assumption via opponent modeling. We present the general case of N LIO agents, indexed by $i \in [N] := \{1, \dots, N\}$. Each agent gives rewards using its incentive function and learns a regular policy with all received rewards. For clarity, we use index i when referring to the reward-giving part of an agent, and we use j for the part that learns from received rewards. For each agent i, let $o^i := O^i(s) \in \mathcal{O}$ denote its individual observation at global state s; $a^i \in \mathcal{A}^i$ its action; and $-i$ a collection of all indices except i. Let $\mathbf{a}$ and $\boldsymbol{\pi}$ denote the joint action and the joint policy over all agents, respectively.

A reward-giver agent i learns a vector-valued incentive function $r_{\eta^i} : \mathcal{O} \times \mathcal{A}^{-i} \mapsto \mathbb{R}^{N-1}$, parameterized by $\eta^i \in \mathbb{R}^n$, that maps its own observation o^i and all other agents' actions a^{-i} to a vector of rewards for the other $N-1$ agents[3]. Let $r^j_{\eta^i}$ denote the reward that agent i gives to agent j. As we elaborate below, r_{η^i} is separate from the agent's conventional policy and is learned via direct gradient ascent on the agent's own extrinsic objective, involving its effect on all other agents' policies, instead of via RL. Hence, while one may interpret LIO to have an augmented action space with an additional channel of influence on other agents, LIO's learning approach is fundamentally different from a standard RL agent who has a "give-reward" action.

[3] We do not allow LIO to reward itself, as our focus is on influencing *other* agents' behavior. Nonetheless, LIO may be complemented by other methods for learning *intrinsic* rewards [41].

We build on the idea of online cross-validation [33], to capture the fact that an incentive has measurable effect only after a recipient's learning step. As such, we describe LIO in a procedural manner below (Algorithm 1). This procedure can also be viewed as an iterative method for a bilevel optimization problem [6], where the upper level optimizes the incentive function by accounting for recipients' policy optimization at the lower level. At each time step t, each recipient j receives a total reward

$$r^j(s_t, \mathbf{a}_t, \eta^{-j}) := r^{j,\text{env}}(s_t, \mathbf{a}_t) + \sum_{i \neq j} r^j_{\eta^i}(o^i_t, a^{-i}_t)\,, \tag{1}$$

where $r^{j,\text{env}}$ denotes agent j's extrinsic reward. Each agent j learns a standard policy π^j, parameterized by $\theta^j \in \mathbb{R}^m$, to maximize the objective

$$\max_{\theta^j} J^{\text{policy}}(\theta^j, \eta^{-j}) := \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{T} \gamma^t r^j(s_t, \mathbf{a}_t, \eta^{-j})\right]\,. \tag{2}$$

Upon experiencing a trajectory $\tau^j := (s_0, \mathbf{a}_0, r^j_0, \ldots, s_T)$, the recipient carries out an update

$$\hat{\theta}^j \leftarrow \theta^j + \beta f(\tau^j, \theta^j, \eta^{-j}) \tag{3}$$

that adjusts its policy parameters with learning rate β (Algorithm 1, lines 4-5). Assuming policy optimization learners in this work and choosing policy gradient for exposition, the update function is

$$f(\tau^j, \theta^j, \eta^{-j}) = \sum_{t=0}^{T} \nabla_{\theta^j} \log \pi^j(a^j_t|o^j_t) G^j_t(\tau^j; \eta^{-j})\,, \tag{4}$$

where the return $G^j_t(\tau^j, \eta^{-j}) = \sum_{l=t}^{T} \gamma^{l-t} r^j(s_l, \mathbf{a}_l, \eta^{-j})$ depends on incentive parameters η^{-j}.

After each agent has updated its policy to $\hat{\pi}^j$, parameterized by new $\hat{\theta}^j$, it generates a new trajectory $\hat{\tau}^j$. Using these trajectories, each reward-giver i updates its individual incentive function parameters η^i to maximize the following individual objective (Algorithm 1, lines 6-7):

$$\max_{\eta^i} J^i(\hat{\tau}^i, \tau^i, \hat{\boldsymbol{\theta}}, \eta^i) := \mathbb{E}_{\hat{\boldsymbol{\pi}}}\left[\sum_{t=0}^{T} \gamma^t \hat{r}^{i,\text{env}}_t\right] - \alpha L(\eta^i, \tau^i)\,. \tag{5}$$

The first term is the expected extrinsic return of the reward-giver in the new trajectory $\hat{\tau}^i$. It implements the idea that the purpose of agent i's incentive function is to alter other agents' behavior so as to maximize its extrinsic rewards. The rewards it received from others are already accounted by its own policy update. The second term is a cost for giving rewards in the first trajectory τ^i:

$$L(\eta^i, \tau^i) := \sum_{(o^i_t, a^{-i}_t) \in \tau^i} \gamma^t \|r_{\eta^i}(o^i_t, a^{-i}_t)\|_1\,. \tag{6}$$

This cost is incurred by the incentive function and not by the policy, since the latter does not determine incentivization[4] and should not be penalized for the incentive function's behavior (see Appendix A.1 for more discussion). We use the ℓ_1-norm so that cost has the same physical "units" as extrinsic rewards. The gradient of (6) is directly available, assuming r_{η^i} is a known function approximator (e.g., neural network). Letting $J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}})$ denote the first term in (5), the gradient w.r.t. η^i is:

$$\nabla_{\eta^i} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) = \sum_{j \neq i} (\nabla_{\eta^i} \hat{\theta}^j)^T \nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}})\,. \tag{7}$$

The first factor of each term in the summation follows directly from (3) and (4):

$$\nabla_{\eta^i} \hat{\theta}^j = \beta \sum_{t=0}^{T} \nabla_{\theta^j} \log \pi^j(a^j_t|o^j_t) \left(\nabla_{\eta^i} G^j_t(\tau^j; \eta^{-j})\right)^T\,. \tag{8}$$

Note that (3) does not contain recursive dependence of θ^j on η^i since θ^j is a function of incentives in *previous* episodes, not those in trajectory τ^i. The second factor in (7) can be derived as

$$\nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) = \mathbb{E}_{\hat{\boldsymbol{\pi}}}\left[\nabla_{\hat{\theta}^j} \log \hat{\pi}^j(\hat{a}^j|\hat{o}^j) Q^{i,\hat{\boldsymbol{\pi}}}(\hat{s}, \hat{\mathbf{a}})\right]\,. \tag{9}$$

[4]Note that the outputs of the incentive function and policy are conditionally independent given the agent's observation, but their separate learning processes are coupled via the learning process of other agents.

Algorithm 1 Learning to Incentivize Others

procedure TRAIN LIO AGENTS
  Initialize all agents' policy parameters θ^i, incentive function parameters η^i
  for each iteration **do**
    Generate a trajectory $\{\tau^j\}$ using $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$
    For all reward-recipients j, update $\hat{\theta}^j$ using (3)
    Generate a new trajectory $\{\hat{\tau}^i\}$ using new $\hat{\boldsymbol{\theta}}$
    For reward-givers i, compute new $\hat{\eta}^i$ by gradient ascent on (5)
    $\theta^i \leftarrow \hat{\theta}^i$, $\eta^i \leftarrow \hat{\eta}^i$ for all $i \in [N]$.
  end for
end procedure

In practice, to avoid manually computing the matrix-vector product in (7), one can define the loss

$$\text{Loss}(\eta^i, \hat{\tau}^i) := -\sum_{j \neq i} \sum_{t=0}^{T} \log \pi_{\hat{\theta}^j}(\hat{a}_t^j | \hat{o}_t^j) \sum_{l=t}^{T} \gamma^{l-t} r^{i,\text{env}}(\hat{s}_l, \hat{\mathbf{a}}_l)\,, \tag{10}$$

and directly minimize it via automatic differentiation [1]. Crucially, $\hat{\theta}^j$ must preserve the functional dependence of the policy update step (4) on η^i within the same computation graph. Derivations of (9) and (10) are similar to that for policy gradients [35] and are provided in Appendix C.

LIO is compatible with the goal of achieving emergent cooperation in fully-decentralized MARL, as agents already learn individual sets of parameters to maximize individual objectives. One may directly apply opponent modeling [3] when LIO can observe, or estimate, other agents' egocentric observations, actions, and individual rewards, and have common knowledge that all agents conduct policy updates via reinforcement learning. These requirements are satisfied in environments where incentivization itself is feasible, since these observations are required for rational incentivization. LIO may then fit a behavior model for each opponent, create an internal model of other agents' RL processes, and learn the incentive function by differentiating through fictitious updates using the model in place of (3). We demonstrate a fully-decentralized implementation in our experiments.

3.1 Relation to opponent shaping via actions

LIO conducts opponent shaping via the incentive function. This resembles LOLA [11], but there are key algorithmic differences. Firstly, LIO's incentive function is trained separately from its policy parameters, while opponent shaping in LOLA depends solely on the policy. Secondly, the LOLA gradient correction for agent i is derived from $\nabla_{\theta^i} J^i(\theta^i, \theta^j + \Delta\theta^j)$ under Taylor expansion, but LOLA disregards a term with $\nabla_{\theta^i}\nabla_{\theta^j} J^i(\theta^i, \theta^j)$ even though it is non-zero in general. In contrast, LIO is constructed from the principle of online cross-validation [33], not Taylor expansion, and hence this particular mixed derivative is absent—the analogue for LIO would be $\nabla_{\eta^i}\nabla_{\theta^j} J^i(\theta^i, \theta^j)$, which is zero because incentive parameters η^i affect all agents *except* agent i. Thirdly, LOLA optimizes its objective assuming one step of opponent learning, *before* the opponent actually does so [22]. In contrast, LIO updates the incentive function *after* recipients carry out policy updates using received incentives. This gives LIO a more accurate measurement of the impact of incentives, which reduces variance and increases performance, as we demonstrate experimentally in Appendix E.1 by comparing with a 1-episode variant of LIO that does not wait for opponent updates. Finally, by adding differentiable reward channels to the environment, which is feasible in many settings with side payments [17], LIO is closer in spirit to the paradigm of optimized rewards [30, 18, 39].

3.2 Analysis in Iterated Prisoner's Dilemma

Table 1: Prisoner's Dilemma

A1/A2	C	D
C	(-1, -1)	(-3, 0)
D	(0, -3)	(-2, -2)

LIO poses a challenge for theoretical analysis in general Markov games because each agent's policy and incentive function are updated using different trajectories but are coupled through the RL updates of all other agents. Nonetheless, a complete analysis of exact LIO—using closed-form gradient ascent without policy gradient approximation—is tractable in repeated matrix games. In the stateless Iterated Prisoner's Dilemma (IPD), for example, with payoff matrix in Table 1, we prove in Appendix B the following:

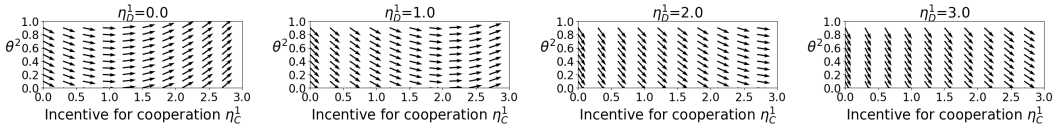


Figure 2: Exact LIO in IPD: probability of recipient cooperation versus incentive for cooperation.

Proposition 1. *Two LIO agents converge to mutual cooperation in the Iterated Prisoner's Dilemma.*

Moreover, we may gain further insight by visualizing the learning dynamics of exact LIO in the IPD, computed in Appendix B. Let $\eta^1 := [\eta_C^1, \eta_D^1] \in [0, 3]^2$ be the incentives that Agent 1 gives to Agent 2 for cooperation (C) and defection (D), respectively. Let θ^2 denote Agent 2's probability of cooperation. In Figure 2, the curvature of vector fields shows guaranteed increase in probability of recipient cooperation θ^2 (vertical axis) along with increase in incentive value η_C^1 received for cooperation (horizontal axis). For higher values of incentive for defection η_D^1, greater values of η_C^1 are needed for θ^2 to increase. Figure 7 shows that incentive for defection is guaranteed to decrease.

4 Experimental setup

Our experiments demonstrate that LIO agents are able to reach near-optimal individual performance by incentivizing other agents in cooperation problems with conflicting individual and group utilities. We define three different environments with increasing complexity in Section 4.1 and describe the implementation of our method and baselines in Section 4.2.

4.1 Environments

Iterated Prisoner's Dilemma (IPD). We test LIO on the memory-1 IPD as defined in [11], where agents observe the joint action taken in the previous round and receive extrinsic rewards in Table 1. This serves as a test of our theoretical prediction in Section 3.2.

N-Player Escape Room (ER). We experiment on the N-player Escape Room game shown in Figure 1 (Section 1). By symmetry, any agent can receive positive extrinsic reward, as long as there are enough cooperators. Hence, for methods that allow incentivization, every agent is both a reward giver and recipient. We experiment with the cases $(N = 2, M = 1)$ and $(N = 3, M = 2)$. We also describe an asymmetric 2-player case and results in Appendix E.1.

Cleanup. Furthermore, we conduct experiments on the Cleanup game (Figure 3) [16, 39]. Agents get +1 individual reward by collecting apples, which spawn on the right hand side of the map at a rate that decreases linearly to zero as the amount of waste in a river approaches a depletion threshold. Each episode starts with a waste level above the threshold and no apples present. While an agent can contribute to the public good by firing a cleaning beam to clear waste, it can only do so at the river as its fixed orientation points upward. This would enable other agents to defect and selfishly collect apples, resulting in a difficult social dilemma. Each agent has an egocentric RGB image observation that spans the entire map.

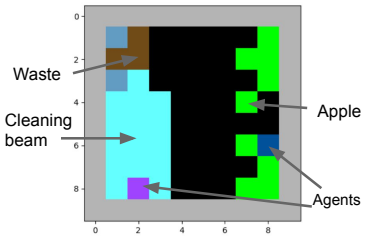
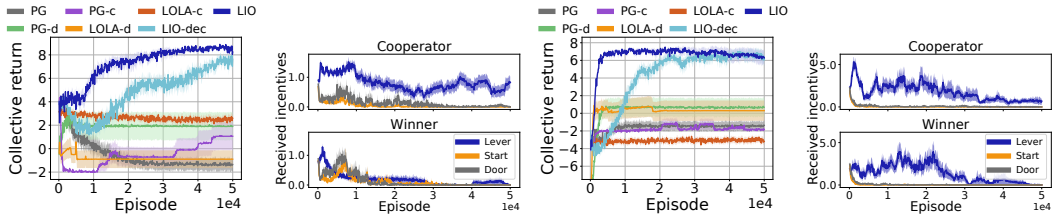


Figure 3: *Cleanup* (10x10 map): apple spawn rate decreases with increasing waste, which agents can clear with a cleaning beam.

4.2 Implementation and baselines

We describe key details here and provide a complete description in Appendix D.2. In each method, all agents have the same implementation without sharing parameters. The incentive function of a LIO agent is a neural network defined as follows: its input is the concatenation of the agent's observation and all other agents' chosen actions; the output layer has size N, sigmoid activation, and is scaled element-wise by a multiplier $R_{\max}$; each output node j, which is bounded in $[0, R_{\max}]$, is interpreted as the real-valued reward given to agent with index j in the game (we zero-out the value it gives to itself). We chose $R_{\max} = [3, 2, 2]$ for [IPD, ER, Cleanup], respectively, so that incentives can overcome any extrinsic penalty or opportunity cost for cooperation. We use on-policy learning with policy gradient for each agent in IPD and ER, and actor-critic for Cleanup. To ensure that all agents' policies perform sufficient exploration for the effect of incentives to be discovered, we include an exploration lower bound ϵ such that $\tilde{\pi}(a|s) = (1 - \epsilon)\pi(a|s) + \epsilon/|\mathcal{A}|$, with linearly decreasing ϵ.



(a) Collective return in N=2 (b) Incentives in N=2 (c) Collective return in N=3 (d) Incentives in N=3

Figure 5: Escape Room. (a,c) LIO agents converge near the global optimum with value 9 (N=2) and 8 (N=3). (b,d) Incentives received for each action by the agent who ends up going to the lever/door.

Fully-decentralized implementation (LIO-dec). Each decentralized LIO agent i learns a model of another agent's policy parameters θ^j via $\theta^j_{\text{estimate}} = \text{argmax}_{\theta^j} \sum_{(o_t^j, a_t^j) \in \tau} \log \pi_{\theta^j}(a_t^j | o_t^j)$ at the end of each episode τ. With knowledge of agent j's egocentric observation and individual rewards, it conducts incentive function updates using a fictitious policy update in (3) with $\theta^j_{\text{estimate}}$ in place of θ^j.

Baselines. The first baseline is independent policy gradient, labeled **PG**, which has the same architecture as the policy part of LIO. Second, we augment policy gradient with discrete "give-reward" actions, labeled **PG-d**, whose action space is $\mathcal{A} \times \{\text{no-op, give-reward}\}^{N-1}$. We try reward values in the set $\{2, 1.5, 1.1\}$. Giving reward incurs an equivalent cost. Next, we design a more flexible policy gradient baseline called **PG-c**, which has continuous give-reward actions. It has an augmented action space $\mathcal{A} \times [0, R_{\text{max}}]^{N-1}$ and learns a factorized policy $\pi(a_d, a_r|o) := \pi(a_d|o)\pi(a_r|o)$, where $a_d \in \mathcal{A}$ is the regular discrete action and $a_r \in [0, R_{\text{max}}]^{N-1}$ is a vector of incentives given to the other $N-1$ agents. Appendix D.2 describes how PG-c is trained. In ER, we run **LOLA-d** and **LOLA-c** with the same augmentation scheme as PG-d and PG-c. In Cleanup, we compare with independent actor-critic agents (**AC-d** and **AC-c**), which are analogously augmented with "give-reward" actions, and with inequity aversion (**IA**) agents [16]. We also show the approximate upper bound on performance by training a fully-centralized actor-critic (**Cen**) that is (unfairly) allowed to optimize joint reward.

5 Results

We find that LIO agents reach near-optimal *collective* performance in all three environments, despite being designed to optimize only *individual* rewards. This arose in ER and Cleanup because incentivization enabled agents to find an optimal division of labor[5] and in IPD where LIO is proven to converge to the CC solution. In contrast, various baselines displayed competitive behavior that led to suboptimal solutions, were not robust across random seeds, or failed to cooperate altogether. We report the results of 20 independent runs for IPD and ER, and 5 runs for Cleanup.

Iterated Prisoner's Dilemma. In accord with the theoretical prediction of exact LIO in Section 3.2 and Appendix B, two LIO agents with policy gradient approximation converge near the optimal CC solution with joint reward -2 in the IPD (Figure 4). This meets the performance of LOLA-PG and is close to LOLA-Ex, as reported in [11]. In the asymmetric case (LIO-asym) where one LIO agent is paired with a PG agent, we indeed find that they converge to the DC solution: PG is incentivized to cooperate while LIO defects, resulting in collective reward near -3.

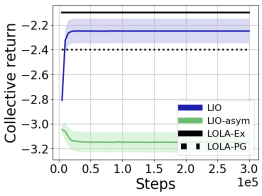
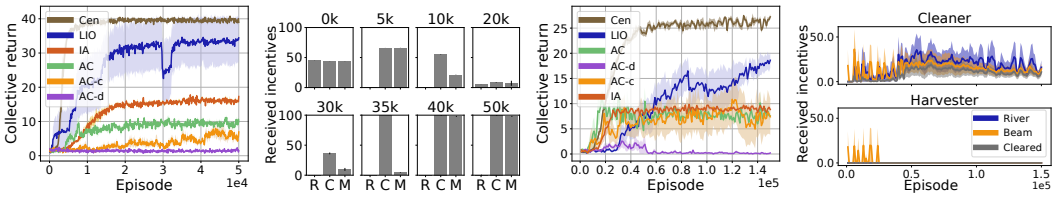


Figure 4: The sum of all agents' rewards in IPD.

Escape Room. Figures 5a and 5c show that groups of LIO agents are able to discover a division of labor in both the $N = 2$ and $N = 3$ cases, whereby some agent(s) cooperate by pulling the lever to allow another agent to exit the door, such that collective return approaches the optimal value (9 for the 2-player case, 8 for the 3-player case). Fully-decentralized LIO-dec successfully solved both cases, albeit with slower learning speed. As expected, PG agents were unable to find a cooperative solution: they either stay at the start state or greedily move to the door, resulting in negative collective return. The augmented baselines PG-d and PG-c sometimes successfully influence the learning of another agent to solve the game, but exhibit high variance across independent runs. This is strong evidence that conventional RL alone

[5]Learned behavior in Cleanup can be viewed at `https://sites.google.com/view/neurips2020-lio`



(a) Cleanup 7x7 (b) Scripted agent incentives (c) Cleanup 10x10 (d) Incentives in training

Figure 6: Results on Cleanup. (a,c) Emergent division of labor between LIO agents enables higher performance than AC and IA baselines, which find rewards but exhibit competitive behavior. (b) Behavior of incentive function in 7x7 Cleanup at different training checkpoints, measured against three scripted opponents: R moves within river without cleaning; C successfully cleans waste; M fires the cleaning beam but misses waste (mean and standard error of 20 evaluation episodes). (d) 10x10 map: the LIO agent who becomes a "Cleaner" receives incentives, while the "Harvester" does not.

is not well suited for learning to incentivize, as the effect of "give-reward" actions manifests only in future episodes. LOLA succeeds sometimes but with high variance, as it does not benefit from the stabilizing effects of online cross-validation and separation of the incentivization channel from regular actions. Appendix E.1 contains results in an asymmetric case (LIO paired with PG), where we compare to an additional heuristic two-timescale baseline and a variant of LIO.

To understand the behavior of LIO's incentive function, we classify each agent *at the end of training* as either a "cooperator" or "winner" based on whether its final policy has greater probability of going to the lever or door, respectively. For each agent type, aggregating over all agents of that type, we measure incentives received by that agent type when it takes each of the three actions during training. Figures 5b and 5d show that the cooperator was correctly incentivized for pulling the lever and receives negligible incentives for noncooperative actions. Asymptotically, the agent who "wins" the game receives negligible incentives from the cooperator(s), who learned to avoid the cost for incentivization (6) when doing so has no benefits itself.

Cleanup. Figures 6a and 6c show that LIO agents collected significantly more extrinsic rewards than AC and IA baselines in Cleanup, and approach the upper bound on performance as indicated by Cen, on both a 7x7 map and a 10x10 map with more challenging depletion threshold and lower apple respawn rates. LIO agents discovered a division of labor (Figure 11a), whereby one agent specializes to cleaning waste at the river while the other agent, who collects almost all of the apples, provides incentives to the former. In contrast, AC baselines learned to perform cleaning for apples to spawn but subsequently compete to collect apples, which is suboptimal for the group (Figure 11b). Due to continual exploration by all agents, an agent may change its behavior if it receives incentives for "wrong actions": e.g., near episode 30k in Figure 6a, an agent temporarily stopped cleaning the river despite having consistently done so earlier. We can further understand the progression of LIO's incentive function during training as follows. First, we classify LIO agents *at the end of training* as a "Cleaner" or a "Harvester", based on whether it primarily cleans waste or collected apples, respectively. Periodically during training, we evaluated each agent against each of three hand-scripted agents: an R agent moves in the river but does not clean, a C agent successfully cleans waste, and an M agent fires the cleaning beam but misses waste. Figure 6b shows the incentives given by a Harvester to these scripted agents at different times in training. At episodes 10k, 30k and 35k, it gave more incentives to C than to M, meaning that it distinguished between successful and unsuccessful cleaning, which explains how its actual partner in training was incentivized to become a Cleaner. Figure 6d shows the actual incentives received by Cleaner and Harvester agents when they are positioned in the river, fire the cleaning beam, or successfully clear waste during training. We see that asymptotically, only Harvesters provide incentives to Cleaners and not the other way around.

6 Conclusion and future directions

We created Learning to Incentivize Others (LIO), an agent who learns to give rewards directly to other RL agents. LIO learns an incentive function by explicitly accounting for the impact of incentives on its own extrinsic objective, through the learning updates of reward recipients. In the Iterated Prisoner's Dilemma, an illustrative *Escape Room* game, and a benchmark social dilemma problem called *Cleanup*, LIO correctly incentivizes other agents to overcome extrinsic penalties so as

to discover cooperative behaviors, such as division of labor, and achieve near-optimum collective performance. We further demonstrated the feasibility of a fully-decentralized implementation of LIO.

Our approach to the goal of ensuring cooperation in a decentralized multi-agent population poses many open questions. How should one analyze the simultaneous processes of learning incentive functions, which continuously modifies the set of equilibria, and learning policies with these changing rewards? How can one account for the cost for incentives in a more adaptive way? How should agents better account for the longer-term effect of incentives? Should social factors among an agent population modulate the effect of incentives? We hope our work will spark further interest in this research endeavor.

Broader Impact

Our work is a step toward the goal of ensuring the common good in a potential future where independent reinforcement learning agents interact with one another and/or with humans in the real world. We have shown that cooperation can emerge by introducing an additional learned incentive function that enables one agent to affect another agent's reward directly. However, as agents still independently maximize their own individual rewards, it is open as to how to prevent an agent from misusing the incentive function to exploit others. One approach for future research to address this concern is to establish new connections between our work and the emerging literature on reward tampering [10]. By sparking a discussion on this important aspect of multi-agent interaction, we believe our work has a positive impact on the long-term research endeavor that is necessary for RL agents to be deployed safely in real-world applications.

Acknowledgements

We thank Thomas Anthony, Jan Balaguer, and Thore Graepel at DeepMind for insightful discussions and feedback.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.* (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[2] Akçay, E. and Roughgarden, J. (2011). The evolution of payoff matrices: providing incentives to cooperate. *Proceedings of the Royal Society B: Biological Sciences*, **278**(1715), 2198–2206.

[3] Albrecht, S. V. and Stone, P. (2018). Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, **258**, 66–95.

[4] Baumann, T., Graepel, T., and Shawe-Taylor, J. (2018). Adaptive mechanism design: Learning to promote cooperation. *arXiv preprint arXiv:1806.04067*.

[5] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., *et al.* (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

[6] Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization. *Annals of operations research*, **153**(1), 235–256.

[7] Delfgaauw, J. and Dur, R. (2008). Incentives and workers' motivation in the public sector. *The Economic Journal*, **118**(525), 171–191.

[8] Doxey, M. P. (1980). *Economic sanctions and international enforcement*. Springer.

[9] Eccles, T., Hughes, E., Kramár, J., Wheelwright, S., and Leibo, J. Z. (2019). Learning reciprocity in complex sequential social dilemmas. *arXiv preprint arXiv:1903.08082*.

[10] Everitt, T. and Hutter, M. (2019). Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective. *arXiv preprint arXiv:1908.04734*.

[11] Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. (2018a). Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130. International Foundation for Autonomous Agents and Multiagent Systems.

[12] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018b). Counterfactual multi-agent policy gradients. In *Thirty-second AAAI conference on artificial intelligence*.

[13] Fong, Y.-f. and Surti, J. (2009). The optimal degree of cooperation in the repeated prisoners' dilemma with side payments. *Games and Economic Behavior*, **67**(1), 277–291.

[14] Güth, W. (1995). An evolutionary approach to explaining cooperative behavior by reciprocal incentives. *International Journal of Game Theory*, **24**(4), 323–344.

[15] Harstad, B. (2008). Do side payments help? collective decisions and strategic delegation. *Journal of the European Economic Association*, **6**(2-3), 468–477.

[16] Hughes, E., Leibo, J. Z., Phillips, M., Tuyls, K., Dueñez-Guzman, E., Castañeda, A. G., Dunning, I., Zhu, T., McKee, K., Koster, R., *et al.* (2018). Inequity aversion improves cooperation in intertemporal social dilemmas. In *Advances in neural information processing systems*, pages 3326–3336.

[17] Jackson, M. O. and Wilkie, S. (2005). Endogenous games and mechanisms: Side payments among players. *The Review of Economic Studies*, **72**(2), 543–566.

[18] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., *et al.* (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, **364**(6443), 859–865.

[19] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[20] Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems.

[21] Lerer, A. and Peysakhovich, A. (2017). Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *arXiv preprint arXiv:1707.01068*.

[22] Letcher, A., Foerster, J., Balduzzi, D., Rocktäschel, T., and Whiteson, S. (2019). Stable opponent shaping in differentiable games. In *International Conference on Learning Representations*.

[23] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier.

[24] Lupu, A. and Precup, D. (2020). Gifting in multi-agent reinforcement learning. In *Proceedings of the 19th Conference on Autonomous Agents and MultiAgent Systems*, pages 789–797. International Foundation for Autonomous Agents and Multiagent Systems.

[25] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., *et al.* (2015). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529–533.

[26] Olson, M. (1965). *The Logic of Collective Action*. Harvard University Press.

[27] Rapoport, A. (1974). Prisoner's dilemma—recollections and observations. In *Game Theory as a Theory of a Conflict Resolution*, pages 17–34. Springer.

[28] Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4295–4304.

[29] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.* (2017). Mastering the game of go without human knowledge. *Nature*, **550**(7676), 354–359.

[30] Singh, S., Lewis, R. L., and Barto, A. G. (2009). Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society.

[31] Sodomka, E., Hilliard, E., Littman, M., and Greenwald, A. (2013). Coco-q: Learning in stochastic games with side payments. In *International Conference on Machine Learning*, pages 1471–1479.

[32] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., *et al.* (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems.

[33] Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*, pages 171–176.

[34] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[35] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[36] Veroff, J. and Veroff, J. B. (2016). *Social incentives: A life-span developmental approach*. Elsevier.

[37] Vinitsky, E. (2020). *Sequential Social Dilemma Games*. `https://github.com/eugenevinitsky/sequential_social_dilemma_games`.

[38] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., *et al.* (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, **575**(7782), 350–354.

[39] Wang, J. X., Hughes, E., Fernando, C., Czarnecki, W. M., Duéñez-Guzmán, E. A., and Leibo, J. Z. (2019). Evolving intrinsic motivations for altruistic behavior. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 683–692. International Foundation for Autonomous Agents and Multiagent Systems.

[40] Xu, Z., van Hasselt, H. P., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pages 2396–2407.

[41] Zheng, Z., Oh, J., and Singh, S. (2018). On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 4644–4654.

A Further discussion

A.1 Cost for incentivization

We justify the way in which LIO accounts for the cost of incentivization as follows. Recall that this cost is incurred in the objective for LIO's incentive function (see (5) and (6)), instead of being accounted in the total reward (1) that is maximized by LIO's policy. Fundamentally, the reason is that the cost should be incurred only by the part of the agent that is directly responsible for incentivization. In LIO, the policy and incentive function are separate modules: while the former takes regular actions to maximize *external* rewards, only the latter produces incentives that directly and actively shape the behavior of other agents. The policy is decoupled from incentivization, and it would be incorrect to penalize it for the behavior of the incentive function. Instead, we need to attribute the cost directly to the incentive function parameters via (6). From a more intuitive perspective, LIO is constructed with the knowledge that it can perform two fundamentally different behaviors—1) take regular actions that affect the Markov game transition, and 2) give incentives to shape other agents' learning—and it knows not to penalize the former behavior with the latter behavior. In contrast, if one were to augment conventional RL with reward-giving actions (as we do for baselines in Section 4.2), then the cost for incentivization should indeed be accounted by the policy. One may consider other mechanisms for cost, such as budget constraints [24].

In our experiments, we find the coefficient α in the cost for incentivization is a sensitive parameter. At the beginning of training, (6) immediately drives the magnitude of incentives to zero. However, both the reward-giver and recipients require sufficient time to learn the effect of incentives, which means that too large an α would lead to the degenerate result of $r_{\eta^i} = \mathbf{0}$. On the other extreme, $\alpha = 0$ means there is no penalty and may result in profligate incentivization that serves no useful purpose. While we found that values of 10^{-3} and 10^{-4} worked well in our experiments, one may consider adaptive and dynamic computation of α for more efficient training.

B Analysis in Iterated Prisoner's Dilemma

Proposition 1. *Two LIO agents converge to mutual cooperation in the Iterated Prisoner's Dilemma.*

Proof. We prove this by deriving closed-form expressions for the updates to parameters of policies and incentive functions. These updates are also used to compute the vector fields shown in Figure 2. Let θ^i for $i \in \{1, 2\}$ denote each agent's probability of taking the cooperative action. Let $\eta^1 := [\eta^1_C, \eta^1_D] \in \mathbb{R}^2$ denote Agent 1's incentive function, where the values are given to Agent 2 when it takes action $a^2 = C$ or $a^2 = D$. Similarly, let η^2 denote Agent 2's incentive function. The value function for each agent is defined by

$$\begin{aligned} V^i(\theta^1, \theta^2) &= \sum_{t=0}^{\infty} \gamma^t p^T r^i = \frac{1}{1-\gamma} p^T r^i \,, \\ \text{where} \quad p &= \left[\theta^1\theta^2, \theta^1(1-\theta^2), (1-\theta^1)\theta^2, (1-\theta^1)(1-\theta^2)\right] \,. \end{aligned} \tag{11}$$

The total reward received by each agent is

$$r^1 = \left[-1 + \eta^2_C, -3 + \eta^2_C, 0 + \eta^2_D, -2 + \eta^2_D\right] \,, \tag{12}$$

$$r^2 = \left[-1 + \eta^1_C, 0 + \eta^1_D, -3 + \eta^1_C, -2 + \eta^1_D\right] \,. \tag{13}$$

Agent 2 updates its policy via the update

$$\begin{aligned} \hat{\theta}^2 &= \theta^2 + \alpha \nabla_{\theta^2} V^2(\theta^1, \theta^2) \\ &= \theta^2 + \frac{\alpha}{1-\gamma} \nabla_{\theta^2} \left(\theta^1\theta^2(-1 + \eta^1_C) + \theta^1(1-\theta^2)\eta^1_D \right. \\ &\quad \left. + (1-\theta^1)\theta^2(-3 + \eta^1_C) + (1-\theta^1)(1-\theta^2)(-2 + \eta^1_D)\right) \\ &= \theta^2 + \frac{\alpha}{1-\gamma} \left(\eta^1_C - \eta^1_D - 1\right) \,, \end{aligned} \tag{14}$$

and likewise for Agent 1:

$$\hat{\theta}^1 = \theta^1 + \frac{\alpha}{1-\gamma} \left(\eta^2_C - \eta^2_D - 1\right) \,. \tag{15}$$

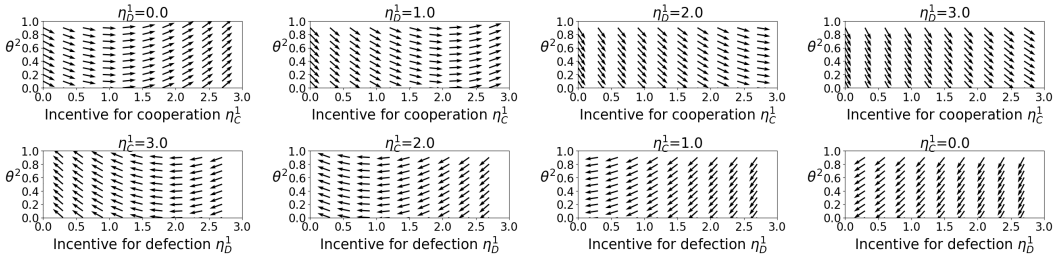


Figure 7: Vector fields showing the probability of recipient cooperation versus incentive value given for cooperation (top row) and defection (lower row). Each plot has a fixed value for the incentive given for the other action.

Let $\hat{p}$ denote the joint action probability under updated policies $\hat{\theta}^1$ and $\hat{\theta}^2$, and let $\Delta^2 := (\eta_C^1 - \eta_D^1 - 1)\alpha/(1-\gamma)$ denote Agent 2's policy update. Agent 1 updates its incentive function parameters via

$$\begin{aligned}
\eta^1 &\leftarrow \eta^1 + \beta \nabla_{\eta^1} \frac{1}{1-\gamma} \hat{p}^T r^1 \\
&= \eta^1 + \frac{\beta}{1-\gamma} \nabla_{\eta^1} \Big[\hat{\theta}^1(\theta^2 + \Delta^2)(-1 + \eta_C^2) + \hat{\theta}^1(1 - \theta^2 - \Delta^2)(-3 + \eta_C^2) \\
&\quad + (1 - \hat{\theta}^1)(\theta^2 + \Delta^2)\eta_D^2 + (1 - \hat{\theta}^1)(1 - \theta^2 - \Delta^2)(-2 + \eta_D^2) \Big] \\
&= \eta^1 + \frac{\beta\alpha}{(1-\gamma)^2} B_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix},
\end{aligned} \tag{16}$$

where the scalar B_2 is

$$B_2 = \hat{\theta}^1(-1 + \eta_C^2) - \hat{\theta}^1(-3 + \eta_C^2) + (1 - \hat{\theta}^1)\eta_D^2 - (1 - \hat{\theta}^1)(-2 + \eta_D^2) = 2\,. \tag{17}$$

By symmetry, with $B_1 = 2$, Agent 2 updates its incentive function via

$$\eta^2 \leftarrow \eta^2 + \frac{\beta\alpha}{(1-\gamma)^2} B_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \tag{18}$$

Note that each η^i is updated so that η_C^i increases while η_D^i decreases. Referring to (14) and (15), one sees that the updates to incentive parameters lead to updates to policy parameters that increase the probability of mutual cooperation. This is consistent with the viewpoint of modifying the Nash Equilibrium of the payoff matrices. With incentives, the players have payoff matrices in Table 2. For CC to be the global Nash Equilibrium, such that cooperation is preferred by an agent i regardless of the other agent's action, incentives must satisfy $\eta_C^i - \eta_D^i - 1 > 0$. This is guaranteed to occur by incentive updates (16) and (18). □

Table 2: Payoff matrices for row player (left) and column player (right) with incentives.

A1	C	D
C	-1 + η_C^2	-3 + η_C^2
D	0 + η_D^2	-2 + η_D^2

A2	C	D
C	-1 + η_C^1	0 + η_D^1
D	-3 + η_C^1	-2 + η_D^1

C Derivations

The factor $\nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}})$ (9) in the incentive function's gradient (7) is derived as follows. For brevity, we will drop the "hat" notation—recall that it indicates a quantity belongs to a new trajectory after a regular policy update—as all quantities here have "hats". Let ∇_j denote $\nabla_{\hat{\theta}^j}$ and π denote $\pi(a_t|s_t)$. Let $V^{i,\boldsymbol{\pi}}(s)$ and $Q^{i,\boldsymbol{\pi}}(s, \mathbf{a})$ denote the global value and action-value function for agent i's reward

under joint policy $\boldsymbol{\pi}$. Then the gradient of agent i's expected extrinsic return with respect to agent j's policy parameter can be derived in a similar manner as standard policy gradients [35]:

$$\begin{aligned}
&\nabla_j J^i(\tau, \boldsymbol{\theta}) = \nabla_j V^{i,\boldsymbol{\pi}}(s_0) = \nabla_j \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s_0) Q^{i,\boldsymbol{\pi}}(s_0, \mathbf{a}) \\
&= \sum_{\mathbf{a}} \pi^{-j} \left((\nabla_j \pi^j) Q^{i,\boldsymbol{\pi}}(s_0, \mathbf{a}) + \pi^j \nabla_j Q^{i,\boldsymbol{\pi}}(s_0, \mathbf{a}) \right) \\
&= \sum_{\mathbf{a}} \pi^{-j} \left((\nabla_j \pi^j) Q^{i,\boldsymbol{\pi}} + \pi^j \nabla_j \left(r^i + \gamma \sum_{s'} P(s'|s_0, \mathbf{a}) V^{i,\boldsymbol{\pi}}(s') \right) \right) \\
&= \sum_{\mathbf{a}} \pi^{-j} \left((\nabla_j \pi^j) Q^{i,\boldsymbol{\pi}} + \gamma \pi^j \sum_{s'} P(s'|s_0, \mathbf{a}) \nabla_j V^{i,\boldsymbol{\pi}}(s') \right) \\
&= \sum_{x} \sum_{k=0}^{\infty} P(s_0 \to x, k, \boldsymbol{\pi}) \gamma^k \sum_{\mathbf{a}} \pi^{-j} \nabla_j \pi^j Q^{i,\boldsymbol{\pi}}(x, \mathbf{a}) \\
&= \sum_{s} d^{\boldsymbol{\pi}}(s) \sum_{\mathbf{a}} \pi^{-j} \nabla_j \pi^j Q^{i,\boldsymbol{\pi}}(s, \mathbf{a}) \\
&= \sum_{s} d^{\boldsymbol{\pi}}(s) \sum_{\mathbf{a}} \pi^{-j} \pi^j \nabla_j \log \pi^j Q^{i,\boldsymbol{\pi}}(s, \mathbf{a}) \\
&= \mathbb{E}_{\boldsymbol{\pi}} \left[\nabla_j \log \pi^j(a^j|s) Q^{i,\boldsymbol{\pi}}(s, \mathbf{a}) \right]
\end{aligned}$$

Alternatively, one may rely on automatic differentiation in modern machine learning frameworks [1] to compute the chain rule (7) via direct minimization of the loss (10). This is derived as follows. Let the notation $\neq j, i$ denote all indices except j and i. Note that agent i's updated policy $\hat{\pi}^i$ is not a function of η^i, as it does not receive incentives from itself. Recall that a recipient j's updated policy $\hat{\pi}^j$ has explicit dependence on a reward-giver i's incentive parameters η^i. Also note that

$$\nabla_{\eta^i} \hat{\pi}^{-i} = \sum_{j \neq i} (\nabla_{\eta^i} \hat{\pi}^j) \hat{\pi}^{\neq j, i}$$

by the product rule. Then we have:

$$\begin{aligned}
&\nabla_{\eta^i} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) = \nabla_{\eta^i} V^{i,\hat{\boldsymbol{\pi}}}(\hat{s}_0) = \nabla_{\eta^i} \sum_{\hat{\mathbf{a}}} \hat{\pi}^i(\hat{a}^i|\hat{s}_0) \hat{\pi}^{-i}(\hat{a}^{-i}|\hat{s}_0) Q^{i,\hat{\boldsymbol{\pi}}}(\hat{s}_0, \hat{\mathbf{a}}) \\
&= \sum_{\hat{\mathbf{a}}} \hat{\pi}^i \left(\sum_{j \neq i} (\nabla_{\eta^i} \hat{\pi}^j) \hat{\pi}^{\neq j, i} Q^{i,\hat{\boldsymbol{\pi}}} + \hat{\pi}^{-i} \nabla_{\eta^i} Q^{i,\hat{\boldsymbol{\pi}}} \right) \quad \text{(by the remarks above)} \\
&= \sum_{\hat{\mathbf{a}}} \hat{\pi}^i \left(\sum_{j \neq i} (\nabla_{\eta^i} \hat{\pi}^j) \hat{\pi}^{\neq j, i} Q^{i,\hat{\boldsymbol{\pi}}} + \gamma \hat{\pi}^{-i} \sum_{s'} P(s'|\hat{s}_0, \hat{\mathbf{a}}) \nabla_{\eta^i} V^{i,\hat{\boldsymbol{\pi}}}(s') \right) \\
&= \sum_{x} \sum_{k=0}^{\infty} P(s_0 \to x, k, \hat{\boldsymbol{\pi}}) \gamma^k \sum_{\hat{\mathbf{a}}} \hat{\pi}^i \sum_{j \neq i} (\nabla_{\eta^i} \hat{\pi}^j) \hat{\pi}^{\neq j, i} Q^{i,\hat{\boldsymbol{\pi}}} \\
&= \sum_{\hat{s}} d^{\hat{\boldsymbol{\pi}}}(\hat{s}) \sum_{\hat{\mathbf{a}}} \hat{\pi}^i \sum_{j \neq i} \hat{\pi}^j (\nabla_{\eta^i} \log \hat{\pi}^j) \hat{\pi}^{\neq j, i} Q^{i,\hat{\boldsymbol{\pi}}} \\
&= \sum_{\hat{s}} d^{\hat{\boldsymbol{\pi}}}(\hat{s}) \sum_{\hat{\mathbf{a}}} \hat{\pi}^i \sum_{j \neq i} (\nabla_{\eta^i} \log \hat{\pi}^j) \hat{\pi}^{-i} Q^{i,\hat{\boldsymbol{\pi}}} \\
&= \sum_{\hat{s}} d^{\hat{\boldsymbol{\pi}}}(\hat{s}) \sum_{\hat{\mathbf{a}}} \hat{\boldsymbol{\pi}} \sum_{j \neq i} (\nabla_{\eta^i} \log \hat{\pi}^j) Q^{i,\hat{\boldsymbol{\pi}}} = \mathbb{E}_{\hat{\boldsymbol{\pi}}} \left[\sum_{j \neq i} (\nabla_{\eta^i} \log \hat{\pi}^j) Q^{i,\hat{\boldsymbol{\pi}}} \right]
\end{aligned}$$

Hence descending a stochastic estimate of this gradient is equivalent to minimizing the loss in (10).

D Experiments

D.1 Environment details

This section provides more details on each experimental setup.

IPD. We used the same definition of observation, action, and rewards as Foerster *et al.* [11]. Each environment step is one round of the matrix game. Each agent observes the joint action taken by both agents at the previous step, along with an indicator for the first round of each episode. We trained for 60k episodes, each with 5 environments steps, which gives the same total number of environment steps used by LOLA [11].

Escape Room. Each agent observes all agents' positions and can move among the three available states: lever, start, and door. At every time step, all agents commit to and disclose their chosen actions, compute the incentives based on their observations of state and others' actions (only for LIO and augmented baselines that allow incentivization), and receive the sum of extrinsic rewards and incentives (if any). LIO and augmented baselines also observe the cumulative incentives given to the other agents within the current episode. An agent's individual reward is zero for staying at the current state, -1 for movement away from its current state if fewer than M agents move to (or are currently at) the lever, and +10 for moving to (or staying at) the door if $\geq M$ agents pull the lever. Each episode terminates when an agent successfully exits the door, or when 5 time steps elapse.

Cleanup. We built on a version of an open-source implementation [37]. The environment settings for 7x7 and 10x10 maps are given in Table 3. To focus on the core aspects of the common-pool resource problem, we removed rotation actions, set the orientation of all agents to face "up", and disabled their "tagging beam" (which, if used, would remove a tagged agent from the environment for a number of steps). These changes mean that an agent must move to the river side of the map to clear waste successfully, as it cannot simply stay in the apple patch and fire its cleaning beam toward the river. Acting cooperatively as such would allow other agents to collect apples, and hence our setup increases the difficulty of the social dilemma. Each agent receives an egocentric normalized RGB image observation that spans a sufficiently large area such that the entire map is observable by that agent regardless of its position. The cleaning beam has length 5 and width 3. For LIO and the AC-c baseline, which have a separate module that observes other agents' actions and outputs real-valued incentives, we let that module observe a multi-hot vector that indicates which agent(s) used their cleaning beam.

Table 3: Environment settings in Cleanup

Parameter	7x7	10x10
appleRespawnProbability	0.5	0.3
thresholdDepletion	0.6	0.4
thresholdRestoration	0.0	0.0
wasteSpawnProbability	0.5	0.5
view_size	4	7
max_steps	50	50

D.2 Implementation

This subsection provides more details on implementation of all algorithms used in experiments. We use fully-connected neural networks for function approximation in the IPD and ER, and convolutional networks to process image observations in Cleanup. The policy network has a softmax output for discrete actions in all environments. Within each environment, all algorithms use the same neural architecture unless stated otherwise. We applied the open-source implementation of LOLA [11] to ER. We use an exploration lower bound ϵ that maps the learned policy π to a behavioral policy $\tilde{\pi}(a|s) = (1-\epsilon)\pi(a|s) + \epsilon/|\mathcal{A}|$, with ϵ decaying linearly from ϵ_{start} to ϵ_{end} by ϵ_{div} episodes. We use discount factor $\gamma = 0.99$. We use gradient descent for policy optimization, the Adam optimizer [19] for training value functions (in Cleanup), and Adam optimizer for LIO's incentive function.

The augmented policy gradient and actor-critic baselines, labeled as PG-c and AC-c, which have continuous "give-reward" actions in addition to regular discrete actions, are trained as follows. These

baselines have an augmented action space $\mathcal{A} \times \mathbb{R}^{N-1}$ and learns a factorized policy $\pi(a_d, a_r|o) := \pi(a_d|o)\pi(a_r|o)$, where $a_d \in \mathcal{A}$ is a regular discrete action and $a_r \in \mathbb{R}^{N-1}$ is the reward given to the other $N-1$ agents. The factor $\pi(a_d|o)$ is a standard categorical distribution conditioned on observation. The factor $\pi(a_r|o)$ is defined via an element-wise sigmoid $\sigma(\cdot)$ applied to samples from a multivariate diagonal Gaussian, so that $\pi(a_r|o)$ is bounded. Specifically, we let $u \sim \mathcal{N}(f_\eta(o), \mathbf{1})$, where $f_\eta(o)\colon \mathcal{O} \mapsto \mathbb{R}^{N-1}$ is a neural network with parameters η, and let $a_r = R_{\text{max}}\sigma(u)$. By the change of variables formula, $\pi(a_r|o)$ has density $\pi(a_r|o) = \mathcal{N}(\mu_\eta, \mathbf{1}) \prod_{i=1}^{N-1} (\mathrm{d}a_r[i]/\mathrm{d}u[i])^{-1}$, which can be used to compute the log-likelihood of $\pi(a_d, a_r|o)$ in the policy gradient.

Let β denote the coefficient for entropy of the policy, α_θ the policy learning rate, α_η the incentive learning rate, α_ϕ the critic learning rate, and R_a the value of the discrete "give-reward" action.

IPD. The policy network and the incentive function in LIO have two hidden layers of size 16 and 8.

Table 4: Hyperparameters in IPD.

Parameter	Value	Parameter	Value
β	0.1	α_θ	1e-3
ϵ_{start}	1.0	α_η	1e-3
ϵ_{end}	0.01	α	0
ϵ_{div}	5000	R_{max}	3.0

ER. The policy network has two hidden layers of size 64 and 32. LIO's incentive function has two hidden layers of size 64 and 16. We use a separate Adam optimizer for the cost part of the incentive function's objective (5), with learning rate 1e-4, with $\alpha_\eta =$ 1e-3, and set $\alpha = 1.0$. Exploration and learning rate hyperparameters were tuned for each algorithm via coordinate ascent, searching through ϵ_{start} in [0.5, 1.0], ϵ_{end} in [0.05, 0.1, 0.3], ϵ_{div} in [100, 1000], β in [0.01, 0.1], α_θ, α_η, and α_{cost} in [1e-3, 1e-4]. LOLA performed best with learning rate 0.1 and $R_a = 2.0$, but it did not benefit from additional exploration. LIO and PG-c have $R_{\text{max}} = 2.0$. PG-d used $R_a = 2.0$.

Table 5: Hyperparameters in Escape Room.

	$N=2$				$N=3$			
Parameter	LIO	PG	PG-d	PG-c	LIO	PG	PG-d	PG-c
β	0.01	0.01	0.01	0.1	0.01	0.01	0.01	0.1
ϵ_{start}	0.5	0.5	0.5	1.0	0.5	0.5	0.5	1.0
ϵ_{end}	0.1	0.05	0.05	0.1	0.3	0.05	0.05	0.1
ϵ_{div}	1e3	1e2	1e2	1e3	1e3	1e2	1e2	1e3
α_θ	1e-4	1e-4	1e-4	1e-3	1e-4	1e-4	1e-4	1e-3

Cleanup. All algorithms are based on actor-critic for policy optimization, whereby each agent j's policy parameter θ^j is updated via

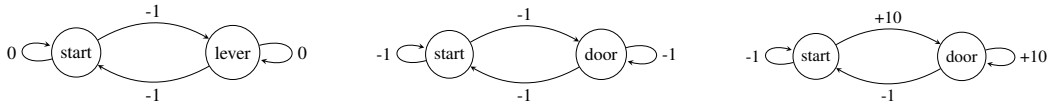
$$\hat{\theta}^j \leftarrow \theta^j + \mathbb{E}_{\boldsymbol{\pi}}\left[\nabla_{\theta^j} \log \pi_{\theta^j}(a^j|o^j)\left(r^j + \gamma V_{\phi^j}(s') - V_{\tilde{\phi}^j}(s)\right)\right], \tag{19}$$

and the critic parameter ϕ^j is updated by minimizing the temporal difference loss

$$L(\phi^j) = \mathbb{E}_{s,s'\sim\boldsymbol{\pi}}\left[\left(r^j + \gamma V_{\tilde{\phi}^j}(s') - V_{\phi^j}(s)\right)^2\right] \tag{20}$$

The target network [25] parameters $\tilde{\phi}^j$ are updated via $\tilde{\phi}^j \leftarrow \tau\phi^j + (1-\tau)\tilde{\phi}^j$ with $\tau = 0.01$.

The policy and value networks have an input convolutional layer with 6 filters of size [3, 3], stride [1, 1], and ReLU activation. The output of convolution is flattened and passed through two fully-connected (FC) hidden layers both of size 64. The policy output is a softmax over discrete actions; the value network has a linear scalar output. LIO's incentive function uses the same input convolutional layer, except that its output is passed through the first FC layer, concatenated with its observation of other agents' actions, then passed through the second FC layer and finally to a linear output layer. Inequity Aversion agents [16] have an additional 1D vector observation of all agents' temporally



(a) Agent A2 incurs an extrinsic penalty for any change of state.

(b) Agent A1 is penalized at every step if A2 does not pull the lever.

(c) A1 get +10 at the door if A2 pulls the lever.

Figure 8: Asymmetric *Escape Room* game involving two agents, A1 and A2. (a) In the absence of incentives, A2's optimal policy is to stay at the start state and not pull the lever. (b) Hence A1 cannot exit the door and is penalized at every step. (c) A1 can receive positive reward if it learns to incentivize A2 to pull the lever. Giving incentives is not an action depicted here.

smoothed rewards—this is concatenated with the output of the first FC hidden layer and sent to the second FC layer. Entropy coefficient was held at 0.1 for all methods.

LIO and AC-c have $R_{\text{max}} = 2.0$. AC-d used $R_a = 2.0$. Inequity aversion agents have disadvantageous aversion coefficient value 0, advantageous aversion coefficient value 0.05, and temporal smoothing parameter $\lambda = 0.95$. We use critic learning rate $\alpha_\phi = 10^{-3}$ for all methods. LIO used α_η =1e-3 and cost coefficient $\alpha = 10^{-4}$. Exploration and learning rate hyperparameters were tuned for each algorithm via coordinate ascent, searching through ϵ_{start} in [0.5, 1.0], ϵ_{end} in [0.05, 0.1], ϵ_{div} in [100, 1000, 5000], α_θ, α_η, and α_{cost} in [1e-3, 1e-4].

Table 6: Hyperparameters in Cleanup.

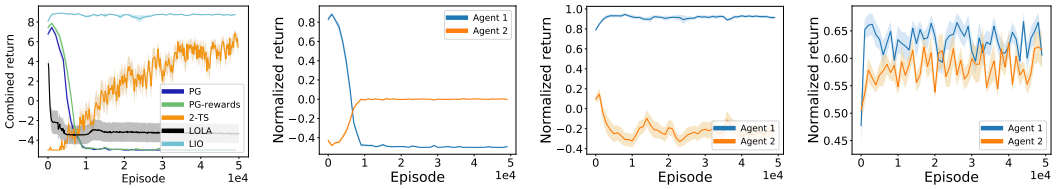
	7x7					10x10				
Parameter	LIO	AC	AC-d	AC-c	IA	LIO	AC	AC-d	AC-c	IA
ϵ_{start}	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
ϵ_{end}	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{div}	100	100	100	100	1000	1000	5000	1000	1000	5000
α_θ	1e-4	1e-3	1e-4	1e-4	1e-3	1e-4	1e-3	1e-3	1e-3	1e-3

E Additional results

E.1 Asymmetric Escape Room

We conducted additional experiments on an asymmetric version of the Escape Room game between two learning agents (A1 and A2) as shown in Figure 8. A1 gets +10 extrinsic reward for exiting a door and ending the game (Figure 8c), but the door can only be opened when A2 pulls a lever; otherwise, A1 is penalized at every time step (Figure 8b). The extrinsic penalty for A2 discourages it from taking the cooperative action (Figure 8a). The global optimum combined reward is +9, and it is impossible for A2 to get positive extrinsic reward. Due to the asymmetry, A1 is the reward-giver and A2 is the reward recipient for methods that allow incentivization. Each agent observes both agents' positions, and can move between the two states available to itself. We allow A1 to observe A2's current action before choosing its own action, which is necessary for methods that learn to reward A2's cooperative actions. We use a standard policy gradient for A2 unless otherwise specified.

In addition to the baselines described for the symmetric case—namely, policy gradient (PG-rewards) and LOLA with discrete "give-reward" actions—we also compare with a two-timescale method, labeled **2-TS**. A 2-TS agent has the same augmented action space as the PG-rewards baseline, except that it learns over a longer time horizon than the reward recipient. Each "epoch" for the 2-TS agent spans multiple regular episodes of the recipient, during which the 2-TS agent executes a fixed policy. The 2-TS agent only caries out a learning update using a final terminal reward, which is the average extrinsic rewards it gets during *test* episodes that are conducted at the end of the epoch. Performance on test episodes serve as a measure of whether correct reward-giving actions were taken to influence the recipient's learning during the epoch. To our knowledge, 2-TS is a novel baseline but has key limitations: the use of two timescales only applies to the asymmetric 2-player game, and requires fast learning by the reward-recipient, chosen to be a tabular Q-learning, to avoid intractably long epochs.



(a) Sum of agent rewards (b) Two PG agents (c) LIO (A1) and PG agent (A2) (d) 1-episode LIO and PG agent

Figure 9: Results in asymmetric 2-player Escape Room. (a) LIO (paired with PG) converges rapidly to the global optimum, 2-TS (paired with tabular Q-learner) converges slower, while policy gradient baselines could not cooperate. (b) Two PG agents cannot cooperate, as A2 converges to "do-nothing". (c) A LIO agent (A1) attains near-optimum reward by incentivizing a PG agent (A2). (d) 1-episode LIO has larger variance and lower performance. Normalization factors are 1/10 (A1) and 1/2 (A2).

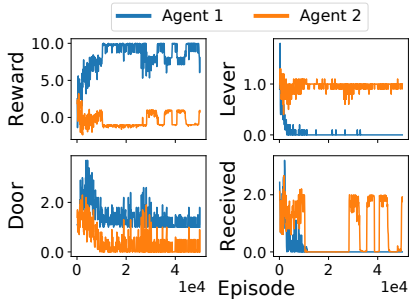
Figure 9 shows the sum of both agents' rewards for all methods on the asymmetric 2-player game, as well as agent-specific performance for policy gradient and LIO, across training episodes. A LIO reward-giver agent paired with a policy gradient recipient converges rapidly to a combined return near 9.0 (Figure 9a), which is the global maximum, while both PG and PG-rewards could not escape the global minimum for A1. LOLA paired with a PG recipient found the cooperative solution in two out of 20 runs; this suggests the difficulty of using a fixed incentive value to conduct opponent shaping via discrete actions. The 2-TS method is able to improve combined return but does so much more gradually than LIO, because an epoch consists of many base episodes and it depends on a highly delayed terminal reward. Figure 9b for two PG agents shows that A2 converges to the policy of not moving (reward of 0), which results in A1 incurring penalties at every time step. In contrast, Figure 9c verifies that A1 (LIO) receives the large extrinsic reward (scaled by 1/10) for exiting the door, while A2 (PG) has average normalized reward above -0.5 (scaled by 1/2), indicating that it is receiving incentives from A1. Average reward of A2 (PG) is below 0 because incentives given by A1 need not exceed 1 continually during training—once A2's policy is biased toward the cooperative action in early episodes, its decaying exploration rate means that it may not revert to staying put even when incentives do not overcome the penalty for moving. Figure 9d shows results on a one-episode version of LIO where the same episode is used for both policy update and incentive function updates, with importance sampling corrections. This version performs significantly lower for A1 and gives more incentives than is necessary to encourage A2 to move. It demonstrates the benefit of learning the reward function using a separate episode from that in which it is applied.

E.2 Symmetric Escape Room

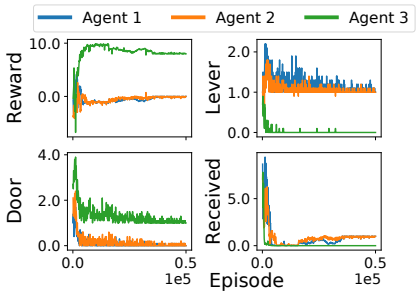
Figure 10 shows total reward (extrinsic + received - given incentives), counts of "lever" and "door" actions, and received incentives in one training run each for ER(2,1) and ER(3,2). In Figure 10a, A1 becomes the winner and A2 the cooperator. It is not always necessary for A1 to give rewards. The fact that LIO models the learning updates of recipients may allow it to find that reward-giving is unnecessary during some episodes when the recipient's policy is sufficiently biased toward cooperation. In Figure 10b, A3 converges to going to the door, as it incentives A1 and A2 to pull the lever.

E.3 Cleanup

Figure 11a is a snapshot of the division of labor found by two LIO agents, whereby the blue agent picks apples while the purple agent stays on the river side to clean waste. The latter does so because of incentives from the former. In contrast, Figure 11b shows a time step where two AC agents compete for apples, which is jointly suboptimal. Figure 11c shows the received incentives during training in the 7x7 map, for each of two LIO agents that were classified after training as a "Cleaner" or "Harvester". Figure 11d shows the incentives given by a "Harvester" agent to three scripted agents during each training checkpoint.

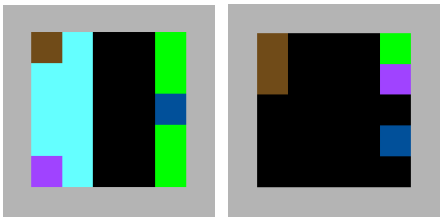


(a) Actions and incentives



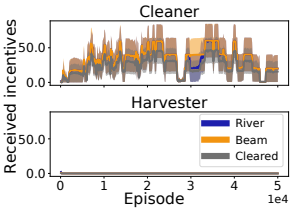
(b) Actions and incentives

Figure 10: Training dynamics: (a) $N = 2$: Agent 1 exits the door by giving incentives to Agent 2. (b) $N = 3$: Agent 3 exits the door by giving incentives to Agents 1 & 2 (1 run).

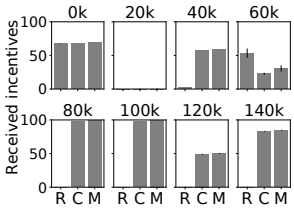


(a) Division of labor

(b) AC agents compete



(c) Cleaner's incentives



(d) 10x10 map

Figure 11: (a) In 7x7 Cleanup, one LIO agent learns to focus on cleaning waste, as it receives incentives from the other who only collects apple. (b) In contrast, AC agents compete for apples after cleaning. (c) Incentives received during training on 7x7 Cleanup. (d) Behavior of incentive function against scripted opponent policies on 10x10 map.