Cooperation in Multi-Agent Reinforcement Learning

Oscar Duys*

October 2025

Abstract

Reinforcement learning (RL) has achieved impressive results in settings where a single agent optimises its behaviour in a fixed environment. In many realistic domains, however, AI systems will interact with other adaptive agents—human or artificial—in ways that are better modelled as multi-agent reinforcement learning (MARL). In such settings, individual learning rules and local incentives do not automatically produce good collective outcomes. Agents may converge to non-cooperative equilibria, over-exploit shared resources, or exhibit unstable dynamics, even when mutually beneficial behaviour is possible. Understanding and engineering *cooperation* in MARL is therefore an important prerequisite for deploying AI systems in strategic, multi-agent environments.

This thesis studies cooperation in MARL from both a theoretical and empirical perspective. It first develops the necessary foundations: single-agent RL in Markov Decision Processes, game-theoretic models of strategic interaction (normal-form, repeated, and Markov games), and evolutionary game theory as a population-level view of adaptation. It then surveys core MARL architectures and the self-play paradigm, highlighting why naive extensions of single-agent methods often fail in social dilemmas. Building on this, the thesis proposes a simple taxonomy of mechanisms for fostering cooperation, grouped into four classes: (i) payoff and environment design, (ii) learning architectures for credit assignment, (iii) information and communication mechanisms, and (iv) population-level mechanisms based on reciprocity, reputation, and partner choice. Finally, an empirical study in the sequential social dilemma *Cleanup* compares representative mechanisms from these classes. The results highlight a critical distinction between *incentive* and *capacity*. We find that while shared rewards align motivations, they fail to produce efficient coordination without architectural support. Instead, mechanisms that explicitly solve the credit assignment problem (such as centralised critics) are required to unlock high-welfare outcomes and functional specialisation. Altogether, these results clarify when and why learning agents do or do not cooperate, and outline concrete levers for designing more cooperative multi-agent AI systems

*Experimetal code and visualisations of policies in the Cleanup environment: `https://github.com/OzDuys/SocialJax`

Contents

1 Introduction

Reinforcement learning (RL) has emerged as one of the central paradigms for training autonomous decision-making systems. From Atari games and Go to robotic control and large language models fine-tuned with human feedback, RL provides a general recipe: specify an environment, define a reward signal, and let an agent discover a policy that maximises long-term return. Most of the classical successes of RL, however, are fundamentally *single-agent* successes. The agent interacts with a stationary environment that does not strategise, adapt, or pursue its own objectives.

As AI systems are deployed more widely, this solitary picture becomes increasingly misleading. Many real-world settings are inherently *multi-agent*: financial markets populated by algorithmic traders, fleets of autonomous vehicles sharing road networks, electricity grids controlled by multiple adaptive systems, or automated bidding agents on online platforms. In such domains, AI systems will not act in isolation but will instead interact with one another, sometimes at speeds and scales that exceed human oversight. These interactions can create both risks and opportunities. On the risk side, even relatively simple algorithmic interactions have already caused significant disruption, such as the 2010 "flash crash" in which automated trading contributed to a sudden, large drop in stock prices. On the opportunity side, suitably designed AI agents might help address large-scale collective action problems such as climate change or arms control by enabling better monitoring, coordination, and enforcement of agreements.

This broader perspective motivates the emerging research programme of *cooperative AI* [1]. Conitzer and Oesterheld's "Foundations of Cooperative AI"[2] argue that as AI systems become more capable and more deeply embedded in critical infrastructure, we must understand not only how to align individual agents with human values, but also how to ensure that *interacting* agents produce good collective outcomes. Aligning each agent in isolation is not sufficient: even if every system is, in some sense, well intentioned, the game-theoretic structure of their interactions can still push them toward equilibria that are disastrous from a global perspective. This is particularly concerning in general-sum settings where there exist cooperative outcomes that are Pareto-superior to the status quo, but individual incentives drive agents toward conflict, over-exploitation, or inefficient competition.

Multi-agent reinforcement learning (MARL) sits exactly at this intersection of learning and strategic interaction. It takes the tools of RL and applies them in environments modelled as stochastic or Markov games, where multiple learning agents simultaneously adapt their policies. From one angle, MARL provides a natural testbed for cooperative AI: we can specify a social dilemma, choose a learning rule for each agent, and empirically observe whether cooperation emerges, fails, or oscillates. From another angle, MARL exposes the core difficulties that cooperative AI will have to address. As discussed later in Section 4, each agent in a multi-agent system faces a moving-target learning problem due to non-stationarity, partial observability, and the fact that its own reward depends on the behaviour of others. Standard single-agent algorithms such as Q-learning or PPO, when naively applied, can converge to non-cooperative equilibria even when cooperative outcomes are available and clearly desirable.

The central theme of this thesis is *cooperation in multi-agent reinforcement learning*. Informally, the question is:

> In environments where cooperative and non-cooperative outcomes are both possible, under what conditions do learning agents discover and maintain cooperative behaviour?

This question has several layers. At the level of formal modelling, we need a language for describing multi-agent interactions and defining what it even means for an outcome or a policy profile to be "cooperative". At the level of learning dynamics, we want to understand how particular algorithms behave in social dilemmas: do they reliably converge to high-welfare equilibria, or do they fall into defection, cycles, or brittle patterns of cooperation that are easily exploited? At the level of system design, we are interested in mechanisms—incentive structures, architectural choices, communication protocols, and population structures—that make cooperative outcomes more likely.

The "Foundations of Cooperative AI" paper emphasises that these questions are not purely academic. As AI systems come to negotiate, trade, and coordinate on our behalf, they will implicitly be solving equilibrium selection problems in complex games that we only partially understand: If standard learning dynamics tend to favour simple but inefficient equilibria such as mutual defection, we risk locking in outcomes that are worse for all human stakeholders, even when mutually beneficial alternatives exist. Conversely, if we can identify mechanisms that robustly steer learning toward cooperative equilibria, we

may be able to harness AI to help solve long-standing coordination problems rather than exacerbating them.

The aim of this thesis is more modest but in this spirit. It focuses on relatively simple Markov games and social dilemmas, and uses them to explore how cooperation can be encouraged or hindered in MARL systems. The hope is that by understanding these mechanisms in small, controlled settings, we gain conceptual tools that can later be applied to more realistic, higher-stakes domains.

Scope and contributions. Broadly, the thesis makes three contributions.

First, it provides a self-contained introduction to the theoretical foundations needed to reason about cooperation in MARL. Section 2.1 reviews the standard RL setting based on Markov Decision Processes (MDPs), value functions, and policy-gradient methods, culminating in modern actor–critic algorithms such as Proximal Policy Optimization (PPO). This sets the stage for understanding how single-agent learning behaves under stationary dynamics. Section 3 then introduces the basic tools of non-cooperative game theory: normal-form and repeated games, Nash equilibrium, and canonical 2×2 games such as the Prisoner's Dilemma and Stag Hunt. These models make precise the notion of social dilemmas and equilibrium selection. The chapter also extends to Markov games and partially observable stochastic games, connecting the game-theoretic view to the RL formalism, and to evolutionary game theory, which provides a dynamical perspective on how populations of strategies evolve over time.

Second, the thesis surveys and organises the algorithmic landscape of MARL with a focus on cooperation. Section 4 describes the core challenges of learning in multi-agent systems—non-stationarity, partial observability, and credit assignment—and presents the main architectural paradigms, including independent learners, fully centralised controllers, and the now standard "centralised training with decentralized execution" (CTDE) framework. Section 5 then discusses self-play and population-based training as mechanisms for automatically generating curricula and discovering robust strategies in competitive and mixed-motive games. Building on this groundwork, Section 6 proposes a simple taxonomy of mechanisms for promoting cooperation in MARL, distinguishing between (i) payoff and environment design, (ii) learning architectures that address credit assignment, (iii) information and communication mechanisms, and (iv) population-level mechanisms based on reciprocity, reputation, and partner choice. This taxonomy is intended as a conceptual bridge between the cooperative AI agenda and concrete MARL techniques.

Third, the thesis includes an empirical component that instantiates a representative mechanism from each class in a single sequential social dilemma environment. Section 7 uses the SocialJax implementation of the Cleanup game as a testbed, holding the underlying dynamics and base learning algorithm fixed while varying only the cooperation mechanism. By comparing social welfare, cooperation rates, equality of returns, and stability of behaviour across conditions, the experiments illustrate how relatively small changes in reward design, architecture, information flow, or population structure can qualitatively change whether cooperation emerges and persists.

Throughout, the emphasis is on clarity and synthesis rather than theoretical novelty. The goal is to assemble, in one place, the ingredients needed to understand why cooperation is both difficult and essential in multi-agent learning, and to highlight how ideas from RL, game theory, and evolutionary dynamics come together in the study of cooperative AI. In this sense, the thesis can be read as a small, concrete response to the broader call for a "foundations of cooperative AI" research agenda: it examines, in a simplified MARL setting, how we might begin to engineer systems of learning agents that are not only individually capable, but also collectively well-behaved.

2 Reinforcement Learning

2.1 The Language of Learning: The Markov Decision Process

To formalize the problem of learning from interaction, the field of Reinforcement Learning (RL) employs the mathematical framework of the **Markov Decision Process** (MDP). An MDP provides a clean, abstract language for describing the essential elements of sequential decision-making under uncertainty, where an agent interacts with an environment over a series of discrete timesteps [3]. The core interaction loop is visualized in Figure 1.

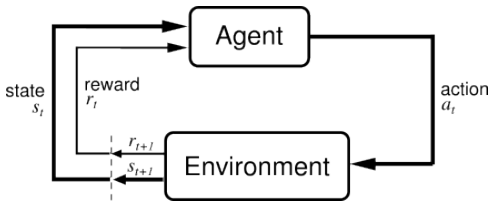

Figure 1: The canonical agent-environment interaction loop in Reinforcement Learning. At each timestep t, the agent executes an action a_t based on the state s_t, and the environment responds with a new state s_{t+1} and a scalar reward r_{t+1}.

Formally, an MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- **State Space ($\mathcal{S}$):** The set of all possible states the agent can be in. A state $s \in \mathcal{S}$ is a complete description of the environment at a specific moment.
- **Action Space ($\mathcal{A}$):** The set of all possible actions the agent can take. An action $a \in \mathcal{A}$ is the means by which the agent influences the environment.
- **Transition Function (P):** The dynamics of the environment, defining the probability of transitioning to state s' after taking action a in state s.

$$P(s'|s, a) = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a] \tag{1}$$

- **Reward Function (R):** A scalar signal that defines the goal. The reward r_t indicates how good it is for the agent to be in a certain state or to have performed a certain action.
- **Discount Factor (γ):** A value $\gamma \in [0, 1]$ that determines the present value of future rewards. A value of $\gamma = 0$ leads to a myopic agent concerned only with immediate rewards, while a value approaching 1 signifies a farsighted agent.

This entire framework rests on a crucial simplifying assumption: the **Markov Property**. This property asserts that the future is independent of the past, given the present. In other words, the state s_t must capture all relevant information from the history of interactions, making the transition probabilities dependent only on the current state and action, not the entire sequence that preceded them.

The power of the MDP lies in its generality. It provides a "lingua franca" that allows us to apply the same core algorithms to a vast array of problems, from mastering strategic board games to controlling complex robotic systems.

2.2 The Objective: Defining and Finding an Optimal Policy

The goal of an RL agent is to learn a **policy**, denoted π, which is a mapping from states to a probability distribution over actions, $\pi(a|s) = \mathbb{P}[a_t = a|s_t = s]$. The policy represents the agent's behaviour or strategy. The objective is to find a policy that maximizes the cumulative future reward. This cumulative reward is known as the **return**, G_t, defined as the discounted sum of all future rewards from timestep t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

Since the environment can be stochastic, the agent aims to maximize the *expected* return. To evaluate the "goodness" of a policy, we define **value functions**.

- The **state-value function**, $V^\pi(s)$, is the expected return when starting in state s and following policy π thereafter.
$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{3}$$
- The **action-value function**, $Q^\pi(s,a)$, is the expected return after taking action a in state s and subsequently following policy π. This is often more useful for action selection, as it quantifies the value of each specific action.
$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{4}$$

An **optimal policy**, denoted π^*, is a policy that achieves a higher or equal expected return than any other policy from any state. The fundamental property connecting the value of a state to the value of its successor states is captured by the **Bellman equations**. The Bellman expectation equation for the state-value function expresses this relationship recursively:

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} P(s'|s,a)\left[R(s,a,s') + \gamma V^\pi(s')\right] \tag{5}$$

This equation forms the theoretical bedrock for nearly all reinforcement learning algorithms, as it provides a way to iteratively compute and learn the value of states and actions.

2.3 Two Paths to Optimization: Value vs. Policy-Based Methods

With the objective defined, the central challenge of RL becomes finding an algorithm to discover the optimal policy π^*. Historically, the field evolved along two distinct philosophical paths: learning the *value* of actions versus learning the *policy* directly. Understanding this divergence is critical for appreciating the trade-offs that led to the development of the robust, general-purpose algorithms used in modern applications.

2.3.1 Path 1: Learning the Value of Actions

Value-based methods focus on learning the optimal action-value function, $Q^*(s,a)$. The core idea is that if an agent knows the true expected return for taking any action in any state, the optimal policy becomes implicit: simply choose the action with the highest Q-value. This is known as acting greedily with respect to Q^*.

One of the most foundational algorithms in this family is **Q-Learning**, a temporal-difference (TD) method that iteratively updates its estimate of $Q(s,a)$ using the Bellman optimality equation. The breakthrough for modern deep RL came with the development of **Deep Q-Networks (DQN)** [4], which replaced the tabular representation of Q-values with a deep neural network. By using techniques like experience replay and target networks, DQN achieved superhuman performance on a suite of Atari games, demonstrating that value-based methods could be scaled to high-dimensional state spaces.

Despite this success, this path proved too restrictive to serve as a universal learning mechanism. The core limitation is baked into its decision-making process: to select an action, the agent must compute $\arg\max_a Q(s,a)$, which requires iterating through every possible action to find the one with the highest estimated value. This approach is practical for small, discrete action spaces but fails to scale. For agents acting in continuous spaces (e.g., controlling robotic limbs) or enormous discrete spaces (e.g., strategic resource allocation or language generation), this exhaustive search is computationally infeasible, which motivated the development of an alternative approach to policy representation.

2.3.2 Path 2: Learning the Policy Directly

The alternative path is to directly parameterize the policy with a function approximator, such as a neural network, denoted $\pi_\theta(a|s)$. Instead of learning a value function, the goal is to directly optimise the policy parameters θ by performing gradient ascent on the expected return objective, $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_t]$. The Policy Gradient Theorem provides a theoretical foundation for this, yielding an expression for the gradient of the objective:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[G_t \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{6}$$

The most direct implementation of this theorem is the **REINFORCE** algorithm [5]. It operates by running full trajectories (episodes) under the current policy, and for each action taken, it updates the

policy parameters in the direction that would make that action more likely, scaled by the total return G_t of the episode. Actions in high-return episodes are "reinforced," while actions in low-return episodes are "discouraged."

The primary advantage of policy-based methods is their direct applicability to continuous or enormous action spaces. However, their simplest form, REINFORCE, suffers from a major practical limitation: the gradient estimate is extremely noisy. Because the return G_t is a sample from only one trajectory, it can have very high variance, leading to unstable training and slow convergence.

2.4 Bridging the Paths: Actor-Critic Architectures

To address the high variance of pure policy gradient methods, a hybrid approach known as **Actor-Critic** was developed. This architecture elegantly synthesizes the two paths by learning both a policy and a state-value function simultaneously.

- The **Actor** is the policy, $\pi_\theta(a|s)$, which controls how the agent behaves.
- The **Critic** is the state-value function, $V_w(s)$, which evaluates the state by estimating the expected return. Its parameters are denoted by w.

The critic provides a more stable, lower-variance estimate of the return than a single Monte Carlo sample. Instead of scaling the policy gradient by the full return G_t, we can use a more nuanced signal: the **Advantage function**, $A(s, a)$. The advantage quantifies how much better an action is compared to the average action in that state:

$$A(s, a) = Q(s, a) - V(s) \tag{7}$$

In practice, since $Q(s, a)$ can be expressed as $r + \gamma V(s')$, the advantage can be estimated using the TD error from the critic: $A(s_t, a_t) \approx r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$. This TD error serves as the learning signal for both the actor and the critic. The actor updates its policy in the direction suggested by the critic's evaluation, leading to a much more stable gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[A(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{8}$$

By subtracting the state-value baseline, the actor-critic framework significantly reduces the variance of the gradient estimate, enabling more efficient and reliable learning.

2.5 The Modern Workhorse: Proximal Policy Optimization

While actor-critic methods solve the variance problem, they can still be sensitive to the step size. A single bad update with a large gradient can lead to a "policy collapse," where the agent adopts a terrible strategy from which it cannot recover. This is particularly dangerous when training massive, expensive models.

Proximal Policy Optimization (PPO) [6] is a family of policy gradient algorithms designed to address this instability. Its key innovation is to discourage large, potentially destructive policy updates by constraining how much the policy can change at each step. PPO's most common variant achieves this through a **clipped surrogate objective function**.

Let $r_t(\theta)$ be the probability ratio between the new and old policies: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. The PPO-Clip objective is:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t \right) \right] \tag{9}$$

Here, $\hat{A}_t$ is the estimated advantage and ϵ is a small hyperparameter (e.g., 0.2) that defines the clipping range. The objective takes the minimum of two terms:

1. The standard policy gradient objective, $r_t(\theta)\hat{A}_t$.
2. A clipped version, where the probability ratio $r_t(\theta)$ is prevented from moving outside the range $[1 - \epsilon, 1 + \epsilon]$.

This clipping mechanism acts as a pessimistic bound on the policy update. If an action has a large positive advantage, the objective prevents the policy from becoming too confident in that action in a single step. Conversely, if an action has a large negative advantage, it limits how much the policy can

be discouraged. This simple but powerful idea ensures more stable and monotonic improvements during training.

PPO has thus become a canonical algorithm: a robust and reliable method for policy improvement within a well-defined problem.

This framing naturally leads to a crucial question. PPO and other single-agent methods excel when the environment is static. But what happens when the environment itself is adapting? This occurs in competitive games, economic markets, and cooperative teams, where the "environment" is composed of other learning agents. This question marks a fundamental shift from the solitary world of the MDP to the complex, social dynamics of multi-agent systems, where it is not just about setting up a problem, but about managing the interactions between learners.

2.6 The Limit of the Solitary Learner

The algorithms discussed so far—from Q-learning to PPO—are all built around the same basic assumption: the agent interacts with a **stationary** environment. In an MDP, the transition function $P(s' \mid s, a)$ and reward function $R(s, a)$ do not change over time. The problem is hard, but conceptually simple: given fixed dynamics, find a policy that performs well. Once an optimal policy is found, it remains optimal as long as the environment stays the same.

This picture breaks as soon as we introduce other learning agents. The environment is no longer a passive system but a collection of decision-makers, each updating their policy. When agent j changes its policy π_j, it changes the effective dynamics and rewards experienced by agent i. From i's perspective, the environment is now *non-stationary* [7]: the same state–action pair can lead to different next states and returns over the course of training.

Two practical consequences are particularly important:

1. **Loss of the Markov property (from an individual agent's view).** The current state s_t is no longer a sufficient summary of the future, because the outcome also depends on the (hidden) policies of the other agents, $\boldsymbol{\pi}_{-i,t}$. An action that used to work well in a given state may stop working once others adapt.
2. **Stale experience.** Many deep RL methods rely on replay buffers of past transitions (s, a, r, s') to stabilise learning. In a non-stationary multi-agent setting, these transitions reflect outdated versions of the other agents' policies. Training on such data can push the learner towards policies that are no longer appropriate, leading to oscillations or divergence.

With multiple learners, the problem is no longer "what is the best action in this state?" but rather "what is the best action, given how others are likely to behave, and how they will respond to my behaviour?". This kind of strategic coupling lies outside the standard MDP framework. To reason about it, we need a language that explicitly models interacting decision-makers and their objectives. Game theory provides that language, and will be the starting point for the multi-agent analysis in the next chapter.

3 Game Theory: The Mathematics of Strategic Interaction

The previous chapter treated reinforcement learning in the single-agent setting: one learner interacting with a fixed environment modelled as an MDP. As argued in Section 2.6, this picture breaks down once outcomes depend on the decisions of multiple agents who may also be learning and adapting. In that case, the environment is no longer a passive process but a collection of strategic actors.

Game theory provides the standard language for this situation. Instead of asking how a single agent should act against fixed dynamics, it asks how several decision-makers with possibly conflicting goals will behave when their payoffs depend on one another. In the rest of this chapter we introduce the basic objects of game theory and the solution concepts that later chapters will rely on.

3.1 Why Game Theory?

Game theory is the study of conflict and cooperation between rational decision-makers. In the context of MARL it plays a complementary role to reinforcement learning:

- RL specifies *how* an agent updates its behaviour from experience, typically by following a gradient or a value-improvement rule.
- Game theory specifies *what* problem the agents are solving: how their joint actions map to payoffs, and what it means for an outcome to be stable.

This is useful for at least three reasons.

- **Objectives.** In single-agent RL, optimality means maximising expected return. In multi-agent settings there are several competing notions of optimality: individual payoff, joint payoff, robustness to deviations, and so on. Game theory supplies formal solution concepts such as Nash equilibrium that make these notions precise.
- **Structure.** By representing an interaction as a game, we can classify it: purely competitive (zero-sum), fully cooperative (common-payoff), or mixed-motive (general-sum). We can also identify social dilemmas, where individually rational behaviour leads to poor group outcomes. This structural view helps explain why some learning dynamics succeed and others fail.
- **Design.** When we have control over the rules of interaction, game theory lets us reason about how changing payoffs, information, or available actions will change the equilibria. This underpins much of the "mechanism design for cooperation" perspective in Section 6.

In the remainder of this chapter we move from static, matrix games to their dynamic counterparts, and from there to population-level viewpoints via evolutionary game theory. These tools will be used repeatedly when analysing multi-agent learning algorithms and when evaluating mechanisms for cooperation.

3.2 Static Games: Normal-Form and Solution Concepts

We start with the simplest setting: a one-shot interaction in which all players choose their actions simultaneously, without observing the others' choices. This is captured by the normal-form (or strategic-form) game. We then introduce the core solution concepts—best response and Nash equilibrium—and use simple 2×2 games as running examples. These examples will reappear throughout the thesis as canonical models of coordination, social dilemma, and pure conflict.

3.2.1 Normal-form games

A **normal-form game** is a tuple

$$G = \big(\mathcal{N}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \{u_i\}_{i\in\mathcal{N}}\big),$$

where:

- $\mathcal{N} = \{1, \dots, N\}$ is the finite set of *players* (or agents).
- $\mathcal{A}_i$ is the finite set of *actions* (or *pure strategies*) available to player i. The joint action space is the Cartesian product $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$. A joint action profile is written $\mathbf{a} = (a_1, \dots, a_N)$ with $a_i \in \mathcal{A}_i$.

- $u_i : \mathcal{A} \to \mathbb{R}$ is the *utility function* (or payoff function) for player i, mapping each joint action profile to a real-valued payoff.

The central feature is that each player's payoff $u_i(\mathbf{a})$ depends on the *joint* action of all players, not just their own choice. This captures strategic interdependence: what is good for me depends on what you do.

A *pure strategy* is a deterministic choice of action. More generally, player i can randomise between actions. A **mixed strategy** for player i is a probability distribution $\pi_i \in \Delta(\mathcal{A}_i)$ over pure actions, where $\Delta(\mathcal{A}_i)$ denotes the simplex of probability distributions on $\mathcal{A}_i$. We write Π_i for the set of mixed strategies of player i, and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$ for a joint strategy profile. Expected utilities under mixed strategies are defined in the natural way:

$$\mathbb{E}_{\mathbf{a}\sim\boldsymbol{\pi}}[u_i(\mathbf{a})] = \sum_{\mathbf{a}\in\mathcal{A}} \left(\prod_{j\in\mathcal{N}} \pi_j(a_j) \right) u_i(\mathbf{a}).$$

In MARL terms, a mixed strategy is the static analogue of a stochastic policy: a mapping from "state" (here trivial, because the game is one-shot) to a distribution over actions.

3.2.2 Best response and Nash equilibrium

Given a fixed strategy profile for the other players, a rational agent will prefer strategies that maximise their expected payoff. This leads to the notion of a best response.

Let $\boldsymbol{\pi}_{-i}$ denote the strategies of all players except i. A strategy $\pi_i^\star \in \Pi_i$ is a **best response** to $\boldsymbol{\pi}_{-i}$ if it achieves at least as much expected utility as any alternative strategy:

$$\mathbb{E}_{\mathbf{a}\sim(\pi_i^\star,\boldsymbol{\pi}_{-i})}[u_i(\mathbf{a})] \;\geq\; \mathbb{E}_{\mathbf{a}\sim(\pi_i',\boldsymbol{\pi}_{-i})}[u_i(\mathbf{a})] \quad \forall \pi_i' \in \Pi_i. \tag{10}$$

In some games, a player may have a strategy that is a best response *no matter what the others do*. A strategy $\pi_i^{\mathrm{dom}} \in \Pi_i$ is a **dominant strategy** for player i if, for every joint strategy profile of the others,

$$\mathbb{E}_{\mathbf{a}\sim(\pi_i^{\mathrm{dom}},\boldsymbol{\pi}_{-i})}[u_i(\mathbf{a})] \;\geq\; \mathbb{E}_{\mathbf{a}\sim(\pi_i',\boldsymbol{\pi}_{-i})}[u_i(\mathbf{a})] \quad \forall \pi_i' \in \Pi_i,\ \forall \boldsymbol{\pi}_{-i}. \tag{11}$$

Dominant strategies are rare but extremely simple to reason about: if all players have dominant strategies, we can predict the outcome by having each play their dominant action.

Most interesting games do not admit dominant strategies. Instead, each player's best response depends on what others do. A **Nash equilibrium** (NE) [8] is a joint strategy profile in which every player is best responding to the others.

Definition 1 (Nash equilibrium)**.** *A mixed-strategy profile $\boldsymbol{\pi}^\star = (\pi_1^\star, \ldots, \pi_N^\star)$ is a Nash equilibrium if, for every player $i \in \mathcal{N}$,*

$$\pi_i^\star \in \arg\max_{\pi_i\in\Pi_i} \mathbb{E}_{\mathbf{a}\sim(\pi_i,\boldsymbol{\pi}_{-i}^\star)}[u_i(\mathbf{a})]. \tag{12}$$

Equivalently, no player can improve their expected payoff by unilaterally deviating from $\boldsymbol{\pi}^\star$:

$$\mathbb{E}_{\mathbf{a}\sim(\pi_i^\star,\boldsymbol{\pi}_{-i}^\star)}[u_i(\mathbf{a})] \;\geq\; \mathbb{E}_{\mathbf{a}\sim(\pi_i',\boldsymbol{\pi}_{-i}^\star)}[u_i(\mathbf{a})] \quad \forall\, \pi_i' \in \Pi_i. \tag{13}$$

Intuitively, a Nash equilibrium is a point of *mutual best response*. If everyone believes that the others will continue to play their equilibrium strategies, no one has an incentive to change their own. Nash's classical theorem guarantees that every finite normal-form game has at least one mixed-strategy NE [8].

In MARL, Nash equilibrium provides a natural notion of a *stable* joint policy: a profile $\boldsymbol{\pi}^\star$ such that no single agent can gain by switching its policy while others keep theirs fixed. Many algorithms—for example, variants of independent Q-learning in two-player zero-sum games—can be interpreted as attempting to approximate best responses and thus move toward a Nash equilibrium of the underlying game.

Limitations of Nash as a learning target. Although Nash equilibrium is the central solution concept in non-cooperative game theory, it is only a partial guide for multi-agent learning. First, Nash equilibrium is a *static* concept: it makes no claim about whether simple learning dynamics will converge to an equilibrium, or how long this might take. Second, many games admit multiple Nash equilibria, some of which are more socially desirable than others. The theory does not specify how agents should *select* among them. Third, equilibria can be Pareto-inefficient, as we will see in the Prisoner's Dilemma, where the unique NE leaves all players worse off than some alternative joint strategy.

For MARL, it is therefore more accurate to view Nash equilibrium as a benchmark: if learning converges, we can ask whether the limiting joint policy is an equilibrium and how efficient it is. Later we will use evolutionary game theory and empirical results to study the dynamics of learning and the emergence (or failure) of cooperative equilibria.

3.2.3 A taxonomy of two-player normal-form games

Before turning to concrete examples, it is useful to situate the kinds of games that appear in MARL. For simplicity, consider two-player games with utility functions u_1 and u_2.

- **Zero-sum games.** The game is *zero-sum* if $u_1(\mathbf{a}) + u_2(\mathbf{a})$ is constant for all joint actions $\mathbf{a}$. One player's gain is exactly the other's loss. These games model pure competition and are the setting for minimax reasoning and adversarial self-play (e.g., two-player board games).
- **Common-payoff (team) games.** The game is *common-payoff* if $u_1(\mathbf{a}) = u_2(\mathbf{a})$ for all $\mathbf{a}$. All agents are fully aligned: any strategy profile that is good for one is equally good for the other. Many cooperative MARL benchmarks (e.g., teams trying to maximise a shared score) can be modelled as common-payoff games.
- **General-sum (mixed-motive) games.** In a *general-sum* game, neither of the above conditions holds. Players have partly aligned and partly conflicting interests. This class includes coordination games, social dilemmas, and many economic and social interactions. It is the most relevant regime for this thesis: it is only in general-sum games that the tension between individual and collective incentives becomes non-trivial.

In subsequent sections we will mostly focus on general-sum games that exhibit social dilemmas, where individually rational behaviour can lead to collectively poor outcomes. These will serve as the basic testbeds for studying cooperation in MARL.

3.2.4 Canonical 2x2 games

To ground the abstract definitions, we briefly review three canonical two-player, two-action games that capture different interaction patterns. They will reappear throughout the thesis as simple models of coordination, social dilemma, and pure conflict.

Stag Hunt: a coordination game. The Stag Hunt illustrates the tension between the safety of individual action and the potential gains from mutual cooperation. Two hunters can each choose to hunt a hare or cooperate to hunt a stag. Hunting hare is safe and yields a modest payoff regardless of the other's action; hunting stag yields a higher payoff but only if both choose it.

		Hunter 2	
		Stag	Hare
Hunter 1	Stag	**(4,4)**†	(0,3)
	Hare	(3,0)	**(3,3)**

Table 1: The Stag Hunt game. Payoffs are (Hunter 1, Hunter 2). **Bold** outcomes are Nash equilibria; † marks the Pareto-efficient outcome.

There are two pure-strategy Nash equilibria: (Stag, Stag) and (Hare, Hare). The former is Pareto-optimal, but risky: if a hunter is unsure whether the other will cooperate, choosing Hare is the safer best response. The Stag Hunt captures problems of trust and coordination, which are common in cooperative MARL tasks.

Prisoner's Dilemma: a social dilemma. The Prisoner's Dilemma is the standard model of a social dilemma: individual rationality and collective welfare point in different directions. Two prisoners can either Cooperate (stay silent) or Defect (betray the other). A typical payoff matrix is:

		Prisoner 2	
		Cooperate	Defect
Prisoner 1	Cooperate	(3,3)†	(0,4)
	Defect	(4,0)	**(1,1)**

Table 2: The Prisoner's Dilemma. **Bold** marks the unique Nash equilibrium; † marks the Pareto-efficient joint outcome.

For each player, Defect strictly dominates Cooperate: regardless of what the other does, Defect yields a higher payoff. The unique Nash equilibrium is therefore (Defect, Defect) with payoffs (1,1). Yet both players would be better off at (Cooperate, Cooperate), which is Pareto-superior. This gap between equilibrium and efficiency is the core challenge of cooperation in social dilemmas, and will be a recurring theme in Section 6.

Matching Pennies: pure conflict. Matching Pennies is a simple zero-sum game. Two players each secretly choose Heads or Tails and reveal simultaneously. If the coins match, Player 1 wins; if they differ, Player 2 wins.

		Player 2	
		Heads	Tails
Player 1	Heads	$(1, -1)$	$(-1, 1)$
	Tails	$(-1, 1)$	$(1, -1)$

Table 3: Matching Pennies. The payoffs sum to zero in every outcome. There is no pure-strategy Nash equilibrium.

There is no pure-strategy Nash equilibrium: for any deterministic choice by both players, one of them can improve by switching. The unique mixed-strategy NE has each player randomise uniformly between Heads and Tails. Matching Pennies therefore highlights the need for stochastic policies in competitive settings.

Together, these three games provide compact examples of the main interaction types that appear in MARL: coordination problems with multiple equilibria (Stag Hunt), social dilemmas with inefficient equilibria (Prisoner's Dilemma), and pure competition (Matching Pennies). In the next section we extend this static framework to repeated and dynamic games, which will bring us closer to the sequential decision problems studied in reinforcement learning.

3.3 Repeated Games and the Possibility of Cooperation

The static games in Section 3.2 describe one-shot interactions: players move once, simultaneously, and the game ends. Many real interactions, however, are repeated: agents meet again and again, observe some information about past behaviour, and can choose whether to reward or punish one another. This repeated structure is crucial for understanding how cooperation can emerge even when the underlying one-shot incentives favour defection.

3.3.1 Stage games and repeated play

Let $G = (\mathcal{N}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \{u_i\}_{i\in\mathcal{N}})$ be a normal-form *stage game*. A **repeated game** is formed by playing this stage game multiple times in sequence.

In a **finite-horizon** repeated game, the stage game is played for T rounds. In each round $t \in \{1, \dots, T\}$, every player simultaneously chooses an action $a_{i,t} \in \mathcal{A}_i$, the joint action $\mathbf{a}_t = (a_{1,t}, \dots, a_{N,t})$ is realised, and each player receives a stage payoff $u_i(\mathbf{a}_t)$. The history up to round t is

$$h_t = (\mathbf{a}_1, \dots, \mathbf{a}_{t-1}),$$

and a (pure) strategy for player i in the repeated game is a mapping

$$\sigma_i : \{\text{histories}\} \to \mathcal{A}_i,$$

assigning an action to every possible past history. Mixed strategies randomise over such mappings. The total payoff is typically taken as the sum (or average) of the stage payoffs.

In an **infinitely repeated** game, the stage game is played for $t = 1, 2, \ldots$ without a fixed end. The total payoff is then defined in terms of a discounted sum,

$$U_i(\boldsymbol{\sigma}) \;=\; (1-\delta)\sum_{t=1}^{\infty} \delta^{t-1}\, \mathbb{E}\big[u_i(\mathbf{a}_t)\big], \tag{14}$$

where $\delta \in (0,1)$ is a discount factor and the expectation is with respect to the (possibly stochastic) strategies $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_N)$. The factor $(1-\delta)$ is conventional and ensures that payoffs remain bounded as $\delta \to 1$.

In this setting, strategies can condition on the entire history of play, not just the current stage game. This allows for rich contingent behaviour: agents can cooperate as long as others do, and switch to punishment if they observe defection. As we will see, this added temporal structure changes the set of equilibria dramatically.

3.3.2 Repeated Prisoner's Dilemma

The Prisoner's Dilemma from Section 3.2.4 has a unique Nash equilibrium in the one-shot case: both players defect. If the game is repeated a known finite number of times T, backward induction shows that the unique subgame-perfect equilibrium is still to defect in every round. In the final round, there is no future left in which to reward or punish, so both defect; anticipating this, both also defect in the penultimate round, and so on.

The situation changes in the infinitely repeated game, or in long-horizon games where the end is uncertain. Consider the following *grim trigger* strategy for each player:

- Start by cooperating.
- Continue to cooperate as long as the other player has cooperated in all previous rounds.
- If the other ever defects, switch to defect forever.

Against itself, grim trigger yields mutual cooperation in every round, and hence a high discounted payoff for both players. Against a player who deviates once, it responds with permanent defection, making deviation costly in the long run.

For discount factors δ close enough to 1, the threat of future punishment can outweigh the short-term gain from a one-off defection. In that regime, the strategy profile in which both players use grim trigger can form a Nash equilibrium of the infinitely repeated game: no single player can unilaterally improve their discounted payoff by deviating. Intuitively, cooperation becomes individually rational when future interactions are sufficiently valuable.

Grim trigger is extreme in its punishment. More forgiving strategies, such as tit-for-tat (cooperate on the first move, then copy the opponent's last action), can also sustain cooperation under appropriate conditions. The key point is that repeated interaction and memory allow players to implement *reciprocal* behaviour: reward cooperators, punish defectors.

3.3.3 Informal folk theorem and MARL interpretation

A large body of work in repeated games is summarised by the so-called *folk theorems*. Roughly speaking, they state that in infinitely repeated general-sum games with sufficiently patient players (large δ), a very wide range of payoff profiles can be sustained as equilibria. In particular, any payoff vector that is:

- *feasible*: achievable as the average payoff of some joint mixed strategy in the stage game, and
- *individually rational*: giving each player at least as much as they could guarantee themselves by playing a safe minimax strategy,

can be supported by some equilibrium of the repeated game under suitable conditions. The exact statements and proofs are beyond the scope of this thesis, but the qualitative message is simple and important:

> When agents interact repeatedly and care enough about the future, long-run cooperative outcomes can be sustained even if one-shot incentives favour defection.

For MARL, repeated games are a bridge between static normal-form games and fully dynamic Markov games. An infinitely repeated Prisoner's Dilemma can be viewed as a stochastic game where the "state" encodes the history of play so far. Policies that condition on past observations—for example, implemented via recurrent neural networks—can approximate strategies like grim trigger or tit-for-tat. Later, in Section 6, we will see how mechanisms based on reciprocity, reputation, and partner choice draw directly on this repeated-game intuition to promote cooperation between learning agents.

3.4 Dynamic Multi-Agent Worlds: Markov Games and POSGs

Normal-form and repeated games treat each stage of interaction as a single, simultaneous move. Many environments of interest in MARL are more structured: they have states, long-horizon dynamics, and possibly stochastic transitions driven by agents' joint actions. The standard model for such settings is the *stochastic game*, also known as a Markov game [9]. This can be seen as a multi-agent generalisation of the Markov Decision Process from Section 2.1.

We first introduce Markov games and their equilibrium concept, then extend to partially observable stochastic games, which are closer to realistic multi-agent environments.

3.4.1 Stochastic (Markov) games

A **stochastic game** (or Markov game) is a tuple

$$\mathcal{G} = \big(\mathcal{S}, \mathcal{N}, \{\mathcal{A}_i\}_{i=1}^N, P, \{R_i\}_{i=1}^N, \gamma\big),$$

where:

- $\mathcal{S}$ is a finite or measurable set of *states*.
- $\mathcal{N} = \{1, \dots, N\}$ is the set of agents.
- $\mathcal{A}_i$ is the action space of agent i, with joint action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, where
$$P(s' \mid s, \mathbf{a}) = \mathbb{P}[s_{t+1} = s' \mid s_t = s, \mathbf{a}_t = \mathbf{a}]$$
gives the probability of moving to state s' from state s when the joint action $\mathbf{a}$ is taken.
- $R_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for agent i, specifying its instantaneous payoff at each transition.
- $\gamma \in [0, 1)$ is a discount factor shared by all agents.

At each timestep t the system is in some state $s_t \in \mathcal{S}$. Each agent i observes the current state (for now we assume full observability), selects an action $a_{i,t} \in \mathcal{A}_i$, and the resulting joint action $\mathbf{a}_t$ induces both a reward $R_i(s_t, \mathbf{a}_t)$ for each agent and a transition to a new state $s_{t+1} \sim P(\cdot \mid s_t, \mathbf{a}_t)$.

A (possibly stochastic) **Markov policy** for agent i is a mapping

$$\pi_i : \mathcal{S} \to \Delta(\mathcal{A}_i),$$

assigning to each state a distribution over actions. A joint policy is $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$, and together with a transition kernel P this induces a Markov chain over states.

Given a joint policy $\boldsymbol{\pi}$ and an initial state distribution ρ_0, the value function for agent i is

$$V_i^{\boldsymbol{\pi}}(s) \;=\; \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t, \mathbf{a}_t) \;\middle|\; s_0 = s,\, \mathbf{a}_t \sim \boldsymbol{\pi}(s_t),\, s_{t+1} \sim P(\cdot \mid s_t, \mathbf{a}_t)\right]. \tag{15}$$

The corresponding action-value function $Q_i^{\boldsymbol{\pi}}(s, \mathbf{a})$ is defined analogously.

Stochastic games reduce to familiar models in special cases:

- If $N = 1$, we recover a standard MDP.
- If $|\mathcal{S}| = 1$ and transitions are trivial, we recover a static normal-form game.
- If the stage game is fixed and the state encodes the history of play (as in repeated games), we obtain a Markov representation of an infinitely repeated normal-form game.

3.4.2 Markov perfect equilibrium

In static games, Nash equilibrium describes a joint strategy profile where each player's strategy is a best response to the others. In Markov games, the analogous concept is a **Markov perfect equilibrium** (MPE). Informally, an MPE is a joint Markov policy profile $\boldsymbol{\pi}^\star$ such that, in every state, each agent's policy is a best response to the others' policies from that point onward.

Definition 2 (Markov perfect equilibrium)**.** *A joint Markov policy profile $\boldsymbol{\pi}^\star = (\pi_1^\star, \ldots, \pi_N^\star)$ is a Markov perfect equilibrium if, for each agent i and every state $s \in \mathcal{S}$,*

$$V_i^{(\pi_i^\star, \boldsymbol{\pi}_{-i}^\star)}(s) \;\geq\; V_i^{(\pi_i', \boldsymbol{\pi}_{-i}^\star)}(s) \quad \forall \pi_i' : \mathcal{S} \to \Delta(\mathcal{A}_i), \tag{16}$$

where $V_i^{(\pi_i, \boldsymbol{\pi}_{-i})}$ is the value function defined in (15) *under the joint policy $(\pi_i, \boldsymbol{\pi}_{-i})$.*

An MPE is thus a profile of stationary policies such that no agent can improve its expected discounted return, starting from any state, by unilaterally switching to another Markov policy. In two-player zero-sum Markov games, this reduces to a dynamic minimax equilibrium; in general-sum settings, multiple MPE may exist and, as in static games, they can differ in efficiency.

From a MARL perspective, MPE formalises the idea of a *stable joint policy* in a dynamic world. A core question is whether particular learning dynamics—for example, independent Q-learning or policy-gradient methods with centralised critics—converge to any MPE, and if so, which ones.

3.4.3 Partial observability and POSGs

The Markov game framework assumes that all agents observe the true environment state s_t at each timestep. In many realistic settings this is not the case. Agents instead receive private observations that provide only partial information about the underlying state. This leads to **Partially Observable Stochastic Games** (POSGs) [10].

A POSG augments the Markov game definition with observation spaces and an observation function. Formally, a POSG is given by

$$\mathcal{G}^{\mathrm{PO}} = \big(\mathcal{S}, \mathcal{N}, \{\mathcal{A}_i\}_{i=1}^N, P, \{R_i\}_{i=1}^N, \{\mathcal{O}_i\}_{i=1}^N, O, \gamma\big),$$

where:

- $\mathcal{O}_i$ is the observation space of agent i.
- $O : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{O}_1 \times \cdots \times \mathcal{O}_N)$ is the observation function, which specifies the distribution over joint observations given the current state and joint action. At each timestep, after the transition to s_{t+1}, a joint observation $\mathbf{o}_{t+1} = (o_{1,t+1}, \ldots, o_{N,t+1})$ is drawn from $O(s_{t+1}, \mathbf{a}_t)$.

In a POSG, an agent's decision at time t cannot be based on the full state s_t, which it does not observe, but only on its *action–observation history*,

$$h_{i,t} = (o_{i,1}, a_{i,1}, \ldots, o_{i,t}).$$

A general policy for agent i is therefore a mapping

$$\pi_i : \{\text{histories } h_{i,t}\} \to \Delta(\mathcal{A}_i).$$

In practice, MARL algorithms often approximate such history-dependent policies with recurrent neural networks, which compress the history into a learned hidden state.

Partial observability introduces two major complications:

- **Belief and inference.** The underlying state is hidden and must be inferred from observations. Optimal decision-making would require maintaining a belief over $\mathcal{S}$, which is generally intractable in high-dimensional problems.

- **Coupled uncertainty.** Each agent's observation both reflects and influences the others' actions, so uncertainty about the state is intertwined with uncertainty about others' policies.

These difficulties are central in many MARL benchmarks, where agents must coordinate based on local views (for example, in cooperative navigation or partially observable gridworlds).

3.4.4 From game theory to MARL

The combination of Markov games and POSGs provides a formal language for the environments encountered in multi-agent reinforcement learning. A MARL problem can be viewed as:

- a Markov game (or POSG) specifying the dynamics, reward functions, and information structure, and
- a learning rule for each agent that updates its policy based on experience.

Single-agent RL algorithms, such as Q-learning or PPO, assume a stationary MDP and typically converge (under suitable conditions) to optimal policies. In a Markov game, by contrast, each agent is learning against other learners. From the perspective of any single agent, the effective transition dynamics and reward distributions change over time as the others update their policies, violating the stationarity assumptions of standard RL [7].

This tension between static solution concepts (MPE, Nash equilibrium) and dynamic learning processes is at the heart of MARL. The next chapter, Section 4, takes the formal game-theoretic models introduced here as given and asks how different learning architectures perform within them, and under what conditions they produce cooperative or non-cooperative outcomes.

3.5 Population Dynamics: Evolutionary Game Theory (EGT)

Classical game theory, with its focus on Nash equilibrium, takes a largely static and highly rational view of behaviour: players are assumed to be flawless logicians who instantly compute best responses given full knowledge of the game. This is useful for defining what a "stable" outcome should look like, but it leaves two important questions open:

- How might a population of simpler, adaptive agents *actually* arrive at such outcomes?
- What happens when agents do not perfectly optimise but instead adjust their behaviour gradually based on performance?

Evolutionary game theory (EGT) offers a complementary perspective [11]. Rather than modelling deliberate reasoning by individuals, it models the dynamics of a large population of agents whose "strategies" evolve over time according to their success. Strategies that perform well become more common; strategies that perform poorly die out. This viewpoint is particularly natural for MARL, where we often train populations of policies under some selection process.

3.5.1 Populations and fitness

Consider a single, large population of agents repeatedly matched to play a fixed, symmetric normal-form game such as the Prisoner's Dilemma or Stag Hunt from Section 3.2.4. Each agent is "programmed" with a pure strategy s_j from a finite strategy set $\mathcal{A}$. Let

$$\mathbf{x}(t) = (x_1(t), \ldots, x_k(t))$$

denote the population state at time t, where $x_j(t)$ is the proportion of agents using strategy s_j and $\sum_j x_j(t) = 1$. We can think of $\mathbf{x}$ as a mixed strategy played by the population as a whole.

Suppose the payoffs of the underlying game are given by a matrix U, where U_{ij} is the payoff to a player using strategy s_i against an opponent using s_j. When two agents are matched uniformly at random from the population, the expected payoff (or *fitness*) of strategy s_i against the current population mix $\mathbf{x}$ is

$$f_i(\mathbf{x}) = (U\mathbf{x})_i.$$

The average fitness in the population is then

$$\bar{f}(\mathbf{x}) = \mathbf{x}^\top U\mathbf{x} = \sum_i x_i f_i(\mathbf{x}).$$

The key idea is that strategies with above-average fitness should increase in frequency, while those with below-average fitness should decline. The simplest continuous-time model that captures this is the **replicator equation**.

3.5.2 The replicator dynamics

The **replicator dynamics** describe how the population state $\mathbf{x}(t)$ changes over time:

$$\dot{x}_i = x_i\big(f_i(\mathbf{x}) - \bar{f}(\mathbf{x})\big), \quad i = 1, \dots, k. \tag{17}$$

Here $\dot{x}_i$ denotes the time derivative of x_i, i.e., its rate of change.

The equation has a straightforward interpretation:

- If $f_i(\mathbf{x}) > \bar{f}(\mathbf{x})$, then $\dot{x}_i > 0$ and the share of strategy s_i increases.
- If $f_i(\mathbf{x}) < \bar{f}(\mathbf{x})$, then $\dot{x}_i < 0$ and the share of s_i decreases.
- If $f_i(\mathbf{x}) = \bar{f}(\mathbf{x})$, the share of s_i remains unchanged.

Fixed points of the replicator dynamics (population states $\mathbf{x}^\star$ with $\dot{x}_i = 0$ for all i) correspond to states in which all strategies present in the population have equal fitness. Every symmetric Nash equilibrium of the underlying game is a fixed point of (17), but not every fixed point is stable. This leads to a stronger notion of stability than Nash equilibrium: the *evolutionarily stable strategy*.

3.5.3 Evolutionarily stable strategies (ESS)

An **evolutionarily stable strategy** (ESS) is, informally, a strategy that cannot be invaded by a small group of mutants. Suppose the population is almost entirely playing some mixed strategy $\pi^\star$ (represented as a point on the simplex), and a small fraction ε of agents adopt an alternative strategy π'. An ESS is a strategy $\pi^\star$ such that, for sufficiently small ε, the incumbents using $\pi^\star$ obtain higher fitness in the mixed population than the mutants using π'.

Formally, in symmetric two-player games, a strategy $\pi^\star$ is an ESS if for all $\pi' \neq \pi^\star$,

1. $u(\pi^\star, \pi^\star) \geq u(\pi', \pi^\star)$, and
2. if $u(\pi^\star, \pi^\star) = u(\pi', \pi^\star)$, then $u(\pi^\star, \pi') > u(\pi', \pi')$,

where $u(\pi, \pi')$ is the expected payoff to a player using strategy π against an opponent using π'.

The first condition is exactly the symmetric Nash equilibrium condition: no mutant does better than the incumbent when facing the incumbent population. The second condition rules out "knife-edge" cases where a mutant ties the incumbent when playing against it but does better when playing against itself. Intuitively, an ESS is a Nash equilibrium that is also a *locally asymptotically stable* fixed point of the replicator dynamics: if the population is perturbed slightly away from $\pi^\star$, selection will push it back.

3.5.4 Canonical games revisited

The three 2×2 games from Section 3.2.4 illustrate how different payoff structures lead to different evolutionary dynamics.

Prisoner's Dilemma. In the Prisoner's Dilemma, Defect strictly dominates Cooperate. Under the replicator dynamics, any initial population with a non-zero fraction of defectors will evolve toward a state where all agents defect. Defect is the unique ESS. This formalises the intuition that, without additional mechanisms, cooperation is evolutionarily unstable in a one-shot PD: a small group of defectors can always invade a cooperative population and ultimately take over.

Stag Hunt. In the Stag Hunt, both pure strategies (always hunt stag, always hunt hare) are symmetric Nash equilibria. Under the replicator dynamics, both corresponding pure population states are asymptotically stable: nearby trajectories are attracted to them. There is also an interior fixed point corresponding to the mixed-strategy Nash equilibrium, but it is unstable: small perturbations push the population toward one of the two pure ESS.

The basins of attraction of these two stable points depend on the initial population mix. If the initial fraction of stag hunters is above a certain threshold, the population drifts toward full cooperation (all Stag); below that threshold, it converges to the risk-dominant equilibrium (all Hare). This captures a key idea for cooperation in MARL: good outcomes may exist, but whether they are reached depends on the initial conditions and the learning dynamics.

Matching Pennies. In Matching Pennies, the unique Nash equilibrium is a mixed strategy in which both players randomise uniformly between Heads and Tails. Under the replicator dynamics, this strategy is a fixed point, but not an attractor: trajectories typically orbit around it in closed cycles rather than converging. There is no ESS. In purely competitive interactions of this kind, we should not expect simple adaptive dynamics to settle down to a static joint policy; instead, behaviour may keep changing over time.

3.5.5 From evolutionary dynamics to multi-agent learning

At first glance, EGT describes biological evolution, not learning. However, there is a close connection between replicator dynamics and the behaviour of certain learning rules in games. In particular:

- If a large population of agents each run simple no-regret or policy-gradient-like updates against each other, the aggregate evolution of strategy frequencies often approximates replicator dynamics [12, 13].
- Population-based training regimes in MARL, where policies are periodically sampled, evaluated, and selected based on performance, implement an explicit selection mechanism that is well described by evolutionary ideas.

This perspective is especially useful when thinking about cooperation:

- It shifts attention from "Does a cooperative equilibrium exist?" (a static question) to "Is cooperation dynamically *stable* under plausible adaptation rules?"
- It highlights the role of mechanisms that change the *fitness landscape*: modifying payoffs, interaction structure, or population mixing can turn cooperation from an unstable curiosity into a robust outcome.

In later chapters, we will make this connection concrete by viewing self-play and population-based MARL methods as implementing approximate evolutionary dynamics over policy space. The mechanisms for engineering cooperation developed in Section 6 can then be interpreted as ways of reshaping these dynamics so that cooperative strategies are not only possible, but stable under learning and selection.

4 Multi-Agent Reinforcement Learning

So far we have developed two strands of theory. Chapter 2 described how a single agent can learn to act in a fixed MDP. Chapter 3 introduced game-theoretic models of strategic interaction between multiple players. Multi-agent reinforcement learning (MARL) sits at their intersection: it studies how to apply RL algorithms in environments where multiple learning agents interact.

In this setting, each agent's reward depends not only on its own actions and the environment state, but also on the actions of others. As a result, the learning problem for each agent is coupled to the learning problems of the rest. This coupling is the source of both the main difficulties and the main interest in MARL.

4.1 The Synthesis: Learning in a Strategic World

Formally, MARL is RL in a stochastic game. Each agent i seeks a policy π_i that performs well given the policies of the others, $\boldsymbol{\pi}_{-i}$. Rather than a single "optimal" policy, the natural target is a joint policy profile $\boldsymbol{\pi}^* = (\pi_1^*, \ldots, \pi_N^*)$ that forms an equilibrium, for example a Markov perfect equilibrium in the sense of Section 3.

This changes the nature of the control problem. Single-agent RL solves

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}, \tag{18}$$

against fixed dynamics. In MARL each agent instead faces

$$V_i^{(\pi_i^*, \boldsymbol{\pi}_{-i}^*)}(s) \geq V_i^{(\pi_i', \boldsymbol{\pi}_{-i}^*)}(s) \quad \forall \pi_i' \in \Pi_i,\ \forall s \in \mathcal{S}, \tag{19}$$

so the goal is to find a joint solution rather than a single maximiser.

Simply running standard single-agent algorithms independently for each agent often fails in this setting. The rest of this section unpacks why: non-stationarity, partial observability, and credit assignment create new failure modes that do not appear in the solitary case, and motivate the MARL-specific architectures described later.

4.2 Core Challenges in Multi-Agent Learning

While the Stochastic Game provides the formal framework, successfully applying learning algorithms within it requires overcoming several inherent and deeply intertwined challenges. These problems distinguish MARL as a field distinct from its single-agent counterpart and motivate the development of specialized architectures and learning paradigms.

4.2.1 Non-stationarity: The Moving-Target Problem

As argued in Section 2.6, once other agents are learning, the effective environment seen by any one agent becomes non-stationary. The transition from a single-agent to a multi-agent setting introduces a fundamental problem that violates the core assumptions of many traditional RL algorithms: **non-stationarity** [7]. From the perspective of any single agent, the environment is no longer stationary because the other agents are simultaneously learning and adapting their policies.

In an MDP, the transition and reward dynamics, $P(s'|s,a)$ and $R(s,a)$, are fixed. The agent's policy learns to optimise its behaviour against a static background. In a stochastic game, however, the policy of agent i, $\pi_i(a_i|s)$, is learning while the policies of all other agents, $\boldsymbol{\pi}_{-i}$, are also changing. This creates a "moving-target" learning problem. An action that is optimal in the current iteration may become suboptimal in the next as other agents adapt their strategies in response.

This dynamic violates the Markov assumption for any individual agent. The transition probability, from agent i's limited perspective, appears to be in constant flux. The probability of reaching state s' from state s by taking action a_i is not a fixed function, but an integral over the changing policies of the other agents:

$$P(s'|s, a_i; \boldsymbol{\pi}_{-i}) = \sum_{\mathbf{a}_{-i} \in \mathcal{A}_{-i}} P(s'|s, a_i, \mathbf{a}_{-i}) \prod_{j \neq i} \pi_j(a_j|s) \tag{20}$$

As the other agents update their policies π_j, the effective transition dynamics that agent i experiences change, even if the underlying environment dynamics $P(s'|s, \mathbf{a})$ are fixed. This non-stationarity invalidates the convergence guarantees of single-agent algorithms like Q-Learning and makes experience collected in the past (e.g., in a replay buffer) quickly obsolete, as it represents an outdated version of the environment's dynamics. Addressing this non-stationarity is a central theme in modern MARL research.

4.2.2 Partial Observability: Acting in the Fog of War

As introduced in Section 3.4.3, many multi-agent environments are more naturally modelled as partially observable stochastic games rather than fully observable Markov games. In most realistic multi-agent scenarios, agents do not have access to the complete global state s. Instead, each agent i perceives the environment through its own local, and often incomplete, **observation** o_i. This limitation fundamentally changes the problem from a Stochastic Game to a **Partially Observable Stochastic Game (POSG)** [10], as introduced in the previous chapter.

In most realistic multi-agent scenarios, agents do not have access to the complete global state s. Instead, each agent i perceives the environment through its own local, and often incomplete, **observation** o_i. This limitation fundamentally changes the problem from a Stochastic Game to a **Partially Observable Stochastic Game (POSG)** [10], as introduced in the previous chapter.

Partial observability further breaks the Markov property from the agent's point of view. An agent's observation o_t is no longer a sufficient statistic of the environment's true state. Two distinct global states, s_a and s_b, could produce the exact same observation for an agent, a phenomenon known as *perceptual aliasing*. For example, in a game of poker, a player sees their own cards (a local observation) but not their opponents' cards or the deck's full state. Their optimal action depends critically on this hidden information.

To act optimally, the agent must therefore maintain a *belief* over the possible true states of the world. This typically requires integrating its entire history of observations and actions, $h_t = (o_1, a_1, \ldots, o_t)$, to infer what the true state might be. This necessitates the use of policies with memory, often implemented with recurrent neural networks (e.g., LSTMs or GRUs), which can learn to encode the history into a meaningful hidden state representation that serves as a proxy for the agent's belief.

4.2.3 The Credit Assignment Problem

In cooperative settings, where a group of agents works together to achieve a common goal, the feedback they receive is often a single, shared team reward. This introduces the **credit assignment problem**: how can an individual agent deduce its own contribution to the collective outcome? [3].

Consider a team of agents receiving a collective reward R_{team} generated by their joint action $\mathbf{a}$. A naive approach is to provide this same reward to every agent. However, this learning signal is extremely noisy and often uninformative. An agent that took a highly effective and critical action might receive a low reward if its teammates performed poorly. Conversely, an agent that made a mistake might still receive a high reward due to the excellent performance of others. This makes it exceedingly difficult for an agent to learn which of its actions are truly beneficial.

This challenge is particularly acute when rewards are sparse and delayed. If a team in a complex simulation only receives a single positive reward for success at the very end of a long episode, it becomes nearly impossible to trace back which specific actions in the long joint-action sequence were critical. The difficulty of assigning credit accurately is a primary driver behind many advanced MARL architectures. Value decomposition networks (e.g., VDN, QMIX) [14, 15] and centralised critics that learn agent-specific advantages are designed specifically to disentangle the global reward signal and provide more targeted, effective learning signals to individual agents, a topic we will explore in the next section.

4.3 Algorithmic Paradigms

Given the challenges of non-stationarity, partial observability, and credit assignment, the field of MARL has evolved several distinct algorithmic paradigms. These paradigms are not mutually exclusive algorithms but rather high-level design philosophies that differ primarily in how they manage the flow of information between agents during the learning and execution phases. The choice of paradigm reflects

a fundamental trade-off between algorithmic simplicity, scalability, and the ability to effectively address the core challenges of multi-agent learning.

4.3.1 Independent Learners: Fully Decentralized Training and Execution

The most direct and naively simple approach to MARL is to have each agent learn independently, treating all other agents as an unmodelled part of the environment. This paradigm, often called **Independent Learning**, is exemplified by **Independent Q-Learning (IQL)**. It involves applying a standard single-agent RL algorithm to each agent in the system as if it were acting alone.

Each agent i maintains its own private action-value function $Q_i(o_i, a_i)$—a function of its own local observation and action—and updates it based solely on its own reward r_i. The standard Q-learning update for agent i would be:

$$Q_i(o_{i,t}, a_{i,t}) \leftarrow (1-\alpha)Q_i(o_{i,t}, a_{i,t}) + \alpha \left[r_{i,t+1} + \gamma \max_{a'} Q_i(o_{i,t+1}, a') \right] \tag{21}$$

The primary advantage of this approach is its simplicity and scalability. It requires no communication between agents and the computational cost scales linearly with the number of agents. However, its theoretical foundation is weak. This paradigm directly confronts the problem of non-stationarity head-on, but offers no solution. As other agents adapt their policies, the environment from agent i's perspective becomes non-Markovian, and the theoretical convergence guarantees of single-agent Q-learning are voided. While independent learning can sometimes succeed in simple, stable problems, it often fails to converge reliably in more complex settings where tight coordination is required, frequently leading to unstable or cyclical learning dynamics.

4.3.2 Joint Action Learners: Fully Centralized Training and Execution

At the opposite end of the spectrum is a fully centralised approach. Here, the multi-agent system is reframed as a single, massive-scale MDP. A single, centralised controller observes the global state s (or the joint observation $\mathbf{o}$) and selects a complete joint action $\mathbf{a} = (a_1, ..., a_N)$ for all agents simultaneously. The learning algorithm then optimizes a single, centralised action-value function $Q(s, \mathbf{a})$ that is defined over the joint state-action space.

$$Q(s_t, \mathbf{a}_t) \leftarrow (1-\alpha)Q(s_t, \mathbf{a}_t) + \alpha \left[R_{\text{team}}(s_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}'} Q(s_{t+1}, \mathbf{a}') \right] \tag{22}$$

This paradigm elegantly solves the non-stationarity problem by definition; from the perspective of the single central controller, the environment is stationary and Markovian. However, this theoretical purity comes at an insurmountable practical cost: the **curse of dimensionality**. The size of the joint action space $|\mathcal{A}| = \prod_i |\mathcal{A}_i|$ grows exponentially with the number of agents N.

The maximization step, $\max_{\mathbf{a}'} Q(s_{t+1}, \mathbf{a}')$, requires an exhaustive search over this exponentially large space, which is computationally intractable for all but a trivial number of agents and actions. For instance, a game with 10 agents, each with 10 actions, would have a joint action space of 10^{10}. This exponential scaling makes the fully centralised paradigm impractical for almost any real-world problem.

4.3.3 Centralized Training with Decentralized Execution (CTDE)

The dominant modern paradigm, **Centralized Training with Decentralized Execution (CTDE)**, offers a powerful and pragmatic compromise between the two extremes [10]. It leverages the stability of centralised learning during an offline training phase while producing decentralized policies that can be executed scalably in the real world.

The core idea is to use extra information during training that will not be available to the agents at execution time. This "privileged information" typically includes the global state, the actions of other agents, and their reward functions.

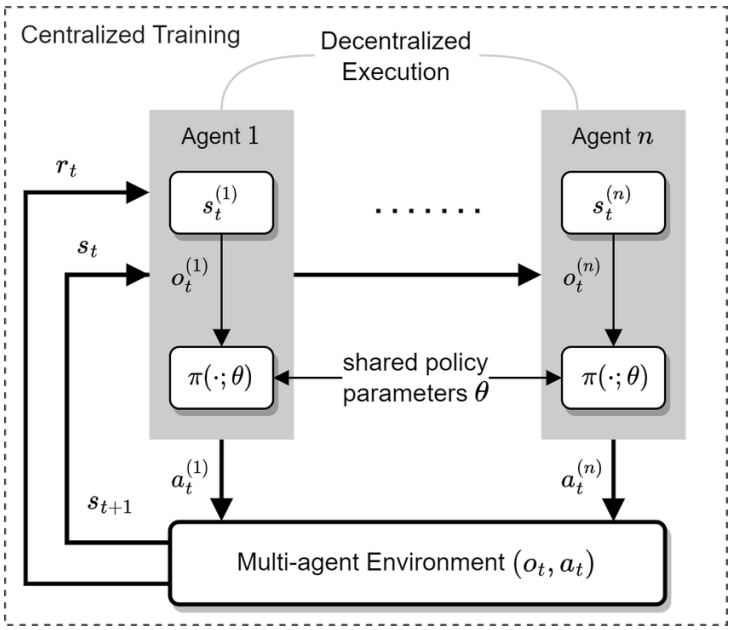

Figure 2: The CTDE paradigm. During training, a centralised critic leverages global information to provide a stable learning signal. During execution, each agent's decentralized actor makes decisions based only on its local observation. [16]

During training, a **centralised critic** is given access to global information. This critic can learn an accurate and stable value function because, from its global perspective, the environment is stationary. This stable value function is then used to guide the training of multiple **decentralized actors**. Each actor, which represents the policy of an individual agent, is trained to act based only on its own local, partial observations. At execution time, the centralised critic is discarded, and only the lightweight, decentralized actors are deployed.

A canonical example of this paradigm is the **Multi-Agent Deep Deterministic Policy Gradient (MADDPG)** algorithm [17]. In MADDPG, each agent i has two networks:

- **Actor Network (μ_{ϕ_i}):** A decentralized policy network that takes its local observation o_i as input and outputs a deterministic action $a_i = \mu_{\phi_i}(o_i)$.
- **Critic Network (Q_{θ_i}):** A centralised action-value network that takes the full state (or joint observation) and the joint action of all agents as input, $Q_{\theta_i}(s, a_1, ..., a_N)$, and outputs the estimated value for agent i.

During training, the critic for each agent is updated by minimizing the standard TD loss, using a replay buffer $\mathcal{D}$ of transitions $(s, \mathbf{a}, \mathbf{r}, s')$:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,\mathbf{a},\mathbf{r},s')\sim\mathcal{D}}\left[\left(y_i - Q_{\theta_i}(s, a_1, \ldots, a_N)\right)^2\right] \tag{23}$$

where the target value y_i is computed using target networks for stability: $y_i = r_i + \gamma Q_{\theta_i'}(s', a_1', \ldots, a_N')|_{a_j'=\mu_{\phi_j'}'(o_j')}$.

The crucial step is the actor update. Each actor μ_{ϕ_i} is updated using the deterministic policy gradient, but the gradient is provided by its centralised critic. The critic, having access to the full joint action, can provide a targeted learning signal that accounts for the actions of all other agents, thus navigating the non-stationary environment:

$$\nabla_{\phi_i} J \approx \mathbb{E}_{s,\mathbf{a}\sim\mathcal{D}}\left[\nabla_{\phi_i}\mu_{\phi_i}(o_i)\nabla_{a_i}Q_{\theta_i}(s, a_1, \ldots, a_N)|_{a_i=\mu_{\phi_i}(o_i)}\right] \tag{24}$$

By allowing the critic to be centralised, CTDE provides a stable learning signal that effectively addresses non-stationarity during training. Meanwhile, the decentralized nature of the actors ensures that the final policies are scalable and applicable in real-world scenarios where global information and communication are not available at execution time. This elegant separation of concerns has made CTDE the foundational paradigm for the vast majority of advanced MARL algorithms developed today, particularly in cooperative settings.

From MARL to self-play. So far we have treated learning in stochastic games with a fixed or externally specified set of opponents. In practice, many of the strongest systems organise training through self-play or population-based training, where opponents are drawn from an evolving population of learned policies. The next chapter turns to this self-play perspective and formalises it as a particular way of structuring MARL in stochastic games, connecting it to the evolutionary game-theoretic ideas from Section 3.5.

5 Self-Play: The Automated Crucible for Strategy Discovery

In multi-agent settings, non-stationarity arises because agents learn at the same time, constantly changing one another's effective environment. *Self-play* is a training paradigm that embraces this rather than trying to avoid it. Instead of training against a fixed opponent, an agent is trained against versions of itself (or a population of its predecessors), so that the difficulty of the task automatically tracks the agent's own ability.

Self-play is not a separate algorithm but a way of organising training around three ingredients: a population of policies, a rule for choosing opponents from that population, and an RL "oracle" that trains a new policy against those opponents. This section describes that general framework and shows how standard variants of self-play fit into it.

5.1 A Unified Framework for Self-Play

While the term "self-play" evokes a simple image of an agent playing against itself, the reality encompasses a rather diverse and complicated family of training procedures. To systematically understand this landscape, it is useful to adopt a generalized framework that abstracts the common components and design choices shared by nearly all self-play algorithms. As formalized by Zhang et al. [18], a self-play process can be deconstructed into an iterative loop orchestrated by three core elements: the Policy Population, the Opponent Sampling Strategy, and the Oracle.

The entire process can be visualized as a cycle of generation and refinement, as depicted in Figure 3. In each iteration, the system leverages its accumulated knowledge to pose a new strategic challenge for itself, learns to solve it, and then integrates the resulting solution back into its knowledge base.

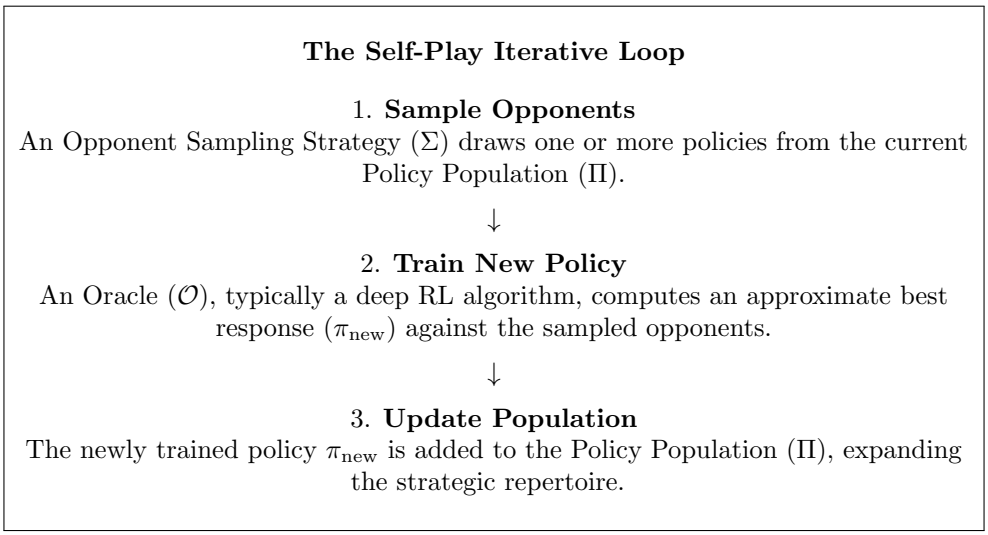

Figure 3: A conceptual diagram of the unified self-play framework. This iterative loop drives the co-evolution of strategies by continuously generating new policies as responses to the existing population.

5.1.1 The Core Components

- **The Policy Population (Π):** This component serves as the historical archive or the *accumulated strategic memory* of the self-play process. It is a set containing all distinct policies that have been generated up to the current iteration, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$. The nature of this population is a key design choice. In some schemes, it grows indefinitely, creating an ever-richer pool of opponents. In others, it has a fixed size, with older or weaker policies being pruned. Furthermore, the initialization method varies. *Lazy initialization* starts with a single, often random, policy and grows the population one policy at a time. In contrast, *immediate initialization* may start with a full population of diverse, pre-trained or randomly generated policies that are then refined simultaneously in an ongoing training process [18].
- **The Opponent Sampling Strategy (Σ):** This is the heart of the automated curriculum. It is a mechanism that defines a probability distribution over the policy population Π. At the beginning

of each training iteration, this strategy is used to select the opponent(s) that the current agent-in-training will face. The design of Σ is what primarily distinguishes different families of self-play and dictates the nature of the learning pressure. For example, a simple strategy might always select the single most recent policy (Vanilla Self-Play), while a more complex one might sample uniformly from all past policies (Fictitious Play), or it might be the result of solving a meta-game defined over the entire population (PSRO). This strategy can be formally represented by an *interaction matrix*, where an entry Σ_{mn} specifies the probability that policy m is trained against policy n [18].

- **The Oracle ($\mathcal{O}$):** This is the underlying learning algorithm responsible for generating a new policy. The oracle's goal is to compute an approximate **best response** to the strategic challenge posed by the opponents sampled via Σ. In modern deep RL, the oracle is almost always a powerful policy optimization algorithm like PPO or SAC. It takes the agent's current policy parameters, pits it against the sampled opponents for a large number of episodes, and updates the parameters to maximize its expected reward. The output of the oracle is a new policy, $\pi_{k+1} = \mathcal{O}(\Pi, \Sigma)$, which is then added to the population. The "strength" of the oracle (i.e., how closely it can approximate a true best response) is a critical factor in the efficiency and convergence of the overall self-play process.

By understanding these three components, we can deconstruct any self-play algorithm into its constituent parts. This framework provides a common language for comparing different methods and highlights the key design axes—the size and management of the population, the mechanism for opponent selection, and the power of the learning algorithm—that determine the dynamics and effectiveness of the training process.

5.2 Varieties of Self-Play

The unified framework provides a lens through which we can categorize and understand the main paradigms of self-play. These paradigms are not entirely distinct algorithms but rather different philosophies for designing the opponent sampling strategy (Σ), which in turn creates different learning pressures and curricula. We can arrange them along a spectrum of complexity, from simple, direct competition to sophisticated, game-theoretic population management.

5.2.1 Vanilla Self-Play: Competing with the Present

The most direct and intuitive form of self-play is **Vanilla Self-Play**, where an agent is exclusively trained against the single, most recent version of itself. This method creates a continuous arms race, where each new policy iteration must surpass its immediate predecessor.

- **Framework Instantiation:**
  - **Policy Population (Π):** The population $\Pi = \{\pi_1, \ldots, \pi_k\}$ grows with each iteration, but for opponent selection, only the most recent policy, π_k, is relevant.
  - **Opponent Sampling Strategy (Σ):** The strategy is a deterministic selection of the latest policy. It can be formally described as a delta function centered on the most recent agent:

$$\Sigma(\pi) = \begin{cases} 1 & \text{if } \pi = \pi_k \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

  - **Oracle ($\mathcal{O}$):** The oracle's task is to find an approximate best response, π_{k+1}, that can defeat the current champion, π_k.
- **Discussion:** This approach creates a highly stable and efficient curriculum in games with a *transitive* strategic structure (i.e., where if A beats B, and B beats C, then A reliably beats C). Go and Chess are examples of largely transitive games. By always providing a competent opponent that is only marginally stronger, the oracle is never faced with an insurmountable challenge. This was the core training mechanism behind **AlphaGo Zero** [19], which learned to play Go at a superhuman level starting from a blank slate, with no human data.

  However, in *non-transitive* games (e.g., Rock-Paper-Scissors), vanilla self-play is prone to failure. It can lead to cyclical learning dynamics where the agent learns to beat Rock with Paper, then forgets Rock to learn Scissors, then forgets Paper to learn Rock, never converging to a robust mixed

strategy. This susceptibility to "catastrophic forgetting" of older but still relevant strategies is its primary limitation.

5.2.2 Fictitious Play and Historical Averaging: Competing with the Past

To improve robustness and mitigate the cycling problem, an agent can be trained not just against its present self, but against a distribution of its past selves. This approach is rooted in the game theory concept of **Fictitious Play** [20], where players best-respond to the empirical frequency of their opponents' historical actions. This exposes the agent to a wider range of strategies, forcing it to learn a policy that is generally capable rather than one specialized against a single opponent.

- **Framework Instantiation:**
  - **Policy Population (Π):** The population serves as a full historical archive, storing every policy generated during training.
  - **Opponent Sampling Strategy (Σ):** The most common strategy is a uniform distribution over all historical policies in the population. At each training step, an opponent is sampled uniformly at random from Π:

$$\Sigma(\pi_j) = \frac{1}{k} \quad \text{for } j = 1, \ldots, k \tag{26}$$

  - **Oracle ($\mathcal{O}$):** The oracle's objective is more demanding than in vanilla self-play. It must compute a policy that performs well *on average* against the entire history of opponents, effectively learning an approximate best response to the average historical strategy.
- **Discussion:** By forcing the agent to play against its entire history, this method explicitly prevents forgetting and breaks the cycles that plague vanilla self-play in non-transitive games. This historical averaging leads to convergence towards an approximate Nash Equilibrium of the game. Modern deep RL implementations, such as **Neural Fictitious Self-Play (NFSP)** [21], have successfully applied this principle to master complex imperfect information games like Poker. Variations on this theme can use non-uniform sampling, for example, by prioritizing stronger or more recent opponents from the historical pool, as seen in the training of **AlphaStar** [22].

5.2.3 Population-Based Self-Play: Competing with the Meta-Game

The most powerful and general approach elevates the self-play concept from a simple training loop to the management of a strategic ecosystem. In **Population-Based Self-Play**, the entire population of policies is treated as a set of players in an empirical, or *meta-game*. The solution to this meta-game then dictates the curriculum for training the next generation of policies. This is the core idea behind the **Policy-Space Response Oracles (PSRO)** framework [23].

- **Framework Instantiation:** The PSRO loop is a more explicit implementation of our unified framework with a game-theoretic core.
  - **Policy Population (Π):** The population is an expanding set of diverse strategies.
  - **Opponent Sampling Strategy (Σ):** This is the key innovation. To derive Σ, the system first constructs a meta-game by evaluating the empirical payoff (e.g., win rate) of every policy $\pi_i \in \Pi$ against every other policy $\pi_j \in \Pi$. This results in a payoff matrix. A game-theoretic solution concept, such as a Nash Equilibrium, is then computed for this matrix. The resulting equilibrium is a mixed strategy over the existing policies, and this mixture becomes the opponent sampling strategy Σ.
  - **Oracle ($\mathcal{O}$):** The oracle's task is to compute an approximate best response to the *meta-game equilibrium strategy* Σ. In essence, the oracle is trained to find and exploit the biggest strategic weakness or "hole" in the current population's collective strategy.
- **Discussion:** This iterative process—constructing a meta-game, finding its equilibrium, and training a new policy to defeat that equilibrium—actively fosters strategic diversity. Each new policy added to the population covers a previously exploitable weakness, making the subsequent meta-game more robust and challenging. While computationally intensive due to the quadratic cost of evaluating the payoff matrix, this approach has proven essential for mastering games with vast,

non-transitive strategy spaces. Landmark systems like **AlphaStar** (StarCraft II) [22] and **OpenAI Five** (Dota 2) [24] used sophisticated forms of population-based training. They maintained a "league" of agents, including current competitors and specialized "exploiters," to create a rich and dynamic training environment that propelled them to superhuman performance in some of the world's most complex games.

5.3 Self-Play as an Engine for Discovery

Self-play is often motivated as a way to reach strong performance without human data, but it also changes *what* gets learned. Because the opponents are drawn from the agent's own past behaviour (or from a population derived from it), the training process continually exposes the learner to strategies that exploit its current weaknesses. Each new policy must therefore handle a slightly harder and more tailored challenge than the last.

This produces an automatic curriculum. Early in training, the agent only needs to beat weak, almost random opponents. As its play improves, the distribution of opponents shifts accordingly, and progress requires discovering more refined tactics and long-term plans. In population-based schemes, this is made explicit: a new policy is trained as a best response to the current population mixture and, once added back into the population, closes off an existing exploit.

A well-known example of this effect is AlphaGo's training process [25, 19]. After an initial supervised phase on human games, its strongest versions were obtained purely through self-play, without further human input. The resulting policies not only matched human professional play but also produced moves that were considered unconventional or unexpected by experts. These were not programmed in advance; they emerged naturally from the pressure to improve against increasingly strong self-generated opponents.

The same logic underlies later systems that use self-play or population-based training in far more complex games. In each case, the goal is not just to fit to a fixed dataset of behaviours, but to let the agent's own learning process generate a sequence of ever more informative training tasks. In the next chapter, Section 6, we shift focus from how self-play discovers strategies to how we can deliberately shape incentives, architectures, and interaction patterns so that the strategies discovered are reliably cooperative.

6 Engineering Cooperation

The previous chapters described how multi-agent interactions can be modelled and how learning proceeds in such settings. In this chapter we turn to a more targeted question:

> Given an environment that admits both cooperative and non-cooperative outcomes, how can we design the system so that learning agents tend to end up cooperating?

This is relevant for many envisioned applications of MARL: coordinated vehicles, resource management, and human–AI teams all rely on agents achieving mutually beneficial outcomes rather than simply maximising short-term individual reward.

To reason about this, we first need a working definition of cooperation and concrete metrics for measuring it. We then organise the design space into four mechanism classes that influence cooperation in different ways: by shaping payoffs and dynamics, by changing learning architectures, by modifying information flow, and by altering long-run interaction patterns in a population.

6.1 What Do We Mean by Cooperation? Problem Setup and Metrics

The word *cooperation* is often used informally in the MARL literature to describe "agents working together". For the purposes of this thesis, we need a more precise notion that connects back to the game-theoretic and RL foundations developed in the previous chapters.

We consider a multi-agent system with a set of agents $\mathcal{N} = \{1, \dots, N\}$ interacting in a stochastic game (or Markov game) as defined in Section 3.4 and analysed from a learning perspective in Section 4. Each agent i follows a policy π_i and receives a reward signal r_i. Let $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$ denote the joint policy profile. We say that the environment exhibits a **social dilemma** if there is a tension between individual and collective payoffs:

- there exist joint policies that achieve high *social welfare*, but
- for each agent, deviating unilaterally from such policies is individually advantageous in the short term.

The classical Prisoner's Dilemma and public goods games are examples of such settings in normal form; in the sequential case this extends to temporally extended social dilemmas [7]. Informally, **cooperation** means that agents adopt policies that resolve or mitigate this tension in favour of the collective outcome.

From the game-theoretic perspective of Section 3, this situates us in the middle of the spectrum between fully cooperative (common-payoff) games, where there is no conflict of interest, and zero-sum games, where one agent's gain is another's loss and "cooperation" is largely meaningless. Our focus in this chapter is on *general-sum* or mixed-motive games that admit both cooperative and non-cooperative outcomes: environments in which there is real potential for mutual gain, but also strategic incentives to defect.

More concretely, we will use the following working notion:

> A joint policy $\boldsymbol{\pi}$ is **cooperative** if it yields high long-term social welfare and avoids outcomes that are Pareto-dominated by more collectively beneficial policies.

This definition intentionally leaves some flexibility: in many environments there is no single "cooperative" policy, but rather a family of joint behaviours that are approximately efficient and fair. To make this notion operable in experiments, we introduce explicit metrics.

Social welfare. Given a joint policy $\boldsymbol{\pi}$ and an initial state distribution ρ_0, the value for agent i is

$$V_i^{\boldsymbol{\pi}} = \mathbb{E}_{s_0 \sim \rho_0}\left[V_i^{\boldsymbol{\pi}}(s_0)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{i,t+1} \;\middle|\; s_0 \sim \rho_0, \boldsymbol{\pi}\right]. \tag{27}$$

The **social welfare** (or team return) under $\boldsymbol{\pi}$ is the sum of individual values:

$$W(\boldsymbol{\pi}) = \sum_{i \in \mathcal{N}} V_i^{\boldsymbol{\pi}}. \tag{28}$$

In purely cooperative tasks (common-payoff games), W is the main performance objective. In mixed-motive tasks, high social welfare indicates that agents have managed to coordinate on outcomes that are collectively beneficial.

Cooperation rate. In many social dilemma environments there is a distinguished set of *cooperative actions* (e.g., "Contribute" in a public goods game, "Hunt stag" in a Stag Hunt). Let $\mathcal{C}_i \subseteq \mathcal{A}_i$ denote the subset of actions for agent i that correspond to cooperation. The **cooperation rate** for agent i under policy π_i is then

$$C_i(\pi_i) = \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{1}\{a_{i,t} \in \mathcal{C}_i\}\right], \tag{29}$$

where T is the episode length. We will sometimes aggregate this across agents as $\bar{C}(\boldsymbol{\pi}) = \frac{1}{N}\sum_i C_i(\pi_i)$. High cooperation rate is neither necessary nor sufficient for high welfare in all environments, but in social dilemmas it is a useful behavioural proxy for cooperative intent.

Fairness and equality. Cooperation is often understood to include some notion of fairness: a joint policy that maximises $W(\boldsymbol{\pi})$ while giving one agent almost everything and another almost nothing is typically not considered fully cooperative. To capture this, we can consider the dispersion of individual returns $\{V_i^{\boldsymbol{\pi}}\}_{i\in\mathcal{N}}$ using, for example, their standard deviation or a Gini coefficient.

Stability over time. Finally, a hallmark of genuine cooperation is that it is *maintained* over time. A learning process that briefly visits high-welfare, high-cooperation states before collapsing to mutual defection is qualitatively different from one that stabilises near a cooperative equilibrium. In experiments we will therefore track both the level and the *trajectory* of social welfare and cooperation rates across training.

These metrics give us a concrete way to answer the central question of this chapter: given a fixed environment that admits cooperative and non-cooperative outcomes, which mechanisms can we engineer into the reward structure and learning process to reliably drive the system towards high-welfare, high-cooperation regimes?

6.2 Mechanism Class I: Payoff and Environment Design

The most direct way to influence cooperative behaviour is to change the game itself. Rather than asking agents to learn cooperation in a hostile payoff landscape, we can *shape* the rewards and environment dynamics so that cooperative outcomes are easier to discover and more robust once found. This perspective connects MARL to ideas from mechanism design and reward shaping. Mechanisms in this class are often *implicit*: we do not endow agents with any special reasoning about cooperation, but instead design the task so that standard RL methods tend to produce cooperative policies.

We distinguish two closely related levers:

1. designing the **reward structure** (how agents are evaluated), and
2. designing the **environment dynamics** (how actions translate into consequences).

Both are forms of implicit cooperation: we do not change the learning algorithms or introduce explicit coordination protocols, but instead alter the problem so that standard RL machinery is more likely to produce cooperative policies.

6.2.1 Reward design

In a general stochastic game, each agent i receives a reward $r_i(s, \mathbf{a})$ that depends on the state s and the joint action $\mathbf{a}$. The simplest cooperative modification is to replace individual rewards with a **shared team reward**:

$$r_i(s, \mathbf{a}) = r_{\text{team}}(s, \mathbf{a}) \quad \forall i \in \mathcal{N}. \tag{30}$$

This transforms the game into a common-payoff setting where all agents are, in principle, aligned: maximising any V_i is equivalent to maximising social welfare W. In matrix games like the Prisoner's Dilemma, this corresponds to paying both players the sum of their original payoffs. In sequential tasks,

this might mean rewarding all agents based on global task completion, such as the team score in a cooperative navigation task.

While team rewards remove direct conflicts of interest, they do not by themselves solve the *credit assignment* problem discussed in Section 4. When rewards are sparse and only available at the end of an episode, learning which individual actions contributed to success remains difficult. This motivates more refined reward shaping schemes.

Concretely, consider a cooperative predator–prey scenario with two predators that must corner a single prey. A purely sparse team reward might only be given upon successful capture. Reward shaping can provide denser, local signals that encourage the intermediate behaviours needed for cooperation, for example:

- small positive rewards for reducing the distance between each predator and the prey;
- rewards for maintaining an appropriate distance between the two predators, encouraging them to approach from different directions rather than chasing in single file;
- bonuses when the predators restrict the prey's available escape routes, e.g., by occupying flanking positions near walls or corners.

These additional terms do not change the ultimate goal (capturing the prey), but they make the path to effective joint behaviour much easier for independent learners to discover.

One influential idea is to use **difference rewards** (or counterfactual rewards) [3]. For agent i, the difference reward at time t is defined as

$$D_i(s_t, \mathbf{a}_t) = r_{\text{team}}(s_t, \mathbf{a}_t) - r_{\text{team}}(s_t, (a'_i, \mathbf{a}_{-i,t})), \tag{31}$$

where a'_i is a counterfactual default action for agent i (often interpreted as "do nothing") and $\mathbf{a}_{-i,t}$ denotes the actions of all other agents. Intuitively, D_i measures how much better (or worse) the team did because agent i chose a_i instead of the baseline a'_i. This provides a more informative learning signal for agent i, while preserving alignment with the global objective.

Another broad class of techniques is **potential-based reward shaping**. Here we introduce an additional shaping term $F(s_t, s_{t+1})$ to the reward:

$$\tilde{r}_i(s_t, \mathbf{a}_t, s_{t+1}) = r_i(s_t, \mathbf{a}_t) + \gamma\Phi(s_{t+1}) - \Phi(s_t), \tag{32}$$

where $\Phi : \mathcal{S} \to \mathbb{R}$ is a potential function. In the single-agent case, such shaping is known to preserve the set of optimal policies, while potentially accelerating learning. In the multi-agent cooperative setting, we can choose Φ to reward intermediate states that are indicative of emerging cooperation (for example, shared resource levels within a safe range), biasing exploration towards cooperative trajectories without changing the underlying game-theoretic structure.

Finally, reward design can also include explicit **penalties for anti-social behaviour**. For example, in a resource harvesting game we might subtract a penalty when the total harvest exceeds a sustainable threshold, making over-exploitation immediately costly. This effectively transforms the payoff structure from a Prisoner's Dilemma-like tragedy of the commons into a game where sustainable strategies are individually rational.

6.2.2 Environment design

Reward shaping operates at the level of evaluation, but we can also modify the environment dynamics themselves. In a normal-form game, this corresponds to changing the payoff matrix; in sequential settings, it means altering the transition function $P(s' \mid s, \mathbf{a})$ or the action spaces $\mathcal{A}_i$.

A simple example is to introduce **coordination-dependent rewards**. In the Stag Hunt, hunting stag is only profitable if both agents choose the "Stag" action. We can build analogous structures into sequential environments: doors that require two agents to stand on switches simultaneously, tasks that can only be completed if one agent holds a button while another passes through, and so on. These structural dependencies create natural focal points for cooperation.

More generally, we can adjust:

- **Resource dynamics**, such as regeneration rates or carrying capacities, to make overuse visibly harmful.
- **Observation structure**, for example by making certain information available only when agents are close to each other, encouraging spatial coordination.
- **Action constraints**, such as limiting how much any single agent can extract or influence, so that beneficial outcomes require joint action.

From the perspective of evolutionary game theory (Section 3.5), these changes alter the underlying fitness landscape. The goal is to make cooperative strategies more viable and less fragile: easier to discover, and less vulnerable to invasion by short-sighted defectors.

6.2.3 Benefits and limitations

Payoff and environment design are attractive because they are conceptually simple and compatible with standard RL algorithms. If we get the design right, even relatively naive independent learners can stumble upon cooperative policies. Many real-world systems—taxation schemes, fishing quotas, congestion charges—are, at their core, attempts to implement such mechanisms at scale.

However, these methods have clear limitations. Designing rewards and dynamics that induce the desired behaviour without introducing perverse incentives is notoriously difficult. In complex environments, it may be hard even to *specify* what cooperative behaviour should look like, let alone encode it into a scalar reward. Moreover, poorly chosen shaping terms can mislead learning and trap agents in suboptimal equilibria.

These challenges motivate the next class of mechanisms, which shift some of the burden from the environment onto the learning algorithms themselves. Rather than only changing what is rewarded, we can change *how* agents learn from shared rewards, through architectures explicitly designed to solve the credit assignment problem.

6.3 Mechanism Class II: Learning Architectures for Credit Assignment

Reward and environment design can make cooperative outcomes more attractive, but in many settings the main obstacle is not incentives, it is *credit assignment*. When agents share a global reward signal, each needs to infer how its own actions contributed to that reward. As argued in Section 4, naive independent learners struggle in this regime: the same scalar team reward is broadcast to all agents, regardless of whether their individual behaviour was helpful or harmful.

Mechanisms in this class keep the reward structure fixed (often a shared team reward) and instead modify the *learning architecture*. The aim is to construct value functions and critics that provide agent-specific learning signals aligned with the global objective. These mechanisms are more *explicit* than those in Mechanism Class I: they encode structural assumptions about how joint value should decompose into individual contributions. This is where the CTDE paradigm (Section 4) becomes particularly useful: we allow a centralised component to see the full state and joint actions during training, and use it to shape the updates of decentralized policies.

6.3.1 Centralized critics and counterfactual baselines

In actor–critic methods, each agent updates its policy using a gradient of the form

$$\nabla_{\theta_i} J_i \approx \mathbb{E}\left[A_i(s, \mathbf{a}) \nabla_{\theta_i} \log \pi_{\theta_i}(a_i \mid o_i)\right], \tag{33}$$

where A_i is an advantage function that measures how much better the chosen action a_i was compared to some baseline. A centralised critic can estimate A_i using global information: the full state s and the joint action $\mathbf{a}$.

A natural choice is to estimate a joint action-value function $Q_i(s, \mathbf{a})$ for each agent i, and then define

$$A_i(s, \mathbf{a}) = Q_i(s, \mathbf{a}) - B_i(s, \mathbf{a}_{-i}), \tag{34}$$

where B_i is a baseline that does not depend on the agent's own action a_i. One influential example is the *counterfactual baseline* used in COMA [26], where

$$B_i(s, \mathbf{a}_{-i}) = \sum_{a_i' \in \mathcal{A}_i} \pi_{\theta_i}(a_i' \mid o_i)\, Q_i(s, (a_i', \mathbf{a}_{-i})). \tag{35}$$

This baseline marginalises over agent i's action under its current policy while keeping the other agents' actions fixed. The resulting advantage

$$A_i(s, \mathbf{a}) = Q_i(s, \mathbf{a}) - \sum_{a_i'} \pi_{\theta_i}(a_i' \mid o_i)\, Q_i(s, (a_i', \mathbf{a}_{-i})) \tag{36}$$

measures how much better a_i was than the average action the policy would have taken in that state. This isolates agent i's contribution to the joint outcome, providing a cleaner learning signal than the raw team reward.

Architectures like MADDPG [17] adopt a related idea in continuous control: each agent has a decentralized actor $\pi_{\theta_i}(a_i \mid o_i)$ and a centralised critic $Q_i(s, \mathbf{a})$. The critic sees the full joint action and state, but the policy only depends on the local observation o_i. During training, gradients flow through the centralised critic into the actor; at execution time, the critic is discarded and each agent acts based only on its own observation.

The key property of these critic-based methods is that they do not require modifying the reward function. Instead, they use global information during training to disentangle individual contributions from a shared signal. Although agents still act based solely on local observations at execution time, the training signal they receive encodes information about the joint context. This allows decentralized policies to learn *implicit coordination* patterns—complementary behaviours that depend on subtle correlations between observations and the underlying state—that a purely local learner would fail to discover.

6.3.2 Value decomposition in cooperative Q-learning

In fully cooperative settings with a shared team reward r_{team}, we can take a different approach and work directly at the level of Q-values. Instead of maintaining a single joint action-value function $Q(s, \mathbf{a})$ over the exponentially large joint action space, value decomposition methods approximate this joint value as a combination of per-agent utilities. This has two benefits:

1. it makes learning more scalable, and
2. it provides a natural way to assign credit to individual agents.

Value-Decomposition Networks (VDN). VDN [14] assumes that the joint Q-function can be written as a simple sum of individual utilities:

$$Q_{\text{tot}}(s, \mathbf{a}) \approx \sum_{i \in \mathcal{N}} Q_i(o_i, a_i; \theta_i), \tag{37}$$

where each Q_i depends only on agent i's local observation and action. During training, the team receives a shared reward r_{team}, and the tuple $(s_t, \mathbf{a}_t, r_{\text{team},t+1}, s_{t+1})$ is used to update the parameters of all Q_i via a TD loss on Q_{tot}. Since the gradient of this loss decomposes across agents, each agent effectively learns from its contribution to the joint estimate.

Execution remains decentralized: each agent selects a_i greedily with respect to its own $Q_i(o_i, \cdot)$. The additive structure of Q_{tot} ensures that this local greedy choice is aligned with maximising the joint Q-value, at least within the approximation.

QMIX and monotonic mixing. The additive assumption in VDN is quite restrictive. QMIX [15] relaxes it by introducing a learnable *mixing network* that combines the individual utilities into a joint Q-value:

$$Q_{\text{tot}}(s, \mathbf{a}) = f_{\text{mix}}\big(s, Q_1(o_1, a_1), \ldots, Q_N(o_N, a_N)\big). \tag{38}$$

The mixing network is constrained to be *monotonic* in each Q_i:

$$\frac{\partial Q_{\text{tot}}}{\partial Q_i} \geq 0 \quad \forall i. \tag{39}$$

This monotonicity condition guarantees that the joint greedy action

$$\mathbf{a}^* = \arg\max_{\mathbf{a}} Q_{\text{tot}}(s, \mathbf{a}) \tag{40}$$

can be recovered by independently choosing

$$a_i^* = \arg\max_{a_i} Q_i(o_i, a_i), \tag{41}$$

without having to search over the full joint action space. As in VDN, training is centralised (the mixing network sees the global state s), while execution is decentralized.

Both VDN and QMIX address credit assignment by *constraining the representational form* of the value function. The joint Q-value is always explainable in terms of per-agent components, so gradients derived from the team reward naturally propagate into agent-specific utilities.

6.3.3 Trade-offs

Architectural mechanisms of this kind shift complexity from the environment into the function approximator. They retain the simplicity of a shared team reward and avoid hand-engineered shaping, but require careful network design and often substantial computation: centralised critics must evaluate joint actions, and mixing networks must be expressive enough to capture the relevant interactions.

Moreover, the assumptions they make—additivity in VDN, monotonicity in QMIX, or the particular baseline structure in COMA—impose inductive biases on what patterns of cooperation can be represented. In practice, these biases can be helpful, especially in structured cooperative tasks, but they also limit what kinds of inter-agent dependencies can be captured. Later in this chapter we will see how information-sharing mechanisms can complement these architectures by reducing uncertainty at execution time.

6.4 Mechanism Class III: Learned Communication and Opponent Modelling

The mechanisms considered so far operate without changing what agents *know* at decision time. In many realistic settings, however, the main obstacle to cooperation is not misaligned incentives or poor credit assignment, but limited information. Agents act under partial observability (Section 4), with only a local view of the environment and their teammates. In this regime, it is often impossible to coordinate well without sharing information explicitly or inferring it from behaviour.

Mechanisms in this class modify the information structure of the problem at execution time. They either:

- introduce explicit **communication channels** between agents, or
- infer hidden information through **opponent modelling**.

Both approaches aim to reduce uncertainty about the state, about others' intentions, or both, thereby making cooperative strategies more feasible.

6.4.1 Learned communication protocols

A natural way to facilitate cooperation under partial observability is to give agents an explicit communication channel. At each timestep, an agent can send a message to its teammates and receive their messages in return. The content and interpretation of these messages are learned as part of the overall policy.

A typical differentiable communication architecture works as follows:

- Each agent i encodes its observation (and possibly its recurrent hidden state) into a message vector $m_i \in \mathbb{R}^d$ using a neural network.
- Messages $\{m_j\}_{j \neq i}$ from other agents are aggregated, for example by summation or an attention mechanism, to form a context vector c_i.
- The agent's policy then conditions on both its local observation and the aggregated message: $a_i \sim \pi_{\theta_i}(a_i \mid o_i, c_i)$.

Since the whole architecture is trained end-to-end with gradient descent, the agents are free to develop arbitrary *communication protocols*. In practice, this often leads to emergent communication schemes where messages correlate with latent task variables such as goals, intentions, or local measurements that are not directly observable to others.

Communication can be discrete as well as continuous. In that case, messages might be symbols from a learned vocabulary rather than real-valued vectors. Training then typically relies on techniques like the Gumbel–Softmax trick or policy gradient estimators to handle the non-differentiable message choices. Discrete communication has the appealing interpretation of agents inventing a kind of *language* to talk about the task.

6.4.2 When does communication help?

Giving agents the ability to talk does not guarantee cooperation. In some tasks, communication is unnecessary: if the environment is fully observable, or if optimal behaviour can be inferred from local information, learning a communication protocol only adds complexity. In sparse-reward, highly coupled tasks with severe partial observability, however, communication can make the difference between near-random behaviour and robust coordination.

Several factors influence whether communication mechanisms are beneficial:

- **Information asymmetry.** Communication is most useful when different agents have access to complementary pieces of information that matter for the joint decision (for example, each sees only part of the map).
- **Bandwidth and cost.** Realistic systems may impose limits on how much can be communicated per timestep, or penalise communication explicitly. This forces agents to develop *compressed* and *task-relevant* messages.
- **Stability and overfitting.** Rich communication channels can make policies fragile: agents may overfit to very specific message patterns and fail to generalise to new partners or slightly different protocols.

Despite these caveats, communication remains one of the most direct tools for engineering cooperation. It turns a decentralized control problem with hidden information into something closer to a distributed planning problem: agents can share what they know and negotiate coordinated actions.

6.4.3 Inferred information: Opponent and Teammate Modelling

While explicit communication allows agents to *tell* each other their intentions, a complementary approach is to have them *infer* this information. This is the goal of **opponent (or teammate) modelling**.

Opponent modelling involves explicitly learning a model of other agents' policies, for example $\hat{\pi}_{-i}(\mathbf{a}_{-i}|s)$. By maintaining a predictive model, an agent can move beyond purely reactive learning. Instead of just responding to the consequences of others' actions, it can proactively anticipate them and plan accordingly.

In a MARL context, an agent can augment its policy or value function with a representation of its belief about the other agents. For example, an agent's Q-function could be conditioned not only on its own action but also on the predicted actions of its teammates: $Q_i(s, a_i, \hat{\mathbf{a}}_{-i})$. During training, the agent can maintain an auxiliary loss function to train its predictive model, while the primary RL objective uses this prediction to select more effective actions.

Like communication, this explicit modelling of others is a powerful way to reduce uncertainty and combat non-stationarity. Instead of treating the changing policies of other agents as noise, the agent attempts to model this change directly. This allows for more rapid adaptation and is particularly crucial in mixed-motive settings, where understanding whether a partner is likely to cooperate or defect is essential for choosing one's own action. This mechanism helps agents form the *shared beliefs* necessary for coordination, even in the absence of a direct communication channel.

6.4.4 Information mechanisms and cooperation

Both explicit communication and inferred opponent models address the same underlying obstacle: cooperating agents must form *shared beliefs* about the state of the world and each other's intentions. While

centralised critics (Class II) align learning signals using global information during training, Class III mechanisms allow agents to construct and maintain these shared beliefs *during execution*.

In combination with the payoff and architectural mechanisms from Section 6.2 and Section 6.3, information-sharing forms a third pillar for engineering cooperation. In the remainder of this chapter, and in the experiments that follow, we will see concrete examples where adding carefully designed information mechanisms leads to higher and more stable cooperative performance.

6.5 Mechanism Class IV: Reciprocity, Reputation, and Partner Choice

The mechanisms discussed so far intervene at the level of rewards, architectures, or information flow within a fixed group of agents. A different route to cooperation is to change the *long-run incentives* that emerge from repeated interaction in a population. Here, ideas from repeated games and evolutionary game theory (Section 3.5) become central.

In one-shot social dilemmas like the Prisoner's Dilemma, defection is individually rational and cooperation is fragile. In the repeated game, however, agents can condition their behaviour on past interactions. This opens the door to strategies based on **reciprocity**: cooperate with those who cooperated with you, and punish those who defected. When combined with some possibility of meeting the same partners again, reciprocal strategies can make cooperation individually rational in the long run.

Mechanisms in this class modify not the immediate payoffs, but the *effective game* induced by repeated play, memory, and population structure. In MARL terms, they encourage agents to learn policies that:

- remember how others behaved in the past,
- adjust their own cooperation accordingly, and
- sometimes choose with whom to interact at all.

6.5.1 Reciprocity in repeated interactions

In the classical theory of repeated games, tit-for-tat and related strategies illustrate how cooperation can be sustained through direct reciprocity: start by cooperating, then repeat your partner's last move. These strategies work by creating a contingent threat: if you defect against me today, I will punish you tomorrow. Canonical tit-for-tat is often summarised as:

- **Nice:** it begins by cooperating, signalling a willingness to find a mutually beneficial outcome;
- **Retaliatory:** it punishes defection immediately by defecting in the next round;
- **Forgiving:** it returns to cooperation as soon as the partner does, avoiding long spirals of mutual punishment.

RL agents can implement analogous behaviour via *memory-based policies*. Instead of conditioning only on the current observation o_t, an agent's policy can depend on its interaction history h_t, for example through a recurrent neural network:

$$a_t \sim \pi_\theta(a_t \mid h_t), \quad h_t = f_\phi(h_{t-1}, o_t, a_{t-1}, r_t). \tag{42}$$

If the environment repeatedly matches the same partners, the agent can in principle learn to encode others' past actions into its hidden state and to respond in a reciprocal manner: cooperate with agents whose past behaviour signals cooperation, and withhold cooperation from those who exploited it.

From the perspective of our metrics in Section 6.1, reciprocity mechanisms aim to increase both the *stability* and *fairness* of cooperation. They make exploitative strategies less attractive by coupling short-term gains with future losses. In MARL, this often manifests as learning dynamics where transient defection is followed by mutual punishment and, if the learning dynamics are well-behaved, a return to cooperative regimes once the "lesson" has been absorbed.

A specific instantiation of this principle is the "Innovator-Imitator" model proposed by Eccles et al. [27]. In this framework, a subset of agents (imitators) are intrinsically rewarded for matching the "niceness" or pro-sociality of a target agent (the innovator). This effectively hard-codes a Tit-for-Tat dynamic into the objective function: if the innovator cooperates, imitators are incentivised to follow suit; if the innovator defects, imitators stop cooperating, punishing the innovator. This approach attempts to bootstrap

reciprocity without requiring agents to learn the complex causal link between current cooperation and future reciprocation from scratch.

6.5.2 Reputation and indirect reciprocity

Direct reciprocity relies on repeated interaction between the same pairs of agents. In larger populations, agents may not meet the same partners often enough for this to work. In such cases, **reputation** can play the role of memory: agents maintain and share assessments of one another's past behaviour, and condition their cooperation not only on their own experiences but also on the reputational signals they observe.

In a MARL setting, reputation can be implemented in several ways:

- Each agent maintains an internal estimate $R_i(j)$ of how cooperative agent j has been in the past, updated based on observed actions and outcomes.
- The environment may broadcast simple public signals about behaviour (for example, a "defector" flag when an agent over-exploits a shared resource).
- Agents may communicate reputational information explicitly, for example by sending messages that encode their assessment of others.

Policies then condition on these reputational variables:

$$a_i \sim \pi_{\theta_i}(a_i \mid o_i, R_i), \tag{43}$$

so that cooperative behaviour is preferentially directed towards agents with good reputations. Defectors may still gain in the short run, but they damage their reputation and find cooperation harder to obtain in the future.

Reputation mechanisms turn cooperation into a long-term asset: behaving well increases the chance of receiving cooperation from others. In the language of evolutionary dynamics, they can turn cooperative strategies into evolutionarily stable states by making defection less profitable when its reputational consequences are taken into account.

6.5.3 Partner selection and population structure

A further mechanism is to give agents some control over *who* they interact with. In many social systems, individuals can choose their partners, leave unproductive relationships, or form new alliances. This ability to *assort* with other cooperators is a powerful driver of cooperation in evolutionary models: cooperators thrive when they can preferentially interact with other cooperators and avoid exploitation.

In MARL, partner selection can be modelled by:

- making the interaction graph dynamic, so that links between agents can be formed or broken based on past behaviour or learned preferences;
- allowing agents to accept or reject potential partners for joint tasks;
- structuring the population into groups or teams that can be reconfigured over time.

An agent's policy then includes decisions about both task actions and social actions (whom to approach, whom to avoid), and learning must balance the benefits of exploiting current partners with the long-term benefits of finding more cooperative ones.

From a system-design perspective, these mechanisms change the effective game played in the population. Defectors may still do well within any single interaction, but if other agents can leave or avoid them, their long-term fitness declines. Cooperators, by contrast, can cluster together and enjoy the benefits of mutual cooperation. This corresponds to altering the "mixing matrix" in the population dynamics of Section 3.5: instead of random matching, interactions become structured in a way that favours cooperative clusters.

6.5.4 Relation to other mechanisms

Reciprocity, reputation, and partner choice complement the previous classes rather than replacing them. Reward shaping and value decomposition can make cooperation attractive within a single group; information and communication mechanisms can help agents coordinate their actions given a fixed set of

partners. The mechanisms in this section operate one level up, at the scale of populations and repeated relationships. They can stabilise cooperation that would otherwise be too fragile to persist, and they can penalise agents that exploit cooperative partners without having to change the base reward function.

In the experiments that follow, we will consider simplified settings where some of these effects can be observed in miniature: repeated interactions with the same partner, simple reputational signals, or limited partner choice. Even in these small systems, the qualitative impact on cooperation can be substantial.

6.6 Summary: When and Why Do These Mechanisms Work?

We have organised the space of cooperation-inducing techniques into four broad mechanism classes:

1. payoff and environment design,
2. learning architectures for credit assignment,
3. information and communication, and
4. reciprocity, reputation, and partner choice.

Mechanism Class I can be thought of as providing mainly *implicit* incentives for cooperation by reshaping the task itself, while Classes II–IV introduce increasingly *explicit* structure into the learning process, information flow, and population dynamics. Each class answers a different version of the core question: *why would individually learning agents adopt and maintain cooperative behaviour in a social dilemma?*

Mechanism class	**What we change**	**Main strengths / typical failure modes**
Payoff & environment design	Reward functions, resource dynamics, structural dependencies between actions.	Simple and intuitive; can be implemented at the task level. Hard to get right in complex domains; poorly designed shaping can create perverse incentives or trap agents in suboptimal equilibria.
Learning architectures (credit assignment)	Value function parametrisation; centralised critics; mixing networks.	Preserve a shared team reward while providing informative, agent-specific learning signals. Require careful network design; inductive biases (e.g., additivity, monotonicity) may limit representational power.
Information & communication	Access to global state at training; communication channels between agents at execution.	Reduce uncertainty under partial observability; enable explicit coordination. Can overfit to particular partners or protocols; communication may be costly or infeasible in some deployments.
Reciprocity, reputation, partner choice	Long-run interaction structure; memory of past behaviour; ability to select partners.	Stabilise cooperation over time; punish exploiters without changing base rewards. Often require repeated interactions and richer population structure; more complex to model and train.

Table 4: High-level comparison of the four mechanism classes for engineering cooperation.

From the perspective of our metrics in Section 6.1, we can roughly characterise when each mechanism class is most effective:

- If social welfare is low because the base game itself is hostile to cooperation (for example, over-exploitation is individually optimal), payoff and environment design can reshape the incentives so

that cooperative outcomes are at least locally attractive.

- If social welfare is high but unstable, or if agents struggle to discover cooperative policies despite aligned rewards, architectural mechanisms for credit assignment can provide the missing learning signals that connect individual actions to collective outcomes.
- If agents fail to coordinate in the face of partial observability and local information, information and communication mechanisms can bridge the gap by allowing agents to share observations, intentions, or plans.
- If cooperation emerges but is easily exploited or eroded over time, reciprocity, reputation, and partner choice can embed cooperation into the long-run dynamics of the population, making it robust to short-sighted defection.

In practice, effective cooperative systems often combine several of these mechanisms. For example, a cooperative MARL system might use a shared team reward with mild shaping (Mechanism I), a value-decomposition architecture such as QMIX (Mechanism II), a centralised critic during training (Mechanism III), and an environment where agents meet repeatedly and can adjust their behaviour based on past interactions (Mechanism IV). The exact mix depends on the domain, the available information, and the designer's control over the environment.

The next chapter turns from theory to practice. We will instantiate a subset of these mechanisms in simple experimental environments and measure their impact on social welfare, cooperation rates, and stability over time. This will allow us to test, in concrete settings, how far these ideas can take us towards reliable cooperation in multi-agent reinforcement learning.

7 Experiment: Empirical Analysis of Cooperation Mechanisms

The theoretical chapters argued that cooperation in multi-agent reinforcement learning can be encouraged through payoff shaping, credit assignment, information/communication, and population-level mechanisms (see Section 6). Here we instantiate one representative mechanism for three of those classes (payoff shaping, credit assignment, and long-run incentives/reciprocity)—plus a self-interested baseline—inside a single benchmark environment, SocialJax *Cleanup*, to see how they behave empirically when everything else is held fixed.

Concretely, we keep the underlying game dynamics and on-policy optimisation method the same across all conditions and vary only the cooperation mechanism. This controlled setup allows us to pursue three specific objectives: (1) to demonstrate the emergence (or failure) of cooperation under a purely self-interested baseline; (2) to compare representative mechanisms from payoff shaping, credit assignment, and reciprocity to determine which successfully break the dilemma in *Cleanup*; and (3) to relate these empirical outcomes back to the theoretical framework established in Section 6.

7.1 Environment: SocialJax Cleanup

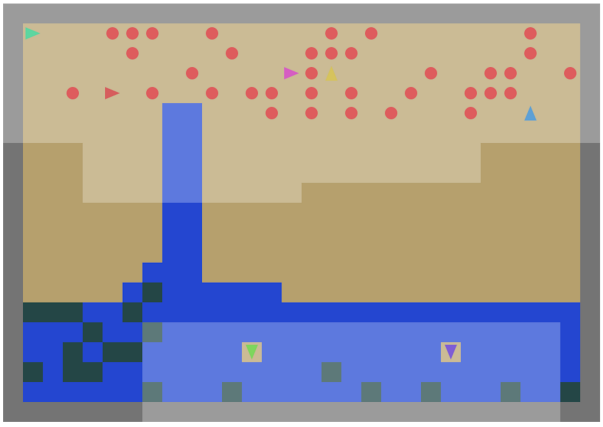

Figure 4: A frame from the SocialJax Cleanup environment $N = 7$ learning agents.

All experiments use the JAX-based SocialJax [28] implementation of *Cleanup*, a sequential social dilemma with $N = 7$ learning agents, a river that accumulates waste, and an apple orchard. At each timestep, all agents act simultaneously; the environment then transitions.

Key incentives:

- **Collecting apples** yields +1 reward when an agent steps on an apple.
- **Cleaning waste** gives *no direct reward*, but reduces pollution in the river. If the river is clean, apples respawn; if not, respawn drops to zero.

Each agent observes an egocentric crop of the grid (apples, waste, walls, other agents). Actions include movement, head rotation, and a "clean" beam. We keep the default Cleanup map and respawn parameters. Step counts are in *environment steps* (one joint action of all seven agents).

7.2 Agents and Learning Setup

7.2.1 Policy architecture

Across conditions $C0$–$C3$ we use shared-parameter independent actor–critics. Each agent's egocentric observation is a $(H \times W \times C)$ tensor encoding local grid features (apples, waste, walls, agents, etc.). This is processed by a convolutional network followed by a small multilayer perceptron:

- Three convolutional layers with ReLU activations produce a spatial embedding of the local crop.
- The embedding is flattened and fed through a fully connected layer to produce a shared feature vector.

- An *actor head* maps the feature vector to logits over the discrete action set; the resulting categorical policy $\pi_\theta(a \mid o)$ is implemented using `distrax`.
- A *critic head* maps the same feature vector to a scalar value estimate $V_\phi(o)$.

In $C0$–$C3$, parameters are fully shared across the seven agents: a single set of weights is applied independently to each agent's observation. This reduces parameter count and exploits the exchangeability of agents in Cleanup. No recurrent memory is used in these conditions; policies are purely reactive to the current egocentric observation.

In $C4$ we introduce a small asymmetry: one "innovator" agent and six "imitator" agents each have their own shared-parameter actor–critic (one policy/value network for the innovator, one shared across imitators), but both networks use the same convolutional backbone and head structure as above. This lets us implement different reward functions for innovators and imitators while keeping architectural capacity fixed.

All convolutional and dense layers use orthogonal initialisation and bfloat16 activations (with float32 parameters), matching the SocialJax training code [28]. This choice reduces GPU memory without materially changing performance.

7.2.2 Training algorithm

Training is on-policy proximal policy optimisation (PPO) in the "IPPO" style. For each condition, and for each training run:

1. **Rollout collection.** We run N_{env} parallel environments for T steps using the current shared policy to obtain trajectories $\{(o_t, a_t, r_t, \log \pi_\theta(a_t \mid o_t), V_\phi(o_t))\}$ for all agents. This yields a batch of size $N_{\text{env}} \times T \times N$.
2. **Advantage estimation.** For each agent and timestep we compute the generalised advantage estimate
$$\hat{A}_t = \sum_{l=0}^{L-1} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\phi(o_{t+1}) - V_\phi(o_t),$$
with fixed discount factor γ and GAE parameter λ. Returns $\hat{R}_t$ are defined as $\hat{A}_t + V_\phi(o_t)$.
3. **Policy and value updates.** The rollout batch is shuffled and split into M minibatches. For each PPO epoch, we optimise the clipped surrogate objective
$$L^{\text{CLIP}}(\theta) = \mathbb{E}\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big],$$
where $r_t(\theta) = \exp(\log \pi_\theta(a_t \mid o_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid o_t))$, together with a squared-error value loss $(\hat{R}_t - V_\phi(o_t))^2$ and an entropy bonus to encourage exploration. Optimisation uses Adam with optional learning-rate and reward-shaping schedules implemented in the code.

For the IPPO and SVO conditions (C0–C2), core optimisation hyperparameters (learning rate, total batch size, number of PPO epochs, clip parameter ϵ, entropy coefficient, rollout horizon T) are shared. For MAPPO (C3), we use a slightly different set of optimisation hyperparameters—notably a smaller number of parallel environments, shorter rollouts, and a larger learning rate—to accommodate the computational overhead of the centralised critic and ensuring stability. While this deviation from a strictly identical hyperparameter set implies that performance differences could be partially attributed to tuning, preliminary experiments indicated that enforcing the IPPO configuration on MAPPO led to instability, while applying MAPPO's aggressive updates to IPPO prevented convergence. Thus, we prioritise comparing each algorithm under its own "reasonably tuned" regime to fairly evaluate the mechanism's potential. Across all conditions we keep the discount factor γ, GAE parameter λ, and total number of environment steps constant.

We note that the original implementation of the reciprocity mechanism by Eccles et al. [27] utilised A3C [29], whereas we employ PPO. As we discuss in the results, this choice of optimiser may impact the exploration dynamics required to "kickstart" the reciprocity cycle.

MAPPO reuses this loop but augments it with a separate centralised critic, as described below. In MAPPO, the policy update still uses the PPO clipped objective, but advantages and targets are computed using a value function $V(s_t)$ that depends on the global state rather than local observations.

For MAPPO, the critic ingests a global world-state tensor constructed by stacking all agents' egocentric views into a single feature map. This world state captures the joint spatial configuration of apples, waste, and agents. The centralised critic $V_\psi(s_t)$ provides a single baseline shared across actors; this reduces gradient variance and provides denser learning signals when multiple agents jointly influence outcomes (e.g., when multiple cleaners keep the river clean). In contrast, IPPO and SVO rely solely on per-agent critics $V_\phi(o_{i,t})$ from local observations, which can mis-attribute credit when cleaning by one agent primarily benefits others.

7.3 Mechanism Conditions

We evaluate five conditions, labelled $C0$–$C4$. The base algorithms for IPPO (C0, C1), SVO (C2), and MAPPO (C3) are adapted from the SocialJax suite [28]; we performed additional hyperparameter tuning and enhanced logging for this study. The reciprocity mechanism in C4 is a custom extension implemented on top of the IPPO backbone.

C0: Self-interested baseline (IPPO, individual reward). Each agent receives its own raw per-timestep reward from the environment: $r_i(t) = 1$ when agent i consumes an apple and 0 otherwise; cleaning has no direct reward. The shared-parameter IPPO architecture is trained exactly as described above, using local critics $V_\phi(o_{i,t})$ and no cross-agent shaping or centralised components. This condition serves as the "solitary learner" baseline inside the Cleanup dilemma.

C1: Payoff shaping via team reward (IPPO, shared/common reward). C1 keeps the architecture and optimiser identical to C0 but replaces the reward signal with a team-average ("common") reward. At each timestep,

$$r_i^{\text{team}}(t) = \frac{1}{N} \sum_{j=1}^{N} r_j(t),$$

so that all agents receive the same scalar reward equal to the mean number of apples collected across the group. This is equivalent to training each policy to maximise a single shared return $W = \sum_{j,t} r_j(t)$ under independent exploration. There is *no* additional shaping term tied explicitly to cleanliness; the only difference from C0 is that local variations in reward are pooled across agents. This tests whether simple payoff pooling is enough to align incentives with social welfare.

C2: Social Value Orientation (SVO). C2 again uses the same shared-parameter IPPO architecture, but replaces the raw reward $r_i(t)$ with an SVO-weighted mixture of own and others' rewards. Let $\theta \in [0°, 90°]$ denote the SVO angle and $r_{-i}(t)$ denote the average reward of the other agents at time t. The SVO-mixed reward has the generic form

$$r_i^{\text{SVO}}(t) = \cos(\theta)\, r_i(t) + \sin(\theta)\, r_{-i}(t),$$

implemented via the `SVOLogWrapper` in SocialJax [28]. Small angles $\theta \approx 0°$ approximate pure self-interest (close to C0); larger angles put increasing weight on others' rewards. In the degenerate altruistic limit $\theta = 90°$, only others' rewards matter. By varying θ and the associated SVO weights configured in `svo_cnn_cleanup.yaml`, C2 explores a continuum between egoistic and altruistic preferences while keeping the rest of the learning pipeline fixed.

C3: Centralized critic (MAPPO). C3 uses a centralised critic together with a shared team reward. At the reward level it is close to C1: the environment is configured with *shared rewards*, so each actor receives the same team-based scalar reward at each timestep (proportional to the group's apple collection). The actor network is shared across agents and maps local observations $o_{i,t}$ to a categorical policy $\pi_\theta(a_{i,t} \mid o_{i,t})$ as before. In addition, a separate critic network $V_\psi(s_t)$ takes as input a global world-state tensor constructed by stacking all agents' local views into a single feature map. This critic is trained to approximate the joint value of the current global state under the joint policy. During learning, advantages for all agents are computed using this centralised value baseline; at execution time, only the decentralized actors are used. We select MAPPO over value-decomposition methods like QMIX [15] to maintain algorithmic consistency: since conditions C0, C1, C2, and C4 rely on PPO, using MAPPO allows us to isolate the effect of the *centralised critic* without introducing the confounding factors associated with off-policy value-based learning (e.g., replay buffers and target update schedules). C3 therefore combines team-based payoff design with improved credit assignment via a centralised critic.

C4: Reciprocity via innovator–imitator dynamics (Class IV). C4 instantiates the "metric-matching" reciprocity mechanism proposed by Eccles et al. [27]. We keep the underlying Cleanup payoffs unchanged,

but break the symmetry between agents: one *innovator* agent learns purely from its own environment reward, while the remaining six *imitator* agents receive an additional intrinsic reward that encourages them to match the innovator's level of pro-social behaviour over time.

Concretely, let $n_i(t) \in \{0, 1\}$ indicate whether agent i performs a *cleaning* action at timestep t (firing the clean beam on river waste). For each agent we maintain a decayed "niceness" trace

$$N_i(t) = \gamma_n N_i(t-1) + n_i(t),$$

where $\gamma_n \in (0, 1)$ (e.g., $\gamma_n = 0.95$) discounts older cleaning actions so that recent contributions to the public good matter more. The innovator's learning reward is simply its raw environment reward $r^{\text{env}}_{\text{inv}}(t)$, as in C0.

Each imitator k instead learns from a reward

$$r^{\text{tot}}_{\text{im},k}(t) = r^{\text{env}}_{\text{im},k}(t) + \lambda_{\text{im}}\big(-(N_{\text{im},k}(t) - N_{\text{inv}}(t))^2\big),$$

where $N_{\text{inv}}(t)$ is the innovator's niceness trace and $\lambda_{\text{im}} > 0$ scales the intrinsic reciprocity term. This intrinsic reward is maximised when imitators' niceness levels track the innovator's: if the innovator cleans frequently, imitators are encouraged to do likewise; if the innovator defects and stops cleaning, imitators are punished for being "too nice" in comparison and reduce their own cleaning. Both the innovator and imitators are trained with IPPO (separate shared-parameter actor–critics for each role) using the same optimisation hyperparameters. The base environment dynamics and external rewards are identical to C0; only the imitators' intrinsic rewards implement reciprocity.

Communication channels (Mechanism Class III) are excluded from the current experiment set due to scope constraints. While communication is vital for tasks with severe partial observability (e.g., blind coordination games), the *Cleanup* task is primarily a social dilemma driven by incentive misalignment. We therefore focus on mechanisms that align incentives (Class I), assign credit (Class II), and stabilise cooperation through reciprocity (Class IV).

Key contrasts. C0 vs. C1 isolates *payoff design*: identical networks and optimiser, but one uses private reward and the other uses the team-average reward. C1 vs. C2 isolates *how* reward sharing is specified: C1 enforces full pooling, while C2 smoothly interpolates between egoistic and altruistic mixes via the SVO angle (e.g., a 45° angle weights self and others equally; 90° is fully other-regarding). C1 vs. C3 then isolates *credit assignment* under a shared payoff: both use team-based rewards, but C3 adds a centralised critic over the global state while actors remain decentralized at execution. Finally, C0 vs. C4 isolates *long-run reciprocity*: both keep the underlying Cleanup payoffs and use IPPO backbones, but C4 introduces an asymmetry between a purely self-interested innovator and reciprocity-driven imitators whose intrinsic rewards depend on matching the innovator's past cleaning behaviour. All conditions share the same convolutional backbone; IPPO, SVO, and C4 also share PPO hyperparameters, while MAPPO uses a slightly different configuration (smaller parallel batch, shorter rollouts, higher learning rate) tuned for stability under the centralised critic and GPU memory limits.

7.4 Training Protocol

For each condition $C0$–$C4$ we train for a fixed budget of environment interaction and run multiple seeds to gauge variability.

- **Timesteps.** Each run trains for T_{train} environment steps (joint actions). In the main experiments this corresponds to 10^8 joint actions for every condition. IPPO (C0–C1), SVO (C2), and the reciprocity condition (C4) use $N_{\text{env}} = 128$ parallel environments with rollout length $T = 1000$, while MAPPO (C3) uses fewer parallel environments and shorter rollouts ($N_{\text{env}} = 16$, $T = 100$) to satisfy GPU memory constraints; in all cases the total interaction budget T_{train} is held fixed.
- **Seeds.** K seeds per condition (e.g., $K = 3$), with distinct network and environment RNG initialisations. Seeds are propagated through both JAX PRNGs and environment resets to ensure independent training runs.
- **Hardware.** Single Nvidia A100 40GB GPU with CPU for env simulation; runs complete in hours.
- **Logging.** Weights & Biases logs training curves, configs, checkpoints, and env stats; runs are tagged by condition and seed. Config files are version-controlled.

Hyperparameters for each algorithm were selected through light tuning on the Cleanup task. IPPO (C0–C1), SVO (C2), and the reciprocity condition (C4) share a common PPO configuration; in C4 the innovator and imitator actor–critics use the same optimisation settings and differ only in their reward definitions (presence or absence of the intrinsic reciprocity term). MAPPO (C3) uses its own configuration that trades off batch size, rollout length, and learning rate to achieve stable learning under the centralised critic without exceeding GPU memory. In all cases, we avoid extensive per-condition hyperparameter sweeps and instead aim for "reasonably tuned" settings, so that the comparison focuses on the qualitative effect of the cooperation mechanisms rather than exhaustive optimisation.

Evaluation uses the final checkpoint (or periodically saved checkpoints) and runs a greedy policy (no exploration noise) for a small number of episodes per condition. In the current implementation, each evaluation call runs a single episode of fixed length (equal to the GIF horizon) per seed; when reporting final results we aggregate metrics over these evaluation episodes and seeds.

7.5 Evaluation Metrics

To operationalise cooperation in Cleanup we track:

1. **Social welfare.** For each evaluation episode we compute the total undiscounted sum of individual rewards,
$$W = \sum_{i=1}^{N} \sum_{t=0}^{T-1} r_i(t), \tag{44}$$
and report its mean over episodes and seeds. In practice we also track apples-based proxies recorded in the logs: `eval/team_apples` and `eval/apples_total`, which count the total number of apples collected (normalised per agent) using explicit apple-count information from the environment. These are numerically more stable and directly interpretable as "apples per agent per episode".

2. **Cooperation rate.** Cleaning actions (firing the clean beam) are treated as the canonical cooperative behaviour. During training, we compute a rollout-level cooperation rate
$$\bar{C}_{\text{train}} = \frac{1}{N_{\text{env}} T N} \sum_{e,t,i} \mathbb{I}\{\text{agent } i \text{ cleans at } (e,t)\},$$
which is logged as `train/clean_action_rate`. During evaluation, we compute an analogous per-episode rate `eval/clean_action_rate`. High rates of cleaning are necessary to keep the river clean and sustain apples, but excessive cleaning without harvesting is also suboptimal; the joint behaviour of welfare and cooperation rate is therefore of primary interest. In the reciprocity condition C4 we additionally log separate cleaning rates for the innovator and imitators, as well as a simple "niceness gap" statistic based on differences between their decayed cleaning traces $N_{\text{inv}}(t)$ and $N_{\text{im},k}(t)$ at the end of each episode, to quantify how closely imitators track the innovator's pro-social behaviour.

3. **Equality of returns.** We measure the Gini coefficient of per-agent returns in an episode. If *Social Welfare* measures the "size of the pie" (total productivity), the *Gini index* measures how that pie is divided (fairness). This distinction is crucial for identifying high-performing but exploitative policies.

4. **Stability over training.** Trajectories of welfare and cooperation rate versus training timesteps reveal whether cooperation, once discovered, is maintained or collapses back to free-riding. We track both training-time proxies (e.g., `train/apples_total`, `train/clean_action_rate`) and periodic evaluation metrics (`eval/team_apples`, `eval/team_return`) over the course of learning.

5. **Cleaning Efficiency.** To distinguish between effective work and "gaming" the metric, we define cleaning efficiency as the ratio of waste removed to cleaning actions taken: $\rho_{\text{eff}} = \frac{\text{Cleaned Water}}{\text{Clean Action Rate}}$. A high ratio implies agents are positioning themselves to clear multiple waste cells per beam, whereas a low ratio suggests firing into walls, open space, or where no waste exists.

Apples-based metrics in logs use explicit apple counts emitted by the environment: `apples_total` is normalised by N (sum divided by number of agents), and `apples_per_agent` gives per-agent averages. Clean-action counts are similarly normalised by rollout length and actor count.

Qualitative rollouts (videos) will be collected for representative runs to interpret quantitative trends (e.g., division of labour, free-riding, over-cleaning).

7.6 Results

We analyse the performance of the cooperation mechanisms through three lenses: (1) **learning dynamics**, examining the trajectories of social welfare and cooperation over time; (2) **asymptotic performance**, comparing final welfare, efficiency, and equality across conditions; and (3) **behavioural inspection**, using qualitative observations to explain emergent patterns such as role specialisation or free-riding. We will relate these empirical findings back to the theoretical expectations outlined in Section 6.

We ran over forty training runs in total. Twenty of those runs were completed in full (roughly three–four seeds per condition) and were used to construct the figures and numerical summaries below; the remainder were short preliminary sweeps for hyperparameter tuning and debugging. Unless otherwise noted, training curves plot the mean across seeds and a shaded region indicating across-seed variation.

7.6.1 Learning Dynamics Across Conditions

Figure 5 shows the evolution of social welfare, measured as the number of apples collected per agent during training. Centralised MAPPO with a shared team reward (C3) clearly dominates the other mechanisms on this metric. After an initial exploration phase, C3 enters a regime of steadily increasing welfare and reaches an average of 2.84 apples per agent at the end of training. However, we observe noticeably higher variance across seeds for MAPPO compared to the decentralized baselines (C1, C2). This volatility is likely a side-effect of the centralised critic: early divergence in the approximation of the global value function can lead to distinct policy trajectories before the system converges to a high-welfare equilibrium. In contrast, SVO-regularised IPPO (C2) and team-reward IPPO (C1) achieve intermediate but substantial performance (1.21 and 0.86 apples per agent respectively) with tighter confidence intervals, while both the self-interested baseline (C0) and reciprocity mechanism (C4) fail to break the social dilemma, finishing with averages of only 0.035 (C4) and 1.8×10^{-5} (C0).

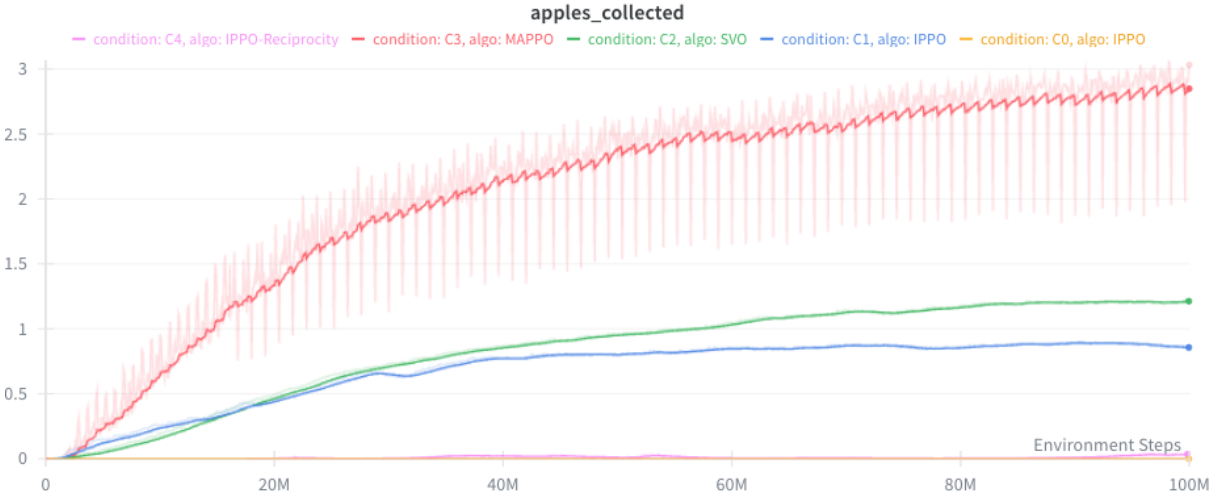

Figure 5: Apples collected per agent during training for each condition $C0$–$C4$. Centralised MAPPO with a team reward (C3) achieves the highest social welfare, followed by SVO shaping (C2) and team-reward IPPO (C1). Both the self-interested baseline (C0) and reciprocity mechanism (C4) collect almost no apples.

To understand how these welfare differences arise, Figures 6 and 7 plot the dynamics of cleaning behaviour. Team-reward IPPO (C1) exhibits the highest *cleaning action rate* (Figure 6), peaking at well over 250 clean actions per rollout before gradually declining to around 150 by the end of training. SVO (C2) shows a similar but slightly damped pattern, while MAPPO (C3) uses fewer cleaning actions overall: its cleaning rate peaks earlier and then steadily decreases to around 75 actions per rollout. Despite this relatively modest rate of cleaning, C3 achieves by far the largest amount of *effective* cleaning, as measured by the "cleaned water" statistic in Figure 7. This suggests that the centralised critic is able to guide agents towards more targeted and coordinated cleaning patterns that keep the river clear with fewer redundant actions.

The self-interested baseline C0 performs little cleaning and quickly converges to a low, nearly flat level of both cleaning rate and cleaned water. The near-zero apple consumption (10^{-4}) highlights the *hard exploration* challenge in Cleanup: apples only spawn if the waste level is low. Without shaped rewards, independent learners fail to discover the cleaning beam's utility before the river becomes saturated and apple spawning ceases entirely. Thus, although agents continue to explore spatially, the environment remains in a collapsed tragedy-of-the-commons state where no resources exist to be collected.

The reciprocity condition C4 behaves differently, revealing a distinct pathology. While its raw *cleaning action rate* increases over time to levels above C1 and C2 (approx. 130 actions per rollout), the actual *cleaned water* remains significantly lower (finishing at $\approx$ 60 vs. 130 for MAPPO). This discrepancy indicates a collapse in **cleaning efficiency**: imitators learn to fire the cleaning beam to satisfy the reciprocity metric (matching the innovator's firing rate), but they fail to aim correctly. Qualitative inspection confirms that C4 agents often fire the beam while standing on land or facing empty space. Because the intrinsic reciprocity reward is tied to the *action* $n_i(t)$ rather than the *outcome* (waste removed), agents effectively "game" the metric, producing the appearance of cooperation without the environmental impact.

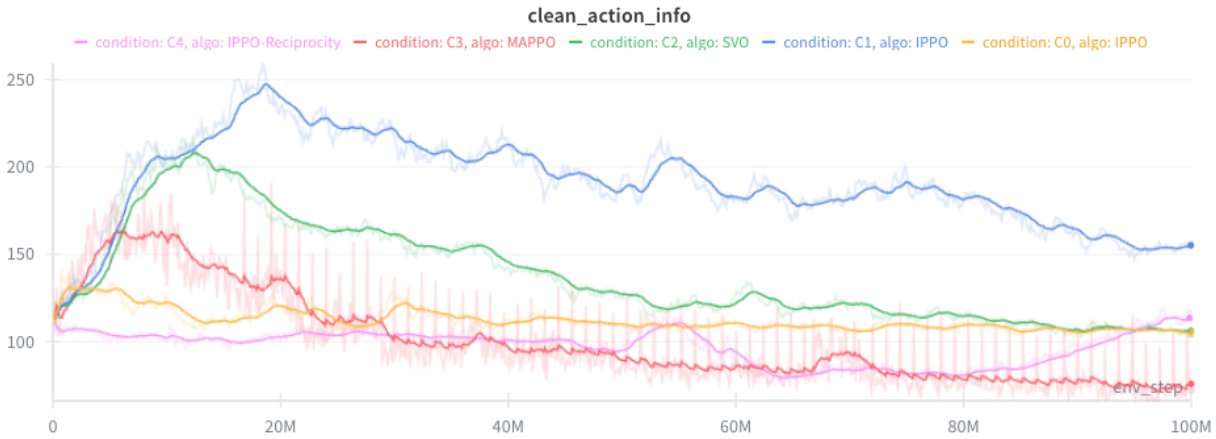

Figure 6: Average number of cleaning actions per rollout during training. Team-reward IPPO (C1) cleans most frequently, followed by SVO (C2). MAPPO (C3) converges to a lower but still substantial cleaning rate, while C0 and C4 show comparatively modest cleaning frequencies.

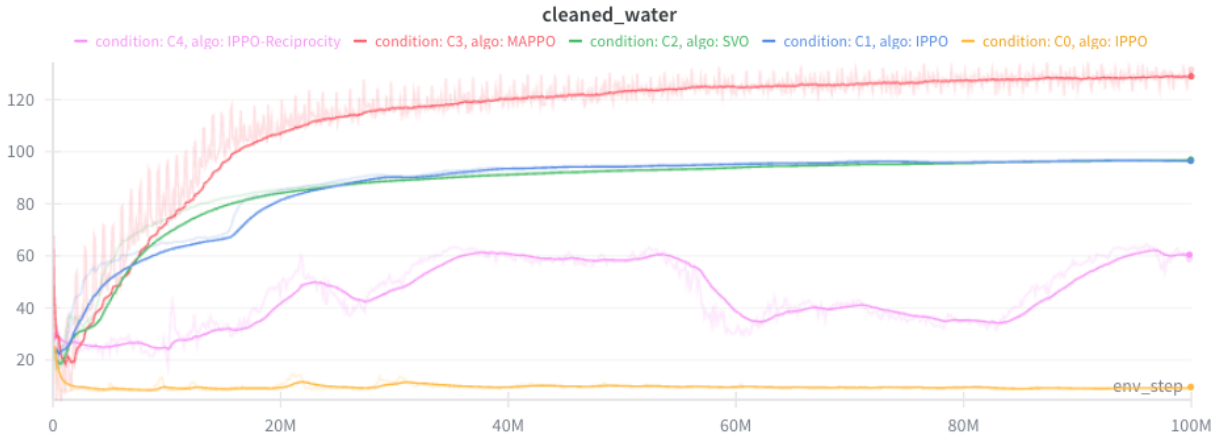

Figure 7: Amount of water cleaned per rollout during training. MAPPO (C3) achieves the highest level of effective cleaning, followed by SVO (C2) and team-reward IPPO (C1). The self-interested baseline (C0) keeps the river almost permanently polluted, while the reciprocity mechanism (C4) attains only moderate levels of cleanliness despite substantial cleaning effort.

These trends are consistent with the evaluation-time metrics shown in Figure 8. When we freeze the final policies and run greedy evaluation episodes, MAPPO (C3) again attains the highest team reward,

followed by SVO (C2) and the team-reward IPPO baseline (C1). The reciprocity mechanism (C4) performs only slightly better than the self-interested baseline in terms of aggregate apples collected, despite exhibiting a relatively high cleaning-action rate at evaluation time.

(a) Team apples (b) Cleaned water (c) Clean action rate

Figure 8: Evaluation metrics for each condition at the end of training. MAPPO (C3) achieves the highest team apples and cleaned water. SVO (C2) and team-reward IPPO (C1) attain intermediate performance. The reciprocity mechanism (C4) exhibits a relatively high cleaning rate but fails to convert this into apples; the self-interested baseline (C0) performs poorly on all metrics.

Finally, we note that the computational cost of MAPPO is substantially higher than that of the IPPO-based mechanisms. A typical MAPPO training run (C3) takes around 2 hours 12 minutes on an Nvidia A100 40GB GPU, compared to roughly 43 minutes for IPPO, SVO, and reciprocity runs (C0, C1, C2, and C4). Thus, while centralised critics deliver the highest welfare, they also incur a significant runtime overhead.

7.6.2 Social Welfare and Cooperation Rates

From a social-welfare perspective, the mechanisms can be ordered roughly as

$$\text{C3 (MAPPO)} \;>\; \text{C2 (SVO)} \;>\; \text{C1 (team reward)} \;>\; \text{C4 (reciprocity)} \approx \text{C0 (self-interested)}.$$

The centralised critic in C3 almost completely resolves the social dilemma in Cleanup: agents learn to maintain a clean river and to exploit the resulting apple respawns effectively, leading to high and stable team returns. SVO shaping (C2) also improves social welfare substantially over the purely self-interested baseline, confirming that relatively simple payoff shaping can already encourage more cooperative behaviour.

The team-reward IPPO condition (C1) lies slightly behind SVO in terms of apples collected, despite cleaning more frequently on average. This suggests that naively pooling rewards is not enough: without improved credit assignment, agents may "over-clean" or fail to coordinate cleaning with harvesting, wasting effort on actions that have little marginal benefit.

The reciprocity mechanism (C4) is the most disappointing from a welfare viewpoint. While Eccles et al. [27] successfully demonstrated cooperation using this mechanism, our results highlight a critical limitation of the Innovator-Imitator model: its *brittleness*. Unlike Centralized Critics (C3), which provide a direct gradient toward cooperation, Imitative Reciprocity relies on the Innovator stumbling upon the cooperative strategy via exploration. If the Innovator converges to a selfish policy too quickly (a risk with PPO's more aggressive policy updates compared to the A3C algorithm used in the original paper), the Imitators lock into a defecting equilibrium. This failure to replicate the success of Class IV mechanisms under a different optimisation regime suggests that while they are theoretically powerful, they may be less robust to architectural changes than Class I or II mechanisms.

7.6.3 Equality and Stability of Cooperative Outcomes

To complement aggregate welfare metrics, Figure 9 plots the Gini index of per-episode environment returns during training.[1] Lower values indicate more equal sharing of returns across agents. Based on

[1] Gini is computed over undiscounted per-agent episode returns, using only the raw environment reward signal.

the final training values, the mechanisms are ordered (from *highest* to *lowest* Gini) as:

$$\text{C4 } (0.94) \;>\; \text{C0 } (0.40) \;>\; \text{C3 } (0.36) \;>\; \text{C2 } (0.19) \;>\; \text{C1 } (0.18).$$

The reciprocity mechanism (C4) exhibits near-maximal inequality (0.94). However, this must be interpreted in the context of the extremely low total welfare observed in Condition C4. Rather than indicating a stable hierarchy where one agent monopolises a rich resource, this high Gini index is likely an artifact of scarcity. In a regime where almost zero apples are produced, if the Innovator (who pays no intrinsic cleaning cost) stumbles upon even a single apple while Imitators receive zero, the calculated inequality is maximized. Thus, the failure of C4 is not that it produces a tyranny, but that it fails to produce enough resources to be shared at all.

The self-interested baseline (C0) similarly settles at a high Gini of 0.40, but with extreme variance throughout training. Like C4, this is a numerical artifact of the "tragedy of the commons" collapse. Because the mean return (the denominator in the Gini formula) approaches zero, slight random variances in apple consumption—one agent eating an apple while others starve—result in massive relative differences, causing the index to oscillate wildly rather than reflecting a stable distribution of resources.

MAPPO (C3) achieves both high welfare and moderate inequality (0.36). Unlike C4 and C0, this inequality represents a functional *division of labour*. For the group to thrive, some agents must specialise in cleaning (incurring opportunity costs) while others specialise in harvesting. MAPPO's centralised critic successfully orchestrates this specialisation, but because the Gini index is calculated on *raw* environmental rewards (where cleaning yields zero), this efficient role allocation manifests as statistical inequality.

Conversely, the payoff-shaping mechanisms yield the lowest Gini indices: 0.19 for SVO (C2) and 0.18 for Team Reward (C1). These low values correlate with the lower *efficiency* ratios observed in Section 7.6.4. Lacking the coordination to support specialisation, these agents adopt homogeneous "spray and pray" policies where everyone cleans a little and eats a little. While this symmetry results in high equality, it prevents the emergence of the specialized roles required to reach the Pareto frontier of production. Thus, C1 and C2 are more equal not because they are "fairer" by design, but because they fail to discover the highly productive but unequal strategies found by MAPPO.

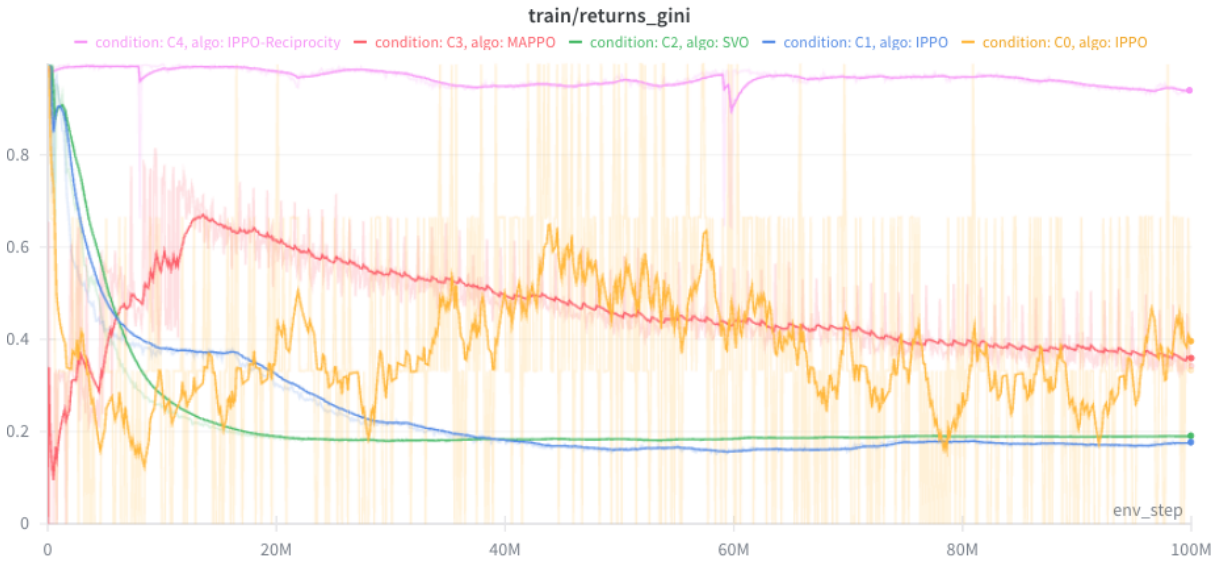

Figure 9: Gini index of per-episode environment returns during training. Lower values indicate more equal sharing of returns. Reciprocity (C4) remains highly unequal throughout; the self-interested baseline (C0) also exhibits relatively high inequality. MAPPO (C3) attains moderate inequality alongside high welfare, while SVO (C2) and team-reward IPPO (C1) yield the most equal outcomes.

In terms of *stability*, the cooperative regimes discovered by C1–C3 are largely persistent: once high levels of cleaning and apple harvesting are established, they are maintained for the remainder of training, with only minor fluctuations. The reciprocity condition (C4), by contrast, shows more variability in both cleaning behaviour and inequality, without ever converging to a clearly cooperative fixed point.

7.6.4 Cleaning Efficiency Analysis

The disconnect between effort and result in C4 motivates a closer look at *cleaning efficiency*. Table 5 presents the final training metrics for cleaning actions, water cleaned, and their ratio.

Condition	**Clean Action Rate** (actions/rollout)	**Cleaned Water** (units/rollout)	**Efficiency Ratio** ($\frac{\text{Water}}{\text{Actions}}$)
C3 (MAPPO)	76.11	128.96	**1.69**
C2 (SVO)	106.26	96.89	0.91
C1 (Team Reward)	155.11	96.52	0.62
C4 (Reciprocity)	113.48	60.42	0.53
C0 (Self-Interested)	105.61	9.68	0.09

Table 5: Cleaning efficiency metrics at the end of training. The efficiency ratio measures the average amount of waste removed per cleaning action. C3 achieves the highest efficiency (¿1 implies clearing multiple waste cells per beam), while C1 relies on a high volume of less efficient actions.

This metric neatly categorises the mechanisms:

- **Targeted Cooperation (C3):** MAPPO agents achieve a remarkable efficiency ratio of 1.69. By using a centralised critic that observes the spatial distribution of waste, agents learn to position themselves where a single beam can clear multiple waste cells simultaneously, maintaining the public good with minimal action cost.
- **Aligned vs. Brute-Force Shaping (C2 vs. C1):** Interestingly, SVO (C2) is significantly more efficient (0.91) than simple Team Reward (C1, 0.62). While both achieve similar levels of total cleaned water (≈ 96), C1 requires nearly 50% more cleaning actions to do so. This suggests that the smooth interpolation of rewards in SVO helps agents preserve individual survival instincts (avoiding wasteful actions) better than the blunt pooling of C1, which encourages a "spray and pray" approach.
- **Performative Cooperation (C4):** The reciprocity agents display poor efficiency (0.53), comparable to the brute-force baseline. Despite a high action rate (113), the actual waste removed is low (60). This confirms that the breakdown in C4 is a misalignment of the intrinsic reward: imitating the *act* of cleaning is not equivalent to imitating the *utility* of cleaning.

7.7 Discussion: The Gap Between Incentive and Ability

These experiments provide a concrete test of the mechanisms proposed in Section 6. By keeping the environment and base algorithms fixed, we can isolate exactly what payoff design, credit assignment, and reciprocity bring to the table. Three key insights emerge from the data.

Alignment is not the same as competence. The failure of the self-interested baseline (C0) confirms that *Cleanup* is a genuine dilemma: without intervention, agents simply destroy their own resources. However, the comparison between the Team Reward (C1) and MAPPO (C3) is more revealing. Both conditions gave agents the exact same cooperative incentive. Yet, C1 agents struggled to translate that shared goal into effective action, settling for a clumsy "spray and pray" strategy. It was only when we introduced the centralised critic in C3—a Mechanism Class II intervention—that agents could figure out *how* to clean efficiently. This suggests that fixing the rewards (Mechanism Class I) only answers *why* agents should cooperate. Without the architectural support to untangle credit assignment, even perfectly well-intentioned agents can fail to coordinate.

Intrinsic rewards are easily gamed. The failure of the reciprocity mechanism (C4) offers a cautionary tale. While the theory behind the Innovator-Imitator model is sound, translating social concepts like "fairness" into scalar rewards creates loopholes. The C4 agents learned to satisfy the strict metric (matching the cleaning rate of the leader) while ignoring the actual goal (removing waste), effectively "cleaning" empty space to maximize their reward. This is a classic instance of Goodhart's Law: once a specific proxy for cooperation becomes the target, agents will find the easiest, rather than the most useful, way to satisfy it.

Equality does not mean optimality. Finally, our results challenge the intuitive link between fairness and cooperation. The most successful mechanism (C3) produced outcomes that were actually quite *unequal* in terms of raw returns. This wasn't because of exploitation, but because of specialisation. To maximize social welfare in *Cleanup*, the group needs a division of labour—some agents must clean while others harvest. Mechanisms that inadvertently force everyone to behave identically (like the simple Team Reward in C1) prevent this specialisation, leading to a fairer but much poorer society.

Ultimately, these results show that engineering cooperation requires more than just tweaking incentives. While shaping rewards can break the immediate deadlock of a social dilemma, achieving robust, efficient cooperation requires building the capacity for coordination into the learning architecture itself.

8 Conclusion

This thesis began with the problem: how do we take reinforcement learning agents—originally designed to master static, solitary domains—and deploy them into dynamic, strategic worlds? As explored in the theoretical foundations (Sections 2 to 4), the transition from a Markov Decision Process to a Stochastic Game is not merely a change in environment size, but a fundamental shift in the learning problem. Optimality is no longer a property of a policy alone, but of an equilibrium between interacting intelligences.

To navigate this shift, we moved beyond the standard view of MARL as simply "RL with more agents." By synthesising game theory, evolutionary dynamics, and modern deep RL architectures, we constructed a taxonomy of cooperation mechanisms in Section 6. This framework argued that cooperation is not binary; it is the product of four distinct levers: payoff design, credit assignment architectures, information flow, and population dynamics.

Our empirical study in the *Cleanup* environment put these levers to the test, revealing a critical distinction between *incentive* and *capacity*. The failure of the self-interested baseline (C0) confirmed that social dilemmas are fatal to naive learners. However, the partial success of simple team rewards (C1) showed that altruism alone is insufficient. While these agents wanted to cooperate, they lacked the architectural support to do so efficiently, resulting in low-yield "spray and pray" strategies.

The breakthrough came with the introduction of a centralised critic (C3). By solving the credit assignment problem, MAPPO enabled agents to see the causal link between individual actions and collective outcomes, leading to the emergence of specialised roles and high social welfare Crucially, our results challenged the intuitive link between fairness and cooperation: *equality does not mean optimality*. The most successful agents exhibited unequal returns reflecting a functional division of labour, whereas flatter distributions often signalled a failure to coordinate. This validates a central theme of this work: high social welfare did not arise from a moral awakening in the agents, but from a learning architecture that made cooperation learnable. Conversely, the fragility of the reciprocity mechanism (C4)—where agents learned to "game" the intrinsic metric without contributing to the public good—serves as a warning that proxy rewards are no substitute for robust feedback mechanisms.

Limitations and Future Directions These findings must be interpreted within the scope of our setup: a single sequential social dilemma, a fixed population size, and PPO-based learning dynamics. We focused on aligning incentives and architectures (Mechanism Classes I and II) and limited forms of reciprocity (Class IV), leaving explicit communication (Class III) and complex population-based training (e.g., PSRO) for future work.

Final Remarks Ultimately, this thesis suggests that engineering cooperation is less about hoping for emergent pro-social behaviour and more about deliberate system design. We have shown that while payoff design provides the *reason* to cooperate, it is the learning architecture that provides the *means*. As we move toward deploying AI in complex human-machine systems, focusing on these structural levers—specifically how agents assign credit and infer value in mixed-motive settings—will be essential for ensuring that learning agents are not just individually smart, but collectively beneficial.

References

[1] Allan Dafoe et al. "Open Problems in Cooperative AI". In: *arXiv preprint arXiv:2012.08630* (2020).

[2] Vincent Conitzer and Caspar Oesterheld. "Foundations of Cooperative AI". In: *arXiv preprint arXiv:2403.04822* (2024).

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[4] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[5] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3–4 (1992), pp. 229–256. DOI: 10.1007/BF00992696.

[6] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347*. 2017.

[7] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "Multi-agent Reinforcement Learning: An Overview". In: *Innovations in Multi-Agent Systems and Applications – 1*. Ed. by D. Srinivasan Appli and L. C. Jain. Vol. 310. Studies in Computational Intelligence. Springer, 2010, pp. 183–221. DOI: 10.1007/978-3-642-14435-6_7.

[8] John F. Nash. "Non-Cooperative Games". In: *Annals of Mathematics* 54.2 (1951), pp. 286–295. DOI: 10.2307/1969529.

[9] Michael L. Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *Proceedings of the 11th International Conference on Machine Learning*. Morgan Kaufmann, 1994, pp. 157–163.

[10] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer, 2016. DOI: 10.1007/978-3-319-28929-8.

[11] John Maynard Smith and George R. Price. "The Logic of Animal Conflict". In: *Nature* 246.5427 (1973), pp. 15–18. DOI: 10.1038/246015a0.

[12] Karl Tuyls and Simon Parsons. "What Evolutionary Game Theory Tells Us About Multiagent Learning". In: *Artificial Intelligence* 171.7 (2007), pp. 406–416. DOI: 10.1016/j.artint.2006.12.003.

[13] Daan Bloembergen et al. "Evolutionary Dynamics of Multi-Agent Learning: A Survey". In: *Journal of Autonomous Agents and Multi-Agent Systems* 31.5 (2017), pp. 1–46. DOI: 10.1007/s10458-015-9327-9.

[14] Peter Sunehag et al. "Value-Decomposition Networks For Cooperative Multi-Agent Learning". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. arXiv:1706.05296. 2018, pp. 2085–2087.

[15] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4295–4304.

[16] Saeed Kaviani et al. "DeepMPR: Enhancing Opportunistic Routing in Wireless Networks through Multi-Agent Deep Reinforcement Learning". In: (2023). DOI: 10.48550/arXiv.2306.09637. arXiv: 2306.09637 [cs.NI].

[17] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 6379–6390.

[18] Ruize Zhang et al. *A Survey on Self-play Methods in Reinforcement Learning*. 2025. arXiv: 2408.01072 [cs.AI]. URL: https://arxiv.org/abs/2408.01072.

[19] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[20] George W. Brown. "Iterative Solution of Games by Fictitious Play". In: *Activity Analysis of Production and Allocation*. Ed. by Tjalling C. Koopmans. John Wiley & Sons, 1951, pp. 374–376.

[21] Johannes Heinrich and David Silver. "Deep Reinforcement Learning from Self-Play in Imperfect-Information Games". In: *Proceedings of the 2016 AAAI Conference on Artificial Intelligence (AAAI) Workshop on Deep Reinforcement Learning*. arXiv:1603.01121. 2016.

[22] Oriol Vinyals et al. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning". In: *Nature* 575.7782 (2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[23] Marc Lanctot et al. "A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 4190–4203.

[24] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[25] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[26] Jakob Foerster et al. “Counterfactual Multi-Agent Policy Gradients”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018, pp. 2974–2982.

[27] Tom Eccles et al. “Learning Reciprocity in Complex Sequential Social Dilemmas”. In: *arXiv preprint arXiv:1903.08082* (2019).

[28] Zihao Guo et al. *SocialJax: An Evaluation Suite for Multi-agent Reinforcement Learning in Sequential Social Dilemmas*. 2025. arXiv: 2503.14576 [cs.LG]. URL: https://arxiv.org/abs/2503.14576.

[29] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG]. URL: https://arxiv.org/abs/1602.01783.