COOPERATION IN MULTI-AGENT REINFORCEMENT LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Jiachen Yang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Machine Learning
School of Computational Science and Engineering

Georgia Institute of Technology

December 2021

COOPERATION IN MULTI-AGENT REINFORCEMENT LEARNING

Thesis committee:

Dr. Hongyuan Zha, Advisor
School of Data Science
Chinese University of Hong Kong, Shenzhen

Dr. Tuo Zhao, Co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Charles Isbell
School of Interactive Computing
Georgia Institute of Technology

Dr. Matthew Gombolay
School of Interactive Computing
Georgia Institute of Technology

Dr. Daniel Faissol
Computational Engineering Division
Lawrence Livermore National Laboratory

Date approved: December 3, 2020

A thousand goals there have been so far, for there have been a thousand peoples. Only the yoke for the thousand necks is still lacking, the one goal is lacking. Humanity still has no goal.

Nietzsche

For a more cooperative world

ACKNOWLEDGMENTS

I acknowledge my advisor Professor Hongyuan Zha for his breadth of knowledge, originality and sharpness of thought, and teaching by example the key to longevity in research; co-advisor Professor Tuo Zhao for his abundant energy, welcoming inclusiveness for my collaboration with his group, and strong professional support; committee member Professor Charles Isbell for his significant role in sharpening my thesis topic and my perspective of the field of multi-agent learning; committee member Professor Matthew Gombolay for pertinent feedback on my research and chairing my thesis defense; committee member Dr. Daniel Faissol for his far-reaching research ideas and outstanding mentorship; Professor Le Song for providing my very first research opportunity in machine learning, which was the origin of this entire journey.

I acknowledge Daniel Faissol and Brenden Petersen for the privilege of close collaboration on highly motivating and far-reaching projects, for their research acumen and sharpness of thought, and for their dedicated, supportive and positive mentorship.

I acknowledge my numerous research collaborators: Tarik Dzanic, Ketan Mittal and Jun Kudo for domain expertise and efficient support that enabled my contribution to our work at the intersection of numerical methods and machine learning; Rakshit Trivedi for great research acumen and critical advice, and Ethan Wang for outstanding technical work, which enabled our work on incentive design to come to fruition; Yan Li for a valuable collaboration between theoretical and experimental work in reinforcement learning; Zhi Zhang for being an inspiring example of dedication to goals and hard work; Professor Xiaojing Ye for mentorship and collaboration on my first research paper; Huan Xu for a short but incredibly impactful discussion that led to my first research breakthrough and subsequent PhD journey; Professor Zsolt Kira for helpful feedback in my doctoral proposal.

I acknowledge my mentors and collaborators at DeepMind: Ang Li for creative discussions on artificial general intelligence, great mentorship, and strong professional support;

Mehrdad Farajtabar for impactful mentorship that enabled me to overcome key challenges, the opportunity to experience work at DeepMind, and providing my first experience with collaboration on machine learning research; Peter Sunehag and Edward Hughes for critical feedback and encouragement that brought our research to fruition.

I acknowledge my numerous mentors during a series of internships: Ahmad Beirami as an outstanding researcher who inspires others through his dedication to high-quality research and mentorship; Igor Borovikov for positive and dedicated mentorship that enabled me to conduct focused research in an industry setting; Alireza Nakhaei and David Isele for the real-world motivation of this thesis topic.

I acknowledge my colleagues in CSE at Georgia Tech, many of whom have already moved on to excellent positions in academia and industry, for a shared lab environment that served as an approximation of a research group: Mehrdad Farajtabar, Rakshit Trivedi, Elias Khalil, Yujia Xie, Xinshi Chen, Yuyu Zhang, Hanjun Dai.

I acknowledge close friends: Rakshit Trivedi who is the epitome of perseverance toward ideals and goals in the face of all challenges; Jeremy Huey, with whom I have the longest history of friendship, who is an excellent sparring partner for philosophical musings and always serves as a reminder that there exists a world outside of academia; Donald Bistri and Trevor Kickliter for a convivial atmosphere as housemates during my graduate studies.

I acknowledge the role of Georgia Tech staff members: William Powell in CSE and Kelley Ross at ISYE for going above and beyond to support my computational work, especially before research deadlines and in remote work during the coronavirus pandemic; Della Phinisee for managing the complications of research subcontracts; Nirvana Edwards and Stephanie Niebuhr for guidance for every milestone during my studies.

I acknowledge my parents Yang Bin and Jia Tan Li for the overwhelmingly positive impact of adversarial training; my paternal grandmother Chen Zhi Xia for demonstrating the process of living life to completion; my maternal grandparents Jia Xi Pu and Yin Hui E for giving me an indelible impression of life in a harmonious home.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As progress in reinforcement learning (RL) gives rise to increasingly general and powerful artificial intelligence, society needs to anticipate a possible future in which multiple RL agents must learn and interact in a shared multi-agent environment. When a single principal has oversight of the multi-agent system, how should agents learn to cooperate via centralized training to achieve individual and global objectives? When agents belong to self-interested principals with imperfectly-aligned objectives, how can cooperation emerge from fully-decentralized learning? This dissertation addresses both questions by proposing novel methods for multi-agent reinforcement learning (MARL) and demonstrating the empirical effectiveness of these methods in high-dimensional simulated environments.

To address the first case, we propose new algorithms for fully-cooperative MARL in the paradigm of centralized training with decentralized execution. Firstly, we propose a method based on multi-agent curriculum learning and multi-agent credit assignment to address the setting where global optimality is defined as the attainment of all individual goals. Secondly, we propose a hierarchical MARL algorithm to discover and learn interpretable and useful skills for a multi-agent team to optimize a single team objective. Extensive experiments with ablations show the strengths of our approaches over state-of-the-art baselines.

To address the second case, we propose learning algorithms to attain cooperation within a population of self-interested RL agents. We propose the design of a new agent who is equipped with the new ability to incentivize other RL agents and explicitly account for the other agents' learning process. This agent overcomes the challenging limitation of fully-decentralized training and generates emergent cooperation in difficult social dilemmas. Then, we extend and apply this technique to the problem of incentive design, where a central incentive designer explicitly optimizes a global objective only by intervening on the rewards of a population of independent RL agents. Experiments on the problem of optimal taxation in a simulated market economy demonstrate the effectiveness of this approach.

CHAPTER 1
INTRODUCTION

Reinforcement Learning (RL) [1] agents are achieving increasing success on an expanding set of complex tasks [2, 3, 4, 5, 6], leading to the belief that RL provides a path toward general Artificial Intelligence (AI) [7]. This is paralleled by a concurrent increase in research effort on multi-agent reinforcement learning (MARL) [8], which extends the concepts of early foundational work [9, 10, 11, 12, 13, 14, 15] with novel algorithms and agent architectures [16, 17, 18, 19] that address the complexity of high-dimensional real-world applications, such as autonomous navigation [20], game AI micromanagement [17, 18], and traffic signal network optimization [21]. Among the many agendas of research on multi-agent learning [13, 15], one of the strongest justifications of MARL rests on its potential real-world applicability to decentralized optimization of a single global objective in complex distributed systems [20, 22, 23], where many agents must act independently based on their own local information but can be trained or influenced by a central entity. In this setting, direct centralized control is often intractable due to high communication costs and the exponentially large joint action space, or simply impossible when autonomous agents do not live within the purview of the central controller. As a contribution to the goal of increasing the potential applicability of MARL in real-world applications, this dissertation shall focus solely on the agenda of achieving cooperation among learning agents in decentralized optimization.

Other agendas in multi-agent learning include empirical modeling of human learning in the presence of other learners, the study of convergence criteria for a given set of interacting learning algorithms, and the design of a best learning strategy for a competitive agent, given a fixed class of opponent agents [13, 15]. These alternative agendas fall outside the scope of this dissertation: the first is orthogonal to the design of learning algorithms for decentralized

optimization, the second belongs to non-cooperative game theory where there may be no global objective that anyone wishes to optimize, while the last takes a selfish single-agent viewpoint that is antithetical to the goal of optimizing a global system-level objective.

The problem of optimizing a global objective in a multi-agent system consists of two distinct cases. In the first case, commonly termed *centralized training with decentralized execution* (CTDE) [24], the multi-agent system falls under the purview of a central principal who can impose a learning algorithm on all agents and provide global information in the training phase. The principal can train agents to optimize the global objective directly, subject to the constraint of decentralized execution of the resulting policies. This setting includes many real-world problems, such as control of a fleet of autonomous vehicles, a network of traffic signals, and a team of AI in team sports games [25, 21, 26]. In the second case, the central principal does not have sufficient power over the agents to conduct centralized training with global information. Nonetheless, one can guide this population of independent learning agents to improve a system-level objective by setting appropriate *incentives* that affect their learning process and promote the emergence of cooperation [27, 28]. If there is no central entity at all, then it is still possible for agents to adopt learning algorithms that conduct mutual incentivization as a means to encourage mutual cooperation, which may result in higher social welfare despite the lack of direct optimization.

1.1 Thesis statement

This dissertation addresses both aforementioned cases of the problem of optimizing a global objective via multi-agent reinforcement learning, by putting forth and substantiating the follow thesis statement:

Multi-agent reinforcement learning enables a collection of independently acting agents to optimize individual and team goals via centralized training in fully-cooperative settings, and generates emergent cooperation that benefits collective welfare via decentralized and centralized incentivization in mixed-motive settings.

This thesis is substantiated by the design of new multi-agent learning algorithms that are empirically evaluated in high-dimensional simulations of cooperative navigation, autonomous driving, a team sports game, social dilemmas, and a market economy with taxation.

1.2 Contributions

We begin with fully-cooperative MARL in Chapter 3. Here, we have complete oversight of the training process and the goal is to design a learning algorithm—in the paradigm of centralized training with decentralized execution [29]—for agents to maximize a single system-level objective. Specifically, we focus on the case that the system-level objective is defined as achieving the collection of all individual agent goals. This *multi-goal* multi-agent control problem arises in many real-world scenarios that require cooperation among multiple autonomous agents. In autonomous driving, multiple vehicles must execute cooperative maneuvers when their individual goal locations and nominal trajectories are in conflict (e.g., double lane merges) [20]. In social dilemmas, mutual cooperation has higher global payoff but agents' individual goals may lead to defection out of fear or greed [30]. Even settings with a global objective that seem unfactorizable can be formulated as multi-goal problems: for example, in traffic flow optimization, different intersection controllers may have local throughput goals but must cooperate for high global performance [21]. We propose a new multi-agent curriculum learning approach to address the challenge of multi-agent exploration, and we improve multi-agent credit assignment with a multi-agent policy gradient involving a learned credit function between action-goal pairs. The proposed method, called **CM3** for *Cooperative Multi-goal Multi-stage Multi-agent RL* [19], outperforms the previous state-of-the-art methods on benchmark problems such as cooperative navigation and autonomous driving.

Continuing with fully-cooperative MARL, Chapter 4 tackles the challenge of discovering interpretable and useful skills without domain knowledge in complex team environments via

hierarchical learning. A common approach in contemporary MARL is to conduct centralized training at the level of *primitive* actions [31, 17, 18, 19, 32], which are the actions used in the transition function of the Markov game [10]. However, the design of hierarchical agents who can cooperate at a higher level of abstraction in high-dimensional multi-agent environments using temporally-extended *skills* is still an open topic. By "skill", we mean a policy that is executed for an extended duration and generates distinguishable behavior; more specifically, the policy is conditioned on a latent skill variable and produces trajectories from which the latent variable can be decoded [33, 34]. It is also not clear how a team of agents can *discover* useful skills without hand-crafted reward functions for each skill, and how they should use skills effectively in combination to achieve a team objective. We draw inspiration from real-world training practices in human team sports—where multiple players learn to coordinate the use of individual skills under centralized training by a coach, while each player individually masters the primitive actions required to perform a skill—to design a new method for hierarchical MARL with automatic skill discovery, called **HSD** [35]. Evaluated against state-of-the-art non-hierarchical methods in a stochastic high-dimensional simulation of team sports, HSD discovers interpretable and useful skills while maintaining competitive performance, and it generalizes significantly better to ad-hoc cooperation.

In Chapter 5, we discard the possibility of centralized training and investigate the problem of attaining multi-agent cooperation among selfish agents who independently optimize their individual rewards. While much effort is devoted to single-agent environments and fully-cooperative games, advances in AI research is driving human society toward a likely future in which large numbers of RL agents with imperfectly-aligned objectives must interact and continually learn in a shared multi-agent environment. The option of centralized training with a global reward [17, 31, 18] is excluded as it does not scale easily to large populations and may not be adopted by self-interested parties. On the other hand, the paradigm of decentralized training—in which no agent is designed with an objective to maximize collective performance and each agent optimizes its own set of policy parameters—poses

difficulties for agents to attain high individual and collective return [36]. In particular, agents in many real world situations with mixed motives, such as settings with non-excludable 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 [37, 38, 39]. Whether, and how, independent learning and acting agents can cooperate while optimizing their own objectives is an open question. We propose a new agent design based on the insight that cooperation may emerge without centralization when each agent gives incentives to other learning agents and explicitly accounts for the impact of incentives on other agents' learning process. The proposed agent, called **LIO** for *Learning to Incentivize Other* learning agents [40], provably converges to mutual cooperation in the classic Iterated Prisoner's Dilemma, and significantly outperforms numerous baselines in difficult social dilemma problems. Promisingly, LIO converges near the known global optimum social welfare in a deceptively hard pedagogical example that accentuates the difficulty of incentivization for standard RL approaches.

In Chapter 6, we extend the methods in Chapter 5 to address the problem of incentive design, whereby a central incentive designer aims to optimize a system-level objective solely by providing incentives to a population of agents who selfishly optimize their individual rewards. This formulation has broad applicability to complex population-level problems in domains such as the energy grid [27], ride sharing [41], and tax design [42], and more generally provides a path toward *in silico* experimental economics [43]. We extend the effectiveness of meta-gradient RL for learning an incentive function [40] to the population setting, and we show the applicability of effective loss functions such as TRPO and PPO [44, 45] for meta-gradient optimization. The proposed method converges to known global optima in standard benchmark problems, and it generates significantly higher social welfare than the previous state-of-the-art in a complex high-dimensional economic simulation of market dynamics with taxation.

1.3 Background and notation

Just as Markov Decision Processes (MDPs) [46, 47, 48] provide the formalism for reinforcement learning [1], Markov games [49, 10] provide the formalism for multi-agent reinforcement learning. This section introduces the framework and notation that will be used throughout this document, and reviews a selection of standard algorithms in reinforcement learning that are used in the construction of new methods in this thesis.

Definition 1 (Markov game)**.** A Markov game with N agents is a tuple $\langle N, \mathcal{S}, \{\mathcal{A}^n\}_{i=1}^N, \{R^n\}_{i=1}^N, P, \gamma \rangle$, where S is the global state space, $\mathcal{A}^1, \ldots, \mathcal{A}^N$ are individual action spaces for each agent i, $R^i \colon \mathcal{S} \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^N \mapsto \mathbb{R}$ is an individual reward for agent i, $P \colon \mathcal{S} \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^N \to \Delta(\mathcal{S})$ is the global state transition function (with P_0 denoting the start state distribution), and $\gamma \in (0, 1]$ is a discount factor for the infinite horizon case. Each agent i learns policy $\pi^i \colon \mathcal{S} \to \Delta(\mathcal{A}^i)$, mapping from the state to a distribution over its own action space, to maximize its objective $J^i(\pi^1, \ldots, \pi^N)$, which depends on the policies of all agents. In the case of selfish agents, J^i is expected cumulative discounted individual rewards

$$J^i(\pi^1, \ldots, \pi^N) := \mathbb{E}_{\{a_t^j \sim \pi^j(\cdot|s_t)\}_{j=1}^N, s_t \sim P(\cdot|s_{t-1}, \mathbf{a}), s_0 \sim P_0} \left[\sum_{t=0}^{\infty} \gamma^t R^i(s_t, a_t^1, \ldots, a_t^N) \right] . \quad (1.1)$$

In the fully-cooperative case, all agents share the same team objective

$$J(\pi^1, \ldots, \pi^N) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \sum_{i=1}^{N} R^i(s_t, a_t^1, \ldots, a_t^N) \right] , \quad (1.2)$$

where the expectation, whenever unspecified, is taken with respect to all agents' policies and the transition function.

In some practical scenarios, such as physical spaces with occlusion, each individual agent i only receives a local observation $o^i = O^i(s)$ from an observation function O^i that depends on the global state. Hence, agent i's policy $\pi^i(a^i|o^i)$ is conditioned on the local

observation only. This scenario is more appropriately formalized as a Decentralized Partially Observable MDP (POMDP) [24] and treated by methods that reason about probabilities of the true global state, given a sequence of past observations and actions by the agent. However, dealing with partial observability is tangential to the objective this thesis. Our focus is the issue of cooperation in two cases: 1) centralized training, where the global state is available at training time; 2) the problem of learning to incentivize learning agents, where the agents' policy learning method is predetermined and not up to our design. Nonetheless, wherever appropriate, we shall use the notation o^i for observation and $\pi^i(a^i|o^i)$ for an agent's policy.

Wherever clear from context, we shall use the letters i, j, m, or n to index an arbitrary agent. In the sequel, any quantity in boldface shall denote the collection of all agents' individual quantities of the same type, e.g. the joint action $\mathbf{a} := (a^1, \ldots, a^N)$, the joint observation $\mathbf{o} := (o^1, \ldots, o^N)$, and the joint policy parameter $\boldsymbol{\theta} := (\theta^1, \ldots, \theta^N)$ when policies are parameterized as $\pi^i_{\theta^i}$. We denote the space of agent n policies by Π^n, and the joint policy induced by conditionally independent agent policies by $\boldsymbol{\pi}(\mathbf{a}|s) := \{\pi^n(a^n|s)\}_{n=1}^N$ Let the notation $-n$ denote the indices of all agents except that of agent n (and similarly for other letters i, j, m). Let $\mathbb{E}_{\boldsymbol{\pi}}[\cdot]$ denote the expectation with respect to the joint policy and the Markov transition function (omitted). In the case of parameterized policies $\pi^i_{\theta^i}$, we may use $J^i(\boldsymbol{\theta})$ in place of $J^i(\boldsymbol{\pi})$.

1.3.1 Reinforcement Learning

In this section, we give an overview of single-agent reinforcement learning. We use the same notation established above for Markov games, except that we omit the agent index.

Reinforcement learning is a class of methods that use repeated experience in a given MDP to find an optimal policy π^* that solves the following optimization problem:

$$\max_{\pi} J(\pi) := \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{1.3}$$

Whereas dynamic programming [50] solves (Equation 1.3) using exact knowledge of the transition function P and reward function R, RL algorithms directly use sequences of state-action transitions $(s_0, a_0, s_1, a_1, \dots)$, collected via repeated *episodes* of interaction in the MDP, to estimate an optimal policy. This broadens the applicability of RL to a large variety of complex and high-dimensional applications where the transition function is too complex to be modeled or the state-action spaces are too large for the equations of dynamic programming to apply, such as the game of Go [51] or StarCraft II [5]. In this document, we focus on the class of *model-free* RL methods that do not estimate a model of the transition or reward function. The alternative class of model-based approaches often use the same learning updates as model-free methods [52], except for the use of a learned model to provide additional samples to those collected from actual experience. It is conceivable, but detracts from our focus on multi-agent cooperation, to combine model-based learning with the methods proposed in this work.

The Bellman equations of dynamic programming [48] provide the foundation for RL algorithms, and we shall review them here. The *value function* $V^\pi(s)$ measures the expected return from state s, following the given policy π. It is defined as

$$V^\pi(s) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right] . \tag{1.4}$$

Similarly, *action-value function* $Q^\pi(s, a)$ (i.e., "Q-function") measures the expected return upon taking action a at state s, and following the given policy π thereafter. It is defined as

$$Q^\pi(s, a) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s, a_0 = a \right] . \tag{1.5}$$

For clarity, we provide the counterparts of the value and action-value function for an arbitrary

agent n, under the joint policy $\boldsymbol{\pi}$, in the multi-agent setting:

$$V^{n,\boldsymbol{\pi}}(s) := \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t R^n(s_t, \mathbf{a}_t) \mid s_0 = s\right] \tag{1.6}$$

$$Q^{n,\boldsymbol{\pi}}(s, \mathbf{a}) := \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t R^n(s_t, \mathbf{a}_t) \;\middle|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\right] . \tag{1.7}$$

The value and action value functions satisfy the Bellman expectation equations:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right] \tag{1.8}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[R(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\right] . \tag{1.9}$$

The recursive form of the Bellman equations suggests an iterative method for policy evaluation. Given an estimate of the value function $V_k(s)$ at iteration k, the value estimate at all states s is updated by:

$$V_{k+1}(s) = \mathbb{E}_{\pi}\left[R(s_t, a_t) + \gamma V_k(s_{t+1}) | s_t = s\right] \tag{1.10}$$

While this update rule provably converges in finite MDPs, it requires knowledge of the transition and reward functions. It motivates the class of sample-based value estimation algorithms called *temporal difference* (TD) learning [53], which contains the commonly-used TD(0) update rule

$$V(s_t) \leftarrow V(s_t) + \alpha\left(R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)\right) , \tag{1.11}$$

where $\alpha \in \mathbb{R}_+$ is a learning rate and data is generated by the policy whose value is being estimated. Here, the 1-step return $R(s_t, a_t) + \gamma V(s_{t+1})$ is an example of a TD *target*. TD targets are also commonly-used in reinforcement learning with function approximation, such as *deep* RL [2, 54], where the value function $V^{\pi}_{\theta}(s)$ is parameterized by θ and the value

estimate is improved via semi-gradient descent according to

$$\theta_{k+1} \leftarrow \theta_k - \alpha \mathbb{E}\left[\nabla_{\theta_k}\left(R(s_t, a_t) + \gamma \perp V^{\pi}_{\theta_k}(s_{t+1}) - V^{\pi}_{\theta_k}(s_t)\right)^2\right], \tag{1.12}$$

where $\perp$ denotes the stop-gradient operator.

Aside from estimating the value of a given policy, one may wish to find the optimal policy π^*, which can be extracted from the optimal Q-function $Q^* := \max_{\pi \in \Pi} Q^{\pi}$ by selecting optimal actions via $\operatorname{argmax}_{a \in \mathcal{A}} Q^*(a, s)$. Hence, in MDPs with discrete action spaces, one often uses the Q-learning update rule [55]

$$Q^*_{k+1}(s_t, a_t) = Q^*_k(s_t, a_t) + \alpha\left(R(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q^*_k(s_{t+1}, a) - Q^*_k(s_t, a_t)\right) \tag{1.13}$$

to find an optimal Q-function. As an off-policy method, Q-learning may use transitions generated by any behavioral policy. Using function approximation with parameters θ, Q-learning is commonly implemented by semi-gradient descent according to

$$\theta_{k+1} \leftarrow \theta_k - \alpha \mathbb{E}\left[\nabla_{\theta_k}\left(R(s_t, a_t) + \gamma \perp \max_{a \in \mathcal{A}} Q_{\theta_k}(s_{t+1}, a) - Q_{\theta_k}(s_t, a_t)\right)^2\right] \tag{1.14}$$

The class of policy gradient methods provides a more direct way to learn an optimal policy, b ascending the gradient of the objective $\nabla_\theta J(\theta)$ for a policy π_θ that is parameterized by θ. This is especially useful in continuous action spaces where the maximization step in Q-learning itself is a high-dimensional optimization problem. The policy gradient is [56]

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta\left[\nabla_\theta \log \pi_\theta(a|s)\left(Q^\pi(s, a) - b(s)\right)\right], \tag{1.15}$$

where $b(s)$ is any state-dependent function for variance reduction. This is often implemented in an actor-critic framework [57], whereby the critic is a value function V^π_ϕ or action-value function Q^π_ϕ that is parameterized by ϕ, which are improved with TD-learning.

CHAPTER 2
LITERATURE REVIEW

2.1 Cooperative Multi-Agent Learning

Cooperative multi-agent learning is important since many real-world problems can be formulated as distributed systems in which decentralized agents must coordinate to achieve shared objectives [14]. The multi-agent credit assignment problem arises when agents share a global reward [58]. While credit assignment can be resolved when independent individual rewards are available [59], this may not be suitable for the fully cooperative setting: [60] showed that agents whose rewards depend on the success of other agents can cooperate better than agents who optimize for their own success. In the special case when all agents have a single goal and share a global reward, COMA [17] uses a counterfactual baseline, while [61] employs count-based variance reduction limited to discrete-state environments. However, their centralized critic does not evaluate the specific impact of an agent's action on another's success in the general multi-goal setting. When a global objective is the sum of agents' individual objectives, value-decomposition methods optimize a centralized Q-function while preserving scalable decentralized execution [31, 18, 32], but do not address credit assignment. While MADDPG [16] and M3DDPG [62] apply to agents with different rewards, they do not address multi-goal cooperation as they do not distinguish between cooperation and competition, despite the fundamental difference.

Multi-goal MARL was considered in [63], who analyzed convergence in a special networked setting restricted to fully-decentralized training, while we conduct centralized training with decentralized execution [64]. In contrast to multi-*task* MARL, which aims for generalization among *non-simultaneous* tasks [65], and in contrast to hierarchical methods that *sequentially* select subtasks [66, 67], our decentralized agents must cooperate

concurrently to attain all goals. Methods for optimizing high-level agent-task assignment policies in a hierarchical framework [68] are complementary to our work, as we focus on learning low-level cooperation after goals are assigned. Prior application of curriculum learning [69] to MARL include a single cooperative task defined by the number of agents [70] and the probability of agent appearance [71], without explicit individual goals. [72] instantiate new neural network columns for task transfer in single-agent RL. Techniques in transfer learning [73] are complementary to our novel curriculum approach to MARL.

2.2 Hierarchical Reinforcement Learning

Building on the framework of options, temporally-extended actions, and hierarchical single-agent RL [74, 75, 76, 77], early work on hierarchical MARL in discrete state spaces with hand-crafted subtasks [78, 79] showed that learning cooperation at the level of subtasks significantly speeds up learning over flat methods [9, 10, 80]. Recent work built on deep reinforcement learning [2, 51] to demonstrate hierarchical single-agent RL in high-dimensional continuous state spaces, using predefined subgoals [81], end-to-end learning of options [82], and latent directional subgoals [66] in a two-level hierarchy. In hierarchical MARL, different subtasks are chosen concurrently by all agents, whereas only a single subtask is chosen for each segment in single-agent hierarchical RL [82, 66].

Progress in hierarchical learning benefits from a complementary line of work on automatic subgoal discovery [83]. Our work draws inspiration from variational option discovery [34, 84, 33], which—in formal analogy with variational auto-encoders [85]—trains a maximum-entropy policy encoder to map latent context vectors into trajectories from which the context can be recovered by a supervised decoder. In contrast to prior work on single-agent skill discovery that focus on finding distinguishable behavior in simulated robotics environments, option discovery in cooperative MARL poses significant new demands: 1) individually distinguishable behaviors must be useful for the team objective; 2) hoping to discover useful skills by increasing the number of latent skills is impractical for the

exponentially larger action space of MARL; and 3) skills must be discovered in the actual multi-agent environment rather than in an isolated single-agent setting.

The key differences from recent work in hierarchical MARL [86, 87] are that we discover skills with an intrinsic reward instead of hand-crafting subtask-specific rewards [86], and our agents are on equal footing without a dedicated "Manager" [87]. A concurrent work on MARL with latent skills [88] require fully-centralized execution using global state information, while our method enables decentralized execution with local observations. A complementary line of work learns *role*-specific parameters and assignment of roles to agents with unique features, where each role is sustained for an entire episode [89], while our agents can dynamically choose skills multiple times in an episode. We design our hierarchical agents using QMIX [18] and independent DQN [9, 2]; other decentralized cooperative MARL [8] and single-agent RL [1] algorithms are equally applicable.

2.3 Toward Cooperation in Social Dilemmas

Learning to incentivize other learning agents is motivated by the problem of cooperation among independent learning agents in intertemporal social dilemmas (ISDs) [38], 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 [17, 18, 31] 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 [90, 91, 92, 93, 94], which are either hand-crafted or slowly evolved based on other agents' performance and modulate each agent's own total 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 [95].

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 [96] and SOS [97] 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 subsection 5.2.1, 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 [98, 99, 100]. 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 [101]; employ a centralized operator on utilities in 2-player games with side payments [99]; and directly optimize collective performance by centralized rewarding in 2-player matrix games [98]. 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 [102], which is used to optimize hyperparameters in meta-gradient RL [103], and by the optimal reward framework [104], in which a single agent learns an intrinsic reward by ascending the gradient of its own extrinsic objective [100].

2.4 Incentive, Utility, and Mechanism Design

A large body of previous work on incentive, utility, and mechanism design belongs to the analytic paradigm, which faces limitations such as linear agent cost and planner incentive functions [28, 105], finite single-round games [106], state-based potential games [107], or pertain to special problems such as welfare distribution [108] or seller-buyer auctions [109]. These simplifications result in lower applicability to complex, nonlinear, and temporally-

extended environments such as dynamic economies [42]. In contrast, we adopt the paradigm of agent-based simulation [110] and take state-of-the-art agent learning methods (i.e., deep reinforcement learning, at present) as the starting point to inform a method for incentive design, at the cost of discarding analytical tractability.

Previous work on incentive or mechanism design with RL differ from ours in the choice of the algorithm for the incentive designer or the model of agents. [111, 112] apply RL to the upper-level planner for non-RL agents. [113] use perturbation-based gradient ascent to search for hyperparameters of a k-armed bandit algorithm that determines the parameters of an auction. [114] employ Bayesian optimization and treat the lower-level multi-agent RL as a black-box. The central planner in [98] optimizes social welfare in 2-player matrix games by anticipating the players' one-step updates. [115] assume that agents' total payoff is a continuous and differentiable function of the joint strategy—which does not hold in general if agents' original reward can be any combination of discrete and nondifferentiable rules—and differentiate through the variational inequality reformulation of Nash equilibria. The closest work to ours are [40], where fully-decentralized RL agents learn mutual pairwise incentivization, and [42, 116], where a central RL planner optimizes an adaptive tax or price policy at the same time-scale as RL agents' policy optimization.

The technical aspect of our method builds on single-agent meta-gradient RL [103] and discovering intrinsic rewards [100], which we extend to the multi-agent setting and refine with the principle of online cross-validation [102]. Related to but different from the variety of existing single-agent meta-learning methods enumerated in [117, Table 1], our method learns a general neural network representation of an incentive function within a single lifetime, as opposed to methods that optimize hyperparameters [103, 118, 119], learn target functions [117], or use multiple lifetimes over different environments to find general update functions [120, 121].

CHAPTER 3
COOPERATIVE MULTI-GOAL MULTI-AGENT REINFORCEMENT LEARNING

3.1 Introduction

Many real-world scenarios that require cooperation among multiple autonomous agents are *multi-goal* multi-agent control problems: each agent needs to achieve its own individual goal, but the global optimum where all agents succeed is only attained when agents cooperate to allow the success of other agents. In autonomous driving, multiple vehicles must execute cooperative maneuvers when their individual goal locations and nominal trajectories are in conflict (e.g., double lane merges) [20]. In social dilemmas, mutual cooperation has higher global payoff but agents' individual goals may lead to defection out of fear or greed [30]. Even settings with a global objective that seem unfactorizable can be formulated as multi-goal problems: in Starcraft II micromanagement, a unit that gathers resources must not accidentally jeopardize a teammate's attempt to scout the opponent base [122]; in traffic flow optimization, different intersection controllers may have local throughput goals but must cooperate for high global performance [21]. While the framework of multi-agent reinforcement learning (MARL) [10, 80, 13] has been equipped with methods in deep reinforcement learning (RL) [2, 54] and shown promise on high-dimensional problems with complex agent interactions [16, 123, 17, 124, 125], learning multi-agent cooperation in the multi-goal scenario involves significant open challenges.

First, given that exploration is crucial for RL [126] and even more so in MARL with larger state and joint action spaces, how should agents explore to learn both individual goal attainment and cooperation for others' success? Uniform random exploration is common in deep MARL [8] but can be highly inefficient as the value of cooperative actions may be

discoverable only in small regions of state space where cooperation is needed. Furthermore, the conceptual difference between attaining one's own goal and cooperating for others' success calls for more modularized and targeted approaches. Second, while there are methods for multi-agent credit assignment when all agents share a single goal (i.e., a global reward) [58, 17, 61], and while one could treat the cooperative *multi-goal* scenario as a problem with a single joint goal, this coarse approach makes it extremely difficult to evaluate the impact of an agent's action on another agent's success. Instead, the multi-goal scenario can benefit from fine-grained credit assignment that leverages available structure in action-goal interactions, such as local interactions where only few agents affect another agent's goal attainment at any time.

Given these open challenges, this chapter focuses on the cooperative multi-goal multi-agent setting where each agent is assigned a goal[1] and must learn to cooperate with other agents with possibly different goals. To tackle the problems of efficient exploration and credit assignment in this complex problem setting, we develop CM3, a novel general framework involving three synergistic components:

1. We approach the difficulty of multi-agent exploration from a novel curriculum learning perspective, by first training an actor-critic pair to achieve different goals in an *induced single-agent setting* (Stage 1), then using them to initialize all agents in the multi-agent environment (Stage 2). The key insight is that agents who can already act toward individual objectives are better prepared for discovery of cooperative solutions with additional exploration once other agents are introduced. In contrast to hierarchical learning where sub-goals are selected sequentially in time [75], all agents act toward their goals simultaneously in Stage 2 of our curriculum.

2. Observing that a wide array of complex MARL problems permit a decomposition of agents' observations and state vectors into components of self, others, and non-agent

[1]Goal discovery and assignment are challenges for MARL. However, many practical multi-agent problems have clear goal assignments, such as in autonomous driving and soccer. Our work is specific to known goal assignment and is complementary to methods such as [68] for the unknown case.

specific environment information [8], we employ function augmentation to bridge Stages 1-2: we reduce the number of trainable parameters of the actor-critic in Stage 1 by limiting their input space to the part that is sufficient for single-agent training, then augment the architecture in Stage 2 with additional inputs and trainable parameters for learning in the multi-agent environment.

3. We propose a *credit function*, which is an action-value function that specifically evaluates action-goal pairs, for localized credit assignment in multi-goal MARL. We use it to derive a multi-goal multi-agent policy gradient for Stage 2. In synergy with the curriculum, the credit function is constructed via function augmentation from the critic in Stage 1.

We evaluate our method on challenging multi-goal multi-agent environments with high-dimensional state spaces: cooperative navigation with difficult formations, double lane merges in the SUMO simulator [127], and strategic teamwork in a Checkers game. CM3 solved all domains significantly faster than IAC and COMA [9, 17], and solved four out of five environments significantly faster than QMIX [18]. Exhaustive ablation experiments show that the combination of all three components is crucial for CM3's overall high performance.

3.2 Preliminaries

In multi-goal MARL, each agent should achieve a goal drawn from a finite set, cooperate with other agents for collective success, and act independently with limited local observations. We formalize the problem as an episodic multi-goal Markov game, review an actor-critic approach to centralized training of decentralized policies, and summarize counterfactual-based multi-agent credit assignment.

Multi-goal Markov games. A multi-goal Markov game is a tuple $\langle \mathcal{S}, \{\mathcal{O}^n\}, \{\mathcal{A}^n\}, P, R, \mathcal{G}, N, \gamma \rangle$ with N agents labeled by $n \in [N]$. In each episode, each agent n has one fixed goal $g^n \in \mathcal{G}$ that is known only to itself. At time t

and global state $s_t \in \mathcal{S}$, each agent n receives an observation $o_t^n := o^n(s_t) \in \mathcal{O}^n$ and chooses an action $a_t^n \in \mathcal{A}^n$. The environment moves to s_{t+1} due to joint action $\mathbf{a}_t := \{a_t^1, \ldots, a_t^N\}$, according to transition probability $P(s_{t+1}|s_t, \mathbf{a}_t)$. Each agent receives a reward $R_t^n := R(s_t, \mathbf{a}_t, g^n)$, and the learning task is to find stochastic decentralized policies $\pi^n \colon \mathcal{O}^n \times \mathcal{G} \times \mathcal{A}^n \to [0, 1]$, conditioned only on local observations and goals, to maximize $J(\boldsymbol{\pi}) := \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t \sum_{n=1}^{N} R(s_t, \mathbf{a}_t, g^n)\right]$, where $\gamma \in (0, 1)$ and joint policy $\boldsymbol{\pi}$ factorizes as $\boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) := \prod_{n=1}^{N} \pi^n(a^n|o^n, g^n)$ due to decentralization. Let a^{-n} and g^{-n} denote all agents' actions and goals, respectively, *except* that of agent n. Let boldface $\mathbf{a}$ and $\mathbf{g}$ denote the joint action and joint goals, respectively. For brevity, let $\pi(a^n) := \pi^n(a^n|o^n, g^n)$. This model covers a diverse set of cooperation problems in the literature [8], without constraining how the attainability of a goal depends on other agents: at a traffic intersection, each vehicle can easily reach its target location if not for the presence of other vehicles; in contrast, agents in a strategic game may not be able to maximize their rewards in the absence of cooperators [31].

Centralized learning of decentralized policies. A centralized critic that receives full state-action information can speed up training of decentralized actors that receive only local information [16, 17]. Directly extending the single-goal case, for each $n \in [1..N]$ in a multi-goal Markov game, critics are represented by the value function $V_n^{\boldsymbol{\pi}}(s) := \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t R_t^n \;\middle|\; s_0 = s\right]$ and the action-value function $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) := \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t R_t^n \;\middle|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\right]$, which evaluate the joint policy $\boldsymbol{\pi}$ against the reward R^n for each goal g^n.

Multi-agent credit assignment. In MARL with a single team objective, COMA addresses credit assignment by using a counterfactual baseline in an advantage function $A^n(s, \mathbf{a}) := Q^{\boldsymbol{\pi}}(s, \mathbf{a}) - \sum_{\hat{a}^n} \pi^n(\hat{a}^n|o^n) Q^{\boldsymbol{\pi}}(s, (\hat{a}^n, a^{-n}))$ [17, Lemma 1] , which evaluates the contribution of a chosen action a^n versus the average of all possible counterfactuals $\hat{a}^n$, keeping a^{-n} fixed. The analysis in [128] for a formally equivalent action-dependent baseline in RL suggests that COMA is a low-variance estimator for single-goal MARL. We

derive its variance in Section A.3.1. However, COMA is unsuitable for credit assignment in multi-goal MARL, as it would treat the collection of goals $\mathbf{g}$ as a global goal and only learn from total reward, making it extremely difficult to disentangle each agent's impact on other agents' goal attainment. Furthermore, a global Q-function does not explicitly capture structure in agents' interactions, such as local interactions involving a limited number of agents. We substantiate these arguments by experimental results in Section 3.5.

3.3 Methods

We describe the complete CM3 learning framework as follows. First we define a credit function as a mechanism for credit assignment in multi-goal MARL, then derive a new cooperative multi-goal policy gradient with localized credit assignment. Next we motivate the possibility of significant training speedup via a curriculum for multi-goal MARL. We describe function augmentation as a mechanism for efficiently bridging policy and value functions across the curriculum stages, and finally synthesize all three components into a synergistic learning framework.

3.3.1 Credit assignment in multi-goal MARL

If all agents take greedy goal-directed actions that are individually optimal in the absence of other agents, the joint action can be sub-optimal (e.g. straight-line trajectory towards target in traffic). Instead rewarding agents for both individual and collective success can avoid such bad local optima. A naïve approach based on previous works [17, 16] would evaluate the joint action $\mathbf{a}$ via a global Q-function $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a})$ for each agent's goal g^n, but this does not precisely capture each agent's contribution to another agent's attainment of its goal. Instead, we propose an explicit mechanism for credit assignment by learning an additional function $Q_n^{\boldsymbol{\pi}}(s, a^m)$ that evaluates pairs of action a^m and goal g^n, for use in a multi-goal actor-critic algorithm. We define this function and show that it satisfies the classical relation needed for sample-based model-free learning.

Definition 2. For $n, m \in [N]$, $s \in \mathcal{S}$, the *credit function* for goal g^n and $a^m \in \mathcal{A}^m$ by agent m is:

$$Q_n^{\boldsymbol{\pi}}(s, a^m) := \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R_t^n \;\Big|\; s_0 = s, a_0^m = a^m\Big] \tag{3.1}$$

Proposition 1. *For all $m, n \in [N]$, the credit function* (Equation 3.1) *satisfies the following relations:*

$$Q_n^{\boldsymbol{\pi}}(s, a^m) = \mathbb{E}_{\boldsymbol{\pi}}\big[R_t^n + \gamma Q_n^{\boldsymbol{\pi}}(s_{t+1}, a_{t+1}^m) \;\big|\; s_t = s, a_t^m = a^m\big] \tag{3.2}$$

$$V_n^{\boldsymbol{\pi}}(s) = \sum_{a^m} \pi^m(a^m|o^m, g^m) Q_n^{\boldsymbol{\pi}}(s, a^m) \tag{3.3}$$

Derivations are given in Section A.2.1, including the relation between $Q_n^{\boldsymbol{\pi}}(s, a^m)$ and $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a})$. Equation (Equation 3.2) takes the form of the Bellman expectation equation, which justifies learning the credit function, parameterized by θ_{Q_c}, by optimizing the standard loss function in deep RL:

$$L(\theta_{Q_c}) = \mathbb{E}_{\boldsymbol{\pi}}\Big[\big(R_t^n + \gamma Q_n^{\boldsymbol{\pi}}(s_{t+1}, a_{t+1}^m; \theta_{Q_c}) - Q_n^{\boldsymbol{\pi}}(s_t, a_t^m; \theta_{Q_c})\big)^2\Big] \tag{3.4}$$

While centralized training means the input space scales linearly with agent count, many practical environments involving only *local* interactions between agents allows centralized training with few agents while retaining decentralized performance when deployed at scale (evidenced in Section A.5).

3.3.2 Cooperative multi-goal multi-agent policy gradient

We use the credit function as a critic within a policy gradient for multi-goal MARL. Letting θ parameterize $\boldsymbol{\pi}$, the overall objective $J(\boldsymbol{\pi})$ is maximized by ascending the following gradient:

Proposition 2. *The cooperative multi-goal credit function based MARL policy gradient is*

$$\nabla_\theta J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{m,n=1}^{N} (\nabla_\theta \log \pi^m(a^m|o^m, g^m)) A^{\boldsymbol{\pi}}_{n,m}(s, \mathbf{a})\Big] \tag{3.5}$$

$$A^{\boldsymbol{\pi}}_{n,m}(s, \mathbf{a}) := Q^{\boldsymbol{\pi}}_n(s, \mathbf{a}) - \sum_{\hat{a}^m} \pi^m(\hat{a}^m|o^m, g^m) Q^{\boldsymbol{\pi}}_n(s, \hat{a}^m) \tag{3.6}$$

This is derived in Section A.2.2. For a fixed agent m, the inner summation over n considers all agents' goals g^n and updates m's policy based on the advantage of a^m over all counterfactual actions $\hat{a}^m$, as measured by the credit function for g^n. The strength of interaction between action-goal pairs is captured by the extent to which $Q^{\boldsymbol{\pi}}_n(s, \hat{a}^m)$ varies with $\hat{a}^m$, which directly impacts the magnitude of the gradient on agent m's policy. For example, strong interaction results in non-constant $Q^{\boldsymbol{\pi}}_n(s, \cdot)$, which implies larger magnitude of $A^{\boldsymbol{\pi}}_{n,m}$ and larger weight on $\nabla_\theta \log \pi(a^m)$. The double summation accounts for first-order interaction between all action-goal pairs, but complexity can be reduced by omitting terms when interactions are known to be sparse, and our empirical runtimes are on par with other methods due to efficient batch computation (Section A.6). As the second term in $A^{\boldsymbol{\pi}}_{n,m}$ is a baseline, the reduction of variance can be analyzed similarly to that for COMA, given in Section A.3.2. While $A^{\boldsymbol{\pi}}_{n,m} = Q^{\boldsymbol{\pi}}_n(s, \mathbf{a}) - V^{\boldsymbol{\pi}}_n(s)$ (due to (Equation 3.3)), ablation results show stability improvement due to the credit function (Section 3.5). As the credit function takes in a single agent's action, it synergizes with both CM3's curriculum and function augmentation as described in Section 3.3.5.

3.3.3 Curriculum for multi-goal MARL

Multi-goal MARL poses a significant challenge for exploration. Random exploration can be highly inefficient for concurrently learning both individual task completion and cooperative behavior. Agents who cannot make progress toward individual goals may rarely encounter the region of state space where cooperation is needed, rendering any exploration useless for learning cooperative behavior. On the other extreme, exploratory actions taken in situations

that require precise coordination can easily lead to penalties that cause agents to avoid the coordination problem and fail to achieve individual goals. Instead, we hypothesize and confirm in experiments that agents who first learn to reach individual goals in the absence of other agents can more reliably produce state configurations where cooperative solutions are easily discovered with additional exploration in the multi-agent environment[2].

We propose a MARL curriculum that first solves a single-agent Markov decision process (MDP), as preparation for subsequent exploration speedup. Given a cooperative multi-goal Markov game **MG**, we induce an MDP **M** to be the tuple $\langle \mathcal{S}^n, \mathcal{O}^n, A^n, P^n, R, \gamma \rangle$, where an agent n is selected to be the single agent in **M**. Entities $\mathcal{S}^n$, P^n, and R are defined by removing all dependencies on agent interactions, so that only components depending on agent n remain. The reduction to **M** involves only deletion of components associated with all other agents from state vectors (since an agent is uniquely defined by its attributes), deletion of if-else conditions from the reward function corresponding to agent interactions, and likewise from the transition function if a simulation is used. Section A.7 provides practical guidelines for the reduction on various tasks.

This reduction to **M** is possible in almost all fully cooperative multi-agent environments used in a large body of work[3] [8], precisely because they support a variable number of agents, including $N = 1$. Important real-world settings that allow this reduction include autonomous driving, multi traffic light control, and warehouse commissioning (removing all but one car/controller/robot, respectively, from the environment). If an agent's original reward function provides no learning signal in the absence of other agents, such as the case of a binary reward for a goal whose complete attainment requires at least another agent's presence and help, one may modify the reward via reward shaping [129] to enable learning of good intermediate states along the path to goal attainment, without changing the optimal policy in **M**. Based on **M**, we define a *greedy policy* for **MG**.

[2]We provide a synthetic example to aid intuition in Section A.4

[3]Environments include: discrete 2D worlds, continuous 3D physics simulators, StarCraft II, transportation tasks, 3D first-person multiplayer games, etc. Exceptions are settings where a single task is purely defined by inter-agent communication, but these are not multi-goal Markov games.

Definition 3. A *greedy policy* π^n by agent n for cooperative multi-goal **MG** is defined as the optimal policy π^* for the induced MDP **M** where only agent n is present.

This naturally leads to our proposed curriculum: Stage 1 trains a single agent in **M** to achieve a greedy policy, which is then used for initialization in **MG** in Stage 2. This greedy initialization causes agents to encounter the relevant region of state space reliably across many early episodes of Stage 2, so that cooperative behavior can be discovered more quickly. Next we explain in detail how to leverage the structure of decentralized MARL to bridge the two curriculum stages.

3.3.4 Function augmentation for multi-goal curriculum

In Markov games with decentralized execution, an agent's observation space decomposes into $\mathcal{O}^n = \mathcal{O}^n_{\text{self}} \cup \mathcal{O}^n_{\text{others}}$, where $o^n_{\text{self}} \in \mathcal{O}^n_{\text{self}}$ captures the agent's own properties, which must be observable by the agent for closed-loop control, while $o^n_{\text{others}} \in \mathcal{O}^n_{\text{others}}$ is the agent's egocentric observation of other agents. In our work, egocentric observations are private and not accessible by other agents [130]. Similarly, global state s decomposes into $s := (s_{\text{env}}, s^n, s^{-n})$, where s_{env} is environment information not specific to any agent (e.g., position of a landmark), and s^n captures agent n's information. While this decomposition is implicitly available in a wide range of complex multi-agent environments [131, 17, 16, 18, 132, 3], we explicitly use it to implement our curriculum. In Stage 1, as the ability to process o^n_{others} and s^{-n} is unnecessary, we reduce the input space of policy and value functions, thereby reducing the number of trainable parameters and lowering the computation cost. In Stage 2, we restore Stage 1 parameters and activate new modules to process additional inputs o^n_{others} and s^{-n}. This augmentation is especially suitable for efficiently learning the credit function (Equation 3.1) and global Q-function, since $Q(s, a)$ can be augmented into both $Q^{\boldsymbol{\pi}}_n(s, \mathbf{a})$ and $Q^{\boldsymbol{\pi}}_n(s, a^m)$, as explained below.

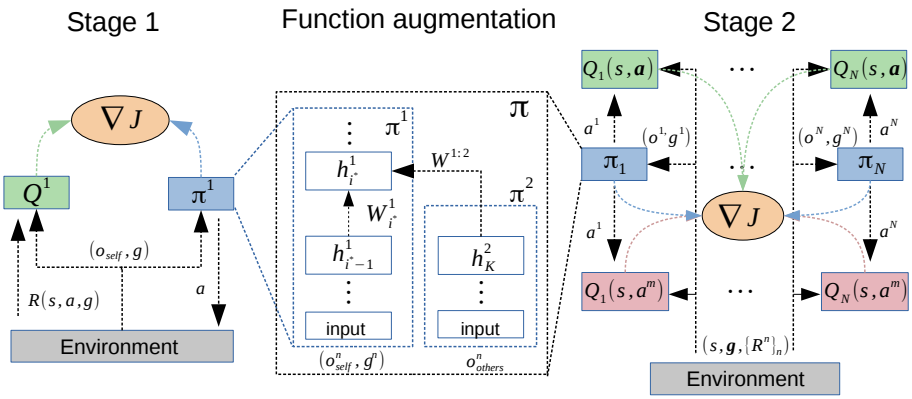


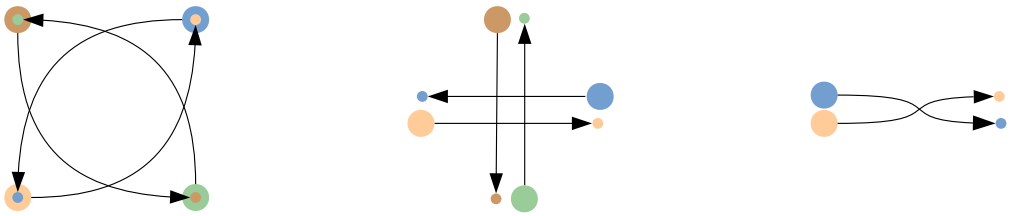
Figure 3.1: In Stage 1, Q^1 and π^1 learn to achieve multiple goals in a single-agent environment. Between Stage 1 and 2, π is constructed from the trained π^1 and a new module π^2; a similar construction is done for $Q_n(s, \mathbf{a})$ and $Q_n(s, a^m)$. In the multi-agent environment of Stage 2, these augmented functions are instantiated for each of N agents (with parameter-sharing).

3.3.5 A complete instantiation of CM3

We combine the preceding components to create CM3, using deep neural networks for function approximation (Figure 3.1 and Algorithm 7). Without loss of generality, we assume parameter-sharing [17] among homogeneous agents with goals as input [133]. The inhomogeneous case can be addressed by N actor-critics. Drawing from multi-task learning [134], we sample goal(s) in each episode for the agent(s), to train one model for all goals.

Stage 1. We train an actor $\pi^1(a|o, g)$ and critic $Q^1(s^1, a, g)$ to convergence according to (Equation 3.4) and (Equation 3.5) in the induced MDP with $N = 1$ and random goal sampling (see Section A.10). This uses orders of magnitude fewer samples than for the full multi-agent environment—compare Figure A.1 with Figure 3.5.

Stage 2. The Markov game is instantiated with all N agents. We restore the trained π^1 parameters, instantiate a second neural network π^2 for agents to process o^n_{others}, and connect the output of π^2 to a selected hidden layer of π^1. Concretely, let $h^1_i \in \mathbb{R}^{m_i}$ denote hidden layer $i \leq L$ with m_i units in an L-layer network π^1, connected to layer $i-1$ via $h^1_i = f(W^1_i h^1_{i-1})$ with $W^1_i \in \mathbb{R}^{m_i \times m_{i-1}}$ and nonlinear activation f. Stage 2 introduces a



(a) Antipodal (b) Cross (c) Merge

Figure 3.2: Cooperative navigation

K-layer network $\pi^2(o^n_{\text{others}})$ with outputs $h^2_K \in \mathbb{R}^{m_K}$, chooses a layer[4] i^* of π^1, and augments $h^1_{i^*}$ to be $h^1_{i^*} = f(W^1_{i^*} h^1_{i^*-1} + W^{1:2} h^2_K)$ with $W^{1:2} \in \mathbb{R}^{m_{i^*} \times m_K}$. Being restored from Stage 1, not re-initialized, hidden layers $i < i^*$ begin with the ability to process (o^n_{self}, g^n), while the new weights in π^2 and $W^{1:2}$ specifically learn the effect of surrounding agents. Higher layers $i \geq i^*$ that already take greedy actions to achieve goals in Stage 1 must now do so while cooperating to allow other agents' success. This augmentation scheme is simplest for deep policy and value networks using fully-connected or convolutional layers.

The middle panel of Figure 3.1 depicts the construction of π from π^1 and π^2. The global $Q^{\boldsymbol{\pi}}(s, \mathbf{a}, g^n)$ is constructed from Q^1 similarly: when the input to Q^1 is $(s_{\text{env}}, s^n, a^n, g^n)$, a new module takes input (s^{-n}, a^{-n}) and connects to a chosen hidden layer of Q^1. Credit function $Q^{\boldsymbol{\pi}}(s, a^m, g^n)$ is augmented from a copy of Q^1, such that when Q^1 inputs are $(s_{\text{env}}, s^n, a^m, g^n)$, the new module's inputs are (s^m, s^{-n}).[5] We train the policy using (Equation 3.5), train the credit function with loss (Equation 3.4), and train the global Q-function with the joint-action analogue of (Equation 3.4).

3.4 Experimental setup

We investigated the performance and robustness of CM3 versus existing methods on diverse and challenging multi-goal MARL environments: cooperative navigation in difficult formations, double lane merge in autonomous driving, and strategic cooperation in a Checkers game. We evaluated ablations of CM3 on all domains. We describe key setup here, with full

[4] Setting i^* to be the last hidden layer worked well in our experiments, without needing to tune.
[5] Input s^m is needed for disambiguation, so that input action a^m is associated with agent m.

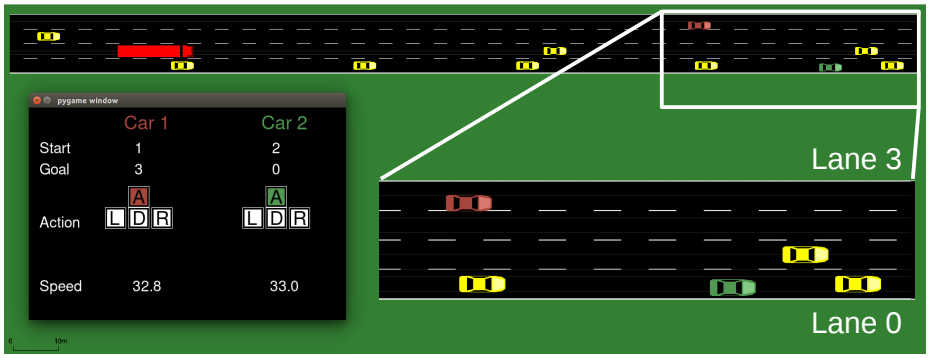


Figure 3.3: Agent sedans must perform double lane merge to reach goal lanes. SUMO controls yellow sedans and trucks. Policy generalization was tested on such traffic conditions.

details in Sections A.7 to A.10[6].

Cooperative navigation: We created three variants of the cooperative navigation scenario in [16], where N agents cooperate to reach a set of targets. We increased the difficulty by giving each agent only an individual reward based on distance to its designated target, not a global team reward, but initial and target positions require complex cooperative maneuvers to avoid collision penalties (Figure 3.2). Agents observe relative positions and velocities (details in Section A.7.1). **SUMO:** Previous work modeled autonomous driving tasks as MDPs in which all other vehicles do not learn to respond to a single learning agent [135, 136]. However, real-world driving requires cooperation among different drivers' with personal goals. Built in the SUMO traffic simulator with sublane resolution [127], this experiment requires agent vehicles to learn double-merge maneuvers to reach goal lane assignments (Figure 3.3). Agents have limited field of view and receive sparse rewards (Section A.7.2).

Checkers: We implemented a challenging strategic game (Section A.7.3, an extension of [31]), to investigate whether CM3 is beneficial even when an agent cannot maximize its reward in the absence of another agent. In a gridworld with red and yellow squares that disappear when collected (Figure 3.4), Agent A receives +1 for red and -0.5 for yellow; Agent B

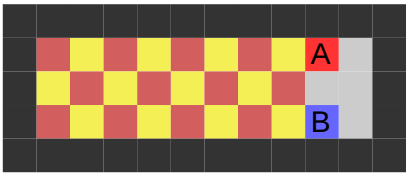


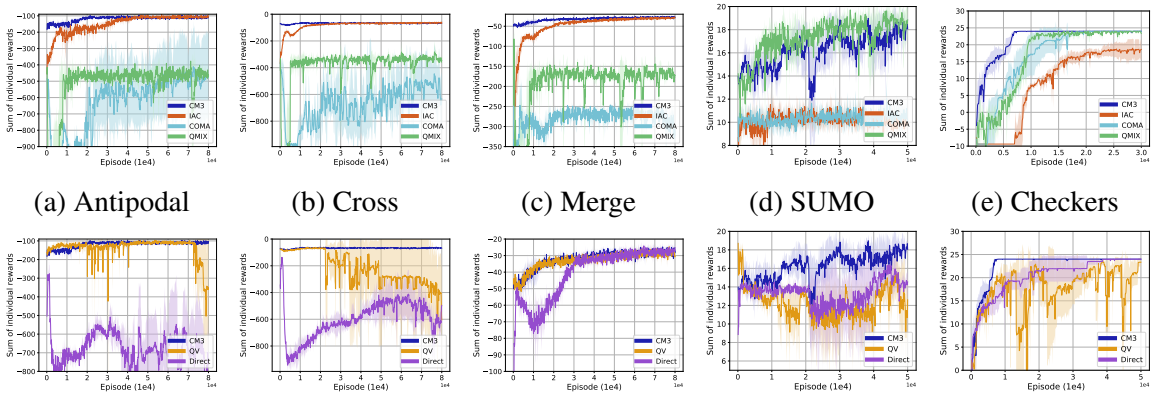
Figure 3.4: Checkers

[6]Code for all experiments is available at https://github.com/011235813/cm3.

receives -0.5 for red and +1 for yellow. Both have a limited 5x5 field of view. The global optimum requires each agent to clear the path for the other.

Algorithm implementations. We describe key points here, leaving complete architecture details and hyperparameter tables to Sections A.8 and A.9. **CM3:** Stage 1 is defined for each environment as follows (Section A.7): in cooperative navigation, a single particle learns to reach any specified landmark; in SUMO, a car learns to reach any specified goal lane; in Checkers, we alternate between training one agent as A and B. Section A.8 describes function augmentation in Stage 2 of CM3. **COMA** [17]: the joint goal $\mathbf{g}$ and total reward $\sum_n R^n$ can be used to train COMA's global Q function, which receives input $(s, o^n, g^n, n, a^{-n}, g^{-n})$. Each output node i represents $Q(s, a^n = i, a^{-n}, \mathbf{g})$. **IAC** [9, 17]: IAC trains each agent's actor and critic independently, using the agent's own observation. The TD error of value function $V(o^n, g^n)$ is used in a standard policy gradient [56]. **QMIX** [18]: we used the original hypernetwork, giving all goals to the mixer and individual goals to each agent network. We used a manual coordinate descent on exploration and learning rate hyperparameters, including values reported in the original works. We ensured the number of trainable parameters are similar among all methods, up to method-specific architecture requirements for COMA and QMIX.

Ablations. We conducted ablation experiments in all domains. To discover the speedup from the curriculum with function augmentation, we trained the full Stage 2 architecture of CM3 (labeled as **Direct**) without first training components π^1 and Q^1 in an induced MDP. To investigate the benefit of the new credit function and multi-goal policy gradient, we trained an ablation (labeled **QV**) with advantage function $A_n^{\boldsymbol{\pi}}(s, \mathbf{a}) := Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) - V_n^{\boldsymbol{\pi}}(s)$, where credit assignment between action-goal pairs is lost. QV uses the same π^1, Q^1, and function augmentation as CM3.



(a) Antipodal (b) Cross (c) Merge (d) SUMO (e) Checkers

(f) Antipodal (g) Cross (h) Merge (i) SUMO (j) Checkers

Figure 3.5: a-e: Comparison against baselines in cooperative navigation (a-c), SUMO (d), Checkers (e). f-j: Comparison against ablations. Average and standard deviation (shaded) of 10 evaluation episodes conducted every 100 training episodes, across 3 independent runs.

3.5 Results and Discussions

CM3 finds optimal or near-optimal policies significantly faster than IAC and COMA on all domains, and performs significantly higher than QMIX in four out of five. We report absolute runtime in Section A.6 and account for CM3's Stage 1 episodes (Section A.10) when comparing sample efficiency.

Main comparison. Over all **cooperative navigation** scenarios (Figures 3.5a to 3.5c), CM3 (with 1k episodes in Stage 1) converged more than 15k episodes faster than IAC. IAC reached the same final performance as CM3 because dense individual rewards simplifies the learning problem for IAC's fully decentralized approach, but CM3 benefited significantly from curriculum learning, as evidenced by comparison to "Direct" in Figure 3.5f. QMIX and COMA settled at suboptimal behavior. Both learn global critics that use all goals as input, in contrast to CM3 and IAC that process each goal separately. This indicates the difficulty of training agents for individual goals under a purely global approach. While COMA was shown to outperform IAC in SC2 micromanagement where IAC must learn from a single team reward [17], our IAC agents have access to individual rewards that resolve the credit assignment issue and improve performance [59]. In **SUMO** (Figure 3.5d), CM3 and QMIX

found cooperative solutions with performances within the margin of error, while COMA and IAC could not break out of local optima where vehicles move straight but do not perform merge maneuvers. Since initial states force agents into the region of state space requiring cooperation, credit assignment rather than exploration is the dominant challenge, which CM3 addressed via the credit function, as evidenced in Figure 3.5i. IAC underperformed because SUMO requires a longer sequence of cooperative actions and gave much sparser rewards than the "Merge" scenario in cooperative navigation. We also show that centralized training of merely two decentralized agents allows them to generalize to settings with much heavier traffic (Section A.5). In **Checkers** (Figure 3.5e), CM3 (with 5k episodes in Stage 1) converged 10k episodes faster than COMA and QMIX to the global optimum with score 24. Both exploration of the combinatorially large joint trajectory space and credit assignment for path clearing are challenges that CM3 successfully addressed. COMA only solved Checkers among all domains, possibly because the small bounded environment alleviates COMA's difficulty with individual goals in large state spaces. IAC underperformed all centralized learning methods because cooperative actions that give no instantaneous reward are hard for selfish agents to discover in Checkers. These results demonstrate CM3's ability to attain individual goals and find cooperative solutions in diverse multi-agent systems.

Ablations. The significantly better performance of CM3 versus "Direct" (Figures 3.5f to 3.5j) shows that learning individual goal attainment prior to learning multi-agent cooperation, and initializing Stage 2 with Stage 1 parameters, are crucial for improving learning speed and stability. It gives evidence that while global action-value and credit functions may be difficult to train from scratch, function augmentation significantly eases the learning problem. While "QV" initially learns quickly to attain individual goals, it does so at the cost of frequent collisions, higher variance, and inability to maintain a cooperative solution, giving clear evidence for the necessity of the credit function.

3.6 Summary

We presented CM3, a general framework for cooperative multi-goal MARL. CM3 addresses the need for efficient exploration to learn both individual goal attainment and cooperation, via a two-stage curriculum bridged by function augmentation. It achieves local credit assignment between action and goals using a credit function in a multi-goal policy gradient. In diverse experimental domains, CM3 attains significantly higher performance, faster learning, and overall robustness than existing MARL methods, displaying strengths of both independent learning and centralized credit assignment while avoiding shortcomings of existing methods. Ablations demonstrate each component is crucial to the whole framework.

CHAPTER 4

HIERARCHICAL COOPERATIVE MARL WITH SKILL DISCOVERY

4.1 Introduction

Fully cooperative multi-agent reinforcement learning (MARL) is an active area of research [14, 8] with a diverse set of real-world application, which include autonomous navigation [20], game AI micromanagement [17, 18], and traffic network optimization [21]. A unique challenge is the need for centralized training for agents to find global optimal cooperative policies, while ensuring scalable decentralized execution whereby agents choose actions independently. In this paradigm of centralized training with decentralized execution [29], a common approach [31, 17, 18, 19, 32] is to conduct centralized training at the level of *primitive* actions, which are the actions used in the transition function of the Markov game [10]. However, the design of hierarchical agents who can cooperate at a higher level of abstraction using temporally-extended *skills* in high-dimensional multi-agent environments is still an open question. A skill is a policy that is conditioned on a latent variable, executed for an extended duration, and generates behavior from which the latent variable can be decoded [33, 34]. It is also not clear how multiple agents can *discover* skills without hand-crafted reward functions for each skill, and how to construct such hierarchical policies to allow human interpretation of skills for potential human-AI cooperation.

In this chapter, we take a hierarchical approach to fully cooperative MARL and address these questions by drawing inspiration from team sports games. At the team level, coaches train human players to execute complementary skills in parallel, such as moving to different field positions in a formation, as well as effective sequences of skills over time, such as switching between offensive and defensive maneuvers when ball possession changes. At the individual level, each player learns a sequence of primitive actions to execute

a chosen skill. Hierarchical approaches inspired from such real-world practices have several benefits for fully cooperative MARL. From an algorithmic viewpoint, a hierarchical decomposition in two key dimensions—over agents, and across time—simultaneously addresses both the difficulty of learning cooperation at the level of noisy low-level actions in stochastic environments and the difficulty of long-term credit assignment due to highly-delayed rewards (e.g., scoring a goal in football) [79, 66]. Hierarchical approaches may also reduce computational complexity [75] to address the exponential increase in sample complexity with number of agents in MARL. From the viewpoint of human-AI cooperation, which has near-term application to video game AI to improve human players' experiences [137], hierarchical policies trained with explicit skills is a key step toward interpretable and modular policies. In this work, we take interpretability to mean the decodability of a latent skill from an agent's observed behavior—i.e., a policy is interpretable if it produces events and actions in a consistent or distinguishable manner. While a flat policy is a black-box, since the action output is purely determined by the agent's observation input, the modularity of hierarchical models also provides an entry point for external control over the skills executed by AI teammates (e.g., execute the offense skill when it observes a human teammate doing so).

However, decomposing a global team objective such as "scoring a goal" into many sub-objectives for training a collection of skills is extremely difficult without expert knowledge, which may be hard to access for complex settings such as competitive team sports. Manually crafting reward functions for each skill in high-dimensional state spaces involving numerous agents is also prone to misspecification and cause unintended behavior [138]. Instead, we investigate a method for hierarchical agents in MARL to discover and learn a set of high-level latent skills. Agents should learn to cooperate by choosing effective combinations of skills with their teammates, and also dynamically choose skills in response to the state of the game. In contrast to prior work in single-agent settings, where motion skills were discovered purely via an intrinsic reward [34, 33], MARL poses significant new challenges

for skill discovery. Merely discovering distinguishable individual motion in an open-ended multi-agent environment may be useless for a team objective. While increasing the number of skills increases the chance that some are useful for a task [33], doing so in the hierarchical multi-agent setting means exponentially increasing the size of a joint high-level action space and will exacerbate the difficulty of learning.

We present a method for training hierarchical policies with unsupervised skill discovery in cooperative MARL, with the following key technical and experimental contributions. 1) We construct a two-level hierarchical agent for MARL by defining a high-level action space as a set of latent variables. Each agent consists of a high-level policy that chooses and sustains a latent variable for many time steps, and a low-level policy that uses both its observation and the selected latent variable to take primitive actions. 2) We use an extrinsic team reward to conduct centralized training of high-level policies for cooperation, while we use a combination of an intrinsic reward and the team reward to conduct decentralized training of low-level policies with independent reinforcement learning (RL). This allows the use of powerful and general algorithms for cooperative MARL and single-agent RL to train high- and low-level policies, respectively. 3) We define the intrinsic reward as the performance of a decoder that predicts the ground truth latent variable from trajectories generated by low-level policies that were conditioned on the latent variables. By dynamically weighting the intrinsic versus extrinsic reward, each low-level policy is trained to reach a balance between decodability and usefulness—it executes a skill, without the need for hand-designed skill-specific reward functions. 4) We applied this algorithm to a highly stochastic continuous state simulation of team sports and performed a detailed quantitative investigation of the learned behaviors. Agents discover useful skills, that affect game events and determine low-level actions in distinct and interpretable ways, such as grouping together to steal possession from an opponent. They learn to choose complementary skills among the team, such as when one agent camps near the opponent goal to get a rebound when its teammate makes a long-range shot attempt. 5) Our hierarchical agents perform higher

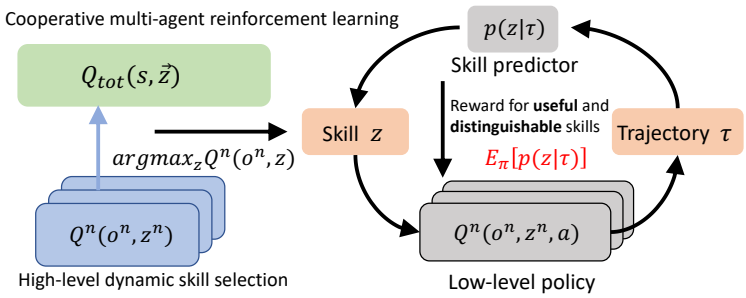


Figure 4.1: Hierarchical MARL with unsupervised skill discovery. At the high level (left), the extrinsic team reward is used to train a centralized action-value function $Q_{\text{tot}}(s, \mathbf{z})$ that decomposes into individual utility functions $Q^n(o^n, z^n)$ for decentralized selection of latent skill variables z. At the low level (right), skill-conditioned action-value functions $Q^n(o^n, z^n, a^n)$ take primitive actions independently. Trajectories τ generated under each z are collected into a dataset $\mathcal{D} = \{(z, \tau)\}$, which is used to train a skill decoder $p(z|\tau)$ to predict z from τ. The probability of selected skills under $p(z|\tau)$ is the intrinsic reward for low-level Q^n.

than flat methods in ad-hoc cooperation when matched with teammates who follow policies that were not encountered in training. This is an encouraging result for the possibility of human-AI cooperation.

4.2 Methods

We present a method for fully-cooperative hierarchical MARL, whereby independently-acting agents learn to cooperate using latent skills that emerge from a combination of intrinsic and extrinsic rewards. Inspired by training practices of real world professional sports teams, we create our method within the paradigm of centralized training with decentralized execution [29]. For ease of exposition and intuition, we assume all agents have the same observation space and action space; nevertheless they take individual actions based on individual observations. In the rest of this section, we define the objective of hierarchical MARL with skill discovery, describe our method to solve the optimization problem, and discuss practical implementation techniques for effective learning.

4.2.1 Combining centralized and decentralized training in hierarchical MARL

We describe a two-level hierarchical MARL setup for training N agents, labeled by $n \in [N]$, as follows. Let $\mathcal{Z}$ denote a set of latent variables z, each of which corresponds to a *skill*. In this work, we use a finite set of latent variables with one-hot encoding; it is possible to generalize $\mathcal{Z}$ to be a learned continuous embedding space [34]. We treat $\mathcal{Z}$ as the *action space* for high-level policies[1] $\mu^n \colon \mathcal{O} \mapsto \mathcal{Z}, \forall n \in [N]$, each of which maps from an agent's observation $o^n \in \mathcal{O}$ to a choice of skill $z^n \in \mathcal{Z}$. Each choice of z^n is sustained for t_{seg} time steps: letting $T = Kt_{\text{seg}}$ denote the length of an episode, there are K time points at which a high-level skill selection is made (see Section 4.2.4). Conditioned on a chosen latent skill and given an agent's observation, a low-level policy $\pi^n \colon \mathcal{O} \times \mathcal{Z} \mapsto \mathcal{A}$ outputs a primitive action a^n in a low-level action space $\mathcal{A}$. Each $z \in \mathcal{Z}$ and the latent-conditioned policy $\pi^n(\cdot; z^n)$ is a skill, in accord with terminology in the literature [84, 33, 34]. Let boldface $\boldsymbol{\mu}, \boldsymbol{\pi}$, and $\mathbf{a}$ denote the joint high-level policy, joint low-level policy, and joint action, respectively. Let $(\cdot)^{-n}$ denote a joint quantity for all agents except agent n. At the high level, $\boldsymbol{\mu}$ learns to select skills to optimize an extrinsic team reward function $R \colon \mathcal{S} \times \{\mathcal{A}\}_{n=1}^N \mapsto \mathbb{R}$ that maps global state and joint action to a scalar reward. At the low level, $\{\pi^n\}_{n=1}^N$ learn to choose primitive actions to produce useful and decodable behavior by optimizing a low-level reward function R_L. Combining the learning at both levels, we view hierarchical MARL as a bilevel optimization problem [139]:

$$\max_{\boldsymbol{\mu}, \boldsymbol{\pi}} \mathbb{E}_{\mathbf{z} \sim \boldsymbol{\mu}, P} \left[\mathbb{E}_{s_t, a_t \sim \boldsymbol{\pi}, P} \left[\sum_{t=1}^{T} \gamma^t R(s_t, \mathbf{a}_t) \right] \right] \tag{4.1}$$

$$\pi^n \in \operatorname*{argmax}_{\pi} \mathbb{E}_{\mathbf{z} \sim \boldsymbol{\mu}, P} \left[\sum_{k=1}^{K} \mathbb{E}_{\tau_k^n \sim \boldsymbol{\pi}, P} \left[R_L(z_k^n, \tau_k^n) \right] \right], \forall n \in [N] \tag{4.2}$$

where τ_k^n is the k-th trajectory segment that consists of a sequence of observations by agent n, P denotes the environment transition probability $P(s_{t+1}|s_t, \mathbf{a})$, and $R_L(z^n, \tau^n) :=$

[1] Without loss of generality, and for consistency with our algorithm implementation below, we use the notation for deterministic policies in this chapter.

$\sum_{(s_t,a_t)\in\tau^n} R_L(z^n, s_t, a_t)$ denotes the sum of agent n's low-level rewards along trajectory τ^n. This may also be viewed as a general-sum meta-game between a $\boldsymbol{\mu}$-player and another $\boldsymbol{\pi}$-player. When R_L is the extrinsic team reward, we have a fully-cooperative meta-game, while the other extreme is where R_L solely promotes decodability. Our approach, explained in Section 4.2.2, lies in between these extremes to strike a balance between usefulness and decodability.

It is difficult to solve (Equation 4.1)-(Equation 4.2) exactly in high-dimensional continuous state spaces. Furthermore, we adjust R_L dynamically to promote skill predictability (see Section 4.2.2). Instead, we approach it using powerful algorithms for MARL and RL. First, we use centralized MARL algorithms to train high-level policies $\boldsymbol{\mu}$ for cooperative high-level skill selection. While cooperative behavior may emerge from flat policies trained by a team reward [132], explicitly training high-level skill-selection policies allows external control over the choice of skills performed (by fixing a latent variable), and subsequent analysis of the behavior for each skill. Second, we apply independent RL to train low-level policies $\{\pi^n\}_{n=1}^N$, each conditioned on a skill selected by the agent's corresponding high-level policy, to take primitive actions to optimize $R_L(z^n, \tau^n)$ (defined below in Section 4.2.2). This reflects the fact that human players in team sports can master skills individually outside of team practice.

4.2.2 Skill discovery via dynamically weighted decoder-based intrinsic rewards

We define the low-level reward by first introducing a skill decoder $p_\psi(z^n|\tau^n)$ that predicts the ground truth latent skill z^n that was used in the low-level policy $\pi(\cdot; z^n)$ that generated the trajectory τ^n. The decoder is trained using a dataset $\mathcal{D} = \{(z, \tau)\}$ of skill-trajectory pairs, where each consists of the z chosen by a high level policy and the corresponding trajectory τ generated by the low level policy given z, over all agents. $\mathcal{D}$ is accumulated in an online manner during training. Hence, training p_ψ alone can be viewed as a supervised learning problem where we have access to the ground truth "label" z associated with each

"datapoint" τ.

We define the intrinsic reward $R_I(z_k^n, \tau_k^n)$ for agent n's k-th trajectory segment τ_k^n via the prediction performance of the skill decoder on the tuple (z_k^n, τ_k^n). Agent n receives this scalar reward upon generating the segment τ_k^n. The key intuition is that a skill in many complex fully-cooperative team games can be inferred from the trajectory of primitive actions that implement the skill [140, 8]. For example, any agent who executes a defensive subtask in soccer will move toward opponents in a consistent way that mainly depends on its own observations, with only weak dependence on the behavior of other physically distant agents[2]. This intrinsic reward encourages the generation of distinguishable behavior for different skills, since only by doing so can the low-level policy produce sufficiently distinct "classes" in the dataset $\mathcal{D}$ for the decoder to achieve high prediction performance. Hence we define the low-level reward R_L as a combination of team reward R and intrinsic reward R_I:

$$R_L(z^n, \tau^n) := \alpha \sum_{s_t, \mathbf{a}_t \in \tau^n} \gamma^t R(s_t, \mathbf{a}_t) + (1-\alpha) R_I(z^n, \tau^n) \tag{4.3}$$

$$\text{where } R_I := p_\psi(z^n | \tau^n) \tag{4.4}$$

$\alpha \in \mathbb{R}$ is a *dynamic* weight (specified below) that determines the amount of intrinsic versus environment reward. In contrast to prior work on single-agent option discovery that do not use an extrinsic reward [84, 33, 34], we take advantage of the team reward in MARL to guarantee that skills are useful for team performance, and rely on the intrinsic reward only to promote the association of latent variables with predictable behavior. This ensures that low-level policies, when conditioned on different latent variables, produce trajectories that are 1) sufficiently different to allow decoding of the latent variable, and 2) useful for attaining the true game reward—e.g. "attack opponent net" and "defend own net". We decrease α from 1.0 to α_{end} via an automatic curriculum in which α decreases by α_{step} only

[2] As a first step, we do not include higher-order skills that involve coordinated behavior of two or more agents. Our method can be extended to higher-order skills by associating multiple agents' concurrent trajectories with a single skill.

when the performance (e.g., win rate) in evaluation episodes, conducted periodically during training, exceeds a threshold $\alpha_{\text{threshold}}$. At high α, low-level policies learn independently to maximize the team reward by taking useful actions, some of which can be composed into interpretable behavior. As α decreases and the skill decoder associates trajectories with latent variables, the low-level policy is increasingly rewarded for generating easily decodable modes of behavior when conditioned on different z. A high $\alpha_{\text{threshold}}$ can be more suitable for highly stochastic games (see Section 4.4.2), so that the weight on the intrinsic reward increases later during training, after agents have learned to take useful actions.

4.2.3 Algorithm

Algorithm 1 is our approach to the optimization problem eqs. (4.1) and (4.2), with skill discovery based on eq. (4.3). We initialize replay buffers $\mathcal{B}_H, \mathcal{B}_L$ for both levels of the hierarchy, for off-policy updates in similar style to DQN [2], and initialize a dataset $\mathcal{D}$ for the decoder (line 2). At the k-th high-level step, which occurs once for every t_{seg} primitive time steps (line 6), we compute the SMDP reward $\tilde{R}_t := \sum_{i=0}^{t_{\text{seg}}-1} \gamma^i R(s_{t-i}, \mathbf{a}_{t-i})$ for the high-level policy (line 8) [75]. Each agent computes its reward and independently selects a new skill to execute for the next high-level step (lines 12-13). We periodically take gradient steps to optimize the high level cooperative skill-selection objective (Equation 4.1) (lines 15-17), using QMIX [18] to train a centralized Q-function $Q_\phi^{\text{tot}}(s_t, \mathbf{z})$ via minimizing the loss:

$$\mathcal{L}(\phi) := \mathbb{E}_{\boldsymbol{\mu},\boldsymbol{\pi}} \left[\frac{1}{2} \left(y_k - Q_\phi^{\text{tot}}(s_k, \mathbf{z}_k) \right)^2 \right] \tag{4.5}$$

$$y_k := \tilde{R}_k + \gamma Q_\phi^{\text{tot}}(s_{k+1}, \mathbf{z}') \big|_{\{z'^n = \text{argmax}_{z^n} Q_\phi^n(o_{k+1}^n, z^n)\}_{n=1}^N} \tag{4.6}$$

Q_ϕ^{tot} is a non-linear function (e.g., neural network) that is monotonic in individual utility functions $Q_\phi^n, n \in [N]$, and we denote $\boldsymbol{\mu}$ as the collection of greedy policies induced by Q_ϕ^n. The hypernetwork of QMIX enforces $\partial Q_\phi^{\text{tot}} / \partial Q_\phi^n > 0$, which is a sufficient condition

for a global argmax to be achieved via decentralized argmax, i.e., $\mathrm{argmax}_{\mathbf{z}} Q_\phi^{\text{tot}}(\cdot, \mathbf{z}) = \{\mathrm{argmax}_{z^n} Q_\phi^n(\cdot, z^n)\}_{n=1}^N$. This allows centralized training with decentralized skill selection. In general, one can choose from a diverse set of cooperative MARL algorithms with decentralized execution [17, 31, 32, 19].

Conditioned on the choices of skills, each agent independently executes primitive actions at every low-level time step (lines 19-20), using the greedy policy π^n induced by low-level Q-functions $Q_\theta^n(o_t^n, z_t^n, a^n)$. We periodically take gradient steps to optimize the low level objective (Equation 4.2) (lines 23-25), by using independent DQN [9, 55, 2] to optimize Q_θ^n via minimizing the loss:

$$\mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{\mu},\boldsymbol{\pi}} \left[\frac{1}{2} \left(y_t^n - Q_\theta^n(o_t^n, z^n, a_t^n) \right)^2 \right] \tag{4.7}$$

$$y_t^n := R_L(z^n, \tau^n) + \gamma \max_{a^n} \hat{Q}_\theta^n(o_{t+1}^n, z^n, a^n), \forall n \in [N] \tag{4.8}$$

$\boldsymbol{\pi}$ denotes the collection of greedy policies induced by all Q_θ^n. The low level reward R_L includes the contribution of the intrinsic reward R_I only at the final time step of each length-t_{seg} trajectory segment, i.e., at every high-level step. $\hat{Q}$ is a target network [2].

Once N_{batch} number of (z^n, τ^n) are collected into the dataset $\mathcal{D}$ (lines 11, 27-29), the skill decoder $p_\psi(z|\tau)$ is trained to predict z given τ via supervised learning on $\mathcal{D}$ by minimizing a standard cross-entropy loss. Each chosen z^n acts as the class label for the corresponding trajectory τ^n. Periodically, we evaluate the agents' performance (e.g., win rate) in seperate evaluation episodes; if performance exceeds $\alpha_{\text{threshold}}$, we decrease the weight α by α_{step} with lower bound α_{end} (Section 4.2.2). While it is extremely challenging to provide theoretical guarantees for hierarchical methods, especially due to the need for nonlinear function approximation to tackle high-dimensional continuous state spaces, simultaneous optimization in hierarchical RL has shown promising practical results [82, 66].

4.2.4 Trajectory segmentation and compression

Hierarchical MARL requires agents to change their choice of skills dynamically at multiple times within an episode, such as in response to a change of ball possession in soccer. This means we use partial segments instead of full episode trajectories for skill discovery, in contrast to the single-agent case [84, 33, 34]. At first glance, using a fixed time discretization hyperparameter t_{seg} for segmentation may pose difficulties for the skill decoder, such as when a segment contains qualitatively different behavior that should correspond to different skills. We address this issue by using the time points at which the high-level policy chooses a new set of skill assignments as the segmentation. Hence, $\boldsymbol{\pi}$ learns to generate trajectory segments in between the time points, and p_ψ learns to associate these segments with the chosen latent variables. We synchronize the time points of all agents' high-level skill choice, and all skills are sustained for t_{seg} low-level steps. This corresponds to a special case of the "any" termination scheme, which is dominant over other termination schemes considered in [141]. A practical approach is to define a range of values based on domain knowledge (e.g., average duration of a player's ball possession) and include it in hyperparameter search. Agents can still learn skills that require more than t_{seg} steps, by sustaining the same skill for multiple high-level steps.

Building on [34], we preprocess each trajectory before using it as input to the decoder. We downsample by retaining every k_{skip} steps, which filters out low-level noise in stochastic environments. We use the element-wise difference between the downsampled observation vectors. This discourages the possibility that more than one skill exhibits stationary behavior (e.g., camping at different regions of a field), as the difference will be indistinguishable for the decoder and result in low intrinsic reward. We reduce the dimension of observation vectors for the decoder by removing entries corresponding to all other agents, while retaining game-specific information (e.g., ball possession). Hence an agent's own trajectory must contain enough information for decoding the latent skill variable.

4.3 Experimental Setup

Our experiments demonstrate that the proposed method discovers interpretable skills that are useful for high-level strategies and has potential for human-AI cooperation in team sports games[3]. We contribute evidence that hierarchical MARL with unsupervised skill discovery can meet or exceed the performance of non-hierarchical methods in high-dimensional environments with only a global team reward. We describe the simulation setup in Section 4.3.1 and provide full implementation details of all methods in Section 4.3.2.

4.3.1 Simple Team Sports Simulator

The Simple Team Sports Simulator (STS2) captures the high-level rules and physical dynamics of general N versus N team sports while abstracting away fine-grained details that do not significantly impact strategic team play [142, 26]. Stochasticity of ball possession and goals makes STS2 a challenging environment for MARL. Complementary to 3D simulations such as [143] that require massively parallelized training, STS2 is a lightweight benchmark where MARL agents can outperform the scripted opponent team within hours on a single CPU. We train in 3v3 mode against the scripted opponent team for 50k episodes. Each episode terminates either upon a goal or a tie at 500 time steps.

State. We define a state representation that is invariant under 180 degree rotation of the playing field and switch of team perspective. For one team, the state vector has the following components, making up total dimension 34: normalized position of the player with possession relative to the goal, and its velocity; a 1-hot vector indicating which team or opponent player has possession; for each team and opponent player, its normalized position and velocity.

Observation. Each agent has its own egocentric observation vector with the following components, making up total dimension 31: normalized position and velocity of the player with possession relative to this agent; a binary indicator of whether this agent has possession;

[3]Code for experiments is available at https://github.com/011235813/hierarchical-marl

a binary indicator of whether its team has possession; its normalized position and its velocity; relative normalized position of each teammate, and their relative velocities; a binary indicator of whether the opponent team has possession; relative normalized position of each opponent player, and their relative velocities.

Action. The low-level discrete set of actions consists of: do-nothing, shoot, pass-1, ... , pass-N, down, up, right, left. Movement and shoot directions are relative to the team's field side. If the agent does not have possession and attempts to shoot or pass, or if it has possession and passes to itself, it is forced to do nothing.

Reward. The team receives reward $+1$ for scoring, -1 when the opponent scores, ± 0.1 on the single step when it regains possession from, or loses possession to, the opponent. We include a reward of $\pm 1/(2 * \text{max steps per episode})$ for having or not having possession.

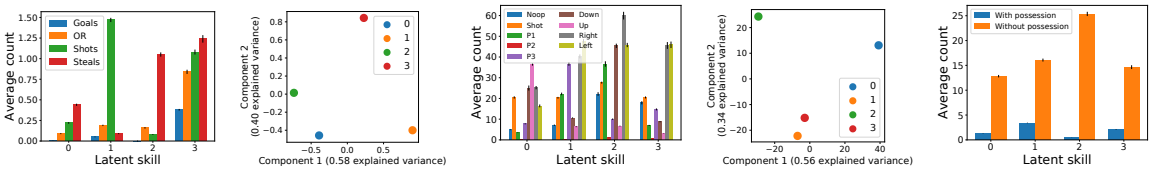
Game events. We define a set of game events, which are frequently used for analyzing team sports [144], to quantify the effect of skills. Goals: agent scored a goal, upon which an episode ends. Offensive rebound: agent's team made a shot attempt, which missed, and the agent retrieved possession. Shot attempts: agent attempted to score a goal. Made or received pass: agent made (received) a successful pass to (from) a teammate. Steals: agent retrieved possession from an opponent by direct physical contact.

4.3.2 Implementation and baselines

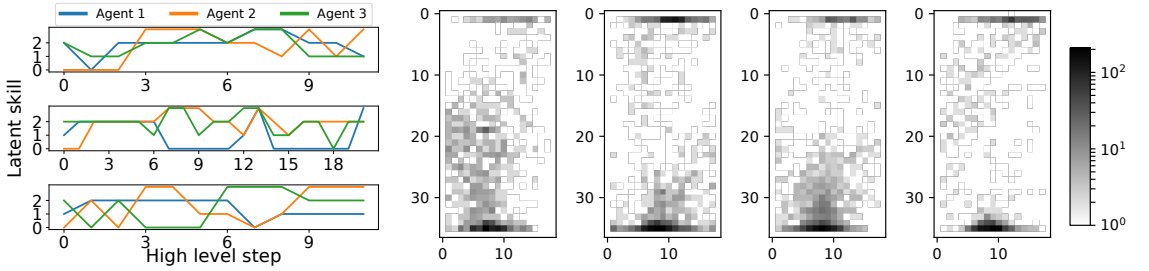
We use parameter-sharing among all agents, as is standard for homogeneous agents in cooperative MARL [8]. For function approximation, we use fully-connected neural networks without recurrent units since the game is fully observable. Each component is depicted in Figure 4.1. The low-level Q-function has two hidden layers, each with 64 units, and one output node per action. The high-level Q-function is a QMIX architecture: the individual utility function has two layers with 128 units per layer, and one output node per skill. Utility values of all agents are passed into a mixer network, whose non-negative weights in two hidden layers are generated by hypernetworks of output dimension 64, and whose final

output is a single global Q value (see [18]). The skill decoder is a bidirectional LSTM [145] with 128 hidden units in both forward and backward cells, whose outputs are mean-pooled over time and passed through a softmax output layer to produce probabilities over skills. We use batch size $N_{\text{batch}} = 1000$ to train the decoder; ϵ-greedy exploration at both high and low levels with ϵ decaying linearly from 0.5 to 0.05 in 1e3 episodes; replay buffers $\mathcal{B}_H$ and $\mathcal{B}_L$ of size 1e5; learning rate 1e-4; and discount $\gamma = 0.99$. High and low level action-value functions are trained using minibatches of 256 transitions every 10 steps at the high and low levels, respectively. Target networks [2] are updated after each training step with update factor 0.01. We conduct 20 episodes of evaluation once every 100 training episodes. We experimented with 4 and 8 latent skills, $t_{\text{seg}} = 10$, and let α decay from 1.0 to a minimum of 0.6 by $\alpha_{\text{step}} = 0.01$ whenever average win rate during evaluation exceeds $\alpha_{\text{threshold}} = 70\%$. We process trajectory segments as described in Section 4.2.4 with $k_{\text{skip}} = 2$.

As we instantiate our general method using QMIX [18] at the high level and independent Q-learning (IQL) [9, 2] at the low level, we compare performance with these two baselines to demonstrate that the new hierarchical architecture maintains performance while gaining interpretability. QMIX uses the same neural architecture as our method, except that the individual utility function outputs action-values for primitive actions instead of action values for high-level skills. IQL uses a two-layer Q-network with 128 units per layer. We first performed a coarse manual search for hyperparameters of QMIX and IQL, and used the same same values for the corresponding subset of hyperparameters in our method. Additional hyperparameters ($\alpha_{\text{threshold}}$, α_{step}, and t_{seg}) in our method were chosen from a coarse manual search, and we show results on hyperparameter sensitivity. We also compared with a variant of our method that uses two hand-scripted subtask reward functions with the same hierarchical architecture. An agent with subtask 1 gets reward +1 for making a goal when having possession; an agent with subtask 2 gets +1 for stealing possession from an opponent. These *individual* rewards mitigate the difficult problem of multi-agent credit assignment, and so this variant gives a rough indication of maximum possible win rate against the scripted



(a) Event distribution (b) PCA of events (c) Action distribution (d) PCA of actions (e) Overall skill usage



(f) Time series of latent skill usage (g) Latent skill usage (skills 0-3, left to right) over the spatial playing field

Figure 4.2: (a-c): Behavioral investigation of one HSD policy, showing average and standard error over 100 test episodes. (a) Distribution of special game events for each latent skill. (b) Projection of each skill's event distribution via PCA. (c) Distribution of primitive actions for each latent skill, where "Px" denotes "pass to teammate x". (d) Projection of each skill's action distribution via PCA. (e) Count of overall skill usage, when agent team has or does not have possession. (f) Time series of skills selected high-level steps, each consisting of $t_{\text{seg}} = 10$ primitive steps; each subplot shows one independent test episode; (g) Count of skill usage over the full continuous playing field, discretized to a 36x18 grid.

opponent team.

4.4 Results

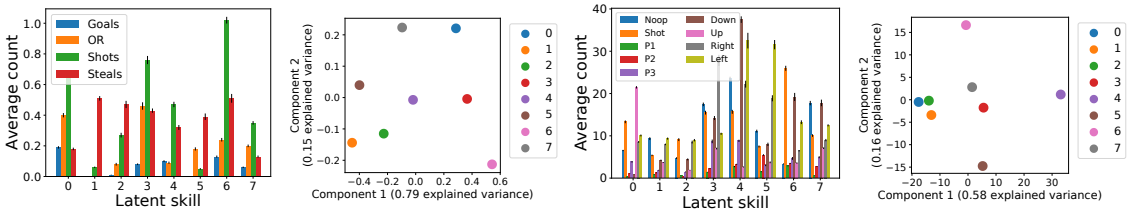
Our method for Hierarchical learning with Skill Discovery, labeled "HSD", learns interpretable skills that are useful for high-level cooperation. HSD meets the performance of QMIX and IQL, exceeds them in ad-hoc cooperation, and enables deeper policy analysis due to its hierarchical structure. Section 4.4.1 provides a detailed quantitative behavioral analysis of learned skills. Section 4.4.2 discusses performance, hyperparameters sensitivity, and ad-hoc cooperation.

4.4.1 Quantitative behavioral analysis

We conducted a quantitative analysis of the discovered skills by measuring the impact of skills on occurrence of game events and primitive actions, agents' choices of skills over an episode, and the spatial occurrence of skills. Figure 4.2 shows results for the case of four latent skills, which we describe immediately below. We describe the case of eight latent skills later in Figure 4.3.

Analysis of game events. Figure 4.2a shows the counts of each game event under each skill, summed over any agent who was assigned to execute the skill, and averaged over 100 test episodes. Skill 1 makes the most shot attempts, Skill 2 provides defense by focusing on steals, while Skill 3 contributes to the most number of successful goals. This difference in game impact, which emerged without any skill-specific reward functions, is also reflected by the large separation of principal components in Figure 4.2b that result from applying PCA to the vector of event counts of Figure 4.2a. Figure 4.2b suggests that component 1 corresponds to tendency to make offensive shots, while component 2 corresponds to tendency to make steals. Figure 4.2c shows the distribution of primitive actions taken by the low-level policy when conditioned on each latent skill. Skill 0 predominantly moves up towards the opponent net to begin offense, Skill 1 is more biased toward the left field, while Skill 2 moves down to defend the home net more than other skills. Figure 4.2e shows the usage of each skill by the high-level policy, under the cases when agent team has possession and when the opponent team has possession. Skill 2 is strongly associated with lack of possession since it is a defensive skill for regaining possession.

Time series of skill usage. Figure 4.2f shows a time series of skill usage over high-level steps by each agent during three different episodes (from top to bottom). Importantly, agents learned to choose complementary skills, such as in Episode 3 when Agent 3 stays for defense while Agents 1 and 2 execute offense via Skills 1 and 3, at step 9. Each individual agent also dynamically switches between skills, such as in Episode 1 when Agents 1 and 3 switch from the defensive Skill 2 to the offensive Skill 3 at step 6. As shown by the extended



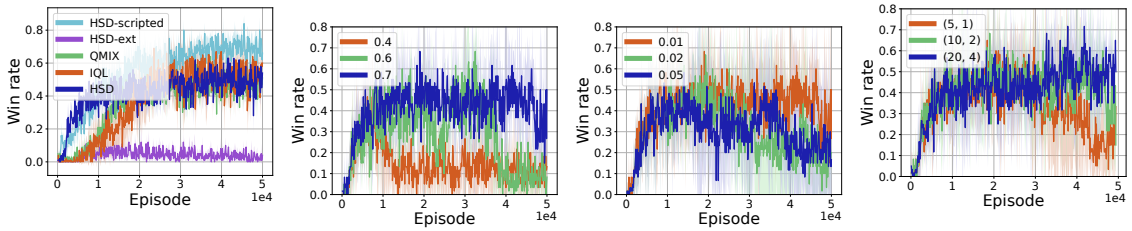
(a) Event distribution (b) PCA of events (c) Action distribution (d) PCA of actions

Figure 4.3: Behavioral analysis of HSD policies with 8 latent skills. (a) Skill 0 makes the most goals, skill 1 focuses on defensive steals, skill 6 makes the most shot attempts. (b) Differences between skills, especially skills 0, 1 and 6, are reflected by the PCA reduction of events. (c) Skill 0 predominantly moves up, which explains its high goal rate, while skill 4 moves down the most (d) These distinguishable characteristics of skills are reflected by their large separation after PCA reduction.

periods in all episodes when all agents play the defensive Skill 2, agents are able to sustain the same skill over multiple consecutive high-level steps, which mitigates the concern over choosing a fixed t_{seg}. Note that at any given time in the game, the defensive Skill 2 is almost always used by some agent either to make steals or cover the home net.

Spatial occurrence of skills. Figure 4.2g is a heatmap of skill usage over the playing field. Consistent with the previous analysis, Skill 0 is used for moving up for offense, Skills 1 and 3 tend to camp near the opponent net (top) to attempt shots, while Skill 2 is concentrated near the home net (bottom) to make defensive steals.

Increasing number of skills. The number of latent skills is also a key design choice to make based on domain knowledge. Figure 4.3 analyzes HSD when trained with eight skills. Skills 0, 3, and 6 focus on shot attempts and offensive rebounds (Figure 4.3a), and they have high values of the first principal component (Figure 4.3b). Skills 1 and 2 focus on defensive steals. Figure 4.3c shows that Skill 0 moves up for offense the most, while Skill 4 moves down to play defense. This is reflected by their large separation in the first principal component (Figure 4.3d).



(a) Win rate against scripted bots

(b) Varying $\alpha_{\text{threshold}}$

(c) Varying α_{step}

(d) Varying t_{seg} and k_{skip}

Figure 4.4: Win rate against scripted opponent team over training episodes. Each curve is the mean over random seeds (5 for (a) and 3 for (b-d)) with shaded region representing 95% confidence interval. (a) HSD is within margin of error with QMIX and IQL. HSD-scripted has the same hierarchical architecture as HSD but is trained with hand-scripted subtask rewards. HSD-ext does not use extrinsic rewards. (b-d) Learning is more stable with high $\alpha_{\text{threshold}}$, small α_{step}, and longer t_{seg}.

4.4.2 Performance and parameter sensitivity

Figure 4.4 shows win rate against the scripted opponent team over training episodes for HSD and baselines, each with 5 independent runs, and for varying hyperparameter settings of HSD, each with 3 independent runs. HSD agents learn faster than QMIX and IQL, consistent with findings on hierarchical versus non-hierarchical methods in early work [79], while their final performance are within the margin of error (Figure 4.4a). HSD-ext does not have access to extrinsic rewards and underperforms the rest. This supports our hypothesis that the extrinsic team reward is needed in combination with the intrinsic reward to promote useful behavior. HSD-scripted outperformed other methods, showing that using cooperative learning at the high-level and independent learning at the low level is a strong approach, and improvement to skill discovery is possible.

We investigated the effect of varying the key hyperparameters of HSD. Figure 4.4b shows that larger values of $\alpha_{\text{threshold}}$ gives higher performance and lower variance. A small $\alpha_{\text{threshold}}$ increases the likelihood that a spuriously high evaluation performance crosses $\alpha_{\text{threshold}}$, which would cause a re-weighting of the extrinsic versus intrinsic reward even when the agents have not yet adapted to the current reward. This explains the instability of $\alpha_{\text{threshold}} = 0.4$ in Figure 4.4b. Likewise, Figure 4.4c shows that a smaller value of α_{step}

performs better, because each adjustment of the low-level reward is smaller and hence the automatic curriculum is easier for learning. Figure 4.4d shows that agents who sustain high-level skills for 10 or 20 time steps perform better than agents who sustain only for 5 steps. A smaller t_{seg} means that agents make more frequent decisions to sustain or switch their choice of skill, which allows for more flexible policies but increases the difficulty of learning.

Table 4.1: Win/lose percentage of final policies over 100 test episodes and 5 seeds, matched with different teammates.

	HSD		QMIX		IQL	
Teammate	Win	Lose	Win	Lose	Win	Lose
Training	46 (4)	39 (4)	55 (3)	23 (3)	36 (7)	46 (4)
1 scripted	49 (4)	45 (3)	48 (4)	44 (4)	32 (3)	54 (4)
2 scripted	52 (3)	45 (1)	45 (2)	51 (2)	37 (2)	58 (1)
1 defensive	43 (5)	42 (4)	-	-	-	-
1 offensive	45 (2)	41 (1)	-	-	-	-

Ad Hoc cooperation. We investigated the test performance of agents in ad-hoc cooperation, by giving them teammate(s) with whom they never previously trained [146]. This mimics the setting where AI agents must cooperate with a human player in team sports games. Table 4.1 shows the win and lose percentage of HSD, QMIX, and IQL (draws are possible). HSD agents perform as well or better when one or two of their teammates are replaced by scripted bots, possibly due to independently-trained low-level policies in HSD. However, QMIX agents performed significantly worse when paired with scripted bots, likely because the out-of-training behavior of bots pose difficulties for QMIX agents who underwent fully-centralized training. IQL agents also lost significantly more often with scripted teammates. For HSD, we can also fix one agent to always play a defensive or offensive skill. Based on Figure 4.2a, we chose Skill 1 for offense and Skill 2 for defense. HSD agents are able to maintain their performance within the margin of error.

4.5 Summary

We presented a method for hierarchical multi-agent reinforcement learning that discovers useful skills for strategic teamwork. We train cooperative decentralized policies for high-level skill selection and train independent low-level policies to execute chosen skills, which emerge from a dynamically weighted combination of intrinsic and extrinsic rewards. We demonstrated the emergence of quantifiable, distinct and useful skills in stochastic team sports simulations without assigning a reward to each skill. These findings are a step toward multi-agent game AI that execute realistic high-level strategies and can cooperate with human players.

Algorithm 1 Hierarchical MARL with unsupervised skill discovery

procedure ALGORITHM
  Initialize high-level Q_ϕ, low-level Q_θ, decoder p_ψ, high-level replay buffer $\mathcal{B}_H$, low-level replay buffer $\mathcal{B}_L$, and trajectory-skill dataset $\mathcal{D}$
  for each episode **do**
    $s_t, \mathbf{o}_t =$ env.reset()
    Initialize trajectory storage $\{\tau^n\}_{n=1}^N$ of max length t_{seg}
    for each step $t = 1, \ldots, T$ in episode **do**
      if $t \mod t_{\text{seg}} = 0$ **then**
        if $t > 1$ **then**
          Compute $\tilde{R}_t := \gamma^{t_{\text{seg}}} * \sum_{k=0}^{t_{\text{seg}}} R_{t-k}$
          Store $(s_{t-t_{\text{seg}}}, \mathbf{o}_{t-t_{\text{seg}}}, \mathbf{z}, \tilde{R}_t, s_t, \mathbf{o}_t)$ into $\mathcal{B}_H$
          for each agent n **do**
            Store (z^n, τ^n) into $\mathcal{D}$
            Compute intrinsic reward R_I^n using (Equation 4.4)
          end for
        end if
        Select new z^n by ϵ-greedy$(Q_\phi^n(o^n, z)), \forall n \in [N]$
        if # (high level steps) $\mod t_{\text{train}} = 0$ **then**
          Update $Q_\phi(s, \mathbf{z})$ using $\mathcal{B}_H$ and (Equation 4.5)
        end if
      end if
      Get a_t^n from ϵ-greedy$(Q(o_t^n, z_t^n, a))$ for each agent
      $s_{t+1}, \mathbf{o}_{t+1}, R_t =$ env.step$(\mathbf{a}_t)$
      Compute $R_L^n := \alpha R_t + (1-\alpha) R_I^n$ for each agent
      For all agents, store $(o_t^n, a_t^n, R_L^n, o_{t+1}^n, z^n)$ into low-level replay buffer $\mathcal{B}_L$, and append o_t^n to trajectory τ^n
      if # (low-level steps) $\mod t_{\text{train}} = 0$ **then**
        Update $Q_\theta(o^n, z^n, a^n)$ using $\mathcal{B}_L$ and (Equation 4.7)
      end if
    end for
    if size of $\mathcal{D} \geq N_{\text{batch}}$ **then**
      Update decoder $p_\psi(z|\tau)$ using $\mathcal{D}$, then empty $\mathcal{D}$
    end if
    if evaluation win rate exceeds $\alpha_{\text{threshold}}$ **then**
      $\alpha \leftarrow \max(\alpha_{\text{end}}, \alpha - \alpha_{\text{step}})$
    end if
  end for
end procedure

CHAPTER 5
LEARNING TO INCENTIVIZE OTHER LEARNING AGENTS

5.1 Introduction

Reinforcement Learning (RL) [1] agents are achieving increasing success on an expanding set of tasks [2, 3, 4, 5, 6]. 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. The option of centralized training with a global reward [17, 31, 18] is excluded as it does not scale easily to large populations and may not be adopted by self-interested parties. On the other hand, the paradigm of decentralized training—in which no agent is designed with an objective to maximize collective performance and each agent optimizes its own set of policy parameters—poses difficulties for agents to attain high individual and collective return [36]. In particular, 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 [37, 38, 39]. 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 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 scales of human activity and, crucially, that it can be successfully resolved when agents give the right incentives to *alter* the objective of other

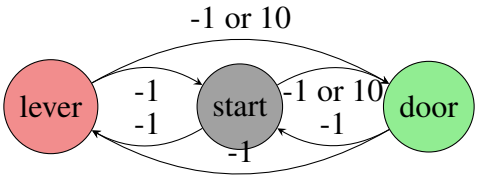


Figure 5.1: The N-player *Escape Room* game ER(N, M). For $M < N$, if fewer than M agents pull the lever, which incurs a cost of -1, then all agents receive -1 for changing positions. Otherwise, the agent(s) who is not pulling the lever can get $+10$ at the door and end the episode.

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 [147, 148, 149]. The rich body of work on game-theoretic side payments [150, 151, 152] further attests to the importance of inter-agent incentivization in society.

Translated to the framework of Markov games for multi-agent reinforcement learning (MARL) [10], 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 [153], 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 5.1. 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 5.4.

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* [91]. Taken together, we believe this is a promising step toward a cooperative multi-agent future.

5.2 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 $i \in [N] := \{1, \ldots, N\}$. 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[1]. 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. Therefore, while it may appear that LIO has an augmented action space that provides an additional channel of influence on other agents, we emphasize that LIO's learning approach does *not* treat the incentive as a standard "give-reward" action.

We build on the idea of online cross-validation [102], to capture the fact that an incentive has measurable effect only after a recipient's learning step. As such, we describe LIO

[1]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 [100].

in a procedural manner below (Algorithm 2). This procedure can also be viewed as an iterative method for a bilevel optimization problem [139], 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

$$r^j(s_t, \mathbf{a}_t, \eta^{-j}) := r^{j,\text{env}}(s_t, \mathbf{a}_t) + \sum_{i \neq j} r^j_{\eta^i}(o^i_t, a^{-i}_t)\,, \tag{5.1}$$

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

$$\max_{\theta^j} J^{\text{policy}}(\theta^j, \eta^{-j}) := \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{T} \gamma^t r^j(s_t, \mathbf{a}_t, \eta^{-j})\right]\,. \tag{5.2}$$

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

$$\hat{\theta}^j \leftarrow \theta^j + \beta f(\tau^j, \theta^j, \eta^{-j}) \tag{5.3}$$

that adjusts its policy parameters with learning rate β (Algorithm 2, lines 4-5). Assuming policy optimization learners in this work and choosing policy gradient for exposition, the update function is

$$f(\tau^j, \theta^j, \eta^{-j}) = \sum_{t=0}^{T} \nabla_{\theta^j} \log \pi^j(a^j_t | o^j_t) G^j_t(\tau^j; \eta^{-j})\,, \tag{5.4}$$

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 2, lines

6-7):

$$\max_{\eta^i} J^i(\hat{\tau}^i, \tau^i, \hat{\boldsymbol{\theta}}, \eta^i) := \mathbb{E}_{\hat{\boldsymbol{\pi}}} \left[\sum_{t=0}^{T} \gamma^t \hat{r}_t^{i,\text{env}} \right] - \alpha L(\eta^i, \tau^i) \,. \tag{5.5}$$

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:

$$L(\eta^i, \tau^i) := \sum_{(o_t^i, a_t^{-i}) \in \tau^i} \gamma^t \| r_{\eta^i}(o_t^i, a_t^{-i}) \|_1 \,. \tag{5.6}$$

This cost is incurred by the incentive function and not by the policy, since the latter does not determine incentivization[2] and should not be penalized for the incentive function's behavior (see Section B.1.1 for more discussion). We use the ℓ_1-norm so that cost has the same physical "units" as extrinsic rewards. The gradient of (Equation 5.6) 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 (Equation 5.5), the gradient w.r.t. η^i is:

$$\nabla_{\eta^i} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) = \sum_{j \neq i} (\nabla_{\eta^i} \hat{\theta}^j)^T \nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) \,. \tag{5.7}$$

The first factor of each term in the summation follows directly from (Equation 5.3) and (Equation 5.4):

$$\nabla_{\eta^i} \hat{\theta}^j = \beta \sum_{t=0}^{T} \nabla_{\theta^j} \log \pi^j(a_t^j | o_t^j) \left(\nabla_{\eta^i} G_t^j(\tau^j; \eta^{-j}) \right)^T \,. \tag{5.8}$$

Note that (Equation 5.3) does not contain recursive dependence of θ^j on η^i since θ^j is a

[2]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 2 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 (Equation 5.3)
        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 (Equation 5.5)
        $\theta^i \leftarrow \hat{\theta}^i$, $\eta^i \leftarrow \hat{\eta}^i$ for all $i \in [N]$.
    end for
end procedure

function of incentives in *previous* episodes, not those in trajectory τ^i. The second factor in (Equation 5.7) can be derived as

$$\nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}}) = \mathbb{E}_{\hat{\boldsymbol{\pi}}} \left[\nabla_{\hat{\theta}^j} \log \hat{\pi}^j(\hat{a}^j | \hat{o}^j) Q^{i,\hat{\boldsymbol{\pi}}}(\hat{s}, \hat{\mathbf{a}}) \right] . \tag{5.9}$$

In practice, to avoid manually computing the matrix-vector product in (Equation 5.7), one can define the loss

$$\text{Loss}(\eta^i, \hat{\tau}^i) := -\sum_{j \neq i} \sum_{t=0}^{T} \log \pi_{\hat{\theta}^j}(\hat{a}_t^j | \hat{o}_t^j) \sum_{l=t}^{T} \gamma^{l-t} r^{i,\text{env}}(\hat{s}_l, \hat{\mathbf{a}}_l) , \tag{5.10}$$

and directly minimize it via automatic differentiation [154]. Crucially, $\hat{\theta}^j$ must preserve the functional dependence of the policy update step (Equation 5.4) on η^i within the same computation graph. Derivations of (Equation 5.9) and (Equation 5.10) are similar to that for policy gradients [56] and are provided in Section B.3.

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 [155] 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 (Equation 5.3). We demonstrate a fully-decentralized implementation in our experiments.

5.2.1 Relation to opponent shaping via actions

LIO conducts opponent shaping via the incentive function. This resembles LOLA [96], 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 [102], 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 [97]. 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 Section B.5.1 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 [150], LIO is closer in spirit to the paradigm of optimized rewards [104, 3, 92].

5.2.2 Analysis in Iterated Prisoner's Dilemma

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

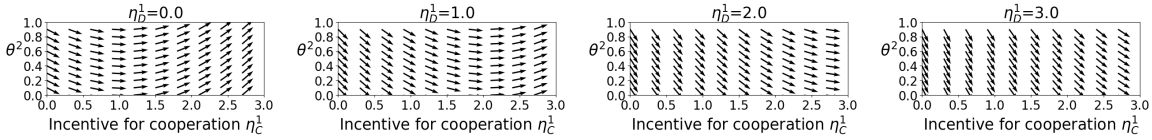


Figure 5.2: Exact LIO in IPD: probability of recipient cooperation versus incentive for cooperation.

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 5.1, we prove in Section B.2 the following:

Proposition 3. *Two LIO agents converge to mutual cooperation in the Iterated Prisoner's Dilemma.*

Table 5.1: Prisoner's Dilemma

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

Moreover, we may gain further insight by visualizing the learning dynamics of exact LIO in the IPD, computed in Section B.2. 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 5.2, 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 B.1 shows that incentive for defection is guaranteed to decrease.

5.3 Experimental setup

Our experiments[3] demonstrate that LIO agents are able to reach near-optimal individual performance by incentivizing other agents in cooperation problems with conflicting individual

[3]Code for all experiments is available at https://github.com/011235813/lio

and group utilities. We define three different environments with increasing complexity in Section 5.3.1 and describe the implementation of our method and baselines in Section 5.3.2.

5.3.1 Environments

Iterated Prisoner's Dilemma (IPD). We test LIO on the memory-1 IPD as defined in [96], where agents observe the joint action taken in the previous round and receive extrinsic rewards in Table 5.1. This serves as a test of our theoretical prediction in Section 5.2.2.

N-Player Escape Room (ER). We experiment on the N-player Escape Room game shown in Figure 5.1 (Section 5.1). 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 Section B.5.1.

Cleanup. Furthermore, we conduct experiments on the Cleanup game (Figure 5.3) [91, 92]. 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.

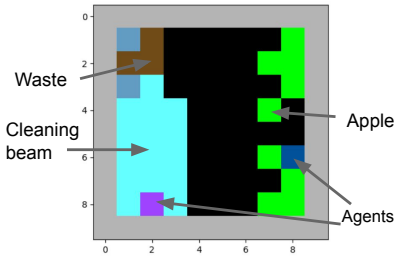


Figure 5.3: *Cleanup* (10x10 map): apple spawn rate decreases with increasing waste, which agents can clear with a cleaning beam.

5.3.2 Implementation and baselines

We describe key details here and provide a complete description in Section B.4.2. 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 $[0, R_{\text{max}}]$, 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 $R_{\text{max}} = [3, 2, 2]$ 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.

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 (Equation 5.3) 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 $\mathcal{A} \times \{\text{no-op}, \text{give-reward}\}^{N-1}$. 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. Section B.4.2 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 [91]. 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.

5.4 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[4] 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 5.2.2 and Section B.2, two LIO agents with policy gradient approximation converge near the optimal CC solution with joint reward -2 in the IPD (Figure 5.4). This meets the performance of LOLA-PG and is close to LOLA-Ex, as reported in [96]. 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.

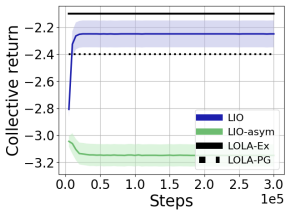
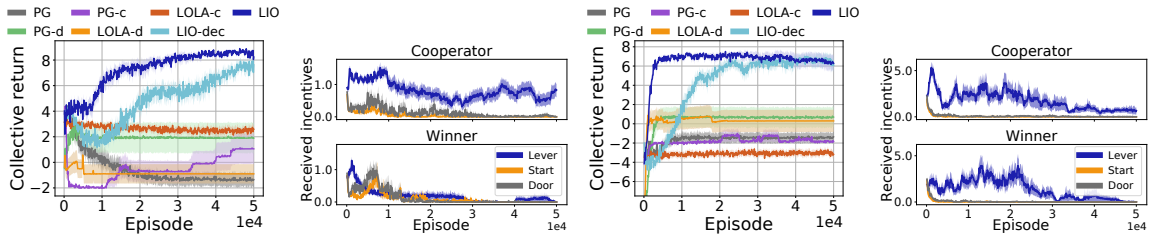


Figure 5.4: The sum of all agents' rewards in IPD.

Escape Room. Figures 5.5a and 5.5c show that groups of LIO agents discover a division of labor in both ER(2,1) and ER(3,2), whereby some agent(s) cooperate by pulling the lever to allow another agent to exit the door, such that collective return approaches the

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

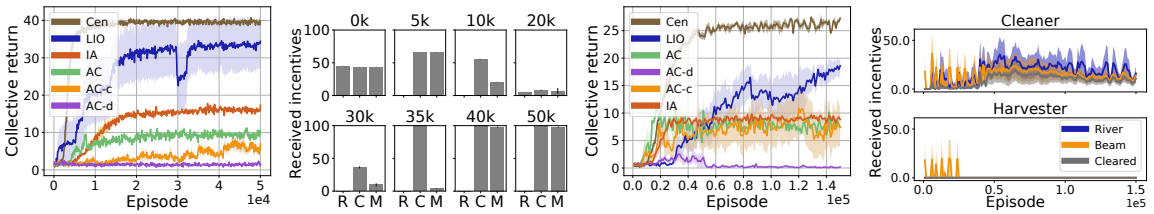


(a) Collective return in N=2 (b) Incentives in N=2 (c) Collective return in N=3 (d) Incentives in N=3

Figure 5.5: 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.

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 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. Section B.5.1 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. Figure B.4c evidences that LIO scales well to larger groups such as ER(5,3), since the complexity of (Equation 5.7) is linear in number of agents.

To understand the behavior of LIO's incentive function, we classify each agent *at the end of training* as 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 5.5b and 5.5d show that the Cooperator was correctly incentivized for pulling the lever and receives negligible incentives



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

Figure 5.6: 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.

for noncooperative actions. Asymptotically, the Winner receives negligible incentives from the Cooperator(s), who learned to avoid the cost for incentivization (Equation 5.6) when doing so has no benefits itself, whereas incentives are still nonzero for the Cooperator.

Cleanup. Figures 5.6a and 5.6c 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 B.5a), 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 clean but subsequently compete to collect apples, which is suboptimal for the group (Figure B.5b). 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 5.6a, 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 collects apples, respectively. Next, we

define 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 5.6b shows the incentives given by a Harvester to these scripted agents when they are tested together periodically during training. At episodes 10k, 30k and 35k, it gave significantly 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. After 40k episodes, it gives nonzero reward for "fire cleaning beam but miss", likely because its actual training partner already converged to successful cleaning (Figure 5.6a), so it may have "forgotten" the difference between successful and unsuccessful usage of the cleaning beam. As shown by results in the Escape Room (Figures 5.5b and 5.5d), correct incentivization can be maintained if agents have a sufficiently large lower bound on exploration rates that pose the risk of deviating from cooperative behavior. Figure 5.6d 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.

5.5 Summary

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.

CHAPTER 6
ADAPTIVE INCENTIVE DESIGN WITH MULTI-AGENT META-GRADIENT REINFORCEMENT LEARNING

6.1 Introduction

As advances in artificial intelligence research drive the growing presence of AI in critical infrastructures—such as transportation [156, 157], information communication [158], financial markets [159] and agriculture [160]—it is increasingly important for AI research to complement the solipsistic view of models and agents acting in isolation with a broader viewpoint: these models and agents may be developed by independent and self-interested principals but are eventually deployed in a shared multi-agent ecosystem. Apart from the special cases of pure common or conflicting interest (i.e., team or zero-sum payoffs), the majority of possible scenarios involve mixed motives [161] whereby cooperation for optimal group payoff is attainable if selfish behavior out of greed or fear—which results in low individual and group payoff—can be overcome. Research on endowing individual agents with social capabilities and designing central institutional mechanisms to safeguard and improve social welfare, recently named "Cooperative AI" [162], is a long-term necessity that requires present-day research efforts.

We focus on one specific pillar of this research agenda: the problem of *adaptive incentive design* [27], whereby a central institution shapes the behavior of self-interested agents to improve social welfare, by introducing an incentive function to modify their individual payoffs. The trivial solution of setting individual payoffs equal to the average system payoff is known to be suboptimal [105]: e.g., uniform redistribution of income leads to low productivity. This problem can be formalized as a reverse Stackelberg game [163, 164], in which the leader first proposes a function (e.g., the incentive function) that maps from

the follower's action space to the leader's decision space, while the follower chooses a best response. This is a difficult bi-level optimization problem even in the linear case [165]. To tackle this problem in high-dimensional multi-agent systems, we propose a model-free method based on meta-gradient reinforcement learning [103] and the principle of online cross-validation [102], for the incentive designer (ID) to explicitly account for the learning dynamics of agents in response to incentives. Potential applications in the long term include: e.g. shaping consumption patterns to improve the efficiency of smart power grids [166] and to mitigate climate crisis [167]; reducing wait times or traffic congestion in taxi dispatch and traffic tolling systems [168, 41]; solving the social dilemma of autonomous vehicle adoption [156]; improving the trade-off between economic productivity and income equality via taxation [42].

In the spirit of complexity economics [169], whereby the tractability of linear models with analytic equilibria [108, 28, 105] is eschewed for the greater realism and richer dynamics of high-dimensional agent-based simulation, we work in the framework of Markov games [10] with reinforcement learning (RL) agents [1]. Within this context, we interpret "incentive design" in the broad sense of influencing agents' behaviors via modifying their individual payoffs. The issue of incentive compatibility, despite being central to analytically tractable applications such as auctions where the goal is to elicit truthful reporting of private valuations [27], do not pertain in general to complex simulations involving RL agents for the following reasons: 1) the concept of private individual preferences may not make sense in the application (e.g., a social dilemma whose payoff is known to all parties); 2) there may be no *a priori* or *fixed* private valuations, because an agent's preference is completely represented by its reward function, which itself depends on the incentive function and hence changes dynamically along with the ID's online optimization process,; and 3) a complex simulation involving nonlinear processes and discrete rules is used to investigate dynamical behavior rather than to converge to equilibria.

Our use of simulation is motivated by two considerations. Firstly, there may arise a future

ecosystem of AI in diverse spaces, such as multiple firms in a financial market [170] and multiple recommendation systems in the same consumer sector [171]. With the increasing success of RL on progressively more difficult tasks [4, 5, 6] and growing efforts to apply RL to real-world problems [172, 173], such real-world *in silico* AI are likely to involve RL for optimization of long-term objectives and will be ontologically equivalent to the entities in our work. Secondly, agent-based simulation is also relevant to incentive design for a group of self-interested humans, firms, or states, by viewing the reward-maximizing behavior of RL agents as an approximation of the bounded rationality of such real world entities [174]. Therefore, we validate our approach in existing simulation benchmarks: *Escape Room* [40] and *Cleanup* [91], which are deceptively hard but easily interpretable benchmark problems, and the complex economic simulation called *Gather-Trade-Build* [42], where agent behavior such as specialization and correlations between taxation and productivity are qualitatively in accord with results from economic theory and reality.

The centralized incentive designer (ID) in our work is related but orthogonal to the centralization commonly seen in the literature on cooperative MARL [14, 175]: centralized training in MARL permits a global entity to update each agent's individual policy parameters using shared global information and thereby directly optimize a single team reward, while the ID in our work can only set an incentive function to affect each agent's total individual reward. Given a fixed incentive function, our agents live in a fully-decentralized POMDP [24]. The ID's intervention on the agents' reward function also differs from the insertion of controlled agents (e.g., centrally-controlled autonomous vehicles) whose actions indirectly regulate the behavior of other individuals (e.g. human drivers) [176].

In short, the algorithmic and experimental contributions of this chapter are: (1) we propose a meta-gradient approach for the reverse Stackelberg game formulation of incentive design in high-dimensional multi-agent reinforcement learning settings; (2) we explain and show experimentally that using standard RL for the incentive designer faces significant difficulties even small finite state finite action Markov games; (3) our proposed method

can converge to known global optima in standard benchmark problems, and it generates significantly higher social welfare than the previous state-of-the-art in a complex high-dimensional economic simulation of market dynamics with taxation.

6.2 Method

We propose a method, called "MetaGrad", for an Incentive Designer (ID) to optimize a measure of social welfare by explicitly accounting for the impact of incentives on the behavior of a population of n independent agents. Each agent $i \in \{1, \ldots, n\}$ has an individual reward function $R^i \colon \mathcal{S} \times \mathcal{A}^n \times \mathcal{U} \to \mathbb{R}$, which depends not only on the global state $s \in \mathcal{S}$ and the joint action $\mathbf{a} \in \mathcal{A}^n$, as in standard Markov games [10] with transition function $P(s'|s, \mathbf{a})$, but also on an incentive that is parameterized by $u \in \mathcal{U}$ for some bounded set $\mathcal{U} \subset \mathbb{R}^l$ (e.g., a vector of marginal tax rates). We assume that R^i is differentiable with respect to the argument u—this holds in the common case of additive incentives such as highway tolling, as well as in complex mechanisms such as a bracketed tax schedule (Equation 6.7). This incentive u is generated by the ID via a learned incentive function $\mu_\eta \colon \mathcal{S} \times \mathcal{A}^n \to \mathcal{U}$, parameterized by η, which adaptively responds to the current system state and joint action of the agents. Each agent i independently learns a policy π_{θ^i}, parameterized by θ^i, to optimize its own individual expected return, while the ID's performance is measured by a social welfare reward $R^{\text{ID}}(s, \mathbf{a})$. Let $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$ denote the agents' joint policy and policy parameters, respectively. For brevity, we use $R^i_\eta(s, \mathbf{a})$ to denote $R^i(s, \mathbf{a}, \mu_\eta(s, \mathbf{a}))$, and we identify $\boldsymbol{\theta}$ with the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ where no confusion arises.

The ID aims to solve the bilevel optimization problem

$$\text{(ID)} \quad \max_{\eta} J^{\text{ID}}(\eta; \widehat{\boldsymbol{\theta}}) := \mathbb{E}_{\boldsymbol{\pi}_{\widehat{\boldsymbol{\theta}}}, P(s'|s_t, \mathbf{a})} \left[\sum_{t=0}^{H} \gamma^t R^{\text{ID}}_t - \psi(s_t) \right] \tag{6.1}$$

$$\text{(Agents)} \; \widehat{\boldsymbol{\theta}} = \left\{ \operatorname*{argmax}_{\theta^i} J^i(\boldsymbol{\theta}; \eta) := \mathbb{E}_{\boldsymbol{\pi}_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{H} \gamma^t R^i_\eta(s_t, \mathbf{a}_t) \right] \right\}_{i=1}^{n} \tag{6.2}$$

over episode horizon H, and ψ is a known cost for incentivization (e.g., sum of all incentives).

We assume that the n agents apply a gradient-based reinforcement learning procedure $\mathrm{RL}(\cdot)$ to update their policies in response to the reward determined by η, i.e., that (Equation 6.2) is achieved by $\hat{\theta}^i = \mathrm{RL}(\theta_0^i; \eta)$, where θ_0^i is an initial policy. In the ideal case, one should measure the population behavior under the final joint policy $\hat{\boldsymbol{\theta}}$, after convergence of the RL process under a fixed η, to evaluate the effectiveness of η in optimizing social welfare and conduct a single η update. However, the high sample count required for convergence of RL in practice is prohibitively expensive, especially if one wishes to apply gradient descent to optimize η. To tackle this challenge, we build on the effectiveness of meta-gradient RL in optimizing hyperparameters and parameterized objectives concurrently with an agent's policy optimization [103, 117, 121]. We apply iterative gradient descent to the upper objective (Equation 6.1) on the same timescale as the agents' policy optimization (Equation 6.2), by explicit differentiating through the agents' policy updates. We emphasize that even though the ID does not wait for convergence of the final $\hat{\boldsymbol{\theta}}$, we follow the principle of online cross-validation [102] and extend it to the optimal control setting : the data used for the η-update is still generated by the agents' *updated* policies, not by any arbitrary policy, in order to measure accurately the impact of η on the ID's objective through the agents' learning process. This differs from previous single-agent meta-gradient RL [103, 117], where the trajectories used for the outer update were not generated by the updated agent policy.

Specifically, we implement the following algorithm (Algorithm 3). Given the current policy $\boldsymbol{\theta}_0$ and incentive function μ_η, agents collect trajectories $\{\{\tau_j^i\}_{i=1}^n\}_{j=0}^{M-1} \sim \boldsymbol{\pi}_{\boldsymbol{\theta}_0}$ (Algorithm 3, line 3) and conduct M policy update steps (Algorithm 3, line 5):

$$\hat{\boldsymbol{\theta}}(\eta) := \boldsymbol{\theta}_M = \boldsymbol{\theta}_0 + \sum_{j=0}^{M-1} \Delta\boldsymbol{\theta}_j(\eta)\,. \tag{6.3}$$

Algorithm 3 Meta-Gradient Incentive Design with pipelining

1: **procedure**
2: Initialize all agents' policy parameters θ^i, incentive function parameters η
3: Generate trajectory $\boldsymbol{\tau}$ using $\boldsymbol{\theta}$ and η
4: **for** each iteration **do**
5: For all agents, update $\hat{\theta}^i$ with $\boldsymbol{\tau}$ using (Equation 6.3)
6: Generate a new trajectory $\hat{\boldsymbol{\tau}}$ using new $\hat{\boldsymbol{\theta}}$
7: Update $\hat{\eta}$ by gradient ascent along (Equation 6.5) using $\boldsymbol{\tau}$ and $\hat{\boldsymbol{\tau}}$
8: $\boldsymbol{\tau} \leftarrow \hat{\boldsymbol{\tau}}$, $\eta \leftarrow \hat{\eta}$, $\theta^i \leftarrow \hat{\theta}^i$ for all $i \in [n]$.
9: **end for**
10: **end procedure**

Each agent's update $\Delta\theta^i_j(\eta) \propto \nabla_{\theta^i_j} J^i(\theta^i_j; \eta, \tau^i_j)$ depends on the fixed η. For example, if agents learn by policy gradient methods, we have $\Delta\theta^i_j \propto \sum_{t=0}^{T} \nabla_{\theta^i_j} \log \pi_{\theta^i_j}(a^i_t|o^i_t) A^i_t(\tau^i_j; \eta)$, where A^i_t is an advantage function that depends on η via $R^i_\eta(s, \mathbf{a})$. Now let $\hat{\boldsymbol{\tau}}$ denote the subsequent trajectory generated by the agents' updated policies $\hat{\boldsymbol{\theta}}$ (Algorithm 3, line 6), which serves as the *validation trajectory* that measures the indirect impact of η on the ID's return through the agents' learning. The ID computes and ascends the gradient of objective (Equation 6.1) w.r.t. η via the chain rule (Algorithm 3, line 7)

$$\nabla_\eta J^{\text{ID}}(\eta; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\tau}}) = \sum_{i=1}^{n} \left(\nabla_\eta \hat{\theta}^i(\eta)\right)^\top \left(\nabla_{\hat{\theta}^i} J^{\text{ID}}(\eta; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\tau}})\right) - \nabla_\eta \psi(\tau) \tag{6.4}$$

where $\nabla_\eta \hat{\theta}^i$ is computed using a replica of the θ^i update step.

Proximal meta-gradient optimization. Instead of computing both factors of (Equation 6.4), we can view (Equation 6.1) as a standard objective in policy-based RL, with the difference that we optimize with respect to η instead of the policy parameters $\hat{\boldsymbol{\theta}}$. Hence, one can apply the policy gradient algorithm [177] by replacing $\nabla_{\hat{\boldsymbol{\theta}}}$ with ∇_η, as shown in [40, Appendix C] and used implicitly by [103, 117]. We extend this viewpoint by showing (in Section C.1.1) that trust-region arguments [44] hold for meta-gradients, which justifies the use of a proximal policy optimization (PPO)-type gradient [45] for the outer

optimization:

$$\nabla_\eta J^{\text{ID}}(\eta; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\tau}}) = \mathbb{E}_{\mathbf{a}_t, s_t \sim \boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}(\eta)}} \left[\min \Big(r(\hat{\boldsymbol{\theta}}; \eta) A_t, \text{clip}(r(\hat{\boldsymbol{\theta}}; \eta), 1-\epsilon, 1+\epsilon) A_t \Big) \right] \tag{6.5}$$

$$r(\hat{\boldsymbol{\theta}}; \eta) := \frac{\nabla_\eta \boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}(\eta)}(\mathbf{a}|s)}{\boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}(\eta)}(\mathbf{a}|s)}\,, \tag{6.6}$$

where $A_t := \sum_{l=t}^{T-1} (\gamma\lambda)^{l-1} \delta_l$ is a generalized advantage estimator computed using $\delta_t := R^{\text{ID}}(s_t, \mathbf{a}_t) + \gamma V(s_{t+1}) - V(s_t)$, critic V for the ID, discount γ and λ-returns.

6.2.1 Technical relation to prior multi-agent learning methods for incentivization

The "AI Economist" [42] treats agents' learning as a black-box: it applies standard RL to a central planner who learns an adaptive tax policy concurrently with the agents' policy learning within a fully-decentralized multi-agent economy. Rather than addressing the bi-level optimization problem, this expands the multi-agent system and exacerbates the already existing problem of non-stationarity in decentralized MARL, which required heuristics such as curriculum learning and tax annealing that are difficult to tune. In contrast, our method to train the incentive function fundamentally differs from standard RL: the gradient (Equation 6.5) is taken with respect to the η variables of the incentive function, *through the policy updates* $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$ of all the regular RL agents, where $\hat{\theta}^i$ preserves the dependence of each agent's update on η in the computational graph.

Our technical method is the centralized analogue of the fully-decentralized pairwise incentivization in LIO [40]. That work begins with the premise that all, or some, agents in the environment are equipped with the LIO learning mechanism, but this may not hold in general environments where no principal opts to use a LIO agent. In contrast, our work only assumes that agents learn from reward functions that depend on and can be differentiated with respect to incentives, which pertains to more general potential applications.

6.3 Experimental setup

We evaluated our approach in three environments: 1) *Escape Room* (ER) [40], a small but deceptively hard pedagogical example that accentuates the core challenges of incentive design for RL agents; 2) *Cleanup* [91], a high-dimensional instance of a sequential social dilemma; and 3) the *Gather-Trade-Build* simulation of a market economy with taxation, trading, and competition for limited resource. Section 6.3.1 summarizes the high-level features of these environments; Section C.2.1 provides complete specifications. Section 6.3.2 describes the implementation of the method and baselines.

6.3.1 Environments

Escape Room [40]. The *Escape Room* game ER(n, m) is a discrete n-player Markov game with individual extrinsic rewards and parameter $m < n$. An agent gets $+10$ extrinsic reward for exiting a door, but this requires m other agents to sacrifice their own self-interest by pulling a lever at a cost of -1 each. We fix all agents to be standard independent RL agents without "give-reward" capabilities, and we introduce a central incentive designer who can modify each agent's reward by adding a scalar bounded in $(0, 2)$, since an incentive value of $1 + \epsilon$ for $\epsilon > 0$ is sufficient for an agent to overcome the -1 penalty for pulling the lever. The ID's reward R^{ID} is the sum of all agents' rewards, and the cost of incentivization ψ is the sum of all scalar incentives. Hence, the global optimum reward for the incentive designer is $10(n - m) - m - m(1 + \epsilon)$.

Cleanup [91]. The *Cleanup* scenario is a high-dimensional gridworld sequential social dilemma that serves as a difficult benchmark for independent learning agents. Agents get +1 individual reward by collecting "apples", whose spawn rate decreases in proportion to the amount of "waste". Each agent can contribute to the public good by firing a cleaning beam to clear waste, but doing so would enable other agents to defect and selfishly collect apples, hence posing a difficult social dilemma. We extended the open-source implementation [178]

to provide a global observation image for the incentive designer.

Gather-Trade-Build (GTB) [42]. The GTB simulation is a 2D grid world in which agents with varying skill levels collect resources that replenish stochastically, spend resources to build houses for coins (i.e., income), and trade coins for resources in an auction system, at the expense of labor costs for each action. It is not known to be a sequential social dilemma. GTB has a much larger state and action space than *Escape Room* and *Cleanup* due to the auction system, which provides 44 trading actions and supplements each agent's spatial observation with the current and historical market information (e.g., counts of bids and asks for various price levels for each resource). At time t, the system productivity prod_t is defined as the sum of all agents' coins, and equality eq_t is defined such that $\text{eq} = 1$ corresponds to uniform coin over agents and $\text{eq} = 0$ means only one agent has non-zero coin. The incentive designer is a central tax planner who optimizes a trade-off between productivity and equality, defined as $\sum_{t=1}^{H} \text{prod}_t \cdot \text{eq}_t$ over horizon H, by imposing taxes according the following mechanism. Each episode lasts for $H = 100$ time steps and consists of 10 tax periods. At the start of each tax period, the ID sets a tax schedule $T\colon \mathbb{R} \to \mathbb{R}$ that determines the tax $T(z)$ applied to an agent's income z earned within the period. T is a bracketed tax schedule based on the US federal taxation scheme: given a set of income thresholds $\{m_b\}_{b=0}^{B=7}$ with $m_0 = 0 < m_1 < \cdots < m_B = \infty$, the ID defines T by setting a vector of *marginal tax rates* $[\tau_b]_{b=1}^{B}$, where $\tau_b \in [0, 1]$ applies to bracket (m_b, m_{b+1}), such that the total tax on income z is given by

$$T(z) = \sum_{b=0}^{B-1} \tau_b \left((m_{b+1} - m_b)\mathbf{1}_{z > m_{b+1}} + (z - m_b)\mathbf{1}_{m_b < z \leq m_{b+1}} \right) \tag{6.7}$$

where $\mathbf{1}_p = 1$ if p is true and 0 otherwise. At the end of each tax period, the total collected tax is evenly distributed back to all agents: if agent i gets total income z_i^p within period p,

then the agent's final adjusted income at the end of the tax period is given by:

$$z_i^p = z_i^p - T(z_i^p) + \frac{1}{N}\sum_{j=1}^{n} T(z_j^p)\,. \tag{6.8}$$

We used the 15×15 map called "env-pure_and_mixed-15x15" [42], which features similar spatial distribution of resource spawn points as the original 25×25 map. Each agent's observation consists of an 11×11 egocentric spatial window, their resource inventories, collection and building skills, personal and other agents' bids and asks, and quantities derived from the current period's tax rate. The ID observes the complete spatial world state, agents' inventories and incomes, cumulative bids and asks, and all derived tax quantities, but does not know agents' private skill and utility functions. Agents have the same discrete action space consisting of movements, building, and trading actions. Section C.2.1 provides more information on observation/action spaces and agent utilities.

6.3.2 Implementation and baselines

We describe the key implementation of all methods here and include all remaining details in Section C.2.2. We use $M = 1$ for MetaGrad across all experiments, such that an ID update occurs after each policy update by agents. We employ pipelining to improve the efficiency of MetaGrad: the validation trajectory $\hat{\boldsymbol{\tau}}$ generated by agents' updated policies (Algorithm 3, line 6), which is required for the ID's update step, is used for the agents' policy update in the next iteration (Algorithm 3, line 5). To differentiate through the agents' learning step, MetaGrad has access to agents' policy parameters and gradients. This assumption can be removed by using behavioral cloning to obtain surrogate models of agents, which has been demonstrated by existing methods that rely on knowledge of agent parameters [96, 40].

Our main baseline is termed **dual-RL**, in which the incentive designer itself is a standard RL agent who optimizes the system-level objective at the same time-scale as the original RL agents. Dual-RL is the centralized analogue of the decentralized agents with "give-

reward" actions, introduced as a baseline in [40]. It is also formally equivalent to the method called "AI economist" in [42]. In *Escape Room* , we compare with discrete-action and continuous-action variants of dual-RL, labeled "dual-RL (d)" and "dual-RL (c)". In *Cleanup*, we compare with "dual-RL (c)". In GTB, we implemented the core aspects of the "AI Economist" based on available information in [42] (relabeled as "dual-RL" here), and also compare with the static US federal tax rates. We tuned hyperparameters for all methods equally using a successive elimination method, detailed in Section C.2.3.

Escape Room. We used policy gradient [177] without parameter sharing as the base agent implementation for all methods. The incentive function μ_η in MetaGrad is a neural network that maps the global state and agents' joint action to a scaled sigmoid output layer of size $|\mathcal{A}| = 3$ (the number of possible agent actions), such that the value of each output node i lies in $(0, 2)$ and is interpreted as the incentive for action i taken by any agent. This parameterization enables MetaGrad to scale to larger number of agents, e.g. ER$(10, 5)$. The cost for incentivization is the sum of all incentives given to agents, and is accounted by $\psi_t(\eta)$ in MetaGrad's loss function.[1] For **dual-RL (d)**, we tried three different sets of discrete incentives $S_r = \{0, 1.1\}$, $S_r = \{0, 1.1, 2.0\}$, and $S_r = \{0, 0.5, 1.0, 1.5, 2.0\}$. The designer's action space is Discrete$(|S_r|^{|\mathcal{A}|})$. Hence, the designer's action is an assignment of a scalar incentive value to each possible agent action, and the policy output is a categorical distribution. In **dual-RL (c)**, the designer's action space is $(0, 2)^n$, and its policy $\pi(a_{\text{ID}}|o)$ is defined by sampling $u \sim \mathcal{N}(\mu_\eta(s, \mathbf{a}), \mathbf{1})$ with neural network $\mu_\eta \colon \mathcal{S} \times \mathcal{A}^n \to \mathbb{R}^n$, then applying the same sigmoid output layer σ as MetaGrad to get $a_{\text{ID}} = \sigma(u)$. The total incentives given to agents are subtracted from R^{ID}. To compare with the method of [98], we implemented separate experiments with actor-critic for agents' policy updates.

Cleanup. We used actor-critic agents with TD(0) critic updates [1] and parameter-sharing as the base agent for all methods. MetaGrad and dual-RL (c) have the same

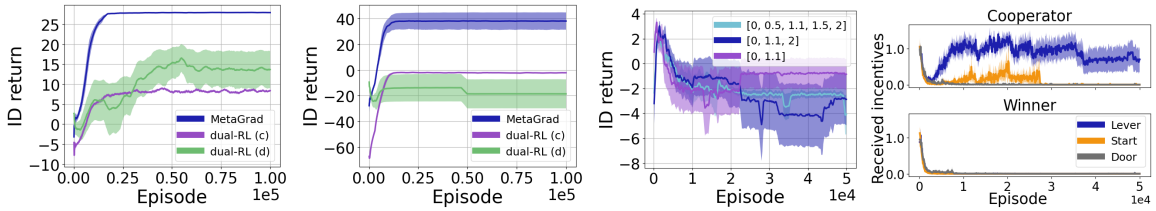
[1] At *train time*, one cannot for cost in R^{ID} of the current episode because MetaGrad only learns from R^{ID} in the next episode, not the episode where incentives are given. At test time, incentives are subtracted from R^{ID} so that comparison to baselines is fair.

architecture as for Escape Room, except that: 1) the observation input for agents and the ID is an RGB image; 2) the ID has a vector observation indicating whether or not each agent performed a cleaning action; 3) each output node of the incentive function is interpreted as the incentive for an action type in the set $\{$fire cleaning beam, collect apples, else$\}$.

Gather-Trade-Build. We used PPO agents [45] with parameter sharing for all methods. The incentive function in MetaGrad has $B = 7$ output nodes, where the value τ_b at each node b is capped by sigmoid activation to lie in $(0, 1)$ and is interpreted as the tax rate τ_b for bracket (m_b, m_{b+1}). By (Equation 6.7), (Equation 6.8), and (Equation C.17), each agent's policy update is a differentiable function of the incentive function parameters η. The ID has seven action subspaces (one for each of the $B = 7$ tax brackets), each with 21 discrete actions that choose the marginal tax rate in $\{0, 0.05, \ldots, 1.0\}$. Dual-RL applies standard RL to the ID, whose action space is a direct product of seven action subspaces (one for each of the $B = 7$ tax brackets), each with 21 discrete choices of the marginal tax rate in $\{0, 0.05, \ldots, 1.0\}$. [42] reported the need for a two-phase curriculum with tax annealing to stabilize training for Dual-RL, which may face difficulties with non-stationarity of the expanded multi-agent system. Hence, in the curriculum version of GTB experiments, we first train agents in a free-market (zero tax) scenario in Phase 1, while the RL-based tax policy is introduced in Phase 2 with a gradual annealing schedule on the maximum tax rate. We repeated experiments without curriculum to investigate the stability of MetaGrad. We trained four independent models per method per case. For measurements of economic activity and tax rates produced by trained models, we report the mean and standard error over the four trained models of the mean over 100 test episodes per model.

6.4 Results

Escape Room. MetaGrad converged to the known global optimum value of approximately 26 in ER$(5, 2)$ and 40 in ER$(10, 5)$ (Figures 6.1a and 6.1b, respectively). Figure 6.1d shows the dynamics of incentivization during training in the case of ER$(2, 1)$, where we labeled



(a) ER(5,2) (b) ER(10,5) (c) Dual-RL (d) (d) Incentives

Figure 6.1: Escape Room. 8 independent runs per method. (a) In ER$(5, 2)$, MetaGrad converges to the global optimum where two agents are incentivized by $1 + \epsilon$ (for some small $\epsilon > 0$) to pull the lever with -1 penalty, three other agents exit the door with +10 reward, which results in a total ID reward near 26. (b) MetaGrad also converges to the global optimum in ER$(5, 2)$ with ID reward near 40. (c) Dual-RL (discrete) was unstable and had high variance across seeds, for various choices of discrete action spaces for the incentive values. (d) Incentives given by MetaGrad for each action by each agent in ER$(2, 1)$.

each agent *at the end of training* as either a "cooperator" or a "winner" based on whether the agent primarily pulls the lever or exits the door, respectively. We see that the cooperator consistently receives incentives of $1 + \epsilon$ during the majority of episodes, which explains the emergence of its cooperative behavior, whereas the winner receives zero incentives asymptotically, which shows the designer learned to avoid unnecessary costs. In contrast, both dual-RL (d) and dual-RL (c) did not solve ER, even including various choices of the discrete action space (Figure 6.1c). This is because a standard RL incentive designer optimizes the expected return of *one* episode, but the impact of its "give-reward" action only appears after agents have conducted learning updates over *many* episodes. In any given episode, the ID's reward contains no information about the impact of the actions it chose during that episode. The only conceivable way that dual-RL learns is by serendipity: the ID's action a in a *previous* episode i led to a change in agents' behavior that results in positive reward for the ID during the *current* episode j, and the ID happened to take a again in episode j, which results in correct credit assignment when learning from episode j. In fact, among eight independent runs for dual-RL, the only run that succeeded had the same random seed as in hyperparameter search. Section C.3, Figure C.1, shows that MetaGrad outperforms the incentive design method of [98], which faced numerical instabilities when extended from matrix games to Markov games.

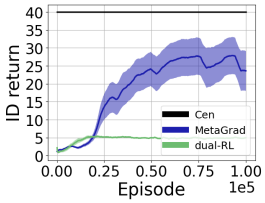


Figure 6.2: 7x7 map

Cleanup. As shown in Figure 6.2, MetaGrad achieved a high level of social welfare, which is only possible because one agent received incentives to take cleaning actions while another agent collects apples. The method labeled "Cen" trains a single policy that acts for both agents; it serves as an empirical upper bound on performance. Under dual-RL, agents did not receive appropriate incentives and hence behaved selfishly—occasionally using the cleaning beam but immediately competing for any apples that spawned—and therefore converged to low social welfare.

6.4.1 Gather-Trade-Build

Social welfare. MetaGrad outperforms both dual-RL and US federal without requiring heuristics such as curriculum learning and tax annealing (Figure 6.3a). It discovers tax rates which differ from the static US federal tax rates in two notable aspects (compare Figure 6.4 and Figure 6.5). Firstly, MetaGrad imposes much higher taxes than US federal on the lowest income bracket (e.g., 0-10 coin), but chooses relatively lower taxes for the next income bracket (10-39). Hence, compared to US federal, there is less incentive for agents to fall in the lowest bracket, which may explain the higher income of agents under MetaGrad versus US federal (Figure 6.12). Secondly, the highest income bracket does not necessarily face significantly higher tax rates than other brackets, and even

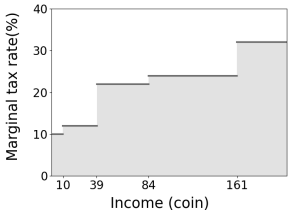
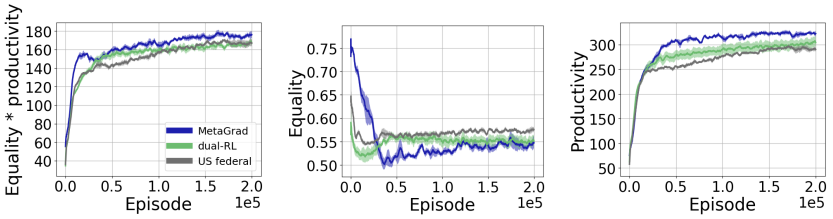


Figure 6.5: US federal



(a) Social welfare (b) Equality (c) Productivity

Figure 6.3: GTB without curriculum: MetaGrad finds tax policies that induce higher social welfare than baselines, by promoting higher productivity at similar levels of equality.

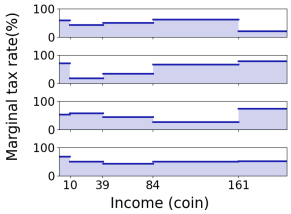
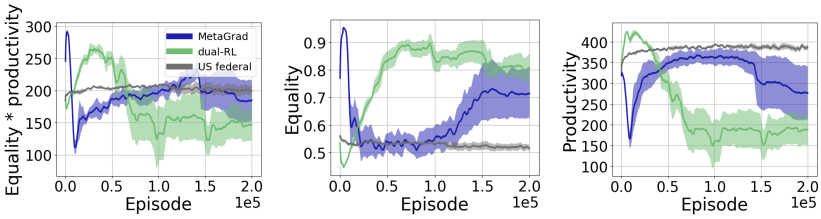


Figure 6.4: MetaGrad tax rates for each independent run.



(a) Social welfare (b) Equality (c) Productivity

Figure 6.6: GTB with curriculum. Both MetaGrad and dual-RL find transient states of high social welfare but are less stable than US federal.

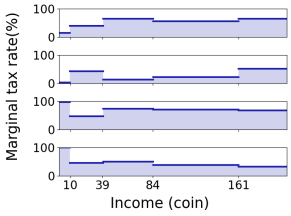
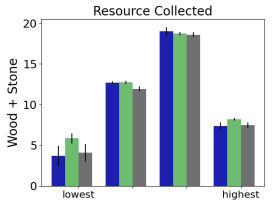


Figure 6.7: MetaGrad tax rates with curriculum.

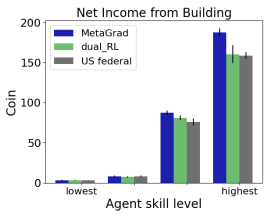
gets the lowest rate in one instance. While this results in lower equality than US federal (Figure 6.3b), the increase in economic activity—such as resource collection, building, and trading (Figure 6.8)—improves system productivity (Figure 6.3c) and ultimately produces significantly higher social welfare.

When curriculum learning is applied to all methods—i.e., for all methods, initializing agents with the same policy that was pre-trained in a free market context—MetaGrad also exceeds the performance of dual-RL. Except for one out of four runs where social welfare abruptly dropped around 150k episodes, MetaGrad also outperforms US federal. Similar to the training dynamics observed in [42], we also observe a transient period where dual-RL passes through unstable local optima (Figure 6.6a, near 25k episodes); however, in contrast to their results, dual-RL did not manage to surpass US federal in asymptotic performance. This is likely because the sudden introduction of a tax planner in Phase 2, after agents have been trained in the free market setting of Phase 1, may do more harm than good for stability, especially when the extra hyperparameters introduced by curriculum learning and tax annealing are hard to tune. Both MetaGrad and dual-RL enact higher taxes at lower income brackets than US federal.

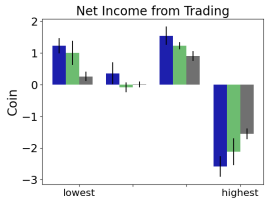
Economic activity. Because skill levels determine the amount of coins per house built, differences in agents' skill levels generate income inequality and behavioral specialization. For example, agents with second-highest skill tend to collect the most resources and sell them for income, whereas agents with the highest skill spend less effort on resource collection



(a) Resource collected



(b) Income from building



(c) Income from trading

Figure 6.8: GTB without curriculum: economic activity after 200k training episodes.

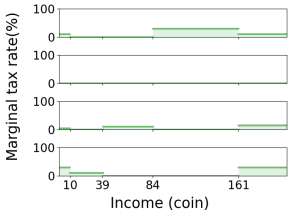
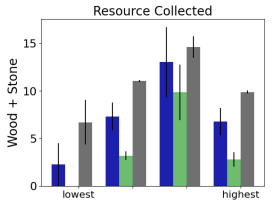
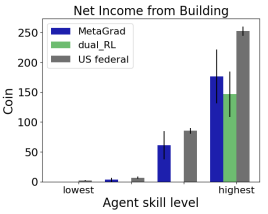


Figure 6.9: Dual-RL tax rates for each independent run.

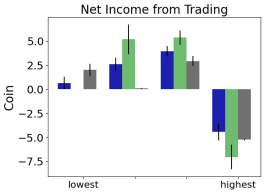
but generate income by building houses from purchased resources (Figures 6.8a and 6.8c). Notably, under all tax policies, all except the highest skill agents receive net positive income from trading (Figure 6.8c). Even though resource collection is comparable across methods, tax policies found by MetaGrad encouraged highest trading activity and hence highest overall income from building, compared to US federal and dual-RL. In the curriculum case, dual-RL tax policies impose high taxation (above 50%) on the three lowest income brackets Figure 6.11. This may explain the fact that the lowest-skilled agent collects zero resources (Figure 6.10a), which lowers overall system productivity.



(a) Resource collection



(b) Building income



(c) Trading income

Figure 6.10: GTB with curriculum: economic activity after 200k training episodes in Phase 2.

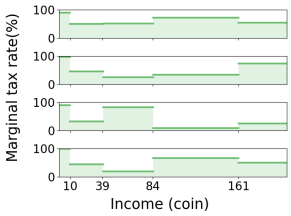


Figure 6.11: Dual-RL tax with curriculum.

Taxation and income. Agents with the two highest skill levels pay significantly less tax for tax policies found by MetaGrad than they do for the US federal tax rates, whereas agents with the two lowest skill pay comparably equal tax (Figure 6.12). This means that MetaGrad does better than US federal at encouraging higher skilled agents to increase economic activity such as building and trading, without affecting resource collection by lower skill agents (as can be seen in Figure 6.8a). While this comes at the expense of lower equality

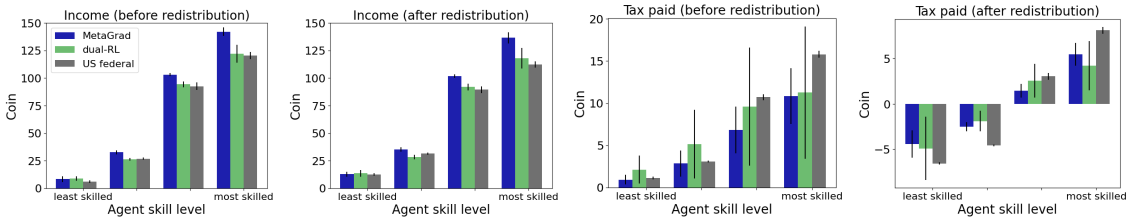


Figure 6.12: GTB without curriculum: income and tax before and after redistribution, after 200k training episodes.

(Figure 6.3b), the incomes of lower-skilled agents both before and after redistribution are actually higher for MetaGrad than US federal, because lower-skilled agents benefit from increased trading activity (Figure 6.8c). Dual-RL tax policies caused agents of all skill levels to pay more taxes than MetaGrad, which is correlated in overall lower building and trading activity.

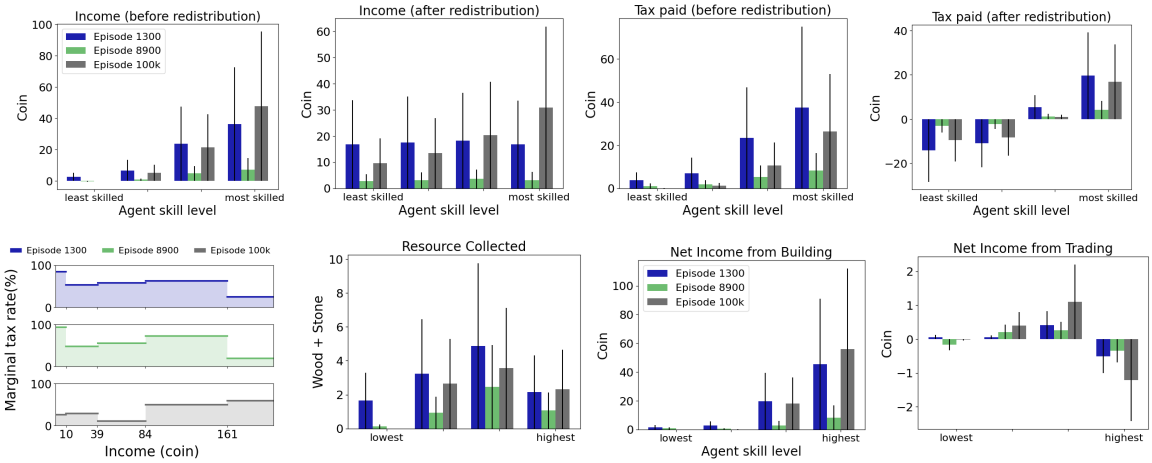


Figure 6.13: Training dynamics. Status of agent and designer behavior at 1300, 8900, and 100k episodes. Top row: income and tax before and after redistribution. Bottom row: tax rates and economic activity.

Training dynamics. In the curriculum setting, the changing tax rates under MetaGrad produced a more dynamical social welfare curve than the fixed US federal rates during training (Figure 6.6a). This can be useful for extracting potential causal relations between taxation and agents' economic behavior. For one particular run of MetaGrad, we measured taxation, income, and economic activity over 100 test episodes at 1300, 8900, and 100k episodes during training, corresponding to the early peak, valley, and steady rising region

in the social welfare curve in Figure 6.6a. These measurements are shown in Figure 6.13, which we use to make the following observations. Social welfare is highest at episode 1300, but agents actually have lower income (before redistribution) at episode 1300 than at episode 100k. This means MetaGrad quickly learned that social welfare can be artificially inflated any productivity level by finding a tax scheme to increase the equality index—hence the sharp early peak in Figure 6.6b. As shown in Figure 6.13, MetaGrad's tax rates at episode 1300 produced nearly uniform income (after redistribution) over all skill levels, by enacting high taxes on the higher skilled agents. This tax policy clearly disincentivizes agents from earning high income, since episode 1300 is followed by a precipitous drop in productivity (see Figure 6.6c) that reaches a global minimum near episode 8900, where agent activity levels (resource collection, building, trading) are lowest. Nonetheless, MetaGrad adapted its tax policy to produce a recovery of productivity at relatively constant equality, from episodes 25k to 100k (Figures 6.6b and 6.6c). The tax policy at episode 100k resembles the progressive schedule of the US federal policy, albeit with significantly lower rates for the 39-84 income bracket that applies to most of the highest-skilled agents, except for the very top earners whose income exceeds 84.

Difficulty of GTB versus Escape Room and Cleanup. Given the clear advantage of MetaGrad over dual-RL in *Escape Room* and *Cleanup* experiments (even though MetaGrad is not tailored to social dilemmas in particular), one may wonder why the advantage is not as stark in GTB. This is because the incentivization problem in GTB is conceptually easier for an incentive designer who uses conventional RL. In GTB, agents can learn from positive rewards regardless of the tax rate, which implies that the ID always receives a learning signal that tracks the agents' behaviors. For dual-RL, the ID receives such feedback in the *next* episode, but at least it is non-zero. However, in the other two problems, if incentives do not pass a threshold ($1 + \epsilon$ to overcome the environment penalty in *Escape Room*; a more complex opportunity cost for sacrificing self-interest in *Cleanup*), then the agents do not receive any predefined environment rewards, which means an RL-based designer

does not receive any feedback at all. In this case, MetaGrad's knowledge of the way that agents' policy parameters change in response to the incentive function, and the use of online cross-validation to learn from the ID's returns in subsequent episodes, is crucial.

6.5 Summary

We proposed the use of complex simulations involving reinforcement learning agents as an *in silico* experimental approach to problems of incentive design. To tackle the issue of delayed impact of incentives, which poses difficulties for directly applying standard RL to the incentive designer, we proposed a meta-gradient approach for the incentive designer to account exactly for the agents' learning response to incentives. The new method significantly outperforms baselines on benchmark problems and also improves the trade-off between productivity and equality in a complex simulated economy.

CHAPTER 7
CONCLUSION

As long as artificial intelligence continues to advance in generality and performance, a multi-agent ecosystem of AI is inevitable. Humanity, still struggling with great collective challenges in the present day, has not had the benefit of prescient design for cooperation in the face of shared interests, not to mention conflicting objectives. Must AI struggle likewise when their time comes? This dissertation puts forth the thesis that cooperation in multi-agent systems is possible via multi-agent reinforcement learning, both in the case of pure common interest where a single principal can directly optimize individual and team-goals via centralized training, and in the mixed-motive setting where cooperation for high social welfare may emerge via centralized or decentralized incentivization. The thesis is substantiated by the design and empirical evaluation in simulation of new multi-agent reinforcement learning algorithms, as summarized below.

7.1 Summary of Contributions

Chapter 3 proposes a fully-cooperative MARL algorithm for the multi-goal setting, where global system optimality is defined as achieving different individual goals of all agents. This multi-goal setting accentuates two key challenges: 1) multi-agent credit assignment, the problem of effectively attributing a reward signal received by one agent to its own actions or the actions of other agents; and 2) cooperative multi-agent exploration, the problem of efficiently finding relevant regions of an exponential global state space where optimal cooperation can occur. The proposed algorithm, called **CM3**, addresses the first challenge by a credit function, a multi-agent variant of the action-value function, that evaluates pairs of actions and goals of different agents. The second challenge of exploration is addressed by a multi-agent curriculum method that first conducts rapid single-agent RL training in

an induced MDP, which primes the agents with greedy behaviors that easily discover the relevant states in the full multi-agent environment. CM3 performs uniformly the best or equal to the best state-of-the-art method over varied cooperation problems in simulation, and extensive ablations show the benefits of each component technique.

Whereas goal assignment are predetermined and fixed within an episode in Chapter 3, Chapter 4 focuses on discovering and learning temporally-extended skills—which can be viewed as sub-goals—for a team of hierarchical agents to maximize a team objective. The proposed method, called **HSD**, is motivated by the team-sports setting: teams employ centralized coaching to coordinate skills, while fully-decentralized players execute skills via low-level actions at match time. Specifically, the method uses centralized training with decentralized execution for high-level policies that select skill variables at a slow timescale; in turn, fully-decentralized low-level policies are conditioned on the selected skills and take primitive actions at a fast timescale. The set of initially meaningless skill variables induce distinguishable agent behavior—and hence acquire semantic meaning—via a feedback loop: conditioned on skill variables, low-level policies are encouraged by a decodability reward to generate trajectory segments that are easy for a decoder to classify the skill variable that was used. Implemented with baseline MARL methods as subcomponents, HSD is shown to discover interpretable skills that are useful in a challenging team sports simulation, making HSD competitive with non-hierarchical baselines in terms of win rate and outperform baselines on generalization.

Chapter 5 tackles the more challenging problem of cooperation among selfish independent agents in mixed-motive settings where no central entity directly optimizes global social welfare. In principle, without any form of centralization, cooperation is unattainable. Nonetheless, Chapter 5 shows that it is possible for cooperation to emerge if certain conditions are met: 1) some, or all, agents in the environment are equipped with the ability to give incentives to other agents; 2) incentives have the same normative value as environment rewards; 3) an agent who gives incentives accounts for the indirect impact of incentives

on their own future performance, through the direct impact on recipients' learning process. Specifically, we design an agent called **LIO** that learns to incentivize other learning agents to cooperate, by extending the principle of online cross validation and meta-gradient reinforcement learning to the multi-agent setting. LIO provably attains the global optimum in classical matrix game social dilemmas, and empirically approaches the optimum in difficult high-dimensional intertemporal social dilemma benchmarks.

Chapter 6 removes the assumption in Chapter 5 that agents themselves are equipped with the ability to give incentives; instead, we instate a central designer who explicitly optimizes social welfare by intervening on the reward functions of an agent population. We show the compatibility of the technical method in Chapter 5 with more effective objective functions in reinforcement learning. This enables us to tackle the problem of optimal tax design for optimizing a trade-off between productivity and equality in a larger and more complex multi-agent RL simulation of resource collection, market dynamics and taxation. The learning-aware nature of our method leads to consistently higher social welfare than the state-of-the-art, and its fast adaptation to agents' response enables us to conduct behavioral analysis to extract domain insights regarding taxation.

7.2 Opportunities for Future Work

Fully-cooperative multi-agent learning. The credit function and curriculum in CM3 are effective heuristics for multi-goal MARL. However, the credit function is limited to evaluating pairwise interactions between an agent's action and another agent's long-term value. Hence, there is room to design more general methods that capture higher-order interaction among actions and goals of many agents, for example by using graph neural networks [179], and to investigate the question of optimal multi-agent credit assignment. Furthermore, the curriculum is specifically designed to target the challenge of exploration in the multi-goal setting. The effectiveness of the curriculum depends on the ability of a single agent to make progress toward its goal in the absence of other agents in Stage

1. While this is true in many application contexts, predominantly in physical navigation tasks, where one can rely on reward shaping to ensure meaningful single-agent learning, one may construct failure cases with strict constraints where reward shaping is not allowed and single-agent pre-training does not produce a meaningful initialization for Stage 2. This motivates further research on an overarching framework to organize the myriad of possible dimensions of multi-agent curricula—such as varying the number of agents, the nature of tasks, or a more complex co-evolution of agents and tasks—that can be used to circumvent limitations and constraints in especially difficult scenarios. One may also generalize the problem formulation of multi-goal MARL to the case of inhomogeneous agents and settings without known goal assignments.

Hierarchical multi-agent learning. One may generalize the approach used in HSD for skill discovery without hand-crafted rewards to discover other abstractions in model-free MARL, such as different roles [89] to fulfill based on each agent's unique features in a heterogeneous team. Roles can also bias an agent's choice of skills. Whereas HSD assumed that all agents synchronously choose new skill variables after every certain number of time steps, one may build on methods for asynchronous termination of options in the single-agent setting [82] to allow learning a larger space of policies. Optimizing the number of skills is also a natural generalization. It would be interesting to apply curriculum-learning approaches that initialize skill-conditioned low-level policies from pretraining in an induced single-agent setting, such as shown in CM3 [19], or using expert data, analogous to professional players practicing skills outside of team matches. This may speed up training since low-level policies can already generate useful trajectories that can be segmented into distinguishable skills.

Decentralized incentivization as an approach for emergent cooperation. Our design of the LIO agent to encourage the emergence of cooperation among decentralized agents in social dilemmas poses many new questions. On the theoretical side, more work is needed to analyze the simultaneous processes of updating incentive functions, which continuously

modifies the game and the set of equilibria, and updating policies with these changing incentives. Previous work that analyze the convergence of gradient-based learning in differentiable games with fixed rewards [180, 97] and the convergence of learning in Stackelberg games [181] are relevant starting points. There are many more open topics on the algorithmic and agent design aspects of incentivization. How can an agent account for the cost of incentives in an adaptive way? An improvement to LIO would be a handcrafted or learned mechanism that prevents the cost from driving the incentive function to zero before the effect of incentives on other agents' learning is measurable. How should agents better account for the longer-term effect of incentives? One possibility is to differentiate through a sequence of gradient descent updates by recipients, during which the incentive function is fixed, but this trades off the quality of a long-term measurement with the frequency of updates to the incentive function. Can multi-agent reinforcement learning generate emergent social factors that modulate the effect of incentives in an agent population? LIO assumes that recipients cannot reject an incentive, but a more intelligent agent may selectively accept a subset of incentives based on its appraisal of the other agents' behavior, such as whether their behavior abides by social norms within the agent population. An agent population also provides fertile ground for more rigorous study of the emergence of social norms [182]. How should one solve higher-order social dilemmas that come with incentivization? Since agents incur a cost for giving incentives, agents have the temptation to free-ride on the effort that other agents spend on sending incentives, and this poses a second-order social dilemma.

Mechanism design. Beyond incentive design, one may consider the extension of ideas in this work to the context of *mechanism* design, interpreted in the general sense of modifying the underlying dynamics of the environment [183] to shape agents' behavior and optimize a system-level objective. Even when the agents' learning process is not differentiable with respect to the change in environment dynamics, our work has shown the importance of evaluating an intervention by its long-term effect through the agents' learning—the principle of online cross validation still applies. More generally, our application to the

Gather-Trade-Build simulation benchmark shows the feasibility of a path toward a data and simulation-driven approach for improving complex systems in society. The realization of this research agenda must build upon advances in transfer learning and robustness, in order to handle the gap between simulation and reality.

Appendices

APPENDIX A

COOPERATIVE MULTI-GOAL MULTI-AGENT REINFORCEMENT LEARNING

A.1 Algorithm

Algorithm 4 Initialization of networks

procedure INITIALIZE(c)
  if $c = 1$ **then**
    Set number of agents $N = 1$
    Initialize Stage 1 main networks $Q_g := Q = Q^1, \pi := \pi^1$ with parameters $\theta_{Q^1}, \theta_{\pi^1}$
    Initialize target networks with $\theta'_{\pi^1}, \theta'_{Q^1}$
  else if $c = 2$ **then**
    Instantiate $N > 1$ agents
    Construct global $Q_g := Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) = \{Q^1, Q_g^2\}$, credit function $Q_c := Q_n^{\boldsymbol{\pi}}(s, a^m) = \{Q^1, Q_c^2\}$ and $\pi := \{\pi^1, \pi^2\}$ using function augmentation with parameters $\theta_{Q_g}, \theta_{Q_c}, \theta_\pi$
    Initialize target networks with $\theta'_{Q_g}, \theta'_{Q_c}, \theta'_\pi$
    Restore values of trained parameters $\theta_{Q^1}, \theta_{\pi^1}$ into the respective subsets of $\theta_{Q_g}, \theta_{Q_c}, \theta_\pi$
  end if
  Return all trainable parameters
end procedure

Algorithm 5 Collect one episode of experience

procedure RUNEPISODE
  Assign goal(s) g_e^n to agent(s) according to given distribution
  Get initial state s_1 and observation(s) $\mathbf{o}_1$
  for $t = 1$ to T **do** ▷ execute policies in environment
    Sample action $a_t^n \sim \pi(a_t^n | o_t^n; \theta_\pi, \epsilon)$ for each agent.
    Execute action(s) $\mathbf{a}_t$, receive $\{r_t^n\}_n$, s_{t+1}, and $\mathbf{o}_{t+1}$
    Store $(s_t, \mathbf{o}_t, \mathbf{g}_e, \mathbf{a}_t, \{r_t^n\}_n, R_t^g, s_{t+1}, \mathbf{o}_{t+1})$ into B
    $s_t \leftarrow s_{t+1}, \mathbf{o}_t \leftarrow \mathbf{o}_{t+1}$
  end for
end procedure

Algorithm 6 Train step

procedure TRAIN
  for epochs $1 \dots K$ **do**
    Sample minibatch of S transitions $(s_i, \mathbf{o}_i, \mathbf{g}_i, \mathbf{a}_i, \{r_i^n\}_n, s_{i+1}, \mathbf{o}_{i+1})$ from B
    Compute global target for all n: $x_i^n = r_i^n + \gamma Q(s_{i+1}, \mathbf{a}_{i+1}, g_i^n; \theta'_{Q_g})|_{\mathbf{a}_{i+1} \sim \boldsymbol{\pi}'}$
    Gradient descent on $\mathcal{L}(\theta_{Q_g}) = \frac{1}{S} \sum_i \frac{1}{N} \sum_{n=1}^{N} \left(x_i^n - Q(s_i, \mathbf{a}_i, g_i^n; \theta_{Q_g}) \right)^2$
    if c = 1 **then**
      $A^{\pi}(s_i, a_i) = Q^1(s_i, a_i, g_i; \theta_{Q^1}) - \sum_{\hat{a}_i} \pi(\hat{a}_i | o_i, g_i) Q^1(s_i, \hat{a}_i, g_i; \theta_{Q^1})$
    else if c = 2 **then**
      $\forall m, n \in [1..N]$, compute target $y_i^n = r_i^n + \gamma Q(s_{i+1}, a_{i+1}^m, g^n; \theta'_{Q_c})|_{a_{i+1}^m \sim \pi'^m}$
      Minimize (Equation 3.4):
$\mathcal{L}(\theta_{Q_c}) = \frac{1}{S} \sum_i \frac{1}{N^2} \sum_{n=1}^{N} \sum_{m=1}^{N} \left(y_i^n - Q(s_i, a_i^m, g_i^n; \theta_{Q_c}) \right)^2$
      Compute advantage:
$A^{\boldsymbol{\pi}}_{n,m}(s_i, \mathbf{a}_i) := Q(s_i, \mathbf{a}_i, g_i^n; \theta_{Q_g}) - \sum_{\hat{a}^m} \pi(\hat{a}^m) Q(s_i, \hat{a}^m, g_i^n; \theta_{Q_c})$
    end if
    $\nabla_{\theta_\pi} J(\boldsymbol{\pi}) = \frac{1}{S} \sum_i \sum_{m,n=1}^{N} (\nabla_{\theta_\pi} \log \pi(a_i^m | o_i^m, g_i^m)) A^{\boldsymbol{\pi}}_{n,m}(s_i, \mathbf{a}_i)$
    Update policy: $\theta_\pi \leftarrow \theta_\pi + \beta \nabla_{\theta_\pi} J(\boldsymbol{\pi})$
    Update all target network parameters using: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
    Reset buffer B
  end for
end procedure

Algorithm 7 Cooperative multi-goal multi-stage multi-agent reinforcement learning (CM3)

for curriculum stage $c = 1$ to 2 **do**
  Computational graph $\leftarrow$ INITIALIZE(c) (Algorithm 4)
  Set all target network weights to equal main networks weights
  Initialize exploration parameter $\epsilon = \epsilon_{\text{start}}$ and empty replay buffer B
  for each training episode $e = 1$ to E **do**
    RUNEPISODE() (Algorithm 5)
    if $e \mod E_{\text{train}} = 0$ **then**
      TRAIN() (Algorithm 6)
    end if
    If $\epsilon > \epsilon_{\text{end}}$, then $\epsilon \leftarrow \epsilon - \epsilon_{\text{step}}$
  end for
end for

Off-policy training with a large replay buffer allows RL algorithms to benefit from less correlated transitions [184, 54]. The algorithmic modification for off-policy training is to maintain a circular replay buffer that does not reset (i.e. remove line 38), and conduct training (lines 24-41) while executing policies in the environment (lines 17-22). Despite introducing bias in MARL, we found that off-policy training benefited CM3 in SUMO and Checkers.

A.2 Derivations

A.2.1 Credit function recursion

By stationarity and relabeling t, the credit function can be written:

$$\begin{aligned} Q_n^{\boldsymbol{\pi}}(s, a^m) &:= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, a_0^m = a^m\Big] \\ &= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_1 = s, a_1^m = a^m\Big] \end{aligned}$$

Using the law of iterated expectation, the credit function satisfies the Bellman expectation

equation (Equation 3.2):

$$
\begin{aligned}
Q_n^{\boldsymbol{\pi}}(s, a^m) &= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, a_0^m = a^m\Big] \\
&= \mathbb{E}_{\boldsymbol{\pi}}\Big[R(s_0, \mathbf{a}_0, g^n) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, a_0^m = a^m\Big] \\
&= \mathbb{E}_{s_1, a_1^m | s_0, a_0, \boldsymbol{\pi}}\Bigg[\mathbb{E}_{\boldsymbol{\pi}}\Big[R(s_0, \mathbf{a}_0, g^n) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big| \\
&\qquad s_0 = s, a_0^m = a^m, s_1 = s', a_1^m = \hat{a}^m\Big] \;\Big|\; s_0 = s, a_0^m = a^m\Bigg] \\
&= \mathbb{E}_{s_1, a_1^m | s_0, a_0, \boldsymbol{\pi}}\Bigg[\sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) R(s, (a^m, a^{-m}), g^n) \\
&\quad + \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, a_0^m = a^m, s_1 = s', a_1^m = \hat{a}^m\Big] \;\Big|\; s_0 = s, a_0^m = a^m\Bigg] \\
&= \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) R(s, (a^m, a^{-m}), g^n) + \mathbb{E}_{s_1, a_1^m | s_0, a_0, \boldsymbol{\pi}}\Bigg[\mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big| \\
&\qquad s_0 = s, a_0^m = a^m, s_1 = s', a_1^m = \hat{a}^m\Big] \;\Big|\; s_0 = s, a_0^m = a^m\Bigg] \\
&= \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) R(s, (a^m, a^{-m}), g^n) + \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) \sum_{s'} P(s' | s, (a^m, a^{-m})) \cdot \\
&\qquad \sum_{\hat{a}^m} \pi(\hat{a}^m | o^m(s')) \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_1 = s', a_1^m = \hat{a}^m\Big] \\
&= \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) \Bigg[R(s, (a^m, a^{-m}), g^n) + \gamma \sum_{s'} P(s' | s, (a^m, a^{-m})) \sum_{\hat{a}^m} \pi(\hat{a}^m | o^m(s')) \cdot \\
&\qquad \mathbb{E}_{\pi}\Big[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_1 = s', a_1^m = \hat{a}^m\Big]\Bigg] \\
&= \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m} | s, \mathbf{g}^{-m}) \Big[R(s, (a^m, a^{-m}), g^n) + \\
&\qquad \gamma \sum_{s'} P(s' | s, (a^m, a^{-m})) \sum_{\hat{a}^m} \pi^m(\hat{a}^m | o^m(s')) Q_n^{\boldsymbol{\pi}}(s', \hat{a}^m)\Big] \\
&= \mathbb{E}_{\boldsymbol{\pi}}\Big[R(s_t, \mathbf{a}_t, \mathbf{g}^n) + \gamma Q_n^{\boldsymbol{\pi}}(s_{t+1}, a_{t+1}^m) \Big| s_t = s, a_t^m = a^m\Big]
\end{aligned}
$$

□

The goal-specific joint value function is the marginal of the credit function:

$$\begin{aligned} V_n^{\boldsymbol{\pi}}(s) &= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s\Big] \\ &= \mathbb{E}_{a_0^m|s_0,\boldsymbol{\pi}}\Big[\mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, a_0^m = a^m\Big] \;\Big|\; s_0 = s\Big] \\ &= \sum_{a^m} \pi(a^m|o^m(s), g^m) Q_n^{\boldsymbol{\pi}}(s, a^m) \end{aligned} \qquad \square$$

The credit function can be expressed in terms of the goal-specific action-value function:

$$\begin{aligned} V_n^{\boldsymbol{\pi}}(s) &= \sum_{a^m} \pi(a^m|o^m, g^m) Q_n^{\boldsymbol{\pi}}(s, a^m) \quad \text{by (Equation 3.3)} \\ V_n^{\boldsymbol{\pi}}(s) &= \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) \quad \text{by (Equation A.2)} \\ &= \sum_{a^m} \sum_{a^{-m}} \pi(a^m|o^m, g^m) \boldsymbol{\pi}(a^{-m}|s, g^{-m}) Q_n^{\boldsymbol{\pi}}(s, (a^m, a^{-m})) \\ \Rightarrow Q_n^{\boldsymbol{\pi}}(s, a^m) &= \sum_{a^{-m}} \boldsymbol{\pi}(a^{-m}|s, g^{-m}) Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) \end{aligned} \qquad \square$$

A.2.2 Cooperative multi-goal credit function based MARL policy gradient

First we state some elementary relations between global functions $V_n^{\pi}(s)$ and $Q_n^{\pi}(s, \mathbf{a})$. These carry over directly from the case of an MDP, by treating the joint policy $\boldsymbol{\pi}$ as as an effective "single-agent" policy and restricting attention to a single goal g^n (standard derivations are included at the end of this section).

$$Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) = R(s, \mathbf{a}, g^n) + \gamma \sum_{s'} P(s'|s, \mathbf{a}) V_n^{\boldsymbol{\pi}}(s') \tag{A.1}$$

$$V_n^{\boldsymbol{\pi}}(s) = \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) \tag{A.2}$$

We follow the proof of the policy gradient theorem [56]:

$$
\begin{aligned}
\nabla_\theta V_n^{\boldsymbol{\pi}}(s) &= \nabla_\theta \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) \\
&= \sum_{\mathbf{a}} \Big[\big(\nabla_\theta \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})\big) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) + \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) \nabla_\theta Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) \Big] \\
&= \sum_{\mathbf{a}} \Big[\big(\nabla_\theta \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})\big) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) + \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) \nabla_\theta \big(R(s,\mathbf{a},g^n) + \\
&\qquad \gamma \sum_{s'} P(s'|s,\mathbf{a}) V_n^{\boldsymbol{\pi}}(s') \big) \Big] \\
&= \sum_{\mathbf{a}} \Big[\big(\nabla_\theta \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})\big) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) + \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) \gamma \sum_{s'} P(s'|s,\mathbf{a}) \nabla_\theta V_n^{\boldsymbol{\pi}}(s') \Big] \\
&= \sum_{\hat{s}} \sum_{k=0}^{\infty} \gamma^k P(s \to \hat{s}, k, \boldsymbol{\pi}) \sum_{\mathbf{a}} (\nabla_\theta \boldsymbol{\pi}(\mathbf{a}|\hat{s},\mathbf{g})) Q_n^{\boldsymbol{\pi}}(\hat{s},\mathbf{a}) \quad \text{(by recursively unrolling)} \\
\nabla_\theta J_n(\boldsymbol{\pi}) := \nabla_\theta V_n^{\boldsymbol{\pi}}(s_0) &= \sum_{s} \sum_{k=0}^{\infty} \gamma^k P(s_0 \to s, k, \boldsymbol{\pi}) \sum_{\mathbf{a}} (\nabla_\theta \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) \\
&= \sum_{s} \rho^{\boldsymbol{\pi}}(s) \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) \big(\nabla_\theta \log \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})\big) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) \\
&= \mathbb{E}_{\boldsymbol{\pi}} \left[\left(\nabla_\theta \log \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g})\right) Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) \right] \qquad \text{(A.3)}
\end{aligned}
$$

We can replace $Q_n^{\boldsymbol{\pi}}(s,\mathbf{a})$ by the advantage function $A_n^{\boldsymbol{\pi}}(s,\mathbf{a}) := Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) - V_n^{\boldsymbol{\pi}}(s)$, which does not change the expectation in Equation (A.3) because:

$$
\begin{aligned}
\mathbb{E}_{\boldsymbol{\pi}} \left[\nabla_\theta \log \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) V_n^{\boldsymbol{\pi}}(s) \right] &= \sum_{s} \rho^{\boldsymbol{\pi}}(s) \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) \nabla_\theta \log \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) V_n^{\boldsymbol{\pi}}(s) \\
&= \sum_{s} \rho^{\boldsymbol{\pi}}(s) V_n^{\boldsymbol{\pi}}(s) \nabla_\theta \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s,\mathbf{g}) = 0
\end{aligned}
$$

So the gradient (Equation A.3) can be written

$$
\nabla_\theta J_n(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}} \Big[\big(\nabla_\theta \sum_{m=1}^{N} \log \pi(a^m|o^m,g^m)\big) \big(Q_n^{\boldsymbol{\pi}}(s,\mathbf{a}) - V_n^{\boldsymbol{\pi}}(s)\big) \Big] \qquad \text{(A.4)}
$$

Recall that from (Equation 3.3), for any choice of agent label $k \in [1..N]$:

$$V_n^{\boldsymbol{\pi}}(s) = \sum_{a^k} \pi(a^k|o^k, g^k) Q_n^{\boldsymbol{\pi}}(s, a^k) \tag{A.5}$$

Then substituting (Equation 3.3) into (Equation A.4):

$$\nabla_\theta J_n(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}\Big[\Big(\nabla_\theta \sum_{m=1}^{N} \log \pi(a^m|o^m, g^m)\Big) A_{n,k}^{\boldsymbol{\pi}}(s, \mathbf{a})\Big] \tag{A.6}$$

$$A_{n,k}^{\boldsymbol{\pi}}(s, \mathbf{a}) := Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) - \sum_{\hat{a}^k} \pi(\hat{a}^k|o^k, g^k) Q_n^{\boldsymbol{\pi}}(s, \hat{a}^k) \tag{A.7}$$

Now notice that the choice of k in (Equation A.7) is completely arbitrary, since (Equation 3.3) holds for any $k \in [1..N]$. Therefore, it is valid to distribute $A_{n,k}^{\boldsymbol{\pi}}(s, \mathbf{a})$ into the summation in (Equation A.6) *using the summation index m instead of k*. Further summing (Equation A.6) over all n, we arrive at the result of Proposition 2:

$$\nabla_\theta J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{m=1}^{N}\sum_{n=1}^{N}\Big(\nabla_\theta \log \pi(a^m|o^m, g^m)\Big) A_{n,m}^{\boldsymbol{\pi}}(s, \mathbf{a})\Big]$$

$$A_{n,m}^{\boldsymbol{\pi}}(s, \mathbf{a}) := Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) - \sum_{\hat{a}^m} \pi(\hat{a}^m|o^m, g^m) Q_n^{\boldsymbol{\pi}}(s, \hat{a}^m) \qquad \square$$

The relation between $V_n^{\pi}(s)$ and $Q_n^{\pi}(s, \mathbf{a})$ in (Equation A.1) and (Equation A.2) are

derived as follows:

$$
\begin{aligned}
Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) &:= \mathbb{E}_{\pi}\Big[\sum_t \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\Big] \\
&= \mathbb{E}_{\pi}\Big[R(s_0, \mathbf{a}_0, g^n) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\Big] \\
&= R(s, \mathbf{a}, g^n) + \mathbb{E}_{s_1|s_0,\mathbf{a}_0,\boldsymbol{\pi}}\Big[\mathbb{E}_{\pi}\Big[\sum_{t=1}^{\infty} R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}, s_1 = s'\Big] \;\Big| \\
&\qquad s_0 = s, \mathbf{a}_0 = \mathbf{a}\Big] \\
&= R(s, \mathbf{a}, g^n) + \gamma \sum_{s'} P(s'|s, \mathbf{a}) \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_1 = s'\Big] \\
&= R(s, \mathbf{a}, g^n) + \gamma \sum_{s'} P(s'|s, \mathbf{a}) V_n^{\boldsymbol{\pi}}(s') \\
V_n^{\boldsymbol{\pi}}(s) &:= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s\Big] \\
&= \mathbb{E}_{\mathbf{a}_0|s_0,\boldsymbol{\pi}}\Big[\mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\Big] \;\Big|\; s_0 = s\Big] \\
&= \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{a}_t, g^n) \;\Big|\; s_0 = s, \mathbf{a}_0 = \mathbf{a}\Big] \\
&= \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) \qquad \square
\end{aligned}
$$

A.3 Variance

A.3.1 Variance of COMA gradient.

Let $Q := Q^{\boldsymbol{\pi}}(s, \mathbf{a}, \mathbf{g})$ denote the centralized Q function, let $\pi(a^n) := \pi(a^n|o^n, g^n)$ denote a single agent's policy, and let $\pi(a^{-n}) := \pi(a^{-n}|o^{-n}, g^{-n})$ denote the other agents' joint policy.

In cooperative multi-goal MARL, the direct application of COMA has the following gradient.

$$\nabla_\theta J = \mathbb{E}\Big[\sum_n \nabla_\theta \log \pi(a^n|o^n, g^n)\big(Q - b_n(s, a^{-n}, \mathbf{g})\big)\Big]$$
$$b_n(s, a^{-n}, \mathbf{g}) := \sum_{\hat{a}^n} \pi(\hat{a}^n|o^n, g^n) Q^{\boldsymbol{\pi}}(s, \hat{a}^n, a^{-n}, \mathbf{g})$$

Define the following:

$$z_n := \nabla_\theta \log \pi(a^n|o^n, g^n)$$
$$f_n := \nabla_\theta \log \pi(a^n|o^n, g^n)\big(Q - b_n(s, a^{-n})\big) = z_n\big(Q - b_n(s, a^{-n}, \mathbf{g})\big)$$

Define $M_{nm} := \mathbb{E}_{\boldsymbol{\pi}}[f_n]^T \mathbb{E}_{\boldsymbol{\pi}}[f_m]$ and let $M := \sum_{n,m} M_{nm}$. Then we have $M_{nm} = \mathbb{E}_{\boldsymbol{\pi}}[z_n Q]^T \mathbb{E}_{\boldsymbol{\pi}}[z_m Q]$ since

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\pi}}[z_n b_n] &= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_s \rho^{\boldsymbol{\pi}}(s) \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|s, \mathbf{g}) \nabla_\theta \log \pi(a^n|o^n, g^n) b_n(s, a^{-n}, \mathbf{g})\Big] \\
&= \sum_s \rho^{\boldsymbol{\pi}}(s) \sum_{a^{-n}} \pi^{-n}(a^{-n}|o^{-n}, g^{-n}) \cdot \big[\\
&\quad \sum_{a^n} \pi(a^n|o^n, g^n) \nabla_\theta \log \pi(a^n|o^n, g^n) b_n(s, a^{-n}, \mathbf{g})\big] \\
&= \sum_s \rho^{\boldsymbol{\pi}}(s) \sum_{a^{-n}} \pi^{-n}(a^{-n}|o^{-n}, g^{-n}) \sum_{a^n} \nabla_\theta \pi(a^n|o^n, g^n) b_n(s, a^{-n}, \mathbf{g}) \\
&= \sum_s \rho^{\boldsymbol{\pi}}(s) \sum_{a^{-n}} \pi^{-n}(a^{-n}|o^{-n}, g^{-n}) b_n(s, a^{-n}, \mathbf{g}) \nabla_\theta \sum_{a^n} \pi(a^n|o^n, g^n) = 0
\end{aligned}$$

Since the COMA gradient is $\mathbb{E}_{\boldsymbol{\pi}}[\sum_{n=1}^{N} f_n]$. its variance can be derived to be [128]:

$$\begin{aligned}\mathrm{Var}(\sum_{n=1}^{N} f_n) = &\sum_{n} \mathbb{E}_{\boldsymbol{\pi}}\Big[z_n^T z_n Q^2 - 2b_n z_n^T z_n Q + b_n^2 z_n^T z_n\Big] \\ &+ \sum_{n}\sum_{m\neq n} \mathbb{E}_{\boldsymbol{\pi}}\Big[z_n^T z_m (Q-b_n)(Q-b_m)\Big] - M\end{aligned}$$

A.3.2 Variance of the CM3 gradient

For convenience, let $Q_n := Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) = Q^{\boldsymbol{\pi}}(s, \mathbf{a}, g^n)$ denote the global Q function for goal g^n, and let $\pi(a^m) := \pi(a^m|o^m, g^m)$. The CM3 gradient can be rewritten as

$$\begin{aligned}\nabla_\theta J(\boldsymbol{\pi}) &= \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{n=1}^{N}\sum_{m=1}^{N} \nabla_\theta \log \pi(a^m)\big(Q_n - b_{nm}(s)\big)\Big] \\ b_{nm}(s) &:= \sum_{\hat{a}^m} \pi(\hat{a}^m) Q_n^{\boldsymbol{\pi}}(s, \hat{a}^m)\end{aligned}$$

As before, $z_m := \nabla_\theta \log \pi(a^m)$. Define $h_{nm} := z_m(Q_n - b_{nm}(s))$ and let $h_n := \sum_m h_{nm}$. Then the variance is

$$\begin{aligned}\mathrm{Var}(\sum_n h_n) &= \sum_n \mathrm{Var}(h_n) + \sum_n \sum_{m\neq n} \mathrm{Cov}(h_n, h_m) \\ &= \sum_n \Big(\sum_m \mathrm{Var}(h_{nm}) + \sum_m \sum_{k\neq m} \mathrm{Cov}(h_{nm}, h_{nk})\Big) + \sum_n \sum_{m\neq n} \mathrm{Cov}(h_n, h_m)\end{aligned}$$

A.4 Example of greedy initialization for MARL exploration

A greedy initialization can provide significant improvement in multi-agent exploration versus naïve random exploration, as shown by a simple thought experiment. Consider a two-player **MG** defined by a 4×3 gridworld with unit actions (up, down, left, right). Agent A starts at (1,2) with goal (4,2), while agent B starts at (4,2) with goal (1,2). The *greedy policy* for each agent in **MG** is to move horizontally toward its target, since this is optimal in the induced **M** (when the other agent is absent). Case 1: Suppose that for

$\epsilon \in (0, 1)$, A and B follow greedy policies with probability $1 - \epsilon$, and take random actions ($p(a) = 1/4$) with probability ϵ. Then the probability of a symmetric optimal trajectory is $P(\text{cooperate}) = 2\epsilon^2((1 - \epsilon) + \epsilon/4)^8$. For $\epsilon = 0.5$, $P(\text{cooperate}) \approx 0.01$. Case 2: If agents execute uniform random exploration, then $P(\text{cooperate}) = 3.05\text{e-}5 \ll 0.01$.

A.5 Generalization

Table A.1: Test performance with heavy traffic on difficult initial and goal lanes configurations

Config	Initial lanes	Goal lanes	CM3	IAC	COMA
C1	$[1, 2]$	$[3, 0]$	**16.17**	11.40	10.00
C2	Unif. random	Unif. random	**14.93**	12.20	12.93
C3	$[1, 2]$	$[2, 1]$	**15.85**	14.32	15.00
C4	$[0, 1]$	$[3, 2]$	**16.35**	9.73	8.1

We investigated whether policies trained with few agent vehicles ($N = 2$) on an empty road can generalize to situations with heavy SUMO-controlled traffic. We also tested on initial and goal lane configurations (C3 and C4) which occur with low probability when training with configurations C1 and C2. Table A.1 shows the sum of agents' reward, averaged over 100 test episodes, on these configurations that require cooperation with each other and with minimally-interactive SUMO-controlled vehicles for success. CM3's higher performance than IAC and COMA in training is reflected by better generalization performance on these test configurations. There is almost negligible decrase in performance from train Figure 3.5d to test, giving evidence to our hypothesis that centralized training with few agents is feasible even for deployment in situations with many agents, for certain applications where local interactions are dominant.

A.6 Absolute runtime

CM3's higher sample efficiency does not come at greater computational cost, as all methods' runtimes are within an order of magnitude of one another. Test times have no significant

difference as all neural networks were similar.

Table A.2: Absolute training runtime of all algorithms in seconds

Environment	CM3	IAC	COMA	QMIX
Antipodal	1.1e4±348	0.9e4±20	1.9e4±238	1.0e4±19
Cross	1.9e4±256	1.5e4±26	1.3e4±12	1.1e4±34
Merge	8.5e3±21	6.8e3±105	9.6e3±294	1.2e4±61
SUMO	9.6e3±278	7.0e3±1.5e3	8.7e3±1.3e3	6.3e3±21
Checkers	9.2e3±880	8.5e3±568	7.7e3±2.2e3	11e3±1.4e3

A.7 Environment details

The full Markov game for each experimental domain, along with the single-agent MDP induced from the Markov game, are defined in this section. In all domains, each agent's observation in the Markov game consists of two components, o_{self} and o_{others}. CM3 leverages this decomposition for faster training, while IAC, COMA and QMIX do not.

A.7.1 Cooperative navigation

This domain is adapted from the multi-agent particle environment in [16]. Movable agents and static landmarks are represented as circular objects located in a 2D unbounded world with real-valued position and velocity. Agents experience contact forces during collisions. A simple model of inertia and friction is involved.

State. The global state vector is the concatenation of all agents' absolute position $(x, y) \in \mathbb{R}^2$ and velocity $(v_x, v_y) \in \mathbb{R}^2$.

Observation. Each agent's observation of itself, o_{self}, is its own absolute position and velocity. Each agent's observation of others, o_{others}, is the concatenation of the relative positions and velocities of all other agents with respect to itself.

Actions. Agents take actions from the discrete set do nothing, up, down, left, right, where the movement actions produce an instantaneous velocity (with inertia effects).

Goals and initial state assignment. With probability 0.2, landmarks are given uniform random locations in the set $(-1, 1)^2$, and agents are assigned initial positions uniformly at random within the set $(-1, 1)^2$. With probability 0.8, they are predefined as follows (see Figure 3.2). In "Antipodal", landmarks for agents 1 to 4 have (x, y) coordinates [(0.9,0.9), (-0.9,-0.9), (0.9,-0.9), (-0.9,0.9)], while agents 1 to 4 are placed at [(-0.9,-0.9), (0.9,0.9), (-0.9,0.9), (0.9,-0.9)]. In "Intersection", landmark coordinates are [(0.9,-0.15), (-0.9,0.15), (0.15,0.9), (-0.15,-0.9)], while agents are placed at [(-0.9,-0.15), (0.9,0.15), (0.15,-0.9), (-0.15,0.9)]. In "Merge", landmark coordinates are [(0.9,-0.2), (0.9,0.2)], while agents are [(-0.9,0.2), (-0.9,-0.2)]. Each agent's goal is the assigned landmark position vector.

Reward. At each time step, each agent's individual reward is the negative distance between its position and the position of its assigned landmark. If a collision occurs between any pair of agents, both agents receive an additional -1 penalty. A collision occurs when two agents' distance is less than the sum of their radius.

Termination. Episode terminates when all agents are less than 0.05 distance from assigned landmarks.

Induced MDP. This is the $N = 1$ case of the Markov game, used by Stage 1 of CM3. The single agent only receives o_{self}. In each episode, its initial position and the assigned landmark's initial position are both uniform randomly chosen from $(-1, 1)^2$.

A.7.2 SUMO

We constructed a straight road of total length 200m and width 12.8m, consisting of four lanes. All lanes have width 3.2m, and vehicles can be aligned along any of four sub-lanes within a lane, with lateral spacing $0.8m$. Vehicles are emitted at average speed 30m/s with small deviation. Simulation time resolution was $0.2s$ per step. SUMO file `merge_stage3_dense.rou.xml` contains all vehicle parameters, and `merge.net.xml` defines the complete road architecture.

State. The global state vector s is the concatenation of all agents' absolute position

(x, y), normalized respectively by the total length and width of the road, and horizontal speed v normalized by 29m/s.

Observation. Each agent observation of itself o^n_{self} is a vector consisting of: agent speed normalized by 29m/s, normalized number of sub-lanes between agent's current sub-lane and center sub-lane of goal lane, and normalized longitudinal distance to goal position. Each agent's observation of others o^n_{others} is a discretized observation tensor of shape [13,9,2] centered on the agent, with two channels: binary indicator of vehicle occupancy, and normalized relative speed between agent and other vehicles. Each channel is a matrix with shape [13,9], corresponding to visibility of $15m$ forward and backward (with resolution $2.5m$) and four sub-lanes to the left and right.

Actions. All agents have the same discrete action space, consisting of five options: no-op (maintain current speed and lane), accelerate ($2.5m/s^2$), decelerate ($-2.5m/s^2$), shift one sub-lane to the left, shift one sub-lane to the right. Each agent's action a^n is represented as a one-hot vector of length 5.

Goals and initial state assignment. Each goal vector g^n is a one-hot vector of length 4, indicating the goal lane at which agent n should arrive once it crosses position x=190m. With probability 0.2, agents are assigned goals uniformly at random, and agents are assigned initial lanes uniformly at random at position x=0. With probability 0.8, agent 1's goal is lane 2 and agent 2's goal is lane 1, while agent 1 is initialized at lane 1 and agent 2 is initialized at lane 2 (see Figure 3.3). Departure times were drawn from a normal distribution with mean 0s and standard deviation 0.5s for each agent.

Reward. The reward $R(s_t, \mathbf{a}_t, g^n)$ for agent n with goal g^n is given according to the conditions: -1 for a collision; -10 for time-out (exceed 33 simulation steps during an episode); $10(1 - \Delta)$ for reaching the end of the road and having a normalized sub-lane difference of Δ from the center of the goal lane; and -0.1 if current speed exceeds 35.7m/s.

Termination. Episode terminates when 33 simulation steps have elapsed or all agents have $x >$190m.

Induced MDP. This is the $N = 1$ case of the Markov game defined above, used by Stage 1 of CM3. The single agent receives only o_{self}. For each episode, agent initial and goal lanes are assigned uniformly at random from the available lanes.

A.7.3 Checkers

This domain is adapted from the Checkers environment in [31]. It is a gridworld with 5 rows and 13 columns (Figure 3.4). Agents cannot move to the two highest and lowest rows and the two highest and lowest columns, which are placed for agents' finite observation grid to be well-defined. Agents cannot be in the same grid location. Red and yellow collectible reward are placed in a checkered pattern in the middle 3x8 region, and they disappear when any agent moves to their location.

State. The global state s consists of two components. The first is s_T, a tensor of shape [3,9,2], where the two "channels" in the last dimension represents the presence/absence of red and yellow rewards as 1-hot matrices. The second is s_V, the concatenation of all agents' (x, y) location (integer-valued) and the number of red and yellow each agent has collected so far.

Observation. Each agent's obsevation of others, o^n_{others}, is the concatenation of all other agents' normalized coordinates (normalized by total size of grid). An agent's observation of itself, o^n_{self}, consists of two components. First, $o^n_{\text{self},V}$ is a vector concatenation of agent n's normalized coordinate and the number of red and yellow it has collected so far. Second, $o^n_{\text{self},T}$ is a tensor of shape [5,5,3], centered on its current location in the grid. The tensor has three "channels", where the first two represent presence/absence of red and yellow rewards as 1-hot matrices, and the last channel indicates the invalid locations as a 1-hot matrix. The agent's own grid location is a valid location, while other agents' locations are invalid.

Actions. Agents choose from a discrete set of actions do-nothing, up, down, left, right. Movement actions transport the agent one grid cell in the chosen direction.

Goals. Agent A's goal is to collect all red rewards without touching yellow. Agent B's

goal is to collect all yellow without touching red. The goal is represented as a 1-hot vector of length 2.

Reward. Agent A gets +1 for red, -0.5 for yellow. Agent B gets -0.5 for red, +1 for yellow.

Initial state distribution. Agent A is initialized at (2,8), Agent B is initialized at (4,8). (0,0) is the top-left cell (Figure 3.4).

Termination. Each episode finishes when either 75 time steps have elapsed, or when all rewards have been collected.

Induced MDP. For Stage 1 of CM3, the single agent is randomly assigned the role of either Agent A or Agent B in each episode. Everything else is defined as above.

A.8 Architecture

For all experiment domains, ReLU nonlinearity was used for all neural network layers unless otherwise specified. All layers are fully-connected feedforward layers, unless otherwise specified. All experiment domains have a discrete action space (with $|\mathcal{A}| = 5$ actions), and action probabilities were computed by lower-bounding softmax outputs of all policy networks by $P(a^n = i) = (1 - \epsilon)\text{softmax}(i) + \epsilon/|\mathcal{A}|$, where ϵ is a decaying exploration parameter. To keep neural network architectures as similar as possible among all algorithms, our neural networks for COMA differ from those of [17] in that we do not use recurrent networks, and we do not feed previous actions into the Q function. For the Q network in all implementations of COMA, the value of each output node i is interpreted as the action-value $Q(s, a^{-n}, a^n = i, \mathbf{g})$ for agent n taking action i and all other agents taking action a^{-n}. Also for COMA, agent n's label vector (one-hot indicator vector) and observation o_{self} were used as input to COMA's global Q function, to differentiate between evaluations of the Q-function for different agents. These were choices in [17] that we retain.

A.8.1 Cooperative navigation

CM3. The policy network π^1 in Stage 1 feeds the concatenation of o_{self} and goal g to one layer with 64 units, which is connected to the special layer h^1_* with 64 units, then connected to the softmax output layer with 5 units, each corresponding to one discrete action. In Stage 2, o_{others} is connected to a new layer with 128 units, then connected to h^1_*.

The Q^1 function in Stage 1 feeds the concatenation of state s, goal g, and 1-hot action a to one layer with 64 units, which is connected to the special layer h^1_* with 64 units, then to a single linear output unit. In Stage 2, Q^1 is augmented into both $Q^{\boldsymbol{\pi}}_n(s, \mathbf{a})$ and $Q^{\boldsymbol{\pi}}_n(s, a^m)$ as separate networks. For $Q^{\boldsymbol{\pi}}_n(s, \mathbf{a})$, s^{-n} (part of state s excluding agent n) and a^{-n} are concatenated and connected to a layer with 128 units, then connected to h^1_*. For $Q^{\boldsymbol{\pi}}_n(s, a^m)$, s^m (agent m portion of state s) and s^{-n} are concatenated and connected to a layer with 128 units, then connected to h^1_*.

IAC. IAC uses the same policy network as Stage 2 of CM3. The value function of IAC concatenates o^n_{self} and goal g^n, connects to a layer with 64 units, which connects to a second layer h_2 with 64 units, then to a single linear output unit. o^n_{others} is connected to a layer with 128 units, then connected to h_2.

COMA. COMA uses the same policy network as Stage 2 of CM3. The global Q function of COMA computes $Q(s, (a^n, a^{-n}))$ for each agent n as follows. Input is the concatenation of state s, all other agents' 1-hot actions a^{-n}, agent n's goal g^n, all other agent goals g^{-n}, agent label n, and agent n's observation o^n_{self}. This is passed through two layers of 128 units each, then connected to a linear output layer with 5 units.

QMIX. Individual value functions take input $(o^n_{\text{self}}, o^n_{\text{others}}, g^n)$ and connects to one hidden layer with 64 units, which connects to the output layer. The mixing network follows the exact architecture of [18] with embedding dimension 64.

A.8.2 SUMO

CM3. The policy network π^1 during Stage 1 feeds each of the inputs o_{self} and goal g^n to a layer with 32 units. The concatenation is then connected to the layer h_*^1 with 64 units, and connected to a softmax output layer with 5 units, each corresponding to one discrete action. In Stage 2, the input observation grid o^n_{others} is processed by a convolutional layer with 4 filters of size 5x3 and stride 1x1, flattened and connected to a layer with 64 units, then connected to the layer h_*^1 of π^1.

The Q^1 function in Stage 1 feeds the concatenation of state s, goal g, and 1-hot action a to one layer with 256 units, which is connected to the special layer h_*^1 with 256 units, then to a single linear output unit. In Stage 2, Q^1 is augmented into both $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a})$ and $Q_n^{\boldsymbol{\pi}}(s, a^m)$ as separate networks. For $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a})$, s^{-n} (part of state s excluding agent n), a^{-n}, and g^{-n} are concatenated and connected to a layer with 128 units, then connected to h_*^1. For $Q_n^{\boldsymbol{\pi}}(s, a^m)$, s^m (agent m portion of state s), s^{-n}, and g^{-n} are concatenated and connected to a layer with 128 units, then connected to h_*^1.

IAC. IAC uses the same policy network as Stage 2 of CM3. The value function of IAC concatenates o^n_{self} and g^n, feeds it into a layer with 64 units, which connects to a layer h_2 with 64 units, which connects to one linear output unit. o^n_{others} is processed by a convolutional layer with 4 filters of size 5x3 and stride 1x1, flattened and connected to a layer with 128 units, then connected to h_2.

COMA. COMA uses the same policy network as Stage 2 of CM3. The Q function of COMA is exactly the same as the one in COMA for cooperative navigation defined above.

QMIX. Individual value functions take input (o^n_{self}, g^n) and connects to one hidden layer with 64 units, which connects to layer h_2 with 64 units. o^n_{others} is passed through the same convolutional layer as above and connected to h_2. h_2 is fully-connected to an output layer. The mixing network follows the exact architecture of [18] with embedding dimension 64.

A.8.3 Checkers

CM3. The policy network π^1 during Stage 1 feeds $o^n_{\text{self},T}$ to a convolution layer with 6 filters of size 3x3 and stride 1x1, which is flattened and connected to a layer with 32 units, which is concatenated with $o^n_{\text{self},V}$, previous action, and its goal vector. The concatenation is connected to a layer with 256 units, then to the special layer h^1_* with 256 units, finally to a softmax output layer with 5 units. In Stage 2, o^n_{others} is connected to a layer with 256 units, then to the layer h^1_* of π^1.

The Q^1 function in Stage 1 is defined as: state tensor s_T is fed to a convolutional layer with 4 filters of size 3x5 and stride 1x1 and flattened. $o^n_{\text{self},T}$ is given to a convolution layer with 6 filters of size 3x3 and stride 1x1 and flattened. Both are concatenated with s^n (agent n part of the s_V vector), goal g^n, action a^n and $o^n_{\text{self},V}$. The concatenation is fed to a layer with 256 units, then to the special layer h^1_* with 256 units, then to a single linear output unit. In Stage 2, Q^1 is augmented into both $Q^{\boldsymbol{\pi}}_n(s, \mathbf{a})$ and $Q^{\boldsymbol{\pi}}_n(s, a^m)$ as separate networks. For $Q^{\boldsymbol{\pi}}_n(s, \mathbf{a})$, s^{-n} (part of state vector s_V excluding agent n) and a^{-n} are concatenated and connected to a layer with 32 units, then connected to h^1_*. For $Q^{\boldsymbol{\pi}}_n(s, a^m)$, s^m (agent m portion of state s_V) and s^{-n} are concatenated and connected to a layer with 32 units, then connected to h^1_*.

IAC. IAC uses the same policy network as Stage 2 of CM3. The value function of IAC feeds $o^n_{\text{self},T}$ to a convolutional layer with 6 filters of size 3x3 and stride 1x1, which is flattened and concatenated with $o^n_{\text{self},V}$ and goal g^n. The concatenation is connected to a layer with 256 units, then to a layer h_2 with 256 units, then to a single linear output unit. o^n_{others} is connected to a layer with 32 units, then to the layer h_2.

COMA. COMA uses the same policy network as Stage 2 of CM3. The global $Q(s, (a^n, a^{-n}))$ function of COMA is defined as follows for each agent n. Tensor part of global state s_T is given to a convolutional layer with 4 filters of size 3x5 and stride 1x1. Tensor part of agent n's observation $o^n_{\text{self},T}$ is given to a convolutional layer with 6 filters of size 3x3 and stride 1x1. Outputs of both convolutional layers are flattened, then concatenated

with s_V, all other agents' actions a^{-n}, agent n's goal g^n, other agents' goals g^{-n}, agent n's label vector, and agent n's vector observation $o^n_{\text{self},V}$. The concatenation is passed through two layers with 256 units each, then to a linear output layer with 5 units.

QMIX. Individual value functions are defined as: $o^n_{\text{self},T}$ is passed through the same convolutional layer as above, connected to hidden layer with 32 units, then concatenated with $o^n_{\text{self},V}$, a^n_{t-1}, and g^n. This is connected to layer h_2 with 64 units. o^n_{others} is connected to a layer with 64 units then connectd to h_2. h_2 is fully-connected to an output layer. The mixing network feeds s_T into the same convolutional network as above and follows the exact architecture of [18] with embedding dimension 128.

A.9 Parameters

We used the Adam optimizer in Tensorflow with hyperparameters in Tables A.3 to A.5. ϵ_{div} is used to compute the exploration decrement $\epsilon_{\text{step}} := (\epsilon_{\text{start}} - \epsilon_{\text{end}})/\epsilon_{\text{div}}$.

Table A.3: Parameters used for CM3, ablations, and baselines in cooperative navigation

	CM3						
Parameter	Stage 1	Stage 2	QV	Direct	IAC	COMA	QMIX
Episodes	1e3	8e4	8e4	8e4	8e4	8e4	8e4
ϵ_{start}	1.0	0.5	0.5	1.0	1.0	1.0	1.0
ϵ_{end}	0.01	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{div}	1e3	2e4	2e4	8e4	8e4	2e4	8e4
Replay buffer	1e4	1e4	1e4	1e4	1e4	1e4	1e4
Minibatch size	256	128	128	128	128	128	128
Episodes per train	10	10	10	10	10	10	N/A
Learning rate π	1e-4	1e-4	1e-4	1e-4	1e-4	1e-5	N/A
Learning rate Q	1e-3	1e-3	1e-3	1e-3	N/A	1e-4	1e-3
Learning rate V	N/A	N/A	1e-3	N/A	1e-3	N/A	N/A
Epochs	24	24	24	24	24	24	NA
Steps per train	N/A	N/A	N/A	N/A	N/A	N/A	10
Max env steps	25	50	50	50	50	50	50

Table A.4: Parameters used for CM3 and baselines in SUMO

	CM3						
Parameter	Stage 1	Stage 2	QV	Direct	IAC	COMA	QMIX
Episodes	2.5e3	5e4	5e4	5e4	5e4	5e4	5e4
ϵ_{start}	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
ϵ_{step}	2e3	1e3	4e4	4e4	1e3	1e4	4e4
Replay buffer	1e4	2e4	2e4	2e4	2e4	2e4	2e4
Minibatch size	128	128	128	128	128	128	128
Steps per train	10	10	10	10	N/A	N/A	10
Episodes per train	N/A	N/A	N/A	N/A	10	10	N/A
Learning rate π	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	N/A
Learning rate Q	1e-3	1e-3	1e-3	1e-3	N/A	1e-3	1e-3
Learning rate V	N/A	N/A	1e-3	N/A	1e-3	N/A	N/A
Epochs	N/A	N/A	N/A	N/A	33	33	N/A
Max env steps	33	33	33	33	33	33	33

A.10 Stage 1

The Stage 1 functions Q^1 and π^1 for a single agent are trained with the $N = 1$ equivalents of (Equation 3.4) and (Equation 3.5):

$$L(\theta_Q) = \mathbb{E}_{\boldsymbol{\pi}}\Big[\big(y_i - Q^1_{\theta_Q}(s_i, a_i)\big)^2\Big] \tag{A.8}$$

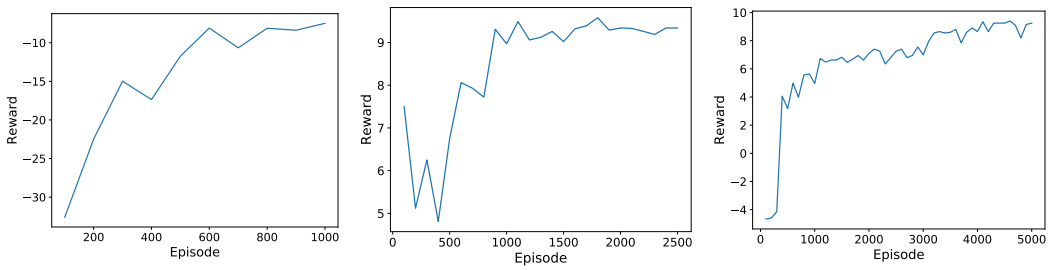
$$y_i := R(s_i, \mathbf{a}_i, g^n) + \gamma Q^1_{\theta_Q}(s_{i+1}, a_{i+1}) \tag{A.9}$$

$$\nabla_\theta J(\pi^1) = \mathbb{E}_{\pi^1}\Big[\nabla_\theta \log \pi(a)\big(Q^{\pi^1}(s, a) - \sum_{\hat{a}} \pi^1(\hat{a}) Q^{\pi^1}(s, \hat{a})\big)\Big] \tag{A.10}$$

Stage 1 training curves for all three experimental domains are shown in Figure A.1.

Table A.5: Parameters used for CM3 and baselines in Checkers

	CM3						
Parameter	Stage 1	Stage 2	QV	Direct	IAC	COMA	QMIX
Episodes	5e3	5e4	5e4	5e4	5e4	5e4	5e4
ϵ_{start}	1.0	0.5	0.5	1.0	1.0	1.0	1.0
ϵ_{end}	0.1	0.1	0.1	0.1	0.1	0.1	0.1
ϵ_{step}	5e2	1e3	1e3	1e4	2e4	1e4	1e4
Replay buffer	1e4	1e4	1e4	1e4	1e4	1e4	1e4
Minibatch size	128	128	128	128	128	128	128
Steps per train	N/A	10	10	10	N/A	N/A	10
Episodes per train	10	N/A	N/A	N/A	10	10	N/A
Learning rate π	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	N/A
Learning rate Q	1e-3	1e-3	1e-3	1e-3	N/A	1e-3	1e-5
Learning rate V	N/A	N/A	1e-3	N/A	1e-3	N/A	N/A
Epochs	10	N/A	N/A	N/A	33	33	N/A
Max env steps	75	75	75	75	75	75	75



(a) Cooperative navigation (b) SUMO (c) Checkers

Figure A.1: Stage 1 reward curves for CM3 in cooperative navigation, SUMO and Checkers.

APPENDIX B

LEARNING TO INCENTIVIZE OTHER LEARNING AGENTS

B.1 Further discussion

B.1.1 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 (Equation 5.5) and (Equation 5.6)), instead of being accounted in the total reward (Equation 5.1) 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 (Equation 5.6). 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 4.3.2), then the cost for incentivization should indeed be accounted by the policy. One may consider other mechanisms for cost, such as budget constraints [95].

In our experiments, we find the coefficient α in the cost for incentivization is a sensitive parameter. At the beginning of training, (Equation 5.6) 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.2 Analysis in Iterated Prisoner's Dilemma

Proposition 3. *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 5.2. 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

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

The total reward received by each agent is

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

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

Agent 2 updates its policy via the update

$$\begin{aligned}\hat{\theta}^2 &= \theta^2 + \alpha \nabla_{\theta^2} V^2(\theta^1, \theta^2) \\ &= \theta^2 + \frac{\alpha}{1-\gamma} \nabla_{\theta^2} \left(\theta^1 \theta^2 (-1 + \eta_C^1) + \theta^1 (1 - \theta^2) \eta_D^1 \right. \\ &\quad \left. + (1 - \theta^1) \theta^2 (-3 + \eta_C^1) + (1 - \theta^1)(1 - \theta^2)(-2 + \eta_D^1) \right) \\ &= \theta^2 + \frac{\alpha}{1-\gamma} \left(\eta_C^1 - \eta_D^1 - 1 \right) ,\end{aligned} \tag{B.4}$$

and likewise for Agent 1:

$$\hat{\theta}^1 = \theta^1 + \frac{\alpha}{1-\gamma} \left(\eta_C^2 - \eta_D^2 - 1 \right) . \tag{B.5}$$

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

$$\begin{aligned}\eta^1 &\leftarrow \eta^1 + \beta \nabla_{\eta^1} \frac{1}{1-\gamma} \hat{p}^T r^1 \\ &= \eta^1 + \frac{\beta}{1-\gamma} \nabla_{\eta^1} \left[\hat{\theta}^1 (\theta^2 + \Delta^2)(-1 + \eta_C^2) + \hat{\theta}^1 (1 - \theta^2 - \Delta^2)(-3 + \eta_C^2) \right. \\ &\quad \left. + (1 - \hat{\theta}^1)(\theta^2 + \Delta^2) \eta_D^2 + (1 - \hat{\theta}^1)(1 - \theta^2 - \Delta^2)(-2 + \eta_D^2) \right] \\ &= \eta^1 + \frac{\beta \alpha}{(1-\gamma)^2} B_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} ,\end{aligned} \tag{B.6}$$

where the scalar B_2 is

$$B_2 = \hat{\theta}^1 (-1 + \eta_C^2) - \hat{\theta}^1 (-3 + \eta_C^2) + (1 - \hat{\theta}^1) \eta_D^2 - (1 - \hat{\theta}^1)(-2 + \eta_D^2) = 2 . \tag{B.7}$$

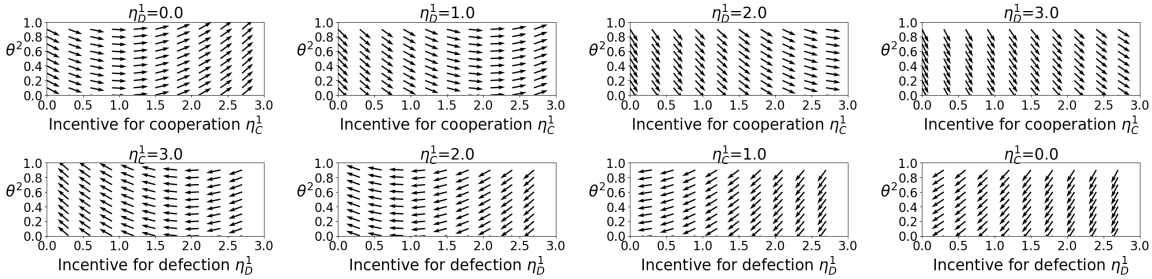


Figure B.1: Vector fields showing the probability of recipient cooperation versus incentive value given for cooperation (top row) and defection (lower row). Each plot has a fixed value for the incentive given for the other action.

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

$$\eta^2 \leftarrow \eta^2 + \frac{\beta\alpha}{(1-\gamma)^2} B_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} . \tag{B.8}$$

Note that each η^i is updated so that η_C^i increases while η_D^i decreases. Referring to (Equation B.4) and (Equation B.5), 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 B.1. 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 (Equation B.6) and (Equation B.8). □

Table B.1: 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

B.3 Derivations

The factor $\nabla_{\hat{\theta}^j} J^i(\hat{\tau}^i, \hat{\boldsymbol{\theta}})$ (Equation 5.9) in the incentive function's gradient (Equation 5.7) 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 [56]:

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

Alternatively, one may rely on automatic differentiation in modern machine learning frameworks [154] to compute the chain rule (Equation 5.7) via direct minimization of the loss (Equation 5.10). 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 (Equation 5.10).

B.4 Experiments

B.4.1 Environment details

This section provides more details on each experimental setup.

IPD. We used the same definition of observation, action, and rewards as [96]. 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 [96].

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 [178]. The environment settings for 7x7 and 10x10 maps are given in Table B.2. 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 B.2: 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

B.4.2 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 [96] 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 [185] 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 B.3: 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 (Equation 5.5), 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 B.4: 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

$$\hat{\theta}^j \leftarrow \theta^j + \mathbb{E}_{\boldsymbol{\pi}}\left[\nabla_{\theta^j} \log \pi_{\theta^j}(a^j|o^j)\left(r^j + \gamma V_{\phi^j}(s') - V_{\tilde{\phi}^j}(s)\right)\right], \tag{B.9}$$

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

$$L(\phi^j) = \mathbb{E}_{s,s'\sim\boldsymbol{\pi}}\left[\left(r^j + \gamma V_{\tilde{\phi}^j}(s') - V_{\phi^j}(s)\right)^2\right] \tag{B.10}$$

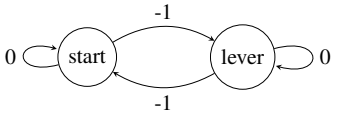
The target network [2] parameters $\tilde{\phi}^j$ are updated via $\tilde{\phi}^j \leftarrow \tau\phi^j + (1-\tau)\tilde{\phi}^j$ with $\tau = 0.01$.

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 [91] have an additional 1D vector observation of all agents' temporally 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.

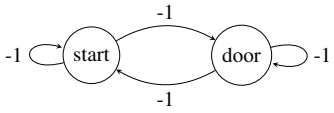
LIO and AC-c have $R_{\text{max}} = 2.0$. AC-d used $R_a = 2.0$. Inequity aversion agents have disadvantageous aversion coefficient value 0, advantageous aversion coefficient value 0.05, and temporal smoothing parameter $\lambda = 0.95$. We use critic learning rate $\alpha_\phi = 10^{-3}$ for all methods. 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 B.5: Hyperparameters in Cleanup.

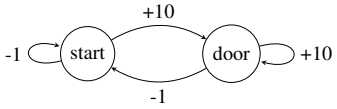
	7x7					10x10				
Parameter	LIO	AC	AC-d	AC-c	IA	LIO	AC	AC-d	AC-c	IA
ϵ_{start}	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
ϵ_{end}	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{div}	100	100	100	100	1000	1000	5000	1000	1000	5000
α_θ	1e-4	1e-3	1e-4	1e-4	1e-3	1e-4	1e-3	1e-3	1e-3	1e-3



(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.



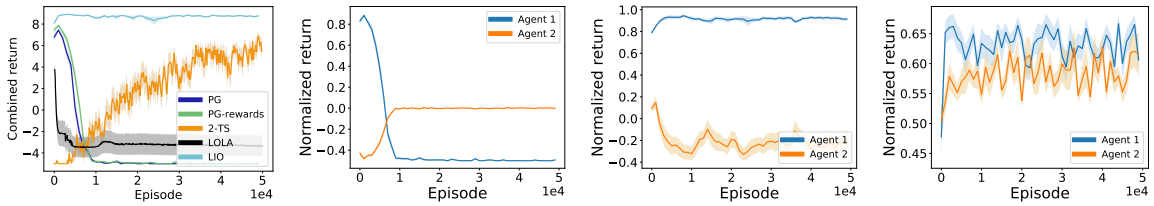
(c) A1 get +10 at the door if A2 pulls the lever.

Figure B.2: 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.

B.5 Additional results

B.5.1 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 B.2. A1 gets +10 extrinsic reward for exiting a door and ending the game (Figure B.2c), but the door can only be opened when A2 pulls a lever; otherwise, A1 is penalized at every time step (Figure B.2b). The extrinsic penalty for A2 discourages it from taking the cooperative action (Figure B.2a). 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



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

Figure B.3: 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).

cooperative actions. We use a standard policy gradient for A2 unless otherwise specified.

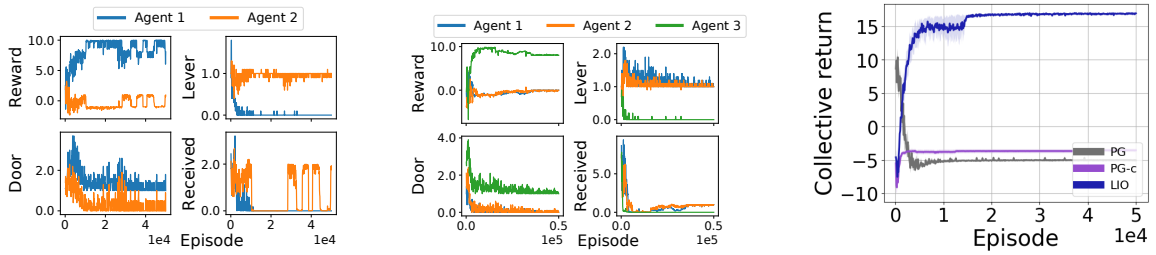
In addition to the baselines described for the symmetric case—namely, policy gradient (PG-rewards) and LOLA with discrete "give-reward" actions—we also compare with a two-timescale method, labeled **2-TS**. A 2-TS agent has the same augmented action space as the PG-rewards baseline, except that it learns 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.

Figure B.3 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 B.3a), 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 B.3b 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 B.3c 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 B.3d 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.

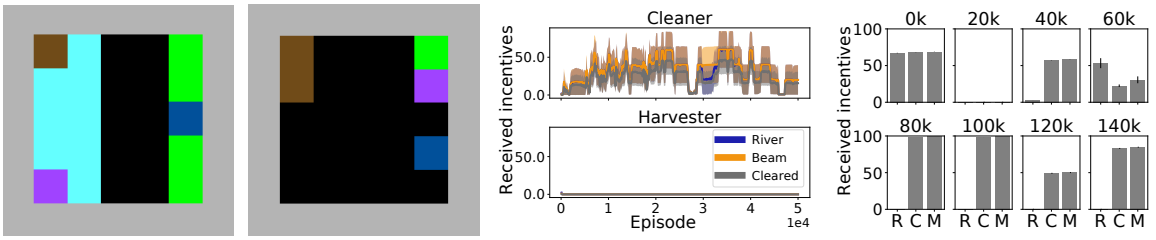
B.5.2 Symmetric Escape Room

Figure B.4 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 B.4a, 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 B.4b, A3 converges to going to the door, as it incentives A1 and A2 to pull the lever.



(a) ER(2,1) (b) ER(3,2) (c) ER(5,3)

Figure B.4: (a,b) Individual actions and incentives in ER(2,1) and ER(3,2). (c) LIO converges to the global optimum in ER(5,3).



(a) Division of labor (b) AC agents compete (c) Cleaner's incentives (d) 10x10 map

Figure B.5: (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.

B.5.3 Cleanup

Figure B.5a 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 B.5b shows a time step where two AC agents compete for apples, which is jointly suboptimal. Figure B.5c 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 B.5d shows the incentives given by a "Harvester" agent to three scripted agents during each training checkpoint.

Agents with hand-designed intrinsic rewards based on social influence [93] also outperform standard RL agents on Cleanup. We can make an indirect comparison to [93] by noting that IA reaches a score around 250 by 1.6×10^8 steps [91, Figure 3a], which outperforms

the score of 200 attained by Social Influence at 3×10^8 steps [93, Figure 1a] in the original Cleanup map with 5 agents. Hence, the fact that LIO outperforms IA in our experiments suggests that LIO compares favorably with Social Influence, provided that LIO uses the same RL algorithm as the latter for policy optimization.

APPENDIX C

ADAPTIVE INCENTIVE DESIGN WITH MULTI-AGENT META-GRADIENT REINFORCEMENT LEARNING

C.1 Additional discussion on methods

C.1.1 Meta-gradient with TRPO/PPO objectives

We recapitulate the derivation of Trust Region Policy Optimization (TRPO) [44] to show that it is compatible with meta-gradient RL. From this, it also follows that PPO [45] is applicable to the upper level incentive designer problem by substituting the gradient with respect to incentive function parameters in the place of the gradient with respect to policy parameters.

TRPO considers the performance of a policy $\hat{\pi}$, parameterized by $\hat{\theta}$, versus another policy π parameterized by θ. Here, the "hat" notation $\hat{(\cdot)}$ has no relation to the "updated policy" in meta-gradient RL. The TRPO objective is

$$J(\pi) := E_{\pi}\left[\sum_{t} \gamma^t R(s_t, a_t)\right] \tag{C.1}$$

It was shown that [44, Equation 2]

$$J(\hat{\pi}) = J(\pi) + \mathbb{E}_{\hat{\pi}}\left[\sum_{t} \gamma^t A_{\pi}(s_t, a_t)\right] \tag{C.2}$$

$$= J(\pi) + \sum_{s} \rho_{\hat{\pi}}(s) \sum_{a} \hat{\pi}(a|s) A_{\pi}(s, a)\,, \tag{C.3}$$

where ρ_{π} is the discounted state visitation frequencies and A_{π} is the advantage function under policy π. TRPO makes a local approximation, whereby $\rho_{\hat{\pi}}$ is replaced by ρ_{π}. One can

define

$$L_\pi(\hat{\pi}) := J(\pi) + \sum_s \rho_\pi(s) \sum_a \hat{\pi}(a|s) A_\pi(s,a)\,, \tag{C.4}$$

and derive the lower bound $J(\hat{\pi}) \geq L_\pi(\hat{\pi}) - C D_{\text{KL}}^{\max}(\pi, \hat{\pi})$, where $D_{\text{KL}}^{\max}$ is the KL divergence maximized over states and C depends on π. The KL divergence penalty can be replaced by a constraint, so the problem becomes

$$\max_{\hat{\theta}} \sum_s \rho_\theta(s) \sum_a \hat{\pi}_{\hat{\theta}}(a|s) A_\theta(s,a) \tag{C.5}$$

$$\text{s.t. } \bar{D}_{\text{KL}}^{\theta}(\theta, \hat{\theta}) \leq \delta\,, \tag{C.6}$$

where $\bar{D}_{\text{KL}}^{\theta}$ is the KL divergence averaged over states $s \sim \rho_\theta$. Using importance sampling, the summation over actions $\sum_a(\cdot)$ is replaced by $\mathbb{E}_{a\sim q}\left[\frac{1}{q(a|s)}(\cdot)\right]$. It is convenient to choose $q = \pi_\theta$, which results in:

$$\max_{\hat{\theta}} \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}\left[\frac{\hat{\pi}_{\hat{\theta}}(a|s)}{\pi_\theta(a|s)} A_\theta(s,a)\right] \tag{C.7}$$

$$\text{s.t. } \mathbb{E}_{s\sim\rho_\theta}\left[D_{\text{KL}}(\pi_\theta(\cdot|s), \hat{\pi}_{\hat{\theta}}(\cdot|s))\right] \leq \delta\,. \tag{C.8}$$

During online learning, the $\hat{\theta}$ that is optimized and the old θ are the same at each iteration, so the gradient estimate is

$$\mathbb{E}_{\pi_\theta}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} A_\theta(s,a)\right]\,. \tag{C.9}$$

Now, making the connection to meta-gradient RL for the incentive design problem, we note the formal equivalency between the TRPO objective $J(\pi)$ (Equation C.1) and the ID's objective $J^{\text{ID}}(\eta; \hat{\boldsymbol{\theta}})$ (Equation 6.1), and between the TRPO agent policy π_θ and the

multi-agent joint policy $\boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}(\eta)}$. [1] Hence, we arrive at the meta-gradient based on TRPO

$$\mathbb{E}_{\boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}}}\left[\frac{\nabla_{\eta}\boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}(\eta)}(\mathbf{a}|s)}{\boldsymbol{\pi}_{\hat{\boldsymbol{\theta}}}(\mathbf{a}|s)}A_{\hat{\boldsymbol{\theta}}}(s,\mathbf{a})\right], \tag{C.10}$$

where only the updated policy $\hat{\theta}$ appears explicitly. This also justifies the use of the PPO loss function as the outer loss for meta-gradient RL.

C.1.2 Q-learning agents

If agents conduct Q-learning with a set of individual Q-functions $\{Q_{\theta^i}(o^i, a^u)\}_{i=1}^n$, we may assume that the induced policy of each agent i is:

$$\pi_{\theta^i}(a^i|o^i) := \frac{\exp(\tau Q_{\theta^i}(o^i,a^i))}{\sum_{a\in\mathcal{A}}\exp(\tau Q_{\theta^i}(o^i,a))} \tag{C.11}$$

where τ is some constant. Agents conduct their standard Q-learning updates as

$$\hat{\theta}^i \leftarrow \theta^i + \alpha f(\theta^i,\eta,\boldsymbol{\tau}) \tag{C.12}$$

$$f(\theta^i,\eta,\boldsymbol{\tau}) := \mathbb{E}_{\boldsymbol{\tau}\sim\boldsymbol{\pi}}\left[\nabla_{\theta^i}\left(R^{i,\text{tot}}(s,\mathbf{a},\eta)\right.\right. \tag{C.13}$$

$$\left.\left. + \gamma\max_a Q'(o^i_{t+1},a) - Q_{\theta^i}(o^i_t,a^i_t)\right)^2\right]. \tag{C.14}$$

We can then use (Equation 6.5) with (Equation C.11) as the agents' policies, and $\nabla_\eta\hat{\theta}^i$ can be computed by differentiating through (Equation C.12).

C.1.3 Relation to hypergradients and implicit differentiation

Hyperparameter optimization seeks the best hyperparameters λ^* such that the validation loss $\mathcal{L}_V$ is minimized by the model whose weights w^* are obtained by minimizing the training

[1] Here, when used with argument η, the "hat" notation is interpreted in the metagradient context, where it denotes the updated policy after learning from incentives.

loss $\mathcal{L}_T$ with λ^*. It can be formulated as a bi-level optimization problem [118]:

$$\lambda^* = \operatorname*{argmin}_{\lambda} \mathcal{L}_V(\lambda, w^*(\lambda)) \tag{C.15a}$$

$$\text{s.t. } w^*(\lambda) = \operatorname*{argmin}_{w} \mathcal{L}_T(\lambda, w) \tag{C.15b}$$

This formulation requires knowing the best-response $w^*(\lambda)$ and the Jacobian $\frac{\partial w^*(\lambda)}{\partial \lambda}$. However, it may not be necessary, or even the best method for fastest training, to know the best-response. A λ corresponds to a certain optimization landscape, and the best response $w^*(\lambda)$ is the minimizer of the training loss of that landscape, with an associated validation loss. It is not clear that the best $w^*(\lambda_k)$ for each intermediate λ_k at training iteration k produces the best direction for the update $\lambda_{k+1} \leftarrow \lambda_k + \Delta\lambda$. We only want the final set of optimal weights, and there is no obvious reason to care about the trajectory of best response weights $w^*(\lambda_1), \ldots, w^*(\lambda_k), \ldots$, especially when this trajectory may not lead to convergence to the minimizer of the validation loss. In our algorithm, we unroll and differentiate through the 1-step inner optimization, without requiring convergence of the inner optimization to a best response.

C.2 Experimental setup

C.2.1 Environment details

Escape Room

Each agent observes a 1-hot encoding of its own position in the three possible states (lever, start, and door), as well as 1-hot encodings of the positions of all other agents. The incentive designer observes a concatenation of 1-hot encodings of all agents' positions. Each agent has has three actions that move it to the three available states: lever, start, and door. At each time step, the designer uses all agents' chosen actions along with the designer's global observation to compute the incentives. Agents receive the sum of predefined environment

rewards and incentives. 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 at least M agents pull the lever. Each episode terminates when any agent successfully exits the door, or when 5 time steps elapse.

Cleanup

We used the open-source implementation based on [178, 40], which contains the following modifications that increase the difficulty of the social dilemma compared to the original version [91]: rotation actions and the tagging beam are disabled, and all agents have a fixed "upward" orientation. Hence, an agent must move to the river to clear waste successfully—it cannot simultaneously stay in the region where apples spawn and fire its cleaning beam toward the river—which incurs the risk that other agents exploit the opportunity to collect apples. 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 incentive designer's input is a global observation of the entire RGB map and a multi-hot vector that indicates which agent(s) used their cleaning beam.

Gather-Trade-Build

Each agent's observation is a pair of tensor with shape 11x11x9 and a vector of size 136, consisting of a limited egocentric spatial window, its own inventories and skills, bids and asks in the market, the tax rates of the current period as well as the marginal rate corresponding to its current income so far. This is in accord with the reverse Stackelberg game formulation, whereby agents observe the designer's chosen function. The ID's observation is a pair of tensor with shape 14x14x8 and a vector of size 112, consisting of complete spatial world state, agents' inventories, all bids and asks, and all derived tax quantities, but does not contain agents' private skill and utility functions.

Each agent's instantaneous reward at time step t is defined as

$$r_t^i := u^i(x_t^i, l_t^i) - u^i(x_{t-1}^i, l_{t-1}^i)\,, \tag{C.16}$$

where x_t^i is the total resources (stone and wood) and coin owned by agent i at time t, and l_t^i is the cumulative labor due to actions by agent i up to time t. The utility function is defined as:

$$u^i(x_t^i, l_t^i) := \text{crra}(x_t^{i,c}) - l_t^i \qquad \text{crra}(z) := \frac{z^{1-\eta} - 1}{1-\eta}, \quad \eta > 0 \tag{C.17}$$

where $x_t^{i,c}$ is the amount of coin owned by agent i.

[42] used a factored discrete action space whereby the tax planner's action is $\tau \in [0, 1]^B$ where $B = 7$ is the number of tax brackets. They designed the discrete action subspace for each bracket to be $\{0, 0.05, \ldots, 1.0\}$. For our meta-gradient approach, we let r_η have 7 real-valued output nodes bounded in $[0, 1]$ and interpret them as the tax rate for each bracket.

We applied an annealing schedule that caps the maximum marginal tax rate chosen by dual-RL, as done in previous work [42]. The cap increases linearly from 0.1 to 1.0 (i.e., no cap) within the first $8k$ episodes.

C.2.2 Implementation

In *Escape Room*, the agent's policy network has two fully-connected (FC) layers of size h_1 and h_2 each, followed by a `softmax` output layer with size equal to the action space size. The MetaGrad incentive designer concatenates its observation with the actions taken by all agents, passes them through two FC layers of sizes d_1 and d_2, then to a `sigmoid` output layer with size equal to the number of possible agent actions. The dual-RL (c) incentive designer has the same architecture as in MetaGrad, except that the output layer is linear and is interpreted as the mean of a Gaussian distribution. The dual-RL (d) incentive designer has the same architecture as the agents.

In *Cleanup*, the agents have the following neural network layers, all with `ReLU` activtion. The input image is passed through one convolutional layer with 6 filters of kernel size $[3, 3]$ and stride $[1, 1]$. This is flattened and passed through two FC layers with sizes h_1 and h_2. For the agents' policy network, there is a final `softmax` output layer that maps to action probabilities. The agents' value function has the same architecture as the policy, except for a linear output layer. The incentive designer has the same convolutional layer as agents' policies. The image part of the designer's observation is passed through that layer, then through an FC layer with size h_1, then concatenated with the vector part of the designer's observation (the multi-hot vector indicating usage of cleaning beam), then passed through the second FC layer with size h_2, then finally to a `sigmoid` output layer with size 3 (one per action type in the set $\{$clean, collect apple, else$\}$. The dual-RL (c) designer has the same architecture as the MetaGrad designer, except for a linear output layer than is interpreted as the mean of a diagonal Gaussian distribution for its continuous action space.

In *Gather-Trade-Build* without curriculum, the agents' policy network consists of the following neural network layers (all with `ReLU` activation) that process the agent's observation (which has a 3D image part and a 1D vector part). The image part of the agent's observation is passed through two convolutional layers with 6 filters of kernel size $[5, 5]$ each and stride $[1, 1]$; this is flattened and passed through a fully-connected (FC) layer whose output size equals the size of the 1D vector observation; this is concatenated with the 1D vector observation, then passed through two FC layers of with 128 nodes each, then finally to a `softmax` output layer size output size equal to the discrete action space size. The incentive designer's policy in dual-RL is exactly the same as the agents' policy architecture, except that the two FC layers have 256 nodes each. The incentive designer in MetaGrad is the same as the dual-RL designer policy, except that the output layer has `sigmoid` activation (rather than `softmax`) and has size 7 (for the 7 tax brackets). The same architecture is used for the agents' value function (which does not share parameters with the policy network), except that we used an LSTM layer of size 128 after the two FC

layers.

In *Gather-Trade-Build* with curriculum, the agent and designer policy and value networks are the same as the case without curriculum, except for these differences: there is an LSTM layer of size 128 after the two FC layers for the incentive designer's policy (i.e., incentive function in the case of MetaGrad, regular policy in the case of dual-RL) and value networks, and for the agents' value network. We omitted the LSTM only for the agents' policy network to avoid complications with second-order gradients in MetaGrad.

The incentive designer in MetaGrad differentiates its objective with respect to the tax function parameters, through the agents' learning, by replicating the chain of calculations from tax $T(z)$ (Equation 6.7) to the agents' final instantaneous reward r_t^i (Equation C.16) within the overall computational graph. All quantities required to compute tax and total reward are saved in the designer's episode buffer to use in the meta-gradient step.

C.2.3 Hyperparameters

We used random uniform sampling with successive elimination for hyperparameter search for all methods. We start with a batch of n_{batch} tuples, where each tuple is a combination of hyperparameter values with each value sampled either log-uniformly from a continuous range or uniformly from a discrete set. We train independently with each tuple for n_{episode} episodes, eliminate the lower half of the batch based on their final performance, then initialize the next set of n_{episode} episodes with the current models for the remaining tuples. We use the hyperparameters of the last surviving model. For *Escape Room*, we used $n_{\text{batch}} = 128$ and $n_{\text{epsiodes}} = 8000$. For *Cleanup*, we used $n_{\text{batch}} = 128$ and $n_{\text{epsiodes}} = 2800$. For *GTB*, we used $n_{\text{batch}} = 128$ and $n_{\text{epsiodes}} = 500$.

Let c_{entropy} denote the policy entropy coefficient, α_θ the agent's policy learning rate, $\alpha_{v,\text{agent}}$ the agent's value function learning rate, α_{ID} the incentive designer's learning rate (dual-RL's policy, MetaGrad's incentive function), $\alpha_{v,\text{ID}}$ the incentive designer's value function learning rate, c_v the value function target update rate (i.e., $\theta'_v \leftarrow c_v\theta_v + (1 - c_v)\theta'_v$,

where θ'_v are parameters of a separate target network and θ_v are parameters of the main value network). For MetaGrad in *Escape Room*, we used a separate optimizer for the cost part of the ID's objective, with learning rate denoted by α_{cost}.

The hyperparameter ranges used for tuning are the following.

- For *Escape Room*: $c_{\text{entropy}} \in (0.01, 10.0)$, $\alpha_\theta \in (10^{-5}, 10^{-3})$, designer $c_{\text{entropy}} \in (0.01, 10.0)$, $\alpha_{\text{ID}} \in (10^{-5}, 10^{-2})$, agent first hidden layer $h_1 \in \{64, 128\}$, agent second hidden layer $h_2 \in \{16, 32, 64\}$, designer first hidden layer $d_1 \in \{64, 128\}$, designer second hidden layer $d_2 \in \{16, 32, 64\}$, $\alpha_{\text{cost}} \in (10^{-5}, 10^{-3})$.

- For *Cleanup*: agent $c_{\text{entropy}} \in (10^{-3}, 1.0)$, agent $\epsilon_{\text{start}} \in \{0.5, 1.0\}$, agent $\epsilon_{\text{end}} \in \{0.05, 0.1\}$, agent $\epsilon_{\text{div}} \in \{100, 1000, 5000\}$, $\alpha_\theta \in (10^{-5}, 10^{-3})$ $\alpha_{v,\text{agent}} \in (10^{-5}, 10^{-3})$, agent $c_v \in (10^{-3}, 1)$, first FC layer $h_1 \in \{64, 128, 256\}$, second FC layer $h_2 \in \{64, 128, 256\}$, designer $c_{\text{entropy}} \in (10^{-3}, 1.0)$, $\alpha_{\text{ID}} \in (10^{-5}, 10^{-3})$, $\alpha_{v,\text{ID}} \in (10^{-5}, 10^{-3})$, designer PPO $\epsilon \in (0.01, 0.5)$, designer $c_v \in (10^{-3}, 1)$

- For *Gather-Trade-Build*: agent $c_{\text{entropy}} \in (10^{-3}, 10.0)$, $\alpha_\theta \in (10^{-5}, 10^{-3})$, $\alpha_{v,\text{agent}} \in (10^{-5}, 10^{-3})$, agent PPO $\epsilon \in (0.01, 0.5)$, agent $c_v \in (10^{-3}, 1)$, designer $c_{\text{entropy}} \in (10^{-3}, 10.0)$, $\alpha_{\text{ID}} \in (10^{-5}, 10^{-3})$, $\alpha_{v,\text{ID}} \in (10^{-5}, 10^{-3})$, designer PPO $\epsilon \in (0.01, 0.5)$, designer $c_v \in (10^{-3}, 1.0)$.

In *Escape Room*, we used discount $\gamma = 0.99$, gradient descent for agents, and AdamOptimizer [154] for the ID. In *Cleanup*, we used discount $\gamma = 0.99$, dual-RL designer GAE $\lambda = 0.99$, gradient descent for agents, and AdamOptimizer for the ID. In *Gather-Trade-Build*, the fixed hyperparameters (i.e., not part of tuning) are: GAE $\lambda = 0.98$ for both agents and the ID, discount factor $\gamma = 0.99$, agent gradient clipping by 10.0 (for dual-RL and US federal only), truncation of an episode rollout to subsequences of length 50 for LSTMs. We used AdamOptimizer for both agents and the ID. For the case without curriculum, and for MetaGrad with curriculum, we applied an exploration lower bound on the agents' policies such that the actual policy is $\pi(a|s) = (1 - \epsilon)\hat{\pi}(a|s) + \epsilon/|\mathcal{A}|$, where $|\mathcal{A}|$ is the size of the

Table C.1: Hyperparameters in Escape Room ER$(5, 2)$.

	ER$(5, 2)$		
Parameter	MetaGrad	dual-RL (c)	dual-RL (d)
Agent c_{entropy}	0.0166	7.07	0.386
α_θ	$9.56 \cdot 10^{-5}$	$7.70 \cdot 10^{-4}$	$3.41 \cdot 10^{-4}$
Designer c_{entropy}	-	-	0.488
α_{cost}	$6.03 \cdot 10^{-5}$	-	-
α_{ID}	$7.93 \cdot 10^{-4}$	$1.17 \cdot 10^{-4}$	$2.47 \cdot 10^{-3}$
h_1	64	64	128
h_2	64	32	64
d_1	64	64	128
d_2	32	64	64

Table C.2: Hyperparameters in Escape Room ER$(10, 5)$.

	ER$(10, 5)$		
Parameter	MetaGrad	dual-RL (c)	dual-RL (d)
Agent c_{entropy}	0.0166	0.345	0.0166
α_θ	$9.56 \cdot 10^{-5}$	$9.05 \cdot 10^{-4}$	$9.56 \cdot 10^{-5}$
Designer c_{entropy}	-	-	0.148
α_{cost}	$6.03 \cdot 10^{-5}$	-	-
α_{ID}	$7.93 \cdot 10^{-4}$	$1.63 \cdot 10^{-4}$	$7.07 \cdot 10^{-3}$
h_1	64	64	64
h_2	64	32	64
d_1	64	128	64
d_2	32	64	32

discrete action space and ϵ linearly decreases from ϵ_{start} to ϵ_{end} over ϵ_{div} episodes. We did not apply this exploration lower bound to dual-RL and US federal in the curriculum case as it was not used in previous work [42].

C.2.4 Computation resources

Experiments were run on the following hardware: Intel(R) Core(TM) i7-4790 CPU with NVIDIA GeForce GTX 750 Ti GPU; Intel(R) Xeon(R) CPU E5-2630 v4 with NVIDIA GeForce GTX 1080 GPU.

Table C.3: Hyperparameters in Cleanup.

Parameter	MetaGrad	dual-RL (c)
Agent c_{entropy}	0.129	0.173
Agent ϵ_{start}	1.0	1.0
Agent ϵ_{end}	0.05	0.1
Agent ϵ_{div}	100	100
α_θ	$6.73 \cdot 10^{-4}$	$6.23 \cdot 10^{-4}$
$\alpha_{v,\text{agent}}$	$5.54 \cdot 10^{-4}$	$4.46 \cdot 10^{-4}$
Agent c_v	$7.93 \cdot 10^{-3}$	0.292
Designer c_{entropy}	-	0.23
α_{ID}	$1.24 \cdot 10^{-5}$	$1.15 \cdot 10^{-5}$
$\alpha_{v,\text{ID}}$	$2.4 \cdot 10^{-5}$	$1.72 \cdot 10^{-5}$
Designer PPO ϵ	0.0172	0.0164
Designer c_v	0.114	0.846
h_1	64	64
h_2	64	64

C.3 Additional results

The central planner in [98] is the centralized analogue of a LOLA agent [96], who learns to shape opponent behavior by anticipating their policy update. [186] showed analytically in certain settings that this can result in aggressive behaviors that try to force opponent compliance, especially in settings with multiple LOLA agents. Moreover, the anticipatory behavior of a LOLA-type central planner does not benefit from the knowledge of the actual impact of incentives on recipients' behavior, as previously shown experimentally in the context of decentralized incentivization [40]. To match the open source implementation, we used a single-layer linear policy network and actor-critic agents. Although [98] report good results in matrix games, we see in Figure C.1 that this method learns slower than MetaGrad. Runs were truncated after 6k episodes as the method was unstable on the temporally-extended Escape Room game.

Table C.4: Hyperparameters in Gather-Trade-Build.

	No curriculum		
Parameter	MetaGrad	dual-RL	US Federal
Agent c_{entropy}	0.184	0.282	0.253
Agent ϵ_{start}	0.5	0.5	0.5
Agent ϵ_{end}	0.05	0.05	0.05
Agent ϵ_{div}	$5k$	$5k$	$5k$
α_θ	$3.30 \cdot 10^{-3}$	$3.92 \cdot 10^{-4}$	$1.39 \cdot 10^{-4}$
$\alpha_{v,\text{agent}}$	$1.67 \cdot 10^{-5}$	$3.74 \cdot 10^{-4}$	$5.0 \cdot 10^{-5}$
Agent PPO ϵ	0.01	0.0201	0.166
Agent c_v	0.0126	$2.80 \cdot 10^{-3}$	$4.36 \cdot 10^{-3}$
Designer c_{entropy}	-	0.203	-
α_{ID}	$1.82 \cdot 10^{-5}$	$7.80 \cdot 10^{-4}$	-
$\alpha_{v,\text{ID}}$	$8.64 \cdot 10^{-4}$	$2.30 \cdot 10^{-4}$	-
Designer PPO ϵ	0.382	0.0216	-
Designer c_v	0.0158	$1.90 \cdot 10^{-3}$	-
Designer ϵ_{start}	-	0.5	-
Designer ϵ_{end}	-	0.05	-
Designer ϵ_{div}	-	$5k$	-

Table C.5: Hyperparameters in Gather-Trade-Build.

	Curriculum		
Parameter	MetaGrad	dual-RL	US Federal
Agent c_{entropy}	0.0316	0.0166	0.0811
Agent ϵ_{start}	0.5	-	-
Agent ϵ_{end}	0.05	-	-
Agent ϵ_{div}	$5k$	-	-
α_θ	$1.09 \cdot 10^{-5}$	$3.33 \cdot 10^{-4}$	$1.32 \cdot 10^{-4}$
$\alpha_{v,\text{agent}}$	$1.14 \cdot 10^{-5}$	$5.58 \cdot 10^{-5}$	$8.87 \cdot 10^{-5}$
Agent PPO ϵ	0.0308	0.084	0.034
Agent c_v	0.0126	0.221	0.702
Designer c_{entropy}	-	0.33	-
α_{ID}	$2.03 \cdot 10^{-4}$	$4.9 \cdot 10^{-5}$	-
$\alpha_{v,\text{ID}}$	10^{-5}	$5.14 \cdot 10^{-4}$	-
Designer PPO ϵ	0.0394	0.0295	-
Designer c_v	0.0573	0.0178	-
Designer ϵ_{start}	-	-	-
Designer ϵ_{end}	-	-	-
Designer ϵ_{div}	-	-	-

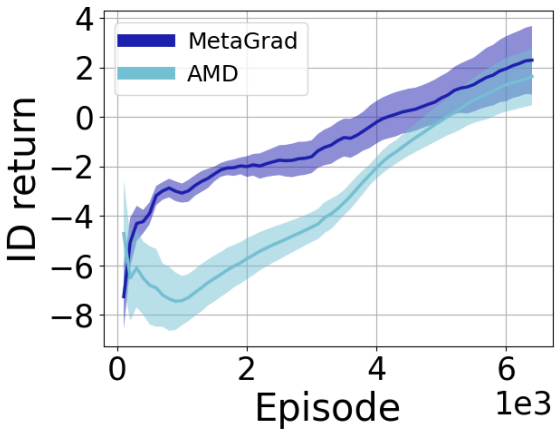


Figure C.1: ER$(2, 1)$

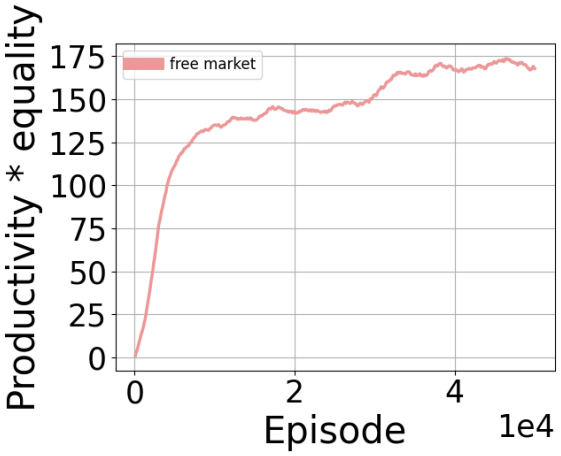


Figure C.2: GTB with curriculum: Phase 1 train curve

REFERENCES

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

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.*, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, p. 859, 2019.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

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

[7] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, p. 103 535, 2021.

[8] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "Is multiagent deep reinforcement learning the answer or the question? a brief survey," *arXiv preprint arXiv:1810.05587*, 2018.

[9] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[10] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 157–163.

[11] J. Hu, M. P. Wellman, *et al.*, "Multiagent reinforcement learning: Theoretical framework and an algorithm.," in *ICML*, Citeseer, vol. 98, 1998, pp. 242–250.

[12] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *Journal of machine learning research*, vol. 4, no. Nov, pp. 1039–1069, 2003.

[13] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: A critical survey," Technical report, Stanford University, Tech. Rep., 2003.

[14] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[15] Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?" *Artificial Intelligence*, vol. 171, no. 7, pp. 365–377, 2007.

[16] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6382–6393.

[17] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[19] J. Yang, A. Nakhaei, D. Isele, K. Fujimura, and H. Zha, "Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2019.

[20] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial informatics*, vol. 9, no. 1, pp. 427–438, 2013.

[21] Z. Zhang, J. Yang, and H. Zha, "Integrating independent and centralized multi-agent reinforcement learning for traffic signal network optimization," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 2083–2085.

[22] K. Tumer and A. Agogino, "Distributed agent-based air traffic flow management," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[23] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[24] F. A. Oliehoek and C. Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016.

[25] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[26] Y. Zhao, I. Borovikov, J. Rupert, C. Somers, and A. Beirami, "On multi-agent learning in team sports games," *arXiv preprint arXiv:1906.10124*, 2019.

[27] L. J. Ratliff, R. Dong, S. Sekar, and T. Fiez, "A perspective on incentive design: Challenges and opportunities," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 305–338, 2019.

[28] L. J. Ratliff and T. Fiez, "Adaptive incentive design," *IEEE Transactions on Automatic Control*, 2020.

[29] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[30] P. A. Van Lange, J. Joireman, C. D. Parks, and E. Van Dijk, "The psychology of social dilemmas: A review," *Organizational Behavior and Human Decision Processes*, vol. 120, no. 2, pp. 125–141, 2013.

[31] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, *et al.*, "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*, International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2085–2087.

[32] K. Son, D. Kim, W. J. Kang, D. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2019.

[33] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *International Conference on Learning Representations*, 2019.

[34] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, "Variational option discovery algorithms," *arXiv preprint arXiv:1807.10299*, 2018.

[35] J. Yang, I. Borovikov, and H. Zha, "Hierarchical cooperative multi-agent reinforcement learning with skill discovery," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1566–1574.

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

[37] A. Rapoport, "Prisoner's dilemma—recollections and observations," in *Game Theory as a Theory of a Conflict Resolution*, Springer, 1974, pp. 17–34.

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

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

[40] J. Yang, A. Li, M. Farajtabar, P. Sunehag, E. Hughes, and H. Zha, "Learning to incentivize other learning agents," *arXiv preprint arXiv:2006.06051*, 2020.

[41] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas, "Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 463–478, 2016.

[42] S. Zheng, A. Trott, S. Srinivasa, N. Naik, M. Gruesbeck, D. C. Parkes, and R. Socher, "The ai economist: Improving equality and productivity with ai-driven tax policies," *arXiv preprint arXiv:2004.13332*, 2020.

[43] D. C. Parkes and M. P. Wellman, "Economic reasoning and artificial intelligence," *Science*, vol. 349, no. 6245, pp. 267–272, 2015.

[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[46] A. G. Barto, R. S. Sutton, and C. Watkins, *Learning and sequential decision making*. University of Massachusetts Amherst, MA, 1989.

[47] R. A. Howard, "Dynamic programming and markov processes.," 1960.

[48] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[49] G. Owen, *Game Theory: Second Edition*. Academic Press, Orlando, Florida, 1982.

[50] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[51] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[52] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*, Elsevier, 1990, pp. 216–224.

[53] ——, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[54] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.

[55] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

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

[57] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[58] Y.-H. Chang, T. Ho, and L. P. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *Advances in neural information processing systems*, 2004, pp. 807–814.

[59] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to communicate at scale in multiagent cooperative and competitive tasks," in *International Conference on Learning Representations*, 2019.

[60] J. L. Austerweil, S. Brawner, A. Greenwald, E. Hilliard, M. Ho, M. L. Littman, J. MacGlashan, and C. Trimbach, "How other-regarding preferences can promote cooperation in non-zero-sum grid games," in *Proceedings of the AAAI Symposium on Challenges and Opportunities in Multiagent Learning for the Real World*, 2016.

[61] D. T. Nguyen, A. Kumar, and H. C. Lau, "Credit assignment for collective multiagent rl with global rewards," in *Advances in Neural Information Processing Systems*, 2018, pp. 8112–8123.

[62] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[63] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 5872–5881.

[64] F. A. Oliehoek, M. T. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.

[65] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *International Conference on Machine Learning*, 2017, pp. 2681–2690.

[66] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3540–3549.

[67] T. Shu and Y. Tian, "M3rl: Mind-aware multi-agent management reinforcement learning," in *International Conference on Learning Representations*, 2019.

[68] N. Carion, G. Synnaeve, A. Lazaric, and N. Usunier, "A structured prediction approach for generalization in cooperative multi-agent reinforcement learning," in *Advances in neural information processing systems*, 2019.

[69] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 41–48.

[70] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, Springer, 2017, pp. 66–83.

[71] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[72] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[73] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[74] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*, 1993, pp. 271–278.

[75] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[76] D. Precup, "Temporal abstraction in reinforcement learning," *Ph. D. thesis, University of Massachusetts*, 2000.

[77] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *International Symposium on abstraction, reformulation, and approximation*, Springer, 2002, pp. 212–223.

[78] R. Makar, S. Mahadevan, and M. Ghavamzadeh, "Hierarchical multi-agent reinforcement learning," in *Proceedings of the fifth international conference on Autonomous agents*, ACM, 2001, pp. 246–253.

[79] M. Ghavamzadeh, S. Mahadevan, and R. Makar, "Hierarchical multi-agent reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 197–229, 2006.

[80] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[81] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, 2016, pp. 3675–3683.

[82] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[83] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proceedings of the Eighteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., 2001, pp. 361–368.

[84] K. Gregor, D. J. Rezende, and D. Wierstra, "Variational intrinsic control," *arXiv preprint arXiv:1611.07507*, 2016.

[85] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations*, 2014.

[86] H. Tang, J. Hao, T. Lv, Y. Chen, Z. Zhang, H. Jia, C. Ren, Y. Zheng, C. Fan, and L. Wang, "Hierarchical deep multiagent reinforcement learning," *arXiv preprint arXiv:1809.09332*, 2018.

[87] S. Ahilan and P. Dayan, "Feudal multi-agent hierarchies for cooperative reinforcement learning," in *4th Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM 2019)*, 2019, p. 57.

[88] Y. Lee, J. Yang, and J. J. Lim, "Learning to coordinate manipulation skills via skill behavior diversification," in *International Conference on Learning Representations*, 2020.

[89] A. Wilson, A. Fern, and P. Tadepalli, "Bayesian policy search for multi-agent role discovery," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

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

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

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

[93] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas, "Social influence as intrinsic motivation for multi-agent deep reinforcement learning," in *International Conference on Machine Learning*, 2019, pp. 3040–3049.

[94] D. E. Hostallero, D. Kim, S. Moon, K. Son, W. J. Kang, and Y. Yi, "Inducing cooperation through reward reshaping based on peer evaluations in deep multi-agent reinforcement learning," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 520–528.

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

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

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

[98] T. Baumann, T. Graepel, and J. Shawe-Taylor, "Adaptive mechanism design: Learning to promote cooperation," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–7.

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

[100] Z. Zheng, J. Oh, and S. Singh, "On learning intrinsic rewards for policy gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 4644–4654.

[101] E. Akçay and J. Roughgarden, "The evolution of payoff matrices: Providing incentives to cooperate," *Proceedings of the Royal Society B: Biological Sciences*, vol. 278, no. 1715, pp. 2198–2206, 2011.

[102] R. S. Sutton, "Adapting bias by gradient descent: An incremental version of delta-bar-delta," in *AAAI*, 1992, pp. 171–176.

[103] Z. Xu, H. P. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," in *Advances in neural information processing systems*, 2018, pp. 2396–2407.

[104] S. Singh, R. L. Lewis, and A. G. Barto, "Where do rewards come from," in *Proceedings of the annual conference of the cognitive science society*, Cognitive Science Society, 2009, pp. 2601–2606.

[105] D. Paccagnan, R. Chandan, and J. R. Marden, "Utility design for distributed resource allocation-part i: Characterizing and optimizing the exact price of anarchy," *IEEE Transactions on Automatic Control*, 2019.

[106] C. Li, F. He, H. Qi, D. Cheng, L. Ma, Y. Wu, and S. Chen, "Potential games design using local information," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 1911–1916.

[107] N. Li and J. R. Marden, "Designing games for distributed optimization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 2, pp. 230–242, 2013.

[108] R. Gopalakrishnan, J. R. Marden, and A. Wierman, "Potential games are necessary to ensure pure nash equilibria in cost sharing games," *Mathematics of Operations Research*, vol. 39, no. 4, pp. 1252–1296, 2014.

[109] W. Shen, P. Tang, and S. Zuo, "Automated mechanism design via neural networks," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 215–223.

[110] L. Tesfatsion and K. L. Judd, *Handbook of computational economics: agent-based computational economics*. Elsevier, 2006.

[111] G. Brero, A. Eden, M. Gerstgrasser, D. C. Parkes, and D. Rheingans-Yoo, "Reinforcement learning of simple indirect mechanisms," *arXiv preprint arXiv:2010.01180*, 2020.

[112] P. Tang, "Reinforcement mechanism design.," in *IJCAI*, 2017, pp. 5146–5150.

[113] D. Pardoe, P. Stone, M. Saar-Tsechansky, and K. Tomak, "Adaptive mechanism design: A metalearning approach," in *Proceedings of the 8th international conference on Electronic commerce*, 2006, pp. 92–102.

[114] D. Mguni, J. Jennings, E. Sison, S. Valcarcel Macua, S. Ceppi, and E. Munoz de Cote, "Coordinating the crowd: Inducing desirable equilibria in non-cooperative systems," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 386–394.

[115] J. Li, J. Yu, Y. Nie, and Z. Wang, "End-to-end learning and intervention in games," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[116] P. Danassis, A. Filos-Ratsikas, and B. Faltings, "Achieving diverse objectives with ai-driven prices in deep reinforcement learning multi-agent markets," *arXiv preprint arXiv:2106.06060*, 2021.

[117] Z. Xu, H. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver, "Meta-gradient reinforcement learning with an objective discovered online," *arXiv preprint arXiv:2007.08433*, 2020.

[118] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 1540–1552.

[119] B. Stadie, L. Zhang, and J. Ba, "Learning intrinsic rewards as a bi-level optimization problem," in *Conference on Uncertainty in Artificial Intelligence*, PMLR, 2020, pp. 111–120.

[120] L. Kirsch, S. van Steenkiste, and J. Schmidhuber, "Improving generalization in meta reinforcement learning using learned objectives," in *International Conference on Learning Representations*, 2019.

[121] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver, "Discovering reinforcement learning algorithms," *arXiv preprint arXiv:2007.08794*, 2020.

[122] Blizzard Entertainment, *Starcraft ii*, https://starcraft2.com/en-us/, Last accessed on 2019-09-07, 2019.

[123] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[124] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 1774–1783.

[125] S. Srinivasan, M. Lanctot, V. Zambaldi, J. Pérolat, K. Tuyls, R. Munos, and M. Bowling, "Actor-critic policy optimization in partially observable multiagent environments," in *Advances in Neural Information Processing Systems*, 2018, pp. 3426–3439.

[126] S. B. Thrun, "Efficient exploration in reinforcement learning," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1992.

[127] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018.

[128] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance reduction for policy gradient with action-dependent factorized baselines," in *International Conference on Learning Representations*, 2018.

[129] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, pp. 278–287.

[130] D. V. Pynadath and M. Tambe, "The communicative multiagent team decision problem: Analyzing teamwork theories and models," *Journal of artificial intelligence research*, vol. 16, pp. 389–423, 2002.

[131] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *International Conference on Learning Representations*, 2018.

[132] S. Liu, G. Lever, J. Merel, S. Tunyasuvunakool, N. Heess, and T. Graepel, "Emergent coordination through competition," in *International Conference on Learning Representations*, 2019.

[133] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, 2015, pp. 1312–1320.

[134] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[135] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2034–2039.

[136] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 204–211.

[137] Y. Zhao, I. Borovikov, A. Beirami, J. Rupert, C. Somers, J. Harder, F. d. M. Silva, J. Kolen, J. Pinto, R. Pourabolghasem, *et al.*, "Winning isn't everything: Training human-like agents for playtesting and game ai," *arXiv preprint arXiv:1903.10545*, 2019.

[138] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[139] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[140] H. M. Le, Y. Yue, P. Carr, and P. Lucey, "Coordinated multi-agent imitation learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1995–2003.

[141] K. Rohanimanesh and S. Mahadevan, "Learning to take concurrent actions," in *Advances in neural information processing systems*, 2003, pp. 1651–1658.

[142] C. Somers, J. Rupert, Y. Zhao, I. Borovikov, and J. Yang, *Simple Team Sports Simulator (STS2)*, version 1.0.0, Feb. 2020.

[143] K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, *et al.*, "Google research football: A novel reinforcement learning environment," *arXiv preprint arXiv:1907.11180*, 2019.

[144] A. Franks, A. Miller, L. Bornn, K. Goldsberry, *et al.*, "Characterizing the spatial structure of defensive skill in professional basketball," *The Annals of Applied Statistics*, vol. 9, no. 1, pp. 94–121, 2015.

[145] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[146] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, "Ad hoc autonomous agent teams: Collaboration without pre-coordination," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

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

[148] J. Delfgaauw and R. Dur, "Incentives and workers' motivation in the public sector," *The Economic Journal*, vol. 118, no. 525, pp. 171–191, 2008.

[149] M. P. Doxey, *Economic sanctions and international enforcement*. Springer, 1980.

[150] M. O. Jackson and S. Wilkie, "Endogenous games and mechanisms: Side payments among players," *The Review of Economic Studies*, vol. 72, no. 2, pp. 543–566, 2005.

[151] B. Harstad, "Do side payments help? collective decisions and strategic delegation," *Journal of the European Economic Association*, vol. 6, no. 2-3, pp. 468–477, 2008.

[152] Y.-f. Fong and J. Surti, "The optimal degree of cooperation in the repeated prisoners' dilemma with side payments," *Games and Economic Behavior*, vol. 67, no. 1, pp. 277–291, 2009.

[153] W. Güth, "An evolutionary approach to explaining cooperative behavior by reciprocal incentives," *International Journal of Game Theory*, vol. 24, no. 4, pp. 323–344, 1995.

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

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

[156] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, pp. 1573–1576, 2016.

[157] D. Bissell, T. Birtchnell, A. Elliott, and E. L. Hsu, "Autonomous automobilities: The social impacts of driverless vehicles," *Current Sociology*, vol. 68, no. 1, pp. 116–134, 2020.

[158] P. N. Howard, S. Woolley, and R. Calo, "Algorithms, bots, and political communication in the us 2016 election: The challenge of automated political communication for election law and administration," *Journal of information technology & politics*, vol. 15, no. 2, pp. 81–93, 2018.

[159] M.-W. Hsu, S. Lessmann, M.-C. Sung, T. Ma, and J. E. Johnson, "Bridging the divide in financial market forecasting: Machine learners vs. financial economists," *Expert Systems with Applications*, vol. 61, pp. 215–234, 2016.

[160] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine learning in agriculture: A review," *Sensors*, vol. 18, no. 8, p. 2674, 2018.

[161] D. Robinson and D. Goforth, *The topology of the 2x2 games: a new periodic table*. Psychology Press, 2005, vol. 3.

[162] A. Dafoe, E. Hughes, Y. Bachrach, T. Collins, K. R. McKee, J. Z. Leibo, K. Larson, and T. Graepel, "Open problems in cooperative ai," *arXiv preprint arXiv:2012.08630*, 2020.

[163] Y.-C. Ho, P. Luh, and R. Muralidharan, "Information structure, stackelberg games, and incentive controllability," *IEEE Transactions on Automatic Control*, vol. 26, no. 2, pp. 454–460, 1981.

[164] H. v. Stackelberg, "Marktform und Gleichgewicht," *Julius Springer*, 1934.

[165] N. Groot, B. De Schutter, and H. Hellendoorn, "Reverse stackelberg games, part i: Basic framework," in *2012 IEEE International Conference on Control Applications*, IEEE, 2012, pp. 421–426.

[166] J. S. Vardakas, N. Zorba, and C. V. Verikoukis, "A survey on demand response programs in smart grids: Pricing methods and optimization algorithms," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 152–178, 2014.

[167] W. Barfuss, J. F. Donges, V. V. Vasconcelos, J. Kurths, and S. A. Levin, "Caring for the future can turn tragedy into comedy for long-term collective action under risk of collapse," *Proceedings of the National Academy of Sciences*, vol. 117, no. 23, pp. 12 915–12 922, 2020.

[168] A. de Palma and R. Lindsey, "Traffic congestion pricing methodologies and technologies," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 6, pp. 1377–1399, 2011.

[169] W. B. Arthur, "Foundations of complexity economics," *Nature Reviews Physics*, pp. 1–10, 2021.

[170] B. Ammanath, D. Jarvis, and S. Hupfer, "Thriving in the era of pervasive AI: Deloitte's state of AI in the enterprise, 3rd edition," Deloitte, Tech. Rep., 2020.

[171] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[172] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," in *NeurIPS*, 2018.

[173] E. Ie, C.-w. Hsu, M. Mladenov, V. Jain, S. Narvekar, J. Wang, R. Wu, and C. Boutilier, "Recsim: A configurable simulation platform for recommender systems," *arXiv preprint arXiv:1909.04847*, 2019.

[174] Y. Niv, "Reinforcement learning in the brain," *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 139–154, 2009.

[175] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *arXiv preprint arXiv:1908.03963*, 2019.

[176] C. Wu, A. M. Bayen, and A. Mehta, "Stabilizing traffic with autonomous vehicles," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 6012–6018.

[177] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, *et al.*, "Policy gradient methods for reinforcement learning with function approximation.," in *NIPs*, Citeseer, vol. 99, 1999, pp. 1057–1063.

[178] E. Vinitsky, *Sequential social dilemma games*, https://github.com/eugenevinitsky/sequential_social_dilemma_games, 2020.

[179] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[180] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games," in *International Conference on Machine Learning*, 2018, pp. 354–363.

[181] T. Fiez, B. Chasnov, and L. Ratliff, "Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study," in *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[182] F. W. Nietzsche and R. J. Hollingdale, *On the genealogy of morals*. Vintage, 1989.

[183] J. Kramár, N. Rabinowitz, T. Eccles, and A. Tacchetti, "Should i tear down this wall? optimizing social metrics by evaluating novel actions," *arXiv preprint arXiv:2004.07625*, 2020.

[184] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[185] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[186] A. Letcher, J. Foerster, D. Balduzzi, T. Rocktäschel, and S. Whiteson, "Stable opponent shaping in differentiable games," in *International Conference on Learning Representations*, 2018.