Intelligent Interaction at Scale

Timon Willi
Exeter College
University of Oxford

Doctor of Philosophy

March 8, 2026

To Brenda, whose love, patience, and support have sustained me throughout this journey—"*For the Quest is achieved, and now all is over. I am glad you are here with me. Here at the end of all things.*" (Tolkien, 1955, p. 228)

Acknowledgements

To my family—my parents, Cornelia and Räto, and my brother, Lukas—who have always provided strength, warmth, and unwavering belief in me. In the words of, again, Tolkien, "*Deep roots are not reached by the frost.*" (Tolkien, 1954, p. 39). Thank you for being my roots.

Multum in parvo, I'd also like to thank all my close collaborators: Chris Lu, Akbir Khan, Johannes Treutlein, Alistair Letcher, Newton Kwan, Andrei Lupu, Pablo Samuel Castro, Karolina Dziugaite, Johan Obando-Ceron, and Matthew Jackson.

I want to express my gratitude to Chris Lu, with whom I have had the privilege and pleasure of sharing my DPhil journey. Even though this journey has ended, I trust the friendship has just started.

To my supervisor, Prof. Jakob Foerster: thank you for seeing a spark where I often saw smoke. In the early years, when I was still auditioning for the role of "hopeless novice", you showed remarkable patience and kept me on the path. When the training wheels should have been off but occasionally wobbled back on, our meetings could grow spirited. Your directness in those moments was always appreciated, constructive, and exactly what I needed. These four years have been my most edifying, equally enlightening, challenging, and occasionally exasperating. I am grateful for every draft you asked me to polish, every question you refused to answer for me, and every idea you insisted could be sharpened one notch further. Whatever clarity appears in these pages is a tribute to your guidance; any remaining rough edges are entirely my own.

To Andrei, thank you for your friendship, the many runs, and at times, patience. To Seb, thank you for being a kindred spirit in humour. To Ola and Marius, thank you for having me on your adventures and for many more in the future. To Ben, thank you for supporting me throughout the job hunting and for the neighbourly desk conversations. To Johannes, thank you for the support in the early days of the DPhil. To Pablo and Karolina, thank you for hosting me at DeepMind and for your invaluable guidance throughout the internship. Thank you, Juliusz and Michal, for the almost philosophical discussions during working hours. Matt and Mikey, thank you for following the bear with me when it called. To Narya, thank you for the emotional support at all times.

To all the Flairies, in the words of Bilbo Baggins, "*I don't know half of you half as well as I should like; and I like less than half of you half as well as you deserve*" (Tolkien, 1954, p. 39).

Also, I am grateful to the staff at Exeter College and the Engineering Department for all the visible and invisible work that made this DPhil possible.

Originality Statement

This thesis presents research conducted between October 2021 and April 2025, culminating in algorithmic and architectural advances for scalable MARL. The core technical contributions are presented across two main parts, with each chapter corresponding to a distinct research paper published during the doctoral studies.

Abstract

Multi-Agent Reinforcement Learning (MARL) aims to develop intelligent agents capable of complex interaction and coordination, but faces significant challenges. Standard single-agent reinforcement learning (RL) algorithms cannot achieve stable, pro-social behaviour in general-sum games, where individual and collective interests may conflict. Simultaneously, scaling learning agents in handling numerous interacting external agents and increasing the internal complexity (parameter count) of individual deep RL agents presents significant challenges; unlike supervised learning, larger deep RL models often fail to yield improved performance.

This thesis addresses these challenges through complementary algorithmic and architectural advances. First, focusing on inter-agent dynamics, we investigate Opponent Shaping (OS). We introduce Consistent Learning with Opponent-Learning Awareness to formally resolve inconsistencies in prior OS methods that inaccurately model co-player adaptation. We further develop Shaper, scaling OS effectively to high-dimensional games with long horizons and demonstrating the benefits of shaping in more complex settings. The open-source JaxMARL library is also presented as a tool to accelerate MARL research.

Second, we explore Mixture-of-Experts (MoE) architectures within deep RL to address the challenge of internal agent complexity and scalability. We demonstrate that MoEs, particularly Soft MoEs, unlock parameter scaling in value-based deep RL. Performance increases with model size, mitigating issues like representational collapse observed in standard architectures. Evaluating MoEs under the amplified non-stationarity of multi-task and continual RL further highlights their capacity for robust learning and specialisation. To motivate future research and keep the multi-agent theme of this thesis, we draw parallels between managing internal expert coordination and external multi-agent challenges.

These contributions advance scalable MARL by providing methods to manage inter-agent influence (OS) and intra-agent complexity (MoE) by improving consistency, scalability, and architectural effectiveness.

Contents

1 Introduction

1.1 Motivation

The landscape of artificial intelligence is increasingly populated by interacting systems. From teams of autonomous vehicles navigating city streets and algorithmic traders competing in financial markets to fleets of coordinating warehouse robots and software assistants collaborating with themselves and with users, the ability of artificial agents to learn, adapt, and interact effectively is paramount (Gronauer and Diepold, 2022). These scenarios demand agents that not only optimise their objectives but also account for, coordinate with, and potentially influence the behaviour of others. Multi-Agent Reinforcement Learning (MARL) is the fundamental framework for analysing and designing such systems to develop agents capable of competent individual and collective behaviour in shared environments. MARL has already impacted diverse fields, demonstrating more efficient, robust, and adaptable autonomous systems tackling problems beyond the reach of single-agent approaches (Silver et al., 2016; Brown and Sandholm, 2017; Vinyals et al., 2019b; Jaderberg et al., 2019).

1.2 Challenges in Learning for Complex Interaction

Despite significant progress in MARL (Silver et al., 2016; Brown and Sandholm, 2017; Rashid et al., 2018; Vinyals et al., 2019a; Jaderberg et al., 2019; Foerster et al., 2019), developing competent interacting agents faces substantial hurdles, which we will broadly categorise into challenges concerning inter-agent dynamics and intra-agent complexity.

First, mastering inter-agent dynamics within MARL is inherently difficult. Agents learning concurrently create a non-stationary environment for one another, destabilising standard reinforcement learning (RL) algorithms. Cooperation, particularly in general-sum or mixed-motive settings where individual incentives may conflict with collective welfare (as exemplified by the Prisoner's Dilemma (Nash, 1951)), often requires mechanisms beyond simple independent learning (Foerster et al., 2018; Letcher et al., 2019b). Moreover, effective credit assignment and coordination complexity grow exponentially more difficult as the number of agents increases, limiting the scalability of many MARL approaches (Gronauer and Diepold, 2022).

Second, the success of modern AI relies on the representational power of deep neural networks (Mnih et al., 2015b; Berner et al., 2019; Vinyals et al., 2019b; Bellemare et al., 2020; Fawzi et al., 2022). However, managing the intra-agent complexity of these models within RL presents unique challenges in scaling (Kaplan et al., 2020a). While supervised learning has benefited from clear scaling laws, where larger models predictably yield better performance, deep RL defies this trend (as detailed further in Section 2.4). Increasing the parameter count of a deep RL agent often leads to performance stagnation or even degradation. This phenomenon is linked to catastrophic forgetting or plasticity loss (difficulty adapting to new information) and parameter underutilisation manifesting as dormant neurons (Ostrovski et al., 2021; Kumar et al., 2021; Lyle et al., 2022; Graesser et al., 2022; Nikishin et al., 2022b; Sokar et al., 2023b; Ceron et al., 2023). These limitations hinder the development of agents with the capacity and adaptability

required for more complex single-agent tasks, let alone sophisticated multi-agent interaction. Intriguingly, using specialised components within a neural network to enhance scaling, like Mixture of Experts (MoE; Shazeer et al., 2017a), has conceptual similarities to coordinating agents within a multi-agent system, suggesting that future research might benefit from insights related to coordination and specialisation.

1.3 Proposed Approaches

This thesis addresses the inter- and intra-agent challenges through complementary algorithmic and architectural advancements:

Opponent Shaping (OS): To tackle the difficulties of inter-agent coordination and cooperation, particularly in mixed-motive scenarios, we investigate Opponent Shaping (OS; Foerster et al., 2018). OS moves beyond naive (or reactive) learning by allowing agents to explicitly model and influence the anticipated learning updates of their co-players. By considering how their actions affect others' learning processes, OS agents guide interactions towards more mutually beneficial or individually advantageous outcomes, often leading to pro-social behaviour where naive methods fail. Addressing these inter-agent dynamics, particularly the challenges of achieving more pro-social outcomes in general-sum settings and managing inconsistent learning assumptions, is the focus of Part I (Chapter 3 and 4)

Mixture of Experts (MoE) Architectures: In Part II (Chapter 6 and 7), we explore the application of Mixture of Experts (MoEs; Shazeer et al., 2017b) to address intra-agent complexity and parameter scaling in deep RL. MoE architectures have proven successful in scaling large supervised learning models. By leveraging modularity and structured sparsity, MoEs offer a potential pathway to build larger yet more efficient and adaptable deep RL agents that overcome the limitations of monolithic network designs.

1.3.1 Part I: Opponent Shaping

This part focuses on improving the consistency and scalability of Opponent Shaping methods for improved inter-agent coordination.

Chapter 3 presents work addressing the fundamental inconsistency of prior Opponent Shaping methods. We introduce Consistent Learning with Opponent-Learning Awareness (COLA), formalise consistency, analyse the relationship with prior methods like CGD and show COLA finds pro-social solutions where others fail. This chapter is based on the publication: ***Timon Willi****, Alistair Letcher*, Johannes Treutlein*, Jakob Foerster. COLA: Consistent Learning with Opponent-Learning Awareness. **In 39th International Conference on Machine Learning (ICML), 2022.**

Contributions: Timon Willi co-developed the core COLA concept and consistency measure, implemented the COLA algorithm, designed and executed the experiments on polynomial and matrix games and co-led the writing of the paper. Timon proposed an initial version of the proof in Appendix A. Johannes Treutlein and Alistair Letcher gave the final proofs in Appendix A-G. Johannes co-led the paper writing. Jakob co-developed the core COLA concept and co-led the paper writing.

Chapter 4 tackles the challenge of scaling Opponent Shaping to high-dimensional games with long horizons and temporally-extended actions. We analyse memory requirements, propose the simplified Shaper algorithm, formalise batch averaging, and demonstrate successful shaping in complex grid world social dilemmas. This chapter is based on the publication: Akbir Khan*, ***Timon Willi****, Newton Kwan*, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, Jakob Foerster. Scaling Opponent Shaping to High Dimensional Games. **In AAMAS, 2024.**

Contributions: Timon Willi co-developed the Shaper algorithm concept, implemented the Shaper architecture and related components, designed and executed experiments in the "in-the-Matrix" grid-worlds, contributed significantly to the analysis of memory roles and batch averaging, and the paper writing. Akbir Khan

wrote the first version of the Shaper algorithm in JAX, co-led experiments, and co-led the paper writing. Newton Kwan co-led the paper writing, experiments, and development of "in-the-Matrix" environments. Andrei Tacchetti provided supervision throughout the project. Chris Lu helped with ideation throughout the project. Edward Grefenstette, Tim Rocktäschel and Jakob Foerster supervised the project.

Chapter 5 introduces JaxMARL, a hardware-accelerated MARL library designed to alleviate computational bottlenecks and improve evaluation practices. It provides efficient JAX implementations of popular MARL environments and baseline algorithms, enabling faster and broader empirical research. This chapter is based on the publication:

Alexander Rutherford*, Benjamin Ellis*, Matteo Gallici*, Jonathan Cook†, Andrei Lupu†, Garðar Ingvarsson†, ***Timon Willi†***, Ravi Hammond†, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu*†, Jakob Foerster. JaxMARL: Multi-Agent RL Environments and Algorithms in JAX. **In NeurIPS Datasets and Benchmarks Track, 2024**. (Note: † indicates Core Contributor)

Contributions: Timon Willi led the implementation of the STORM environments within JaxMARL, with Akbir Khan Alexandra Souly, who contributed to earlier versions of STORM. Timon also contributed to library design discussions and assisted with benchmarking efforts. Alexander Rutherford and Benjamin Ellis co-led the submission efforts, development of IPPO, the JaxMARL API, MPE and SMAXv2 environments, respectively. Matteo Gallici implemented and evaluated the off-policy MARL algorithms and assisted with the Hanabi environment, alongside Ravi Hammond and Jonathan Cook. Christian Schroeder de Witt implemented the predator-prey environment. Andrei Lupu implemented the Overcooked environment. Garðar Ingvarsson helped with the Multi-Agent Brax environments. Chris Lu led the project on ideation and helped with writing. Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, and Robert Lange assisted on various implementations in

JaxMARL. Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, and Jakob Foerster served as advisors, with Jakob acting as primus inter pares.

1.3.2 Part II: Mixture of Experts for Deep Reinforcement Learning

This part shifts focus to intra-agent complexity, exploring MoE architectures to achieve parameter scaling and handle non-stationarity in deep RL, drawing parallels to internal multi-agent coordination.

Chapter 6 demonstrates that incorporating MoE modules into value-based deep RL networks can unlock parameter scaling, improving performance with model size contrary to standard observations. We analyse Soft MoE and Top1-MoE, investigate architectural components, and link performance gains to improved network utilisation. This chapter is based on the publication:

Johan Obando-Ceron*, Ghada Sokar*, ***Timon Willi****, Clare Lyle, Jesse Farebrother, Jakob Foerster, Karolina Dziugaite, Doina Precup, Pablo Samuel Castro. Mixtures of Experts Unlock Parameter Scaling for Deep RL.

In 41st International Conference on Machine Learning (ICML), 2024.

Contributions: Timon Willi co-developed the application of MoEs to value-based RL for scaling, implemented MoE modules (especially Section 5.3), ran part of the core scaling experiments on ALE, and helped with the writing of the paper. Ghada Sokar implemented early versions of the MoE code used in the core environments. Johan Obando-Ceron and Pablo Samuel Castro co-led the experiments, ideation, and paper writing. Clare Lyle led the capacity analysis. Jakob Foerster, Jesse Farebrother, Karolina Dziugaite, and Doina Precup provided insights throughout the project.

Chapter 7 investigates the efficacy of MoE architectures under the amplified non-stationarity of multi-task (MTRL) and continual (CRL) reinforcement learning settings. We compare different MoE placements and routing strategies, providing further evidence for MoEs enhancing learning capacity and robustness framed

through internal expert specialisation and coordination. This chapter is based on the publication:

Timon Willi*, Johan Obando-Ceron*, Jakob Foerster, Karolina Dziugaite, Pablo Samuel Castro. Mixture of Experts in a Mixture of RL settings. **In Reinforcement Learning Conference (RLC), 2024.**

Contributions: Timon Willi co-led the conceptualisation of using MTRL/CRL to test MoE robustness to non-stationarity with Jakob Foerster. Timon implemented the experimental setups with various MoE architectures, designed and ran the comparative experiments, led the analysis of results (specialisation, plasticity), and co-led the writing of the paper with Johan Obando-Ceron. Jakob Foerster, Karolina Dziugaite, and Pablo Samuel Castro provided invaluable supervision throughout the project.

1.4 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides the necessary background on RL, MARL, Opponent Shaping, deep RL scaling challenges, and Mixture of Experts. Part I then delves into Opponent Shaping, with Chapter 3 focusing on consistency (COLA), Chapter 4 on scaling (Shaper), and Chapter 5 on tools (JaxMARL). Part II explores Mixture of Expert architectures, with Chapter 6 demonstrating parameter scaling in single-task deep RL and Chapter 7 investigating MoEs in multi-task and continual RL settings. Finally, Chapter 8 concludes the thesis, summarising the contributions and proposing avenues for future work.

2
Background

This chapter provides the necessary concepts and formalism to understand the research presented in this thesis. First, we introduce the fundamentals of single-agent Reinforcement Learning, followed by its extension to the multi-agent setting, highlighting its unique challenges. We then discuss Opponent Shaping, which addresses concurrent learning dynamics in multi-agent systems. Subsequently, we explore the challenges of scaling deep neural networks within the context of Reinforcement Learning. This motivates the introduction of Mixture of Experts architectures, which we frame through the lens of internal agent coordination. Finally, we touch upon computational tools that enable research at scale.

2.1 Reinforcement Learning

Reinforcement Learning (Sutton et al., 1998) addresses the problem of an agent learning to make a sequence of decisions by interacting with an environment to maximise a cumulative reward signal.

2.1.1 Markov Decision Processes

The standard formalism for single-agent RL is the **Markov Decision Process** (**MDP;** Bellman, 1957). An MDP is defined as a tuple $M = \langle S, A, P, R, \rho_0, \gamma \rangle$,

where:

- S is the set of possible states s the environment can be in.
- A is the set of actions a the agent can take.
- $P : S \times A \to \Delta(S)$ is the state transition probability function, where $P(s'|s, a)$ gives the probability of transitioning to state s' after taking action a in state s. $\Delta(S)$ denotes the set of probability distributions over S.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, where $R(s, a, s')$ is the immediate reward received after transitioning to state s' from state s by taking action a. Sometimes simplified as $R(s, a)$.
- $\rho_0 : S \to [0, 1]$ is the initial state distribution, defining the probability of starting in each state $s \in S$.
- $\gamma \in [0, 1]$ is the discount factor, which determines the present value of future rewards. A value of $\gamma < 1$ ensures that the sum of rewards is finite in infinite-horizon problems and prioritises immediate rewards.

The agent interacts with the environment based on its **policy** π, which defines its behaviour. A policy is a mapping from states to a probability distribution over actions, $\pi : S \to \Delta(A)$, where $\pi(a|s)$ denotes the probability of taking action a in state s.

The agent's goal is to find a policy π that maximises the expected **discounted return**, which is the cumulative sum of discounted rewards from a given time step t: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. The objective function is typically defined as the expected return starting from an initial state $s_0 \sim \rho_0$:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[G_0] = \mathbb{E}_{s_0\sim\rho_0, a_t\sim\pi(\cdot|s_t), s_{t+1}\sim P(\cdot|s_t,a_t)} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right] \tag{2.1}$$

where $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, ...)$ represents a trajectory generated by following policy π.

2.1.2 Value Functions

To evaluate policies and guide learning, RL utilises **value functions**:

- The **state-value function** $V^\pi(s)$ is the expected return starting from state s and subsequently following policy π:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[G_t|S_t = s] \tag{2.2}$$

- The **action-value function** (or Q-function) $Q^\pi(s, a)$ is the expected return starting from state s, taking action a, and thereafter following policy π:

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[G_t|S_t = s, A_t = a] \tag{2.3}$$

Intuitively, $V^\pi(s)$ measures how good it is to be in state s under policy π, while $Q^\pi(s, a)$ measures how good it is to take action a in state s under policy π.

There exists at least one optimal policy π^* such that $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in S$ and all policies π. All optimal policies share the same optimal value functions, denoted V^* and Q^*. These satisfy the Bellman optimality equations (Bellman, 1952):

$$V^*(s) = \max_{a\in A} Q^*(s, a) \tag{2.4}$$

$$Q^*(s, a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s, a, s')+\gamma V^*(s')] = \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s, a, s')+\gamma \max_{a'\in A} Q^*(s', a')] \tag{2.5}$$

2.1.3 Learning Approaches

RL algorithms aim to find π^*, which can be achieved through different approaches.

Value-Based Methods: These methods learn an approximation of the optimal action-value function Q^*, typically parameterised by θ, denoted Q_θ. The optimal policy is then derived implicitly by acting greedily: $\pi(s) = \arg\max_a Q_\theta(s, a)$. Q-learning (Watkins and Dayan, 1992) updates Q_θ using temporal difference (TD) learning, minimizing the difference between the current estimate $Q_\theta(s_t, a_t)$ and a target value, often $y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')$, where θ^- are the parameters

of a slowly updated target network used for stability in Deep Q-Networks (DQN) (Mnih et al., 2013b). Experience replay buffers are also commonly used to break correlations in sampled transitions (Mnih et al., 2015a).

Policy-Based Methods: These methods directly parameterise the policy π_θ and update θ by ascending the gradient of the objective $J(\theta)$. The **Policy Gradient Theorem** (Sutton et al., 1999) provides an expression for this gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\right] \tag{2.6}$$

where $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is the **advantage function**. Algorithms like REINFORCE (Williams, 1992) use Monte Carlo estimates of the return for A^π.

Actor-Critic Methods: These methods combine value-based and policy-based approaches. An **actor** component learns the policy π_θ, and a **critic** component learns a value function (V_ϕ or Q_ϕ) to estimate the advantage A^π. The critic provides lower-variance estimates for the actor's gradient updates compared to REINFORCE. Proximal Policy Optimization (PPO) (Schulman et al., 2017) is a popular actor-critic algorithm that uses a clipped surrogate objective to ensure stable policy updates.

2.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning is essential as real-world scenarios involve interacting entities where single-agent RL, treating others merely as environment dynamics, falls short. MARL's focus on explicitly modeling co-players enables sophisticated strategies like opponent shaping. This endeavour engages fundamental questions, as highlighted by Abel et al. (2025): attributing "agency" is frame-dependent, relying on observer choices about boundaries, goals, and causality. Thus, MARL is not just technically challenging but a necessary scientific inquiry into the principles of interaction and the very nature of agency within complex systems.

2.2.1 Partially Observable Stochastic Games (POSGs)

A general and widely used formalism for multi-agent settings involving sequential decision-making under uncertainty is the **Partially Observable Stochastic Game** (**POSG**; Shapley, 1953; Hansen et al., 2004). POSGs can model cooperative, competitive, and mixed-motive interactions where agents have incomplete information about the true state of the world.

A POSG for N agents is defined as a tuple $\langle N, \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \Omega, O, \gamma \rangle$, where:

- N is the number of agents.
- $\mathcal{S}$ is the set of global states s.
- $\mathcal{A} = \times_{i=1}^{N} \mathcal{A}_i$ is the joint action space, where $\mathcal{A}_i$ is the action space for agent i. A joint action is $a = (a_1, \ldots, a_N)$.
- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition probability function, where $P(s'|s, a)$ gives the probability of transitioning to state s' after the joint action a is taken in state s. $\Delta(\mathcal{S})$ denotes the set of probability distributions over $\mathcal{S}$.
- $\mathcal{R} = (R_1, \ldots, R_N)$ is a vector of reward functions, where $R_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function for agent i. Each agent i receives a reward r_i according to $R_i(s, a, s')$ after the transition to state s'. This allows agents to have potentially different, conflicting, or aligned objectives.
- $\Omega = \times_{i=1}^{N} \Omega_i$ is the joint observation space, where Ω_i is the set of possible observations for agent i. A joint observation is $o = (o_1, \ldots, o_N)$.
- $O = (O_1, \ldots, O_N)$ is a vector of observation functions, where $O_i : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega_i)$ is the observation function for agent i, giving the probability $O_i(o_i|s', a)$ of agent i receiving observation o_i after joint action a leads to state s'.
- $\gamma \in [0, 1)$ is the common discount factor.

Each agent i learns a policy π_i that maps its local observation-action history $h_{i,t} = (o_{i,1}, a_{i,1}, \ldots, o_{i,t}) \in (\Omega_i \times \mathcal{A}_i)^*$ to a distribution over its actions $\mathcal{A}_i$, i.e., $\pi_i(a_{i,t}|h_{i,t})$. The goal of each agent i is typically to maximise its *own* expected discounted return:

$$J_i(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{i,t+1}\right]$$

where $\pi = (\pi_1, \ldots, \pi_N)$ is the joint policy and $r_{i,t+1}$ is the reward received by agent i at step $t+1$.

The relationships between the individual reward functions R_i determine the nature of the game.

If all reward functions are identical ($R_1 = R_2 = \cdots = R_N$), the setting is **fully cooperative**. This specific subclass of POSGs is known as a **Decentralized POMDP** (**Dec-POMDP**; Oliehoek and Amato, 2016).

If the rewards sum to zero ($\sum_i R_i(s, a, s') = 0$ for all s, a, s'), it is a **zero-sum game**. Otherwise, it is a **general-sum game**, which encompasses mixed-motive scenarios with cooperative and competitive elements.

The POSG framework is suitable for modeling the challenges addressed by Opponent-Shaping methods, such as COLA and Shaper, in this thesis. These methods often operate in settings where agents have distinct objectives, and influencing co-player learning becomes strategically important.

2.2.2 Core MARL Challenges

MARL introduces significant challenges beyond single-agent RL (Gronauer and Diepold, 2022).

Non-stationarity: From the perspective of any single agent i, the environment is non-stationary because the other agents' policies π_{-i} are simultaneously changing during learning. This violates the Markov assumption for standard single-agent RL algorithms. Opponent Shaping methods, discussed in Section 2.3 and explored in detail in Part I, directly attempt to address this non-stationarity by modeling opponent adaptation.

Coordination: Agents must coordinate their actions effectively to achieve desired individual or collective outcomes, especially when communication is limited or costly.

Credit Assignment: Determining which agent's actions contributed to a team's success or failure is difficult, particularly with sparse or delayed team-based rewards.

Scalability (Inter-agent): The joint action and observation spaces grow exponentially with the number of agents N, making exploration and learning computationally demanding.

Interestingly, some challenges faced in coordinating multiple external agents, such as achieving effective specialisation of roles and ensuring coherent collective action, find parallels in the *intra-agent* challenges of managing complex internal components within large neural network architectures, a theme explored in Part II of this thesis.

2.3 Opponent Shaping

A naive approach to MARL treats other learning agents simply as part of the environment's dynamics. This can lead to suboptimal outcomes, such as mutual defection in the Iterated Prisoner's Dilemma (IPD; Axelrod and Hamilton, 1981; Foerster et al., 2018). Opponent Shaping methods provide an alternative by explicitly modeling an agent's actions' influence on the learning process and subsequent policy changes of its co-players. The core idea is for an agent i to anticipate how its actions will affect the parameters θ_{-i} of agent $-i$ in the next time step(s), and choose its current action to steer agent $-i$'s learning towards behaviour that is beneficial for agent i.

Learning with Opponent-Learning Awareness (**LOLA**; Foerster et al., 2018) is a pioneering OS method. A LOLA agent i computes its policy update by differentiating through an anticipated learning step of its opponent. Assuming agent $-i$ performs a naive gradient descent step $\Delta\theta_{-i} = -\alpha\nabla_{-i}L_{-i}(\theta_i, \theta_{-i})$ (where L_{-i} is agent $-i$'s loss and α is the assumed learning rate), agent i optimises its

own loss evaluated at the anticipated *post-update* parameters of the opponent: $L_i(\theta_i, \theta_{-i} + \Delta\theta_{-i})$.

While demonstrating success in promoting cooperation (e.g., learning Tit-for-Tat-like strategies in the IPD), the original LOLA formulation suffers from an **inconsistency problem**: it assumes the opponent is a naive learner, which is false if the opponent is also a LOLA agent or any other adaptive agent (Letcher et al., 2019b). This thesis addresses this inconsistency in Chapter 3.

2.3.1 The Differentiable Games Framework

Much of the theoretical analysis of Opponent Shaping, including the methods explored in Part I of this thesis, leverages the framework of **Differentiable Games** (Letcher et al., 2019a;b). This framework provides tools to analyse the learning dynamics when multiple agents simultaneously update their parameters based on interdependent losses.

Formally, a differentiable game involves n players, where each player i controls a parameter vector $\theta_i \in \mathbb{R}^{d_i}$ and aims to minimize a loss function $L^i(\theta_1, ..., \theta_n)$ which depends on the parameters of all players. Let $\theta = (\theta_1, ..., \theta_n) \in \mathbb{R}^d$ be the concatenation of all parameters, where $d = \sum_{i=1}^{n} d_i$.

The learning dynamics are modeled as agents performing simultaneous gradient descent on their respective losses. The instantaneous change in parameters is described by the negative **simultaneous gradient vector field** $\xi(\theta) \in \mathbb{R}^d$:

$$\xi(\theta) = (\nabla_{\theta_1} L^1(\theta), \nabla_{\theta_2} L^2(\theta), ..., \nabla_{\theta_n} L^n(\theta))^\top$$

The dynamics of learning under simultaneous gradient descent are then given by the ordinary differential equation $\dot{\theta} = -\xi(\theta)$.

The stability and convergence properties of these dynamics are analysed by studying the **Jacobian** (or Hessian in the game context) of the vector field ξ:

$$J(\theta) = \nabla_\theta \xi(\theta) = \begin{pmatrix} \nabla_{11} L^1 & \nabla_{12} L^1 & \cdots & \nabla_{1n} L^1 \\ \nabla_{21} L^2 & \nabla_{22} L^2 & \cdots & \nabla_{2n} L^2 \\ \vdots & \vdots & \ddots & \vdots \\ \nabla_{n1} L^n & \nabla_{n2} L^n & \cdots & \nabla_{nn} L^n \end{pmatrix} \tag{2.7}$$

where $\nabla_{ij} L^k = \nabla_{\theta_j} \nabla_{\theta_i} L^k(\theta)$. The eigenvalues of $J(\theta)$ at fixed points (where $\xi(\theta) = 0$) determine the local stability properties.

LOLA can be understood within this framework as modifying the update rule for player i from $-\nabla_i L^i$ to incorporate terms involving the off-diagonal blocks of the Jacobian ($\nabla_{ij} L^i$ for $j \neq i$) by differentiating through the anticipated gradient step of opponents ($\Delta\theta_{-i}$). For example, the first-order Taylor LOLA update involves terms like $\nabla_{i(-i)} L^i \Delta\theta_{-i}$. This explicit consideration of the game's Jacobian structure is what allows LOLA agents to shape opponents.

The Differentiable Games framework, particularly the decomposition of the Jacobian into symmetric and antisymmetric components (related to potential and Hamiltonian games, respectively, see Letcher et al. (2019a)), provides the mathematical tools used in Chapter 3 to analyse the consistency and stability properties of LOLA and develop the COLA algorithm.

2.4 Challenges in Deep RL Scaling

Scaling deep neural networks has proven paramount for recent successes in supervised learning (Brown et al., 2020). For recent RL successes, applying deep neural networks has also been instrumental (Mnih et al., 2015a). However, unlike supervised learning, where larger models consistently yield better performance given sufficient data (Kaplan et al., 2020b), scaling deep networks in RL often fails to deliver performance gains and can even hurt performance (Farebrother et al., 2022; Schwarzer et al., 2023).

Several interconnected issues contribute to this.

Plasticity Loss: Deep RL agents struggle to continuously adapt and learn new information without catastrophically forgetting previously acquired knowledge (Igl et al., 2020; Nikishin et al., 2022a).

Dormant Neurons: A significant fraction of neurons in deep RL networks become inactive during training, effectively reducing the network's capacity and indicating parameter under-utilisation (Sokar et al., 2023a).

Parameter Inefficiency: The parameters in deep RL networks are often not used as effectively as in supervised learning, potentially due to the complex interplay between representation learning and control, and the non-stationary learning signal (Kumar et al., 2021; Lyle et al., 2022).

Sensitivity: Training large deep RL networks can be highly sensitive to hyper-parameter choices and initialization (Engstrom et al., 2020).

These challenges highlight that simply increasing network size is often insufficient for improving multi-agent, let alone single-agent, deep RL agent capabilities, and motivate the exploration of alternative architectures like Mixture of Experts to close the gap with supervised learning, the subject of Part II (Chapter 6 and 7).

2.5 Mixture of Experts

The challenges outlined in the previous section, particularly the difficulty in effectively scaling deep neural networks within RL, motivate the exploration of alternative architectures. **Mixture of Experts (MoE)** architectures have emerged as a powerful technique for building extremely large yet computationally efficient models in supervised learning, particularly in the context of large language models and transformers (Shazeer et al., 2017a; Fedus et al., 2022; Du et al., 2022). MoEs increase model capacity through conditional computation, activating only a subset of the model's parameters for any given input.

An MoE layer typically replaces a standard feed-forward layer within a larger neural network (e.g., a policy network π_θ or a value network Q_θ). It consists of two main components:

1. A set of n **expert networks** $(f_1, ..., f_n)$. These are typically feed-forward networks, acting as the "processing units" within the MoE layer.

2. A **router network** (or gate) g. This network takes the input x (typically an intermediate representation derived from the environment state s or observation o processed by earlier layers) and determines which experts process x.

In standard **sparse MoE**, the router $g(x)$ outputs scores/logits, and a Top-k mechanism selects the k experts with the highest scores ($k \ll n$). The output y of the MoE layer is a weighted combination of the outputs from these k experts, $y = \sum_{j \in \text{TopK}(g(x))} g(x)_j f_j(x)$.

Soft MoE (Puigcerver et al., 2023), the primary variant studied in Part II, provides a differentiable alternative using soft assignments. Let the input to the MoE layer, derived from state/observation s, be represented as m tokens $X \in \mathbb{R}^{m \times d}$. The layer has n experts $\{f_i : \mathbb{R}^d \to \mathbb{R}^d\}_{i=1}^n$, each with p input/output slots parameterised by $\Phi \in \mathbb{R}^{d \times (n \cdot p)}$.

1. **Dispatch Weights D (Routing):** The router, implicitly defined by Φ, computes weights $D_{ij} = \frac{\exp((X\Phi)_{ij})}{\sum_{i'=1}^{m} \exp((X\Phi)_{i'j})}$ that softly allocate each input token X_i to each expert *slot* Φ_j.

2. **Slot Inputs $\tilde{X}$ (Task Allocation):** Each slot receives a weighted combination of all input tokens: $\tilde{X} = D^\top X$.

3. **Expert Computation $\tilde{Y}$ (Specialized Processing):** Each expert f_k processes the inputs for its slots: $\tilde{Y}_j = f_{\lfloor j/p \rfloor}(\tilde{X}_j)$.

4. **Combine Weights C (Output Aggregation):** Weights $C_{ij} = \frac{\exp((X\Phi)_{ij})}{\sum_{j'=1}^{n \cdot p} \exp((X\Phi)_{ij'})}$ are computed to aggregate the expert slot outputs $\tilde{Y}$ for each original input token.

5. **Final Output Y:** The layer's output is $Y = C\tilde{Y}$, which then feeds into subsequent layers to ultimately produce, e.g., action probabilities $\pi_\theta(a|s)$ or Q-values $Q_\theta(s, a)$.

MoEs' potential to address the scaling challenges outlined in Section 2.4 is investigated empirically in Chapter 6 and 7.

MoE as an Internal Multi-Agent System: An Analogy Viewing MoEs through an **internal multi-agent lens**, we can draw parallels between its components and concepts from MARL to motivate future research (Section 2.2).

- **Internal State:** The input x (or tokens X) entering the MoE layer serves as the shared state or context for the internal system.
- **Expert Networks as Specialised Agents:** Each expert f_i can be viewed as an internal agent specialising in processing certain types of inputs or contributing to specific aspects of the overall computation (e.g., estimating the value for certain state regions, contributing to specific action probabilities). They do not have individual policies π_i or rewards R_i in the MARL sense, but their parameters are adjusted to contribute effectively to the single agent's overall RL objective $J(\theta)$.
- **Router as Coordinator:** The router network g (or the mechanism defined by Φ in Soft MoE) acts as an internal coordinator. Its action is the routing decision (hard assignment in sparse MoE, soft weights D and C in Soft MoE). It dynamically allocates the processing load (tasks) based on the input state x, aiming to leverage expert specialisation.
- **Output as Collective Action:** The final output Y represents the aggregated result of the specialised expert computations, analogous to a collective outcome derived from coordinated individual agent contributions.

Limitations of the Analogy and Future Work: This analogy, while conceptually useful to inspire future research in specialisation and efficient parameter use, has limitations and was not the original motivation to conduct the research.

1. **Shared Objective & Parameters:** Unlike MARL agents, which may have conflicting goals (general-sum), all experts and the router share a *single* RL objective $J(\theta)$ of the host agent via standard backpropagation. There is no notion of individual expert rewards or independent learning objectives.

2. **Implicit Coordination:** The coordination achieved by the router is learned implicitly through end-to-end gradient descent. It does not employ explicit MARL coordination protocols, communication channels between experts, or reason about other experts' intentions or beliefs.

3. **Simplified Actions:** The router's action is a routing decision, and experts' actions are intermediate computations, not actions taken within the external environment M.

These limitations, however, inspire future research directions.

- **MARL-inspired Routers:** Could more sophisticated routing mechanisms be designed by drawing inspiration from MARL coordination algorithms (e.g., using attention mechanisms analogous to communication or explicitly modeling expert load/utility)?

- **Internal Credit Assignment:** Is standard backpropagation the most effective way to assign credit to individual experts for their contribution to the overall agent's success? Could techniques inspired by MARL credit assignment improve expert specialisation?

- **Applying MoEs within MARL agents:** Can the enhanced scalability and internal specialisation provided by MoEs benefit individual agents operating within a larger, external multi-agent system?

This internal coordination perspective suggests that MoEs might inherently possess architectural properties suited to tackling the parameter inefficiency and plasticity challenges in deep RL (Section 2.4). By enabling modularity, specialisation, and dynamic allocation of internal resources, MoEs offer a promising path towards more scalable and robust deep RL agents, explored empirically in Part II of this thesis.

2.6 Hardware Acceleration and JAX

In addition to algorithmic advances, auto-differentiation libraries, such as TensorFlow (Abadi et al., 2016), PyTorch (Paszke, 2019), and JAX (Bradbury et al., 2018), facilitated the widespread use of deep learning. Among other benefits, these libraries simplify high-performance computing on hardware accelerators (GPUs, TPUs, etc.).

JAX combines automatic differentiation (autograd) with the XLA compiler. Key features relevant to this thesis include:

- **`jit` (Just-In-Time Compilation):** Compiles Python functions to highly optimised XLA code for execution on accelerators.
- **`vmap` (Vectorisation):** Automatically vectorises functions, enabling efficient batch processing without manual batching logic.
- **`pmap` (Parallelisation):** Allows parallel execution of code across multiple accelerators.

These features enable researchers to write concise Python/NumPy-like code that can be efficiently scaled for large-scale MARL simulations (as demonstrated in JaxMARL, Chapter 5) and the training of complex architectures like MoEs (Chapter 6 and 7).

While supervised learning typically performs the whole training pipeline on GPUs, deep RL typically deploys the environment logic on CPUs. Only the forward and backward steps of backpropagation were computed on the GPUs. This slowed down wall-clock training speeds, as communication between CPUs and GPUs is costly. PureJaxRL (Lu et al., 2022b) and concurrent work changed this paradigm for single-agent RL, allowing the full RL training to be run on the GPU, leading to great speed-ups.

This background sets the stage for the contributions presented in the subsequent chapters, which aim to advance opponent shaping techniques for inter-agent coordination and leverage Mixture of Experts architectures to address the critical challenge of scaling deep reinforcement learning agents.

Part I

Opponent Shaping

3

COLA: Consistent Learning with Opponent-Learning Awareness

As established in the Background (Section 2.3), while Opponent Shaping offers a promising avenue for achieving pro-social behaviour in general-sum MARL settings, pioneering methods like Learning with Opponent-Learning Awareness (LOLA; Foerster et al., 2018) suffer from a critical conceptual flaw: their inherent inconsistency. As detailed in Section 2.3, a standard LOLA agent models its opponent as a naive learner, updating its parameters via simple gradient descent on its own loss. This assumption is violated when interacting with another adaptive agent, particularly another LOLA agent, leading to a mismatch between the modeled and actual learning dynamics of the co-player. Letcher et al. (2019b) hypothesised this inconsistency as a key reason for LOLA's observed shortcomings, such as its empirical convergence failures and its inability to preserve desirable game-theoretic equilibria like Stable Fixed Points (SFPs) even in simple games (Letcher et al., 2019b). Without resolving this foundational issue, robustly applying OS remains challenging.

This chapter directly confronts this inconsistency problem, establishing a more theoretically sound basis for opponent shaping. In Willi et al. (2022), we first formalise the notion of consistency for update functions within the Differentiable Games framework (introduced in Subsection 2.3.1). An update function pair is

consistent if each agent's update rule correctly anticipates the update rule used by its opponent. We analyse higher-order extensions of LOLA (HOLA) and prove that its limit under convergence (infinite-order LOLA, or iLOLA) is indeed consistent, if it exists. Critically, we revisit and correct prior claims regarding Competitive Gradient Descent (CGD; Schäfer and Anandkumar, 2019), demonstrating that CGD does not, in general, recover HOLA nor satisfy the consistency property, thus failing to provide an off-the-shelf solution.

Recognising that the HOLA recursion may diverge or be computationally prohibitive, we introduce Consistent LOLA (COLA) as the chapter's primary contribution. COLA avoids recursion by directly learning a pair of update functions that minimise a differentiable measure of inconsistency. This approach requires only second-order derivatives and demonstrably finds consistent solutions even in cases where HOLA diverges. Our theoretical analysis reveals that COLA solutions are not necessarily unique, and importantly, that consistency alone does not guarantee the preservation of SFPs, challenging previous hypotheses.

Empirically, we evaluate COLA against LOLA, HOLA, and CGD across benchmark polynomial and matrix games. The results confirm COLA's ability to achieve more robust convergence, especially under higher look-ahead rates, and its tendency to find more socially desirable outcomes compared to inconsistent methods. By developing and analysing COLA, this chapter makes a key contribution towards establishing a more principled foundation for opponent shaping, clarifying the role and limitations of consistency in achieving stable and pro-social multi-agent learning.

COLA: Consistent Learning with Opponent-Learning Awareness

Timon Willi [*1] **Alistair Letcher** [*] **Johannes Treutlein** [*2 3] **Jakob Foerster** [1]

Abstract

Learning in general-sum games is unstable and frequently leads to socially undesirable (Pareto-dominated) outcomes. To mitigate this, Learning with Opponent-Learning Awareness (LOLA) introduced *opponent shaping* to this setting, by accounting for each agent's influence on their opponents' anticipated learning steps. However, the original LOLA formulation (and follow-up work) is *inconsistent* because LOLA models other agents as *naive learners* rather than LOLA agents. In previous work, this inconsistency was suggested as a cause of LOLA's failure to preserve stable fixed points (SFPs). First, we formalize consistency and show that higher-order LOLA (HOLA) solves LOLA's inconsistency problem if it converges. Second, we *correct a claim* made in the literature by Schäfer and Anandkumar (2019), proving that Competitive Gradient Descent (CGD) does *not* recover HOLA as a series expansion (and fails to solve the consistency problem). Third, we propose a new method called Consistent LOLA (COLA), which *learns* update functions that are consistent under mutual opponent shaping. It requires no more than second-order derivatives and learns consistent update functions even when HOLA fails to converge. However, we also prove that even consistent update functions do not preserve SFPs, contradicting the hypothesis that this shortcoming is caused by LOLA's inconsistency. Finally, in an empirical evaluation on a set of general-sum games, we find that COLA finds prosocial solutions and that it converges under a wider range of learning rates than HOLA and LOLA. We support the latter finding with a theoretical result for a simple game.

[*]Equal contribution [1]Department of Engineering Science, University of Oxford, United Kingdom [2]Department of Computer Science, University of Toronto, Canada [3]Vector Institute, Toronto, Canada. Correspondence to: <timon.willi@eng.ox.ac.uk>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

1. Introduction

Much research in deep multi-agent reinforcement learning (MARL) has focused on zero-sum games like Starcraft and Go (Silver et al., 2017; Vinyals et al., 2019) or fully cooperative settings (Oroojlooyjadid & Hajinezhad, 2019). However, many real-world problems, e.g. self-driving cars, contain both cooperative and competitive elements, and are thus better modeled as general-sum games. One such game is the famous Prisoner's Dilemma (Axelrod & Hamilton, 1981), in which agents have an individual incentive to defect against their opponent, even though they would prefer the outcome in which both cooperate to the one where both defect. A strategy for the infinitely iterated version of the game (IPD) is tit-for-tat, which starts out cooperating and otherwise mirrors the opponent's last move. It achieves mutual cooperation when chosen by both players and has proven to be successful at IPD tournaments (Axelrod & Hamilton, 1981). If MARL algorithms are deployed in the real world, it is essential that they are able to cooperate with others and entice others to cooperate with them, using strategies such as tit-for-tat (Dafoe et al., 2021). However, naive gradient descent and other more sophisticated methods (Korpelevich, 1977; Mescheder et al., 2017; Balduzzi et al., 2018; Mazumdar et al., 2019; Schäfer & Anandkumar, 2019) converge to the mutual defection policy under random initialization (Letcher et al., 2019b).

An effective paradigm to improve learning in general-sum games is *opponent shaping*, where agents take into account their influence on the anticipated learning step of the other agents. LOLA (Foerster et al., 2018a) was the first work to make explicit use of opponent shaping and is one of the only general learning methods designed for general-sum games that obtains mutual cooperation with the tit-for-tat strategy in the IPD. While LOLA discovers these prosocial equilibria, the original LOLA formulation is inconsistent because LOLA agents assume that their opponent is a *naive learner*. This assumption is clearly violated if two LOLA agents learn together. It has been suggested that this inconsistency is the cause for LOLA's main shortcoming, which is not maintaining the stable fixed points (SFPs) of the underlying game, even in some simple quadratic games (Letcher 2018, pp. 2, 26; see also Letcher et al. 2019b).

Contributions. To address LOLA's inconsistency, we first revisit the concept of *higher-order* LOLA (HOLA) (Foerster et al., 2018a) in Section 4.1. For example, *second-order* LOLA assumes that the opponent is a *first-order* LOLA agent (which in turn assumes the opponent is a naive learner) and so on. Supposing that HOLA converges with increasing order, we define *infinite-order* LOLA (iLOLA) as the limit. Intuitively, two iLOLA agents have a *consistent* view of each other since they accurately account for the learning behavior of the opponent under mutual opponent shaping. Based on this idea, we introduce a formal definition of *consistency* and prove that, if it exists, iLOLA is indeed consistent (Proposition 4.3).

Second, in Section 4.2, we correct a claim made in previous literature, which would have provided a closed-form solution of the iLOLA update. According to Schäfer & Anandkumar (2019), the Competitive Gradient Descent (CGD) algorithm recovers HOLA as a series expansion. If true, this would imply that CGD coincides with iLOLA, thus solving LOLA's inconsistency problem. We prove that this is untrue: CGD's series expansion does, in general, not recover HOLA, CGD does not correspond to iLOLA, and CGD does not solve the inconsistency problem (Proposition 4.4).

In lieu of a closed-form solution, a naive way of computing the iLOLA update is to iteratively compute higher orders of LOLA until convergence. However, there are two main problems with addressing consistency using a limiting update: the process may diverge and typically requires arbitrarily high derivatives. To address these, in Section 4.3, we propose Consistent LOLA (COLA) as a more robust and efficient alternative. COLA learns a pair of consistent update functions by explicitly minimizing a differentiable measure of consistency inspired by our formal definition. We use the representational power of neural networks and gradient based optimization to minimize this loss, resulting in *learned* update functions that are mutually consistent. By reframing the problem as such, we only require up to second-order derivatives.

In Section 4.4, we prove initial results about COLA. First, we show that COLA's solutions are not necessarily unique. Second, despite being consistent, COLA does not recover SFPs, contradicting the prior belief that this shortcoming is caused by inconsistency. Third, to show the benefit of additional consistency, we prove that COLA converges under a wider range of look-ahead rates than LOLA in a simple general-sum game.

Finally, in Sections 5 and 6, we report our experimental setup and results, investigating COLA and HOLA and comparing COLA to LOLA and CGD in a range of games. We experimentally confirm our theoretical result that CGD does not equal iLOLA. Moreover, we show that COLA converges under a wider range of look-ahead rates than HOLA and LOLA, and that it is generally able to find socially desirable solutions. It is the only algorithm consistently converging to the fair solution in the Ultimatum game, and while it does not find tit-for-tat in the IPD (unlike LOLA), it does learn policies with near-optimal total payoff. We find that COLA learns consistent update functions even when HOLA diverges with higher order and its updates are similar to iLOLA when HOLA converges. Although COLA solutions are not unique in theory, COLA empirically tends to find similar solutions over different runs.

2. Related work

General-sum learning algorithms have been investigated from different perspectives in the reinforcement learning, game theory, and GAN literature (Schmidhuber, 1991; Barto & Mahadevan, 2003; Goodfellow et al., 2014; Racanière et al., 2017). Next, we will highlight a few of the approaches to the mutual opponent shaping problem.

Opponent modeling maintains an explicit belief of the opponent, allowing to reason over their strategies and compute optimal responses. Opponent modeling can be divided into different subcategories: There are classification methods, classifying the opponents into pre-defined types (Weber & Mateas, 2009; Synnaeve & Bessière, 2011), or policy reconstruction methods, where we explicitly predict the actions of the opponent (Mealing & Shapiro, 2017). Most closely related to opponent shaping is recursive reasoning, where methods model nested beliefs of the opponents (He & Boyd-Graber, 2016; Albrecht & Stone, 2017; Wen et al., 2019).

In comparison, COLA assumes that we have access to the ground-truth model of the opponent, e.g., the opponent's payoff function, parameters, and gradients, putting COLA into the framework of differentiable games (Balduzzi et al., 2018). Various methods have been proposed, investigating the local convergence properties to different solution concepts (Mescheder et al., 2017; Mazumdar et al., 2019; Letcher et al., 2019b; Schäfer & Anandkumar, 2019; Azizian et al., 2020; Schäfer et al., 2020; Hutter, 2021). Most of the work in differentiable games has not focused on opponent shaping or consistency. Mescheder et al. (2017) and Mazumdar et al. (2019) focus solely on zero-sum games without shaping. To improve upon LOLA, Letcher et al. (2019b) suggested Stable Opponent Shaping (SOS), which applies ad-hoc corrections to the LOLA update, leading to theoretically guaranteed convergence to SFPs. However, despite its desirable convergence properties, SOS still does not solve the conceptual issue of inconsistent assumptions about the opponent. CGD (Schäfer & Anandkumar, 2019) addresses the inconsistency issue for zero-sum games but not for general-sum games. The exact difference between CGD, LOLA and our method is addressed in Section 4.2. Model-Free Opponent Shaping (M-FOS) (Lu et al., 2022)

Table 1. On the Tandem game: (a) Log of the squared consistency loss, where e.g. HOLA6 is sixth-order higher-LOLA. (b) Cosine similarity between COLA and LOLA, HOLA3, and HOLA6 over different look-ahead rates. The values represent the mean of a 1,000 samples, uniformly sampled from the parameter space Θ. Error bars represent one standard deviation over 10 COLA training runs.

(a)

α	LOLA	HOLA3	HOLA6	COLA
1.0	128.0	512	131072	3e-14±2e-15
0.5	12.81	14.05	12.35	2e-14±5e-15
0.3	2.61	2.05	0.66	4e-14±3e-15
0.1	0.08	9e-3	2e-6	6e-14±9e-15
0.01	1e-5	2e-8	4e-14	1e-14±4e-14

(b)

α	LOLA	HOLA3	HOLA6
1.0	0.94±0.04	0.94±0.04	0.94±0.04
0.5	0.88±0.12	0.88±0.12	0.08±0.13
0.3	0.92±0.01	0.91±0.01	0.80±0.01
0.1	0.95±0.01	0.99±0.01	0.99±0.01
0.01	0.99±0.01	1.00±0.00	1.00±0.00

frames opponent shaping as a meta-learning problem requiring only first-order derivatives. M-FOS is consistent in that it does not make any assumption about the learning algorithm of the opponent. However, the work does not investigate consistency specifically.

3. Background

3.1. Differentiable games

The framework of differentiable games has become increasingly popular to model multi-agent learning. Whereas stochastic games are limited to parameters such as action-state probabilities, differentiable games generalize to any real-valued parameter vectors and differentiable loss functions (Balduzzi et al., 2018). We restrict our attention to two-player games, as is standard in much of the literature (Foerster et al., 2018a;b; Schäfer & Anandkumar, 2019).

Definition 3.1 (Differentiable games)**.** In a two-player differentiable game, players $i = 1, 2$ control parameters $\theta_i \in \mathbb{R}^{d_i}$ to minimize twice continuously differentiable losses $L^i : \mathbb{R}^{d_1+d_2} \to \mathbb{R}$. We adopt the convention to write $-i$ to denote the opponent of player i.

A fundamental challenge of the multi-loss setting is finding a good solution concept. Whereas in the single loss setting the typical solution concept are local minima, in multi-loss settings there are different sensible solution concepts. Most prominently, there are Nash Equilibria (Osborne & Rubinstein, 1994). However, Nash Equilibria include unstable saddle points that cannot be reasonably found via gradient-based learning algorithms (Letcher et al., 2019b). A more suitable concept are stable fixed points (SFPs), which could be considered a differentiable game analogon to local minima in single loss optimization. We will omit a formal definition here for brevity and point the reader to previous work on the topic (Letcher et al., 2019a).

3.2. LOLA

Consider a differentiable game with two players. A LOLA agent θ_1 uses its access to the opponent's parameters θ_2 to differentiate through a learning step of the opponent. That is, agent 1 reformulates their loss to $\tilde{L}^1 = L^1\left(\theta_1, \theta_2 + \widehat{\Delta\theta_2}\right)$, where $\widehat{\Delta\theta_2}$ represents the assumed learning step of the opponent. In first-order LOLA we assume the opponent to be a naive learner: $\widehat{\Delta\theta_2} = -\alpha\nabla_2 L^2$. This assumption makes LOLA inconsistent when the opponent is any other type of learner. Here, ∇_2 denotes the gradient with respect to θ_2, and α represents the *look-ahead* rate, which is the *assumed* learning rate of the opponent. This rate may differ from the opponent's actual learning rate, but we will only consider equal learning rates and look-ahead rates across opponents for simplicity. In the original paper the loss was approximated using a Taylor expansion, $\tilde{L}^1 \approx L^1 + (\nabla_2 L^1)^\top \widehat{\Delta\theta_2}$. For agent 1, their first-order *Taylor* LOLA update is then

$$\Delta\theta_1 = -\alpha\left(\nabla_1 L^1 + \nabla_{12} L^1 \widehat{\Delta\theta_2} + \left(\nabla_1 \widehat{\Delta\theta_2}\right)^\top \nabla_2 L^1\right).$$

Alternatively, in *exact* LOLA, the derivative is taken directly with respect to the reformulated loss, yielding the update

$$\Delta\theta_1 = -\alpha\nabla_1\left(L^1\left(\theta_1, \theta_2 + \widehat{\Delta\theta_2}\right)\right).$$

LOLA has had some empirical success, being one of the first general learning methods to discover tit-for-tat in the IPD. However, later work showed that LOLA does not preserve SFPs $\bar{\theta}$, e.g., the rightmost term in the equation for Taylor LOLA can be nonzero at $\bar{\theta}$. In fact, LOLA agents show "arrogant" behavior: they assume they can shape the learning of their *naive* opponents without having to adapt to the shaping of the opponent. Prior work hypothesized that this arrogant behavior is due to LOLA's inconsistent formulation and may be the cause for LOLA's failure to preserve SFPs (Letcher (2018), pp. 2, 26; Letcher et al. (2019b))

3.3. CGD

CGD (Schäfer & Anandkumar, 2019) proposes updates that are themselves Nash Equilibra of a local bilinear approximation of the game. It stands out by its robustness to different look-ahead rates and its ability to find SFPs. However, CGD does not find tit-for-tat on the IPD, instead converging to mutual defection (see Figure 3e). CGD's update rule is

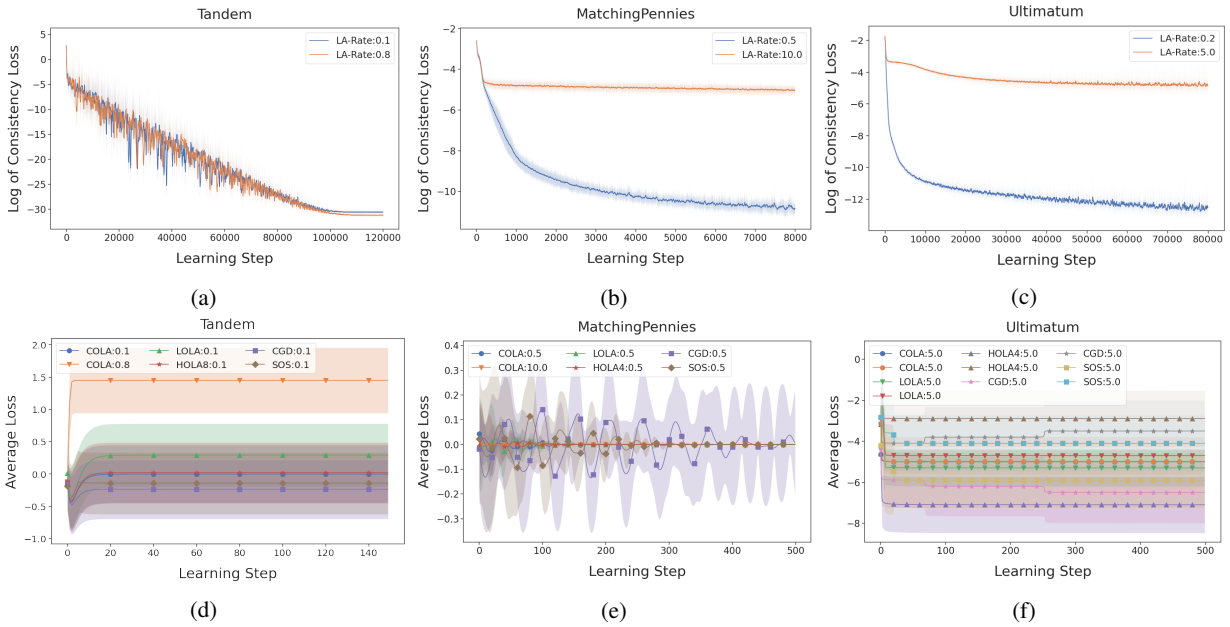


Figure 1. Subfigures (a), (b) and (c): Log of the consistency loss with standard error of 10 independent training runs over the training of the update functions for the Tandem, MP and Ultimatum games. Subfigures (d), (e) and (f): Learning outcomes for the respective games. E.g., "COLA:0.1" is COLA at a look-ahead rate of 0.1. Lines represent payoff means and shaded areas standard deviations over 10 runs.

given by

$$\begin{pmatrix} \Delta\theta_1 \\ \Delta\theta_2 \end{pmatrix} = -\alpha \begin{pmatrix} \mathrm{Id} & \alpha\nabla_{12}L^1 \\ \alpha\nabla_{21}L^2 & \mathrm{Id} \end{pmatrix}^{-1} \begin{pmatrix} \nabla_1 L^1 \\ \nabla_2 L^2 \end{pmatrix}.$$

One can recover different orders of CGD by approximating the inverse matrix via the series expansion $(\mathrm{Id} - A)^{-1} = \lim_{N\to\infty} \sum_{k=0}^{N} A^k$ for $\|A\| < 1$. For example, at N=1, we recover a version called Linearized CGD (LCGD), defined via $\Delta\theta_1 := -\alpha\nabla_1 L^1 + \alpha^2 \nabla_{12} L^1 \nabla_2 L^2$.

4. Method and theory

In this section, we formally define iLOLA and consistency under mutual opponent shaping and show that iLOLA is consistent, thus in principle addressing LOLA's inconsistency problem. We then clear up the relation between CGD and iLOLA, *correcting a false claim* in Schäfer & Anandkumar (2019). Lastly, we introduce COLA as an alternative to iLOLA and present some initial theoretical analysis, including the result that, contrary to prior belief, even consistent update functions do not recover SFPs.

4.1. Convergence and consistency of higher-order LOLA

The original formulation of LOLA is inconsistent when two LOLA agents learn together, because LOLA agents assume their opponent is a naive learner. To address this problem, we define and analyze iLOLA. In this section, we focus on exact LOLA, but we provide a version of our analysis for Taylor LOLA in Appendix C. HOLAn is defined by the recursive relation

$$h_1^{n+1} := -\alpha\nabla_1 \left(L^1(\theta_1, \theta_2 + h_2^n)\right)$$
$$h_2^{n+1} := -\alpha\nabla_2 \left(L^2(\theta_1 + h_1^n, \theta_2)\right)$$

with $h_1^{-1} = h_2^{-1} = 0$, omitting arguments (θ^1, θ^2) for convenience. In particular, HOLA0 coincides with simultaneous gradient descent while HOLA1 coincides with LOLA.

Definition 4.1 (iLOLA)**.** If HOLAn = (h_1^n, h_2^n) converges pointwise as $n \to \infty$, define

$$\text{iLOLA} := \lim_{n\to\infty} \begin{pmatrix} h_1^n \\ h_2^n \end{pmatrix} \text{ as the limiting update.}$$

We show in Appendix A that HOLA does not always converge, even in simple quadratic games. But, unlike LOLA, iLOLA satisfies a criterion of *consistency* whenever HOLA does converge (under some assumptions), formally defined as follows:

Definition 4.2 (Consistency)**.** Any update functions $f_1 \colon \mathbb{R}^d \to \mathbb{R}^{d_1}$ and $f_2 \colon \mathbb{R}^d \to \mathbb{R}^{d_2}$ are *consistent* (under mutual opponent shaping with look-ahead rate α) if for all $\theta_1 \in \mathbb{R}^{d_1}, \theta_2 \in \mathbb{R}^{d_2}$, they satisfy

$$f_1(\theta_1, \theta_2) = -\alpha\nabla_1(L^1(\theta_1, \theta_2 + f_2(\theta_1, \theta_2))) \quad (1)$$
$$f_2(\theta_1, \theta_2) = -\alpha\nabla_2(L^2(\theta_1 + f_1(\theta_1, \theta_2), \theta_2)) \quad (2)$$

Table 2. On the MP game: Over different look-ahead rates we compare (a) the consistency losses and (b) the cosine similarity between COLA and LOLA, HOLA2, and HOLA4. The values represent the mean over 1,000 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA2	HOLA4	COLA
10	0.06	0.70	6.56	2e-3±3e-4
5	5e-3	0.03	0.15	5e-4±5e-5
1.0	9e-6	3e-8	4e-9	3e-6±1e-6
0.5	5e-7	3e-10	5e-12	3e-6±3e-6
0.01	1e-13	6e-17	5e-17	2e-6±2e-6

(b)

α	LOLA	HOLA2	HOLA4
10	0.90±0.01	0.84±0.01	0.74±0.02
5	0.98±0.01	0.97±0.01	0.92±0.01
1.0	1.00±0.00	1.00±0.00	1.00±0.00
0.5	1.00±0.00	1.00±0.00	1.00±0.00
0.01	1.00±0.00	1.00±0.00	1.00±0.00

Proposition 4.3. *Let* HOLA$n = (h_1^n, h_2^n)$ *denote both players' exact n-th order LOLA updates. Assume that* $\lim_{n\to\infty} h_i^n(\theta) = h_i(\theta)$ *and* $\lim_{n\to\infty} \nabla_i h_{-i}^n(\theta) = \nabla_i h_{-i}(\theta)$ *exist for all* $\theta \in \mathbb{R}^d$ *and* $i \in \{1, 2\}$. *Then iLOLA is consistent under mutual opponent shaping.*

Proof. In Appendix B. □

4.2. CGD does not recover higher-order LOLA

Schäfer & Anandkumar (2019) claim that "LCGD [Linearized CGD] coincides with first order LOLA" (page 6), and moreover that the higher-order "series-expansion [of CGD] would recover higher-order LOLA" (page 4). If this were correct, it would imply that *full* CGD is equal to iLOLA and thus provides a convenient closed-form solution. We prove that this is false in general games:

Proposition 4.4. *CGD is inconsistent and does not coincide with iLOLA. In particular, Linearized CGD (LCGD) does not coincide with LOLA and the series-expansion of CGD does not recover HOLA (neither exact nor Taylor). Instead, LCGD coincides with LookAhead (Zhang & Lesser, 2010), an algorithm that lacks opponent shaping, and the series-expansion of CGD recovers higher-order LookAhead.*

Proof. In Appendix D. For the negative results, it suffices to construct a single counterexample: we show that LCGD and LOLA differ almost everywhere in the Tandem game (excluding a set of measure zero). We prove by contradiction that the series-expansion of CGD does not recover HOLA. If it did, CGD would equal iLOLA, and by Proposition 4.3, CGD would satisfy the consistency equations. However, this fails almost everywhere in the Tandem game, concluding the contradiction. □

4.3. COLA

iLOLA is consistent under mutual opponent shaping. However, HOLA does not always converge and, even when it does, it may be expensive to recursively compute HOLAn for sufficiently high n to achieve convergence. As an alternative, we propose COLA.

COLA learns consistent update functions and avoids infinite regress by directly solving the equations in Definition 4.2. We define the *consistency losses* for learned update functions f_1, f_2 parameterized by ϕ_1, ϕ_2, obtained for a given θ as the difference between RHS and LHS in Definition 4.2:

$$C_1(\phi_1, \phi_2, \theta_1, \theta_2) = \left\| f_1 + \alpha \nabla_1 (L^1(\theta_1, \theta_2 + f_2)) \right\|$$
$$C_2(\phi_1, \phi_2, \theta_1, \theta_2) = \left\| f_2 + \alpha \nabla_2 (L^2(\theta_1 + f_1, \theta_2)) \right\| .$$

If both losses are 0 for all θ, then the two update functions defined by ϕ_1, ϕ_2 are consistent. For this paper, we parameterise f_1, f_2 as neural networks with parameters ϕ_1, ϕ_2 respectively, and numerically minimize the sum of both losses over a region of interest.

The parameter region of interest Θ depends on the game being played. For games with probabilities as actions, we select an area that captures most of the probability space (e.g. we sample a pair of parameters $\theta_1, \theta_2 \sim [-7, 7]$, since $\sigma(7) \approx 1$ where σ is the sigmoid function).

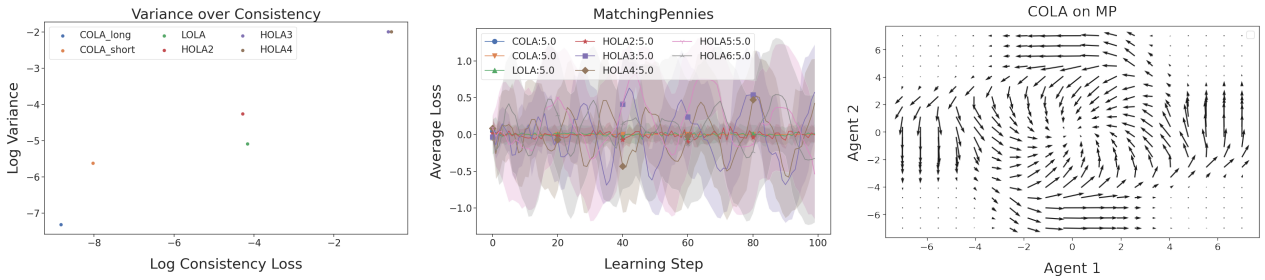
We optimize the mean of the sum of consistency losses,

$$C(\phi_1, \phi_2) := \mathbb{E}_{(\theta_1,\theta_2)\sim\mathcal{U}(\Theta)} \big[C_1(\phi_1, \phi_2, \theta_1, \theta_2) + C_2(\phi_1, \phi_2, \theta_1, \theta_2) \big],$$

by sampling parameter pairs (θ_1, θ_2) uniformly from Θ and feeding them to the neural networks f_1, f_2, each outputting an agent's parameter update. The weights ϕ_1, ϕ_2 are then updated by taking a gradient step to minimize C. We train the update functions until the loss has converged and use the learned update functions to train a pair of agent policies in the given game.

4.4. Theoretical results for COLA

In this section, we provide some initial theoretical results for COLA's uniqueness and convergence behavior, using the Tandem game (Letcher et al., 2019b) and the Hamiltonian game (Balduzzi et al., 2018) as examples. These are simple polynomial games, with losses given in Section 5. Proofs for the following propositions can be found in Appendices E, F and G, respectively.



(a) (b) (c)

Figure 2. Training in MP at look-ahead rate of $\alpha = 10$. (a) Axes are on a log-scale. Shown is the mean variance and consistency over 10 different runs. Each run, COLA was retrained. COLA_long was trained for 80k steps and COLA_short for 800 steps. (b) LOLA and HOLA find non-convergent or even diverging solutions, while COLA's converge. (c) Gradient field learned by COLA on MP at a look-ahead rate of 10.

First, we show that solutions to the consistency equations are in general not unique, even when restricting to linear update functions in the Tandem game. Interestingly, empirically, COLA does seem to consistently converge to similar solutions regardless (see Table 7 in Appendix I.3).

Proposition 4.5. *Solutions to the consistency equations are not unique, even when restricted to linear solutions; more precisely, there exist several linear consistent solutions to the Tandem game.*

Second, we show that consistent solutions do not, in general, preserve SFPs, contradicting the hypothesis that LOLA's failure to preserve SFPs is due to its inconsistency (Letcher (2018), pp. 2, 26; Letcher (2018)). We experimentally confirm this result in Section 6.

Proposition 4.6. *Consistency does not imply preservation of SFPs: there is a consistent solution to the Tandem game with $\alpha = 1$ that fails to preserve **any** SFP. Moreover, for any $\alpha > 0$, there are **no** linear consistent solutions to the Tandem game that preserve more than one SFP.*

Third, we show that COLA can have more robust convergence behavior than LOLA and SOS:

Proposition 4.7. *For any non-zero initial parameters and any $\alpha > 1$, LOLA and SOS have divergent iterates in the Hamiltonian game. By contrast, any linear solution to the consistency equations converges to the origin for any initial parameters and **any** look-ahead rate $\alpha > 0$; moreover, the speed of convergence strictly increases with α.*

5. Experiments

We perform experiments on a set of games from the literature (Balduzzi et al., 2018; Letcher et al., 2019b) using LOLA, SOS and CGD as baselines. For details on the training procedure of COLA, we refer the reader to Appendix H.

First, we compare HOLA and COLA on polynomial general-sum games, including the Tandem game (Letcher et al., 2019b), where LOLA fails to converge to SFPs. Second, we investigate non-polynomial games, specifically the zero-sum Matching Pennies (MP) game, the general-sum Ultimatum game (Hutter, 2021) and the IPD (Axelrod & Hamilton, 1981; Harper et al., 2017).

Polynomial games. Losses in the **Tandem game** (Letcher et al., 2019b) are given by $L^1(x, y) = (x+y)^2 - 2x$ and $L^2(x, y) = (x + y)^2 - 2y$ for agent 1 and 2 respectively. The Tandem game was introduced to show that LOLA fails to preserve SFPs at $x + y = 1$ and instead converges to Pareto-dominated solutions (Letcher et al., 2019b). Additionally to the Tandem game, we investigate the algorithms on the **Hamiltonian game**, $L^1(x, y) = xy$ and $L^2(x, y) = -xy$; and the **Balduzzi game**, where $L^1(x, y) = \frac{1}{2}x^2 + 10xy$ and $L^2(x, y) = \frac{1}{2}y^2 - 10xy$ (Balduzzi et al., 2018).

Matching Pennies. The payoff matrix for MP (Lee & K, 1967) is shown in Appendix I.3 in Table 6. Each policy is parameterized with a single parameter, the log-odds of choosing heads $p_{\text{heads}} = \sigma(\theta_A)$. In this game, the unique Nash equilibrium is playing heads half the time.

Ultimatum game. The binary, single-shot Ultimatum game (Güth et al., 1982; Sanfey et al., 2003; Oosterbeek et al., 2004; Henrich et al., 2006) is set up as follows. Player 1 has access to \$10. They can split the money fairly with player 2 (\$5 for each player) or they can split it unfairly (\$8 for player 1, \$2 for player 2). Player 2 can either accept or reject the proposed split. If player 2 rejects, the reward is 0 for both players. If player 2 accepts, the reward follows the proposed split. Player 1's parameter is the log-odds of proposing a fair split $p_{\text{fair}} = \sigma(\theta_1)$. Player 2's parameter is the log-odds of accepting the unfair split (assuming that player 2 always accepts fair splits) $p_{\text{accept}} = \sigma(\theta_2)$. In terms

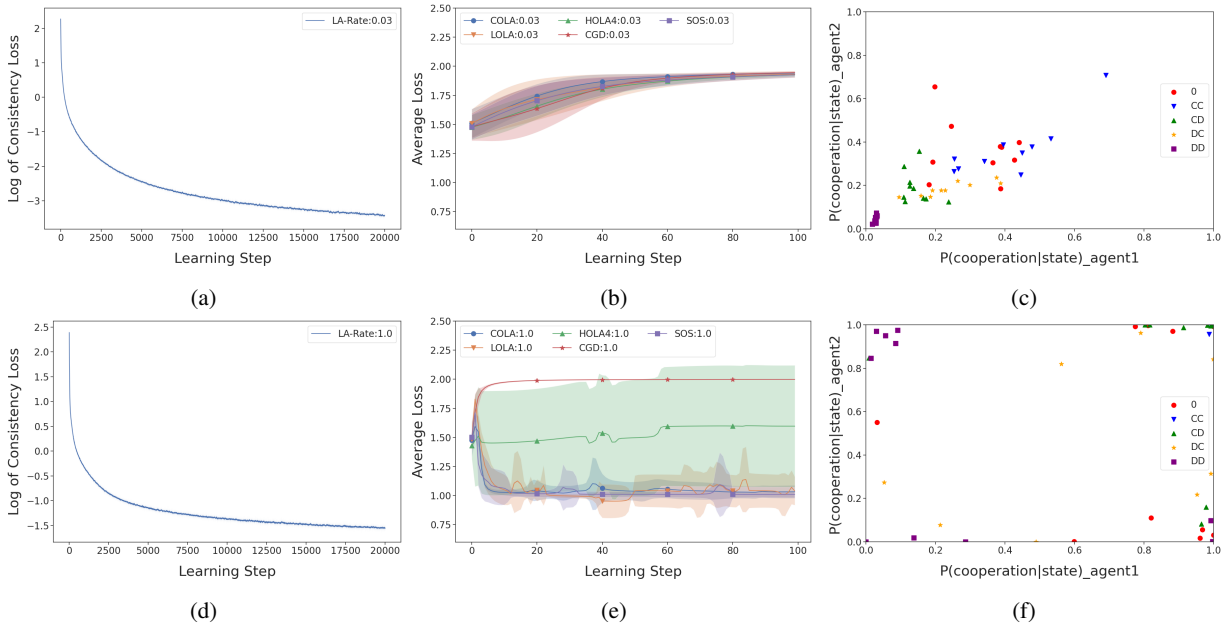


Figure 3. IPD results: Subfigure (a) / (d) show the consistency loss with shaded standard error over 10 independent training runs for look-ahead rates of 0.03 / 1.0, (b) / (e) the average loss and (c) / (f) the policy for the first player, both for the same pair of look-ahead rates. At low look-ahead HOLA defects and at high ones it diverges, also leading to high loss.

of losses, we have

$$L_1 = -\left(5p_{\text{fair}} + 8(1-p_{\text{fair}})p_{\text{accept}}\right)$$
$$L_2 = -\left(5p_{\text{fair}} + 2(1-p_{\text{fair}})p_{\text{accept}}\right).$$

IPD. We investigate the infinitely iterated Prisoner's Dilemma (IPD) (Axelrod & Hamilton, 1981; Harper et al., 2017) with discount factor $\gamma = 0.96$ and the usual payout function (see Appendix I.6). An agent i is defined through 5 parameters, the log-odds of cooperating in the first time step and across each of the four possible tuples of past actions of both players in the later steps.

6. Results

First, we report and compare the learning outcomes achieved by COLA and our baselines. We find that COLA update functions converge even under high look-ahead rates and learn socially desirable solutions. We also confirm our theoretical result (Proposition 4.4) that *CGD does not equal iLOLA*, contradicting Schäfer & Anandkumar (2019), and that COLA does not, in general, maintain SFPs (Proposition 4.6), contradicting the prior belief that this shortcoming is caused by inconsistency.

Second, we provide a more in-depth empirical analysis and comparison of the COLA and HOLA update functions, showing that COLA and HOLA tend to coincide when the latter converges, and that COLA is able to find consistent solutions even when HOLA diverges. Moreover, while COLA's solutions are not unique in theory (Proposition 4.5), we empirically find that in our examples COLA tends to find similar solutions across different independent training runs. Additional results supporting the above findings are reported in Appendix I.

Learning Outcomes. In the Tandem game (Figure 1d), we see that COLA and HOLA8 converge to similar outcomes in the game, whereas CGD does not. This supports our theoretical result that CGD does not equal iLOLA (Proposition 4.4). We also see that COLA does not recover SFPs, thus experimentally confirming Proposition 4.6. In contrast to LOLA, HOLA and SOS, COLA finds a convergent solution even at a high look-ahead rate (see COLA:0.8 in Figure 1d and Figure 4b in Appendix I.1). CGD is the only other algorithm in the comparison that also shows robustness to high look-ahead rates in the Tandem game.

On the IPD, all algorithms find the defect-defect strategy on low look-ahead rates (Figure 3b). At high look-ahead rates, COLA finds a strategy qualitatively similar to tit-for-tat, as displayed in Figure 3f, though more noisy. However, COLA still achieves close to the optimal total loss, in contrast to CGD, which finds defect-defect even at a high look-ahead rate (see Figure 13 in Appendix I.6). The fact that, unlike HOLA and COLA, CGD finds defect-defect, further confirms that CGD does not equal iLOLA.

On MP at high look-ahead rates, SOS and LOLA mostly

Table 3. IPD: Over multiple look-ahead rates we compare (a) the consistency losses and (b) the cosine similarity between COLA and LOLA, HOLA2, and HOLA4. The values represent the mean over 250 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA2	HOLA4	COLA
1.0	39.56	21.16	381.21	1.06±0.09
0.03	2e-3	5e-6	9e-8	0.16±0.02

(b)

α	LOLA	HOLA2	HOLA4
1.0	0.73±0.02	0.63±0.01	0.46±0.03
0.03	0.97±0.01	0.97±0.01	0.97±0.01

don't converge, whereas COLA converges *even faster* with a high look-ahead rate (see Figure 2a), confirming Proposition 4.7 experimentally (also see Figure 6b and 7b in Appendix I.2). To further investigate the influence of consistency on learning behavior, we plot the consistency of an update function against the variance of the losses across learning steps achieved by that function, for different orders of HOLA and for COLA (Figure 2b). At a high look-ahead rate in Matching Pennies, we find that more consistent update functions tend to lead to lower variance across training, demonstrating a potential benefit of increased consistency at least at high look-ahead rates.

For the Ultimatum game, we find that COLA is the only method that finds the fair solution consistently at a high look-ahead rate, whereas SOS, LOLA, and CGD do not (Figure 1f). At low look-ahead rates, all algorithms find the unfair solution (see Figure 10b in Appendix I.4). This demonstrates an advantage of COLA over our baselines and shows that higher look-ahead rates can lead to better learning outcomes.

Lastly, we introduce the Chicken game in Appendix I.5. Both Taylor LOLA and SOS crash, whereas COLA, HOLA, CGD, and exact LOLA swerve at high look-ahead rates (Figure 12d). Crashing in Chicken results in a catastrophic payout for both agents, whereas swerving results in a jointly preferable outcome.[1]

Update functions. Turning to our analysis of COLA and HOLA update functions, we first investigate how increasing the order of HOLA affects the consistency of its updates. As shown in Table 1a, 2a and 3a, HOLA's updates become more consistent with increasing order, but only below a certain, game-specific look-ahead rate threshold. Above that threshold, HOLA's updates become less consistent with increasing order.

Second, we compare the consistency losses of COLA and HOLA. In the aforementioned tables, we observe that COLA achieves low consistency losses on most games. Below the threshold, COLA finds similarly low consistency losses as HOLA, though there HOLA's are lower in the non-polynomial games. Above the threshold, COLA finds consistent updates, even when HOLA does not. A visualization of the update function learned by COLA at a high look-ahead rate on the MP is given in Figure 2c.

For the IPD, COLA's consistency losses are high compared to other games, but much lower than HOLA's consistency losses at high look-ahead rates. We leave it to future work to find methods that obtain more consistent solutions.

Third, we are interested whether COLA and HOLA find similar solutions. We calculate the cosine similarity between the respective update functions over Θ. As we show in Table 1b, 2b and 3b, COLA and HOLA find very similar solutions when HOLA's updates converge, i.e., when the look-ahead rate is below the threshold. Above the threshold, COLA's and HOLA's updates unsurprisingly become less similar with increasing order, as HOLA's updates diverge with increasing order.

Lastly, we investigate Proposition 4.5 empirically and find that COLA finds similar solutions in Tandem and MP over 5 training runs (see Table 7 in Appendix I.3). Moreover, the small standard deviations in Table 3b indicate that COLA also finds similar solutions over different runs in the IPD.

7. Conclusion and Future Work

In this paper, we corrected a claim made in prior work (Schäfer & Anandkumar, 2019), clearing up the relation between the CGD and LOLA algorithms. We also showed that iLOLA solves part of the consistency problem of LOLA. We introduced COLA, which finds consistent solutions without requiring many recursive computations like iLOLA. It was believed that inconsistency leads to arrogant behaviour and lack of preservation of SFPs. We showed that even with consistency, opponent shaping behaves *arrogantly*, pointing towards a fundamental open problem for the method.

In a set of games, we found that COLA tends to find prosocial solutions. Although COLA's solutions are not unique in theory, empirically, COLA tends to find similar solutions in different runs. It coincides with iLOLA when HOLA converges and finds consistent update functions even when HOLA fails to converge with increasing order. Moreover, we showed empirically (and in one case theoretically) that

[1] Interestingly, in contrast to Taylor LOLA, exact LOLA swerves. The Chicken game is the only game where we found a difference in learning behavior between exact LOLA and Taylor LOLA.

COLA update functions converge under a wider range of look-ahead rates than HOLA and LOLA update functions.

This work raises many questions for future work, such as the existence of solutions to the COLA equations in general games and general properties of convergence and learning outcomes. Moreover, additional work is needed to scale COLA to large settings such as GANs or Deep RL, or settings with more than two players. Another interesting axis is addressing further inconsistent aspects of LOLA as identified in Letcher et al. (2019b).

Acknowledgements

A part of this work was done while Timon Willi and Jakob Foerster were at the Vector Institute, University of Toronto. They are grateful for the access to the Vector Institute's compute infrastructure. They are also grateful for the access to the Advanced Research Computing (ARC) infrastructure. Johannes Treutlein is grateful for support by the Center on Long-Term Risk.

References

Albrecht, S. V. and Stone, P. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 547–555, 2017.

Axelrod, R. and Hamilton, W. D. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

Azizian, W., Mitliagkas, I., Lacoste-Julien, S., and Gidel, G. A tight and unified analysis of gradient-based methods for a whole spectrum of differentiable games. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2863–2873, 2020.

Balduzzi, D., Racanière, S., Martens, J., Foerster, J. N., Tuyls, K., and Graepel, T. The mechanics of n-player differentiable games. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 363–372, 2018.

Barto, A. G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discret. Event Dyn. Syst.*, 13(1-2):41–77, 2003.

Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K., and Graepel, T. Open problems in cooperative ai. In *Cooperative AI workshop*, 2021.

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

Foerster, J. N., Farquhar, G., Al-Shedivat, M., Rocktäschel, T., Xing, E. P., and Whiteson, S. Dice: The infinitely differentiable monte carlo estimator. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1524–1533, 2018b.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pp. 2672–2680, 2014.

Güth, W., Schmittberger, R., and Schwarze, B. An experimental analysis of ultimatum bargaining. *Journal of Economic Behavior & Organization*, 3(4):367–388, 1982.

Harper, M., Knight, V., Jones, M., Koutsovoulos, G., Glynatsi, N. E., and Campbell, O. Reinforcement learning produces dominant strategies for the iterated prisoner's dilemma. *PLOS ONE*, 12(12):e0188046, 2017.

He, H. and Boyd-Graber, J. L. Opponent modeling in deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1804–1813, 2016.

Henrich, J., Boyd, R., Bowles, S., Camerer, C., Fehr, E., and Gintis, H. Foundations of Human Sociality: Economic Experiments and Ethnographic Evidence From Fifteen Small-Scale Societies. In *American Anthropologist*, volume 108. 2006.

Hutter, A. Learning in two-player games between transparent opponents. arXiv preprint arXiv:2012.02671, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Korpelevich, G. M. The extragradient method for finding saddle points and other problems. volume 13, pp. 35–49, 1977.

Lee, K. and K, L. *The Application of Decision Theory and Dynamic Programming to Adaptive Control Systems*. Thesis, 1967.

Letcher, A. Stability and exploitation in differentiable games. Master's thesis, University of Oxford, 2018.

Letcher, A., Balduzzi, D., Racanière, S., Martens, J., Foerster, J. N., Tuyls, K., and Graepel, T. Differentiable game mechanics. *J. Mach. Learn. Res.*, 20:84:1–84:40, 2019a.

Letcher, A., Foerster, J. N., Balduzzi, D., Rocktäschel, T., and Whiteson, S. Stable opponent shaping in differentiable games. In *7th International Conference on Learning Representations*, 2019b.

Lu, C., Willi, T., Schroeder de Witt, C., and Foerster, J. Model-free opponent shaping. *arXiv preprint arXiv:2205.01447*, 2022.

Mazumdar, E. V., Jordan, M. I., and Sastry, S. S. On finding local nash equilibria (and only local nash equilibria) in zero-sum games. arXiv preprint arXiv:1901.00838, 2019.

Mealing, R. and Shapiro, J. L. Opponent modeling by expectation-maximization and sequence prediction in simplified poker. *IEEE Trans. Comput. Intell. AI Games*, 9(1):11–24, 2017.

Mescheder, L. M., Nowozin, S., and Geiger, A. The numerics of gans. In *Advances in Neural Information Processing Systems*, volume 30, pp. 1825–1835, 2017.

Oosterbeek, H., Sloof, R., and van de Kuilen, G. Cultural Differences in Ultimatum Game Experiments: Evidence from a Meta-Analysis. *Experimental Economics*, 7(2): 171–188, 2004.

Oroojlooyjadid, A. and Hajinezhad, D. A review of cooperative multi-agent deep reinforcement learning. 2019.

Osborne, M. J. and Rubinstein, A. *A Course in Game Theory*. The MIT Press, Cambridge, MA, 1994.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pp. 8024–8035. 2019.

Racanière, S., Weber, T., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P. W., Hassabis, D., Silver, D., and Wierstra, D. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5690–5701, 2017.

Sanfey, A. G., Rilling, J. K., Aronson, J. A., Nystrom, L. E., and Cohen, J. D. The Neural Basis of Economic Decision-Making in the Ultimatum Game. *Science*, 300(5626): 1755–1758, 2003.

Schäfer, F. and Anandkumar, A. Competitive gradient descent. In *Advances in Neural Information Processing Systems*, volume 32, pp. 7623–7633, 2019.

Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. In Meyer, J. A. and Wilson, S. W. (eds.), *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 222–227. MIT Press/Bradford Books, 1991.

Schäfer, F., Anandkumar, A., and Owhadi, H. Competitive mirror descent. arXiv preprint arXiv:2006.10179, 2020.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

Synnaeve, G. and Bessière, P. A bayesian model for opening prediction in RTS games with application to starcraft. In *IEEE Conference on Computational Intelligence and Games*, pp. 281–288, 2011.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

Weber, B. G. and Mateas, M. A data mining approach to strategy prediction. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, pp. 140–147, 2009.

Wen, Y., Yang, Y., Luo, R., Wang, J., and Pan, W. Probabilistic recursive reasoning for multi-agent reinforcement learning. In *7th International Conference on Learning Representations*, 2019.

Zhang, C. and Lesser, V. R. Multi-agent learning with policy prediction. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

3.1 Limitations and Future Work

The work presented in this chapter introduced COLA, a method that successfully addresses LOLA's theoretical inconsistency by learning mutually consistent update functions. We demonstrate empirically that COLA exhibits more robust convergence behaviour than HOLA, particularly under higher look-ahead rates, and often identifies more cooperative solutions in mixed-motive games. Additionally, we clarify the relationship between OS methods and CGD, showing that consistency alone cannot guarantee the preservation of SFPs.

Next to these contributions, COLA opens avenues for future investigation. Firstly, we conduct the theoretical analysis and empirical evaluations of COLA in low-dimensional polynomial and matrix games. While simplifying the isolation of the effects of consistency, the scalability of the COLA formulation to high-dimensional state and action spaces, typical of deep RL environments, remains an open question.

Computing or approximating second-order derivative information (inherent in differentiating the consistency loss through opponent updates) poses significant computational challenges for large neural network policies. Developing more scalable approximations or implementations of COLA suitable for deep MARL is a key direction for future work.

Secondly, while COLA finds consistent solutions, we demonstrated that these solutions are not necessarily unique. Understanding the properties of the different consistent solutions COLA might converge to and whether specific desirable solutions (e.g., those with higher social welfare) can be targeted or selected requires further investigation.

Thirdly, finding that consistency does not guarantee SFP preservation refutes a common hypothesis but leaves the question of what is required to achieve stability alongside opponent shaping. Future research could explore integrating COLA principles with mechanisms specifically designed for stability, such as those proposed by Letcher et al. (2019b) for SOS.

Finally, the practical application of COLA within standard deep MARL training paradigms (e.g., using actor-critic architectures, experience replay) was not explored

here. Integrating COLA effectively with these techniques, similar to how Proximal Learning With Opponent-Learning Awareness (POLA; Zhao et al., 2022) integrated PPO with LOLA, is necessary to assess its practical utility in complex deep MARL problems. While COLA provides a more robust theoretical grounding for opponent shaping, scaling such second-order OS methods to the complexity encountered in many real-world MARL tasks remains a significant hurdle.

4

Scaling Opponent Shaping to High Dimensional Games

While Chapter 3 addresses the foundational issue of theoretical consistency in Opponent Shaping by introducing COLA (Willi et al., 2022), its evaluation, like that of prior OS methods (Foerster et al., 2018; Letcher et al., 2019b; Schäfer and Anandkumar, 2019), was confined to low-dimensional settings such as matrix games. While helpful in isolating theoretical properties, these environments lack many characteristics of complex, real-world multi-agent interactions. Truly scaling OS requires agents capable of handling high-dimensional observations, partial observability, long time horizons, and decision-making involving temporally-extended actions where the consequences of choices unfold over many steps. The applicability and effectiveness of OS principles in such complex scenarios remained largely unexplored.

Scaling OS presents unique difficulties beyond those faced by standard MARL algorithms. Explicitly calculating or differentiating through opponent updates (as in LOLA/COLA) becomes computationally intractable with the large neural networks required for high-dimensional inputs or introduces high variance in training dynamics. Meta-learning OS approaches, which frame shaping as learning an update rule, offer an alternative by avoiding explicit higher-order derivatives. Methods like

Model-Free Opponent Shaping (M-FOS; Lu et al., 2022b) and The Good Shepherd (GS; Balaguer et al., 2022) fall into this category. However, these methods have limitations. GS lacks the necessary memory to condition shaping on interaction history. At the same time, M-FOS, although memory-equipped, has only shown preliminary results in inadequate benchmark environments, such as the CoinGame (Lerer and Peysakhovich, 2017). A principled and empirically validated method for scaling OS robustly to high-dimensional, partially observable settings with long-term dependencies was lacking.

Presenting the work published in Khan et al. (2024b), this chapter tackles the scaling challenge for opponent shaping. We analyse the architectural components underpinning meta-learning-based OS, proposing and dissecting the roles of history (intra-episodic memory) and context (inter-episodic memory) for tracking opponent adaptation. We identify that both memory types are necessary, but M-FOS's reliance on separate meta- and inner-policies introduces an unnecessary architectural bottleneck.

This chapter's core contribution is developing and evaluating Shaper, a simplified meta-learning OS algorithm. Shaper utilises a single recurrent neural network architecture to capture history and context efficiently, removing the bottleneck while keeping the ability to shape. We further formalise the role of batch averaging, a previously implicit technique, demonstrating its importance for integrating information across parallel environments when opponent learning depends on batched data.

Critically, we evaluate Shaper in novel, challenging general-sum grid world environments derived from the Melting Pot suite (Leibo et al., 2021). These "in-the-Matrix" environments feature temporally-extended actions, partial observability, and significantly larger state spaces than the matrix games used in Chapter 3. Our empirical results demonstrate that Shaper successfully scales OS to these complex settings, effectively shaping and improving individual and collective outcomes where prior methods and baselines struggle. This work advances the practical applicability of opponent shaping, demonstrating its potential beyond simple games towards more realistic MARL problems.

Scaling Opponent Shaping to High Dimensional Games

Akbir Khan*
University College London
akbir.khan.13@ucl.ac.uk

Timon Willi*
University of Oxford
timon.willi@eng.ox.ac.uk

Newton Kwan*
University College London

Andrea Tacchetti
Deepmind

Chris Lu
University of Oxford

Edward Grefenstette
University College London

Tim Rocktäschel
University College London

Jakob Foerster
University of Oxford

ABSTRACT

In multi-agent settings with mixed incentives, methods developed for zero-sum games have been shown to lead to detrimental outcomes. To address this issue, *opponent shaping* (OS) methods explicitly learn to influence the learning dynamics of co-players and empirically lead to improved individual *and* collective outcomes. However, OS methods have only been evaluated in low-dimensional environments due to the challenges associated with estimating higher-order derivatives or scaling model-free meta-learning. Alternative methods that scale to more complex settings either converge to undesirable solutions or rely on unrealistic assumptions about the environment or co-players. In this paper, we successfully scale an OS-based approach to general-sum games with temporally-extended actions and long-time horizons for the first time. After analysing the representations of the meta-state and history used by previous algorithms, we propose a simplified version called Shaper. We show empirically that Shaper leads to improved individual and collective outcomes in a range of challenging settings from literature. We further formalize a technique previously implicit in the literature, and analyse its contribution to opponent shaping. We show empirically that this technique is helpful for the functioning of prior methods in certain environments. Lastly, we show that previous environments, such as the CoinGame, are inadequate for analysing temporally-extended general-sum interactions[1].

KEYWORDS

Multi-Agent Reinforcement Learning, Opponent Shaping, General-Sum Games

ACM Reference Format:
Akbir Khan, Timon Willi[1], Newton Kwan[1], Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. 2024. Scaling Opponent Shaping to High Dimensional Games. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 29 pages.

*Equal Contribution.
[1]Blogpost available at sites.google.com/view/scale-os/

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.

1 INTRODUCTION

From personal assistants and chat-bots to self-driving cars and recommendation systems, the world of software is becoming increasingly *multi-agent* as these systems are continuously learning and interacting with each other in fully cooperative, fully competitive, and general-sum settings.

In this paper we investigate interacting, learning agents in general-sum settings. In such settings, commonly-used RL methods developed for zero-sum games can lead to disastrous outcomes [7]. For example, in real-world scenarios like pollution and international arms races [8, 40], such agents would fail to realise that they're better off cooperating, even if it means they're potentially worse off than their co-players. Poor performance could also lead to being extorted in social-dilemma-like scenarios [30, 33].

While multi-agent learning research has shown great success in strictly competitive [6, 18, 39, 42] and fully cooperative settings [13, 34], this success does not transfer to general-sum settings [12, 28]: In competitive games, agents can learn Nash equilibrium strategies by iteratively best-responding to suitable mixtures of past opponents. Similarly, best-responding to rational co-players leads to the desirable equilibria in cooperative games (assuming joint training). In contrast, many Nash equilibria coincide with globally worst welfare outcomes in general-sum settings, rendering the above learning paradigms ineffective. For example, in the iterated prisoner's dilemma [IPD, 2, 15], naive best-response dynamics converge to unconditional mutual defection [12] rather than Nash equilibria with higher social welfare.

It is important that general-sum learning methods scale to *high-dimensional* settings, such as those with longer-time horizons, larger state spaces and temporally-extended actions, as these environments are more akin to the real world. In matrix games, cooperation and defection are clearly defined atomic actions, whilst in more complex environments such as autonomous driving, cooperation and defection are defined over sequences of actions (e.g. a path towards a cooperative/defective location). Scalable methods [19, 22, 32, 46] that manage to avoid *unconditional defection* in these settings rely heavily on *reward shaping*, which blurs the line between the problem setting ("social dilemma") and the method.

As an alternative approach, *opponent shaping* (OS) methods recognise that the actions of any one agent influence their co-player's policy and seek to use this mechanism to their advantage [12, 20, 27, 44]. However, many past OS methods require privileged

information to shape their opponents and are myopic since anticipating many steps is intractable. Model-Free Opponent Shaping [M-FOS, 30] and The Good Shepherd [GS, 4] solve the issues above by framing opponent shaping as a meta-learning problem, which our method inherits and builds upon. However, M-FOS presents only preliminary results on the higher-dimensional CoinGame [26] benchmark, and GS none at all.

To scale OS agents to higher-dimensional benchmarks, we systematically evaluate the architectural components of the M-FOS and GS algorithms. We identify two forms of memory—*history* and *context*. *History* captures intra-episode information and *context* inter-episode information. We find empirically that both are necessary to achieve shaping. M-FOS captures both types of memory (though not completely), whereas GS does not. However, we identify a bottleneck in the M-FOS method, as M-FOS requires two separate policies to capture *context* and *history*, where only one is necessary. Using this finding, we propose a new method, called Shaper, removing the unnecessary bottleneck from M-FOS.

Beyond these memory components, we uncover another element used implicitly in prior work but never formally introduced or analysed: averaging across the batch of trajectories at each meta step. This ensures that the hidden states of the opponent shaping algorithm carry information from the entire batch, rather than just a single batch dimension. We formalise this technique and empirically investigate its importance. Our analysis shows that while this technique improves previous methods like M-FOS in certain environments, it is not essential for our proposed method, Shaper, in typical environment settings. This highlights the value of our formalisation and empirical analysis in understanding and improving upon existing methods.

Shaper outperforms previous OS methods in general-sum games with long-time horizons and temporally-extended actions. We showcase our performance on the "IPD in the Matrix" and "IMP in the Matrix" games, introduced by the *melting pot* suite [25]. These have a 30x state-space than environments previously used to evaluate OS and contain more complex interaction dynamics. Additionally, we consider shaping on sequential matrix games with 100x longer horizons than their previously used counterparts. We demonstrate empirically that our simplification of M-FOS helps scalability, that only *evolutionary-based* meta-learning is effective in these long-horizon games, and that previous evaluation environments, such as the CoinGame [26], are inadequate for analysing OS in temporally-extended, general-sum interactions.

2 BACKGROUND

What is a Game? We formalise our environments as Partially Observable Stochastic Games [38, POSG]. A POSG is given by the tuple $\mathcal{M} = \langle N, \mathcal{A}, O, S, \mathcal{T}, \mathcal{I}, \mathcal{R}, \gamma \rangle$, where $\mathcal{A}$, O, and S denote the action, observation, and state space, respectively. These parameters can be distinct at every time step and also incorporated into the transition function $\mathcal{T} : S \times \mathbf{A} \to \Delta S$, where $\mathbf{A} \equiv \mathcal{A}^n$ is the joint action of all agents. Each agent draws individual observations according to the observation function $\mathcal{I} : S \times N \to O$ and obtains a reward according to their reward function $\mathcal{R} : S \times \mathbf{A} \times N \to \mathbb{R}$ where $N = \{1, \ldots, n\}$. POSGs represent general-sum games. The single-player case, $N = \{1\}$, of POSGs are Partially Observable Markov Decision Processes (POMDPs).

Algorithm 1 Shaper Update: Given a POSG $\mathcal{M}$, policies $\pi_{\phi_i}, \pi_{\phi_{-i}}$ and their respective initial hidden states h_i, h_{-i} and a distribution of initial co-players ρ_ϕ, this algorithm updates a meta-agent policy ϕ_i over T trials consisting of E episodes.

Require: $\mathcal{M}, \phi_i, \rho_\phi, E, T$
for $t = 0$ **to** T **do**
  Initialise trial reward $\bar{J} = 0$
  Initialise meta-agent hidden state $h_i = \mathbf{0}$
  Sample co-players $\phi_{-i} \sim \rho_\phi$
  for $e = 0$ **to** E **do**
    Initialise co-players' $h_{-i} = \mathbf{0}$
    $J_i, J_{-i}, h_i', h_{-i}' = \mathcal{M}(\phi_i, \phi_{-i}, h_i, h_{-i})$
    Update ϕ_{-i} according to co-players' update rule.
    $h_i \leftarrow h_i'$
    $\bar{J} \leftarrow \bar{J} + J_i$
  end for
  Update ϕ_i with respect to $\bar{J}$
end for

What is Shaping? Shaping is acting to manipulate the co-player's learning dynamics (and subsequent behaviour) [12], where co-players are any other participants in the game. Newer shaping methods frame shaping as a meta-learning problem [4, 20, 30]. We next present the meta-learning problem setting presented by M-FOS since our work is a simplified case of the M-FOS framework.

What is M-FOS? Conceptually, M-FOS separates the task of shaping (the meta-game) from the task of playing the game. Specifically, the meta-game is formulated as a POMDP $\langle \overline{\mathcal{A}}, \overline{O}, \overline{\mathcal{S}}, \overline{\mathcal{T}}, \overline{\mathcal{I}}, \overline{\mathcal{R}}, \bar{\gamma} \rangle$ over an underlying general-sum game, represented by a POSG $\mathcal{M}$, where the overline indicates the single-agent version of the elements defined for POSGs. In the "shaping" POMDP, the meta-state $\overline{\mathcal{S}}$ contains the policies of all players in the underlying POSG: $\bar{s}_e = \left(\phi_i^{e-1}, \phi_{-i}^{e-1}\right) \in \overline{\mathcal{S}}$, where e indexes the episodes and $(i, -i)$ index all agents. The meta-observation is all observations of the previous episode in the underlying game, i.e., $\bar{o}_e = (o_0^{e-1}, o_1^{e-1}, ..., o_K^{e-1})$, where K is the length of an episode. The meta-action space $\overline{\mathcal{A}}$ consists of the policy parameterisation of the inner agent i (in practice a vector conditioning the policy), i.e., $\overline{a}_e = \phi_i^e$.

M-FOS training works as follows. The meta-agent trains over a sequence of T *trials* (denoted "meta-episodes" in the original paper). Each trial contains E episodes. At the end of each episode e within a trial t, conditioned on both agent's policies, the co-players update their parameters with respect to the expected episodic return $J_{-i}^e = \mathbb{E}\left[\sum_{k=0}^K \gamma^k r_{-i}^k(\phi_i^e, \phi_{-i}^e)\right]$, where K is the length of an episode. For example, if the co-players were Naive Learners, i.e., agents not accounting for the learning dynamics of the co-player, with learning rate α, the update is: $\phi_j^{e+1} = \phi_j^e + \alpha \nabla_{\phi_j^e} J_j^e$, for $j \in -i$.

In contrast to the co-player's update, the meta-agent learns an update function for the parameters of their inner agent, i.e., $\bar{a}_e = \phi_i^e \sim \pi_\theta(\cdot \mid \bar{o}_e)$, where θ is the parameters of the meta-agent. The meta-agent optimises the meta-return $\bar{J} = \sum_{e=0}^E J_i^e$ (summed

over all episodes) at the end of a trial t using any model-free optimisation technique, e.g., PPO [37] or Evolution Strategies [ES, 35]. The meta-agent and the inner agent are usually represented as recurrent neural networks, such as LSTMs [17]. The meta-game setup allows the meta-agent to observe the results of the co-player's learning dynamics, enabling it *to learn to shape*. Though not formalized or discussed in detail in the paper, the original M-FOS averages across a batch of trajectories at each update step to ensure access to all information necessary for shaping. In Section 3, we introduce a formal definition of averaging across the batch and investigate its role for shaping. While published concurrently, the Good Shepherd [4] is a simplified version of M-FOS, in which the meta-agent and underlying agent are collapsed into a single agent without memory *that only updates after each trial*. The agent is represented by a feedforward neural network and has no memory. However, as GS was evaluated on infinitely iterated matrix games, where the state usually represents a one-step history, we consider GS to have one-step history.

Where have current shaping algorithms been evaluated? Both M-FOS and GS evaluate their shaping on infinitely-iterated matrix games. While this is a fruitful playground to discover complex strategies, such as tit-for-tat, infinitely-iterated matrix games do not contain temporally-extended actions or high dimensional state spaces. For example, in matrix games, cooperation simply consists of playing the "cooperation" action. However, in the real world, cooperation requires a repeated commitment to a cooperative strategy (where it is often unclear whether a given atomic action is cooperative). The CoinGame [26] supposedly addresses this shortcoming by incorporating IPD-like game dynamics into a gridworld. M-FOS presents very preliminary results on the CoinGame with no detailed analysis of the emerging strategies. However, as we show in Section 4 the CoinGame suffers from pathologies that enable shaping with simple strategies.

3 SHAPER: A SCALABLE OS METHOD

To introduce our method, we first analyse the role of memory in meta-learning-based OS. Memory is important because it enables a meta-agent to adapt its meta-policy within a trial since it only updates *parameters* after a trial.

If the meta-agent cannot adapt their policy within a trial, a co-player could simply learn the best-response to the meta-agent's policy. For example, in Rock-Paper-Scissors, the meta-agent would be forced to play the fully mixed strategy, as any deviation from it will be taken advantage of by the co-player. Instead a meta-agent with memory can adapt to the co-player's best response within a trial and potentially achieve a better meta-return, which we show in Section 5. Thus, memory is important if a meta-agent is to perform well in all general-sum games.

We define two forms of memory: *context* and *history*. Let us define *history* as a trajectory of a (partial) episode e, $\tau_e = \left(o_e^0, a_e^0, r_e^0, ..., r_e^K\right)$, and *context* as a trajectory of a (partial) trial t, $\bar{\tau}_t = (\tau_0, ..., \tau_E)$. *History* captures the dynamics within an episode and is crucial for implementing policies such as TFT that reward/punish the co-player based on past actions. In contrast, *context* captures the learning dynamics of the co-player as it contains the co-player's policy changes over many parameter updates. *Context* is important for shaping when the co-players *update dynamics* are non-stationary across a trial or need to be inferred from the changing policy itself across different episodes. It allows the shaper to adapt its inner policy to, e.g., a change of the co-player's learning rate or implicitely infer their objective function.

Using the above definitions, we express the policies as the following: M-FOS : $a \sim \pi_\phi(\cdot \mid \tau_e, \pi_\theta(\bar{o}_e))$ and GS : $a \sim \pi_\phi(\cdot \mid o_e^t)$. M-FOS captures one-step *context* via the memory of the meta-agent, and *history* via the memory of the inner agent but requires two agents to do so. In contrast, GS captures one-step history (if given by environment) but does not require a separate inner agent.

We propose Shaper, an algorithm requiring only one agent to capture *context and history*. This is accomplished via an RNN that retains its hidden state over episodes and only resets *after each trial*

$$\text{Shaper: } a \sim \pi_\phi(\cdot \mid \tau_e, \bar{\tau}_t) \tag{1}$$

Compared to GS, Shaper has access to history and context by *adding memory to the architecture and retaining the hidden state over the episodes*. Compared to M-FOS, Shaper only requires sampling from one action space. To contrast Shaper to M-FOS in more detail, we refer to Appendix M.

Shaper is trained as follows. Given a POSG $\mathcal{M}$, at the start of a trial, co-players $\phi_{-i} \sim \rho_\phi$ are sampled, where ρ_ϕ is the respective sampling distribution. Shaper's parameters ϕ_i and hidden state h_i are randomly initialised. During an episode of length K, agents take their actions, $a_i^k \sim \pi_{\phi_i}(\cdot \mid o_i^k, h_i^k)$. At each time step in the episode, the hidden state of the meta-agent is updated: $h_i^{k+1} = f(o_i^k, h_i^k)$. On receiving actions, the POSG returns rewards r_i^k, new observations o_i^{k+1} and a done flag d, indicating if an episode has ended.

When an inner episode terminates, the updated co-player ϕ_{-i}^{e+1} *and the meta-agent's hidden state* h_i^K are passed to the next episode. This process is repeated over E episodes in a trial. When a trial terminates, the meta-agent's policy is updated, maximising total *trial* reward, $\bar{J} = \sum_e^E J_i^e$. This leads to the following objective,

$$\max_{\phi_i} \mathbb{E}_{\rho(\phi),\rho(\mathcal{M})}\left[\bar{J}\right]. \tag{2}$$

In practice, the co-players optimise their parameters using some form of gradient descent, which typically involves batching the episodal trajectories. Assume $\phi_{-i}^e = G(\phi_{-i}^{e-1}, \boldsymbol{\tau}_{e-1})$ is some co-player's update function $G : \mathbb{R}^P \times \mathbb{R}^{B\times T} \to \mathbb{R}^P$, where P is the number of parameters of ϕ_{-i}, B is the batch size, i.e., number of environments for parallel training, and T is the length of the trajectory. Shaper then interacts with a co-player over a batch of environments, i.e., $\mathbf{a}_i^k \sim \pi_{\phi_i}(\cdot \mid \mathbf{o}_i^k, \mathbf{h}_i^k)$, where $\mathbf{a}_i^k \in \mathbb{R}^{B\times A}$, $\mathbf{o}_i^k \in \mathbb{R}^{B\times O}$, and $\mathbf{h}_i^k \in \mathbb{R}^{B\times H}$, and A, O, and H are the action-, observation-, and hidden-state-size respectively.

Shaper needs to account for the batched updates of the co-player because opponent shaping requires all the information determining the learning update of the co-player. For example, imagine Shaper plays with a co-player across a batch of environments with different reward functions. While the co-player updates its parameters based on a diverse set of trajectories from many reward functions, each of Shaper's hidden states only observes the trajectory of its respective reward function. Intuitively, if the reward functions are very diverse, the update derived from the whole batch would significantly differ

Table 1: Converged reward per episode (meta-agent, co-player) for agents trained with Naive Learners on the CoinGame, IPDitM and IMPitM. We report reward per episode for better interpretability. The mean is reported across 100 seeds with standard deviations.

	CoinGame		IPD in the Matrix		IMP in the Matrix	
SHAPER	**4.63 ± 0.66**,	**−3.35 ± 0.67**	**22.44 ± 1.12**,	**21.49 ± 0.67**	**0.14 ± 0.06**,	**−0.14 ± 0.06**
M-FOS (ES)	3.13 ± 0.40,	2.27 ± 0.38	15.49 ± 1.28,	23.88 ± 0.93	**0.11 ± 0.07**,	**−0.11 ± 0.07**
M-FOS (RL)	0.94 ± 0.68,	−0.23 ± 0.52	7.42 ± 0.21,	7.28 ± 0.15	0.04 ± 0.00,	−0.04 ± 0.00
GS	**5.44 ± 0.61**,	**−4.17 ± 0.48**	16.16 ± 1.33,	24.35 ± 0.83	0.00 ± 0.00,	0.00 ± 0.00
PT-NL	0.70 ± 0.58,	−0.3 ± 0.47	6.33 ± 0.33,	6.96 ± 0.31	−0.17 ± 0.10,	0.17 ± 0.10
CT-NL	0.47 ± 0.83,	0.26 ± 0.30	5.56 ± 0.02,	5.56 ± 0.02	−0.10 ± 0.06,	0.10 ± 0.06

from the update estimated from a single environment trajectory, as observed by the hidden state. We thus define the *batched context* as a trajectory of a batched (partial) trial $\bar{\tau}_t = (\tau_0, ..., \tau_E)$.

$$\text{Shaper: } a \sim \pi_\phi(\cdot \mid \tau_e, \bar{\tau}_t)$$

This insight leads to the consequence that Shaper needs to consolidate information across its batch of hidden states, at least at every co-player update. To address this issue, Shaper averages over its hidden states across the batch at each step, combined with a skip connection to ensure "situational" awareness of the hidden state's respective environment (see Figure 6).

$$\mathbf{h}_i^{k+1} = \lambda \left(\frac{1}{B} \sum_{l=0}^{B} h_{i,l}^k \right) + (1-\lambda)\mathbf{h}_i^k \tag{3}$$

$$\mathbf{h}_i^{k+1} = f(o_i^k, \mathbf{h}_i^{k+1}) \tag{4}$$

This approach ensures that SHAPER effectively shapes its co-players while accounting for the diverse set of trajectories that inform their gradient updates, captured by the following meta-return function with the expectation over the batched gradient update of the co-player:

$$\begin{aligned} \bar{J}^{\text{ES}} = \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, I_d)} \Bigg[& \mathbb{E}_{\bar{\tau} \sim \pi_{\phi_i + \epsilon\sigma}; \phi^e_{-i} \sim G(\phi^{e-1}_{-i}, \tau_{e-1})} \\ & \times \left[\mathbb{E}_{\tau_e \sim \pi_{\phi^e_{-i}}} \left[\Sigma_{k=0}^{K} \gamma^k r^k(\phi^e_{-i}, \phi_i + \epsilon\sigma) \right] \right] \Bigg] \end{aligned} \tag{5}$$

4 EXPERIMENTS

Here we present the test environments and our evaluation protocol for SHAPER. We also explain our ablation experiments helping us evaluate the role of memory in OS.

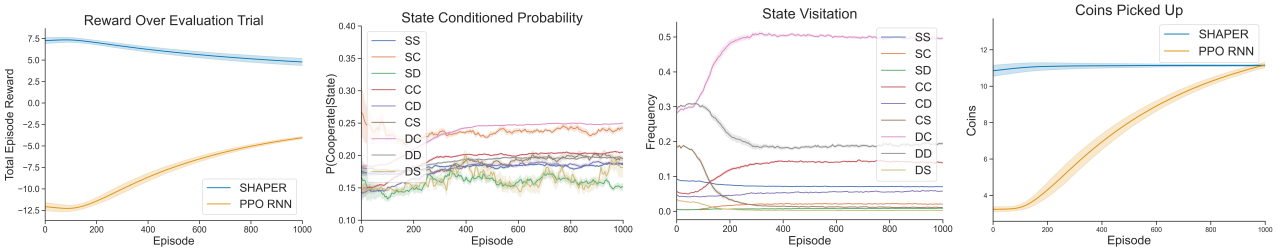
The **Prisoner's Dilemma** is a well-known and widely studied general-sum game illustrating that two self-interested agents do not cooperate even if it is globally optimal. The players either cooperate (C) or defect (D) and receive a payoff according to Table 5a. In the *iterated* prisoner's dilemma (IPD), the agents repeatedly play the prisoner's dilemma and observe the previous action of both players.Past research used the infinite IPD in their experiments [4, 12, 28, 30, 44]. In the infinite version, the exact value function and gradients thereof are calculated directly from the policy weights [12]; In our work, we consider the finitely iterated PD (f-IPD), where we cannot calculate the exact value function and have to rely on sample-based approaches such as RL and ES.

Iterated Matching Pennies (IMP) is an iterated matrix game like the IPD. The players choose heads (H) or tails (T) and receive a payoff according to both players' choices. In contrast to the IPD, a general-sum game, IMP is a zero-sum game. In the IMP one player gets +1 for playing the same action as the other player, while the other player is rewarded for playing a *different* action. Thus, the only equilibrium strategy for each one-memory agent is to play a random policy, resulting in an expected joint payoff of (0,0). Only with intra-episode memory can a meta-agent observe a co-player's current policy and thus shape it.

CoinGame is a multi-agent gridworld environment that simulates social dilemmas (like the IPD) with high-dimensional states and multi-step actions [26]. Two players, blue and orange, move around a wrap-around grid and pick up blue and orange coloured coins. When a player picks up a coin of any colour, this player receives a reward of +1. When a player picks up a coin of the co-player's colour, the co-player receives a reward of −2. Whenever a coin gets picked up, a new coin of the same colour appears in a random location on the grid at the next time step. If both agents reach a coin simultaneously, then both agents pick up that coin (the coin is duplicated). When both players pick up coins without regard to colour, the expected reward is 0. In contrast to matrix games, the CoinGame requires learning from high-dimensional states with multi-step actions.

* **in the Matrix** extends matrix games to gridworld environments [41], where * is any normal-form game. For visual descriptions, see Figures 13(c,d), 14, and15. Agents collect two types of resources into their inventory: *Cooperate* and *Defect* coins. Once an agent has collected any coin, the agent's colour changes, representing that the agent is "ready" for interaction. Agents can fire an 'interact' beam to an area in front of them. If an agent's interact beam catches a "ready" agent, both receive rewards equivalent to playing a matrix game *, where their inventory represents their policy. For example, when agent 1's inventory is 1 *Cooperate* coin and 3 *Defect* coins, agent 1's probability to cooperate is 25%. For all details see Appendix H.1.

* in the Matrix introduces a series of novel complexities for shaping over the CoinGame and finite matrix games. The environment is substantially more demanding than the previous games—it is partially observable, has complex interactions, and much longer



(a) (b) (c) (d)

Figure 1: Evaluation results over a single trial (with co-player) compromising over 100 seeds for the CoinGame. (a) Reward, (b) SHAPER's frequency of picking up its own colour coin, (c) state visitation, and (d) the number of coins picked up per episode. SHAPER successfully elicits exploitation with a co-player with a high state visitation for DC and strong competency.

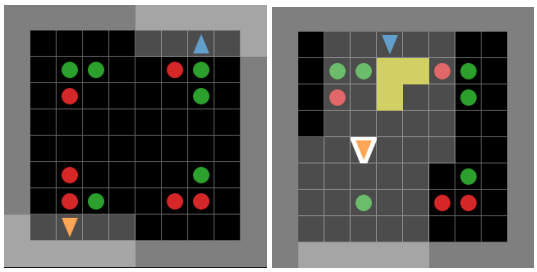


Figure 2: Render of the IPDitM games, a multi-step, gridworld-based general-sum game. Agents with restricted visibility and orientation traverse a grid picking up either *Defect* or *Cooperate* coins. (left) shows an initial state of the game before either agent has a coin. Once agents pick up a coin, their appearance changes, and they can interact. (right) shows the orange agent having collected a coin and the blue agent firing their interact beam.

time horizons. For shaping, partial observability makes temporally-extended actions harder to estimate. Shapers are also incentivised to speed up co-players learning, as the environment only allows interactions after both agents have picked up a coin. We explore two specific implementations of the game: "IPD in the Matrix" (IPDitM) and "IMP in the Matrix" (IMPitM).

For our baseline comparisons, we compare SHAPER against multiple baselines: Naive Learners (NLs), variants of M-FOS, and GS. A NL does not explicitly account for the learning of the co-player across different episodes. In all of our experiments, the co-player is a NL. We train meta-agents until convergence in their respective environments. Then, we evaluate the performance of fixed meta-agents against new co-player (NL) initialisation ϕ_{-i}. Additional implementation details and hyperparameters for each game are provided in Appendix L.[2]

In finite matrix games, our NL is parameterised as a tabular policy trained using PPO. In the gridworld environments, the NL is parameterised by a recurrent neural network and trained using PPO. Furthermore, in gridworlds, we compare to both M-FOS optimised with PPO and by ES. For GS, we only use ES, consistent with the original paper. We compare the performance of SHAPER to two different types of NL pairs: The first type, co-training NL (CT-NL), two NLs are initialised randomly and trained together using independent learning. This shows that avoiding unconditional defection is a challenge in the first place. The second type, pre-trained NL (PT-NL) instead takes an agent from a fully trained CT-NL pair and uses it as a naive shaper baseline, i.e., trains a NL as a best response to the fixed final policy. This ensures that the performance of SHAPER is not simply due to breaking the learning dynamics of the co-player, e.g., because the fully trained NL deprives a randomly initialised agent of all rewards. Specific details are provided in Appendix L. Next, for our **ablations**, we consider three challenges:

Context Challenge: During a trial, after k episodes, the co-player stops updating their parameters. When they stop updating, the shaper's optimal behaviour is to exploit the co-player's fixed policy (effectively stop shaping). We evaluate in the IPD and choose $k = 2$. This challenge tests if shapers: 1) identify the sudden change in a co-player's learning dynamics, and 2) deploy a more suitable exploitative policy. We hypothesise that shapers without context cannot identify the change. We evaluate SHAPER and compare against GS to understand the importance of context for shaping.

History Challenge: We reset the hidden state of SHAPER between episodes, removing its ability to use *context* to shape (SHAPER w/o context). We evaluate in IMP, and agents must infer the co-player's current policy using only history. Finally, we evaluate SHAPER within IMP environment over short and long episode lengths (2 and 100, respectively) to limit the relative strength of *history*.

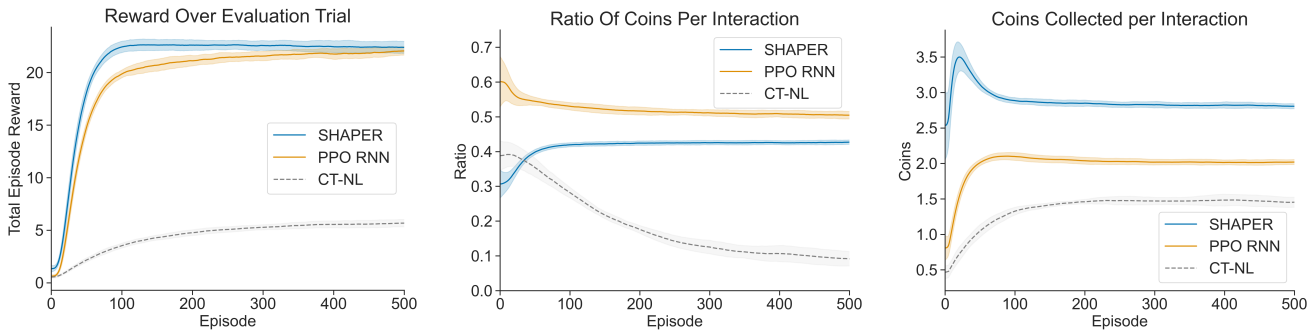
Average Challenge: We also analyse the role of averaging across the batch in matrix games by comparing the performance difference of both MFOS and Shaper with and without averaging.

5 RESULTS

Shaping in Finite Matrix Games: We evaluate SHAPER, M-FOS and GS on finite matrix games, i.e., long-time-horizon variants of the infinite matrix games used in prior work. We recreate previously reported extortion behaviour in a more challenging setting [30].

Insight 1: SHAPER shapes the best in long-horizon iterated matrix games. We inspect the converged reward for each shaping algorithm against a PPO agent in the IPD (see Table 6). Here, SHAPER shapes its co-player more effectively than the baselines, achieving

[2]The codebase is open-source [43].



(a) (b) (c)

Figure 3: Evaluation results over a single trial (with co-player) compromising over 100 seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset. The independent learner is shown to contrast what learning without a meta-agent would look like.

Table 2: Converged reward per step (meta-agent, co-player) for agents against Naive Learners in finite matrix games. SHAPER can shape co-players to exploitative equilibria. We report mean and standard deviation over 20 randomised co-players.

	IPD	IMP
SHAPER	**-0.1 ± 0.02, -2.8 ± 0.05**	**0.9 ± 0.02, -0.9 ± 0.02**
M-FOS	-0.6 ± 0.14, -2.3 ± 0.14	**0.8 ± 0.09, -0.8 ± 0.09**
GS	-1.0 ± 0.03, -1.3 ± 0.10	0.0 ± 0.01, 0.0 ± 0.01
CT-NL	-2.0 ± 0.00, -2.0 ± 0.00	0.0 ± 0.00, 0.0 ± 0.00

an average return of -0.13 per episode. All shaping baselines reach extortion-like policies.

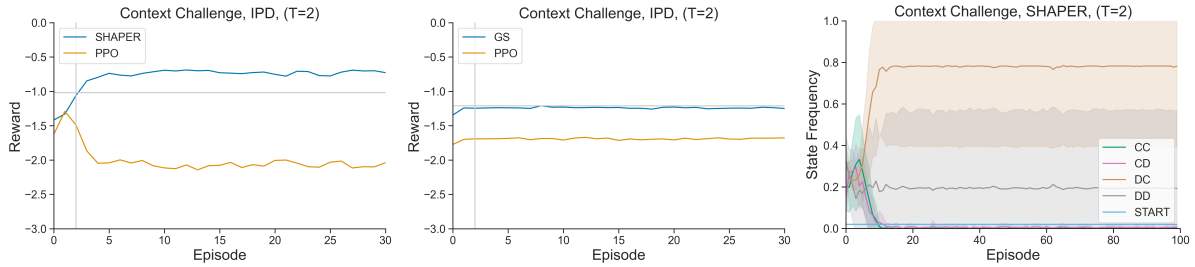
Insight 2: Memory is important for shaping in the IMP. In the IMP, SHAPER exploits its opponent to achieve a score of $(0.9, -0.9)$ (see Table 2). As expected, GS cannot shape the opponent, achieving a score close to the Nash equilibrium, $(0.0, 0.0)$. With only a single-step history, it is impossible to shape the opponent because the opponent can switch to a random strategy between episodes to achieve a score of at least 0. Thus memory is required to find shaping strategies. We find that M-FOS, an agent with memory, shapes too. Next, we present our **CoinGame** results.

Insight 3: Knowing how to navigate the gridworld and pick up coins is already enough to suppress co-player's learning. Towards the end of meta-training, newly initialised co-players have to play against already competent meta-agents who have seen the game many times. We found that in CoinGame, it was sufficient for the meta-agents to pick up all coins before the co-player could reach them to hinder training. Therefore, we suggest checking that the co-player learns against pre-trained Naive Learners. This mitigates behaviours that prohibit the co-players from learning at all. We found that changing from a global to an egocentric observation space in the CoinGame helped co-players learn against pre-trained agents. Examples of sanity tests are found in Appendix E.

Reiterating *Insight 1*, we find meta-agents find extortion-like policies in the CoinGame. To better understand behaviour in CoinGame, we extend the five states from the IPD (S, CC, CD, DC, DD) to include the start states (SS, SC, SD, CS, DS). At the start of an episode, the state is SS until a player picks up a coin. To understand how SHAPER shapes, we inspect the probability of the meta-agent picking up a coin of its own colour at the start, i.e., SC $\rightarrow$ CC. For example, suppose the meta-agent were to cooperate unconditionally in the CoinGame. In that case, it only picks up coins of their own colour no matter the state and would relate to a high probability of cooperating over all states.

Figure 1b demonstrates how SHAPER shapes its co-players effectively already at the start. The difference between cooperating in SC and SS (25%, 15% resp.) highlights how SHAPER uses context to evaluate the exploitability of its co-player. In SS, when both agents have not picked up coins, SHAPER probes for exploitability by not cooperating. In SC, where the co-player has already shown they are cooperative, SHAPER also cooperates. Moreover, Figure 1c shows that CS is visited more often than DS in early episodes (18%, 15% resp.), indicating that SHAPER is shaping the co-player to form a preference for picking up their own colour. This preference is then exploited by SHAPER as indicated by the increasing visitation of DC. The meta-agent's probability of cooperating in DC converges to 25%, i.e., occasionally rewarding the co-player, as never cooperating would probably make the co-player learn pure defection.

Insight 4: CoinGame is not suitable as a multi-step action environment. We found GS produces comparable results to SHAPER. At first, this is surprising since GS is a feedforward network and does not have access to the history (or, at most, one step). Therefore it should not be able to retaliate against a defecting agent since it has no memory of their past actions. However, a close investigation of the problem setting shows that due to particular environment dynamics, the *current state* is often indicative of *past actions*. For example, seeing two agents and a coin on the same square is a strong signal that one of the agents defected since this situation only could have arisen when either all objects spawn on the same square (occurs with a probability of 0.12% and only at the beginning of an episode) or when both agents went for the same coin and the coin respawned on top of them (see Figure 13b). This illustrates that CoinGame allows for simple shaping strategies that



(a) (b) (c)

Figure 4: Hardstop Challenge: Average reward per timestep over an evaluation trial for SHAPER (a) and GS (b) against a Naive Learner in the IPD. Here GS fails to generalise to a co-player that stops learning after an unknown number of timesteps (unseen during training). (c) State Visitation through the evaluation shows SHAPER responds to co-players frozen policy by moving into either DD (the best response to a defective agent) or DC (the best response to a fully cooperative agent).

Table 3: Ablations highlighting the importance of context and history for Shaping. We report converged reward per step (meta-agent, co-player) for agents against Naive Learners.

Context Challenge: IPD	
SHAPER	-0.8, -2.0
SHAPER w/o Context	-1.25, -1.75
History Challenge: IMP (Length=2)	
SHAPER	0.5, -0.5
SHAPER w/o History	0.0, 0.0
History Challenge: IMP (Length=100)	
SHAPER	0.5, -0.5
SHAPER w/o History	0.5, -0.5

do not require *context* or *history*, limiting its utility as a benchmark to measure temporally-extended actions.

We continue with our results for the * **in the Matrix** environments. Motivated by *Insight 3*, we show that co-players learn against pre-trained agents by the number of coins collected in Table 7.

Insight 5: SHAPER outperforms other shaping methods in the IPDitM by a considerable margin SHAPER outperforms other shaping methods in the IPDitM by a considerable margin (see Table 1), e.g., Shaper gets $\sim$ 22.44 points against NL, where M-FOS gets $\sim$ 15.49. Furthermore, SHAPER finds a *collectively better equilibrium for both players* over any other shaping method, e.g., in comparison with M-FOS, Shaper achieves ($\sim$ 22.44, $\sim$ 21.49) and M-FOS gets ($\sim$ 15.49, $\sim$ 23.88).

Insight 6: Shaping in *IPDitM* leads to collectively and individually better outcomes. Table 1 (second column) shows that shaping (SHAPER, M-FOS, and GS) leads to collectively and individually better outcomes in IPDitM compared to PT-NL or CT-NL.

Insight 7: SHAPER shapes by picking up almost all coins at the beginning of a trial. The meta-agent picks up almost all coins in the grid in the first 20 episodes ($\approx$ 3.5, see Figure 3c), especially *Defect* coins. This leaves only *Cooperate* coins for co-players. Interacting with a more cooperative ratio, the co-player receives some reward, reinforcing the co-player to play a cooperative ratio in the future. Figure 3b shows the meta-agent and co-player converge to collecting a large ratio of *Cooperate* coins ($\approx (0.4, 0.6)$), in contrast to independent learners (≈ 0.1) (grey dashed line). Interestingly, a (meta-agent, co-player) pair collects more coins ($\approx (3.0, 2.0)$) than a pair of independent agents ($\approx (1.5, 1.5)$) - this is because the independent learners maximise their return under mutual defection only by increasing interactions within an episode.

In the IMPitM, GS does not learn to shape, as expected from *Insight 2*, whereas M-FOS and SHAPER does. SHAPER and M-FOS achieve similar performances. (see Table 1).

Insight 8: SHAPER empirically tends to find better shaping policies than M-FOS in IPDitM. SHAPER outperforms M-FOS in Table 1, providing evidence that SHAPER scales to more complex policies. SHAPER demonstrates shaping, as indicated by the final rewards, which are significantly higher for both agents than M-FOS IPDitM. We postulate that as M-FOS architecture is as expressive as SHAPER, its complexities and biases hinder ES' ability to find optimal solutions (for training training curves, see Appendix H.2).

In Table 9, we show that SHAPER finds policies leading to improved global welfare in cross-play with M-FOS and GS. In cross-play, the shaping algorithms are trained against Naive Learners and evaluated against each other. This experiment motivates that SHAPER's inductive biases leads to finding more robust policies even when evaluated out of distribution. Note that Shaper vs. Shaper achieves similar scores as M-FOS vs M-FOS. However, Shaper achieves better scores against M-FOS (7.32 ± 0.34, 5.08 ± 0.36) and GS (28.61 ± 1.82, 20.23 ± 1.27). Also, note how GS achieves its highest payoff when playing against Shaper.

In our **ablations**, we find that context is beneficial for shaping in the IMP. In the "Context Challenge", SHAPER (-0.8) outperforms SHAPER w/o Context (-1.25) (see Table 8). For shaping to occur in this challenge, we expect methods to change their strategy at $e = 2$ episodes. SHAPER demonstrates dynamic shaping by switching, yet SHAPER w/o Context's policy does not adapt and does not exploit the stop (see Fig. 4). This result provides evidence that context is needed to shape.

In the "History Challenge", when playing the IMP with a small number of inner-episodes ($e = 2$), we expect meta-agents without context to be unable to identify co-players' current learning

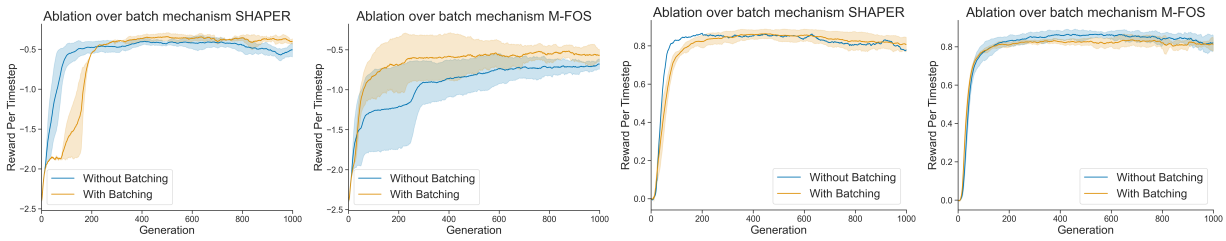


Figure 5: Reward per timestep throughout training for The "Average" challenge. Results are presented over matrix games for 5 seeds. In a) and b) we evaluate OS methods on IPD and in c) and d) we evaluate on IMP. We note that batching only helps M-FOS in the IPD. This indicates batching is only useful in sufficiently diverse environments, relative to the OS method.

Table 4: Episodic reward for in a single evaluation trial against different OS shaping methods in IPDitM. Neither agent takes gradient updates, but those with memory SHAPER and M-FOS use memory to change their policy during the trial. Results are reported for the row player in each match. We report mean and std over 5 seeds.

	SHAPER	GS	M-FOS
SHAPER	16.48 ± 0.88	28.61 ± 1.82	7.32 ± 0.34
GS	20.23 ± 1.27	0 ± 0	1.91 ± 0.27
M-FOS	5.08 ± 0.36	1.35 ± 0.28	16.25 ± 0.95

and thus cannot shape. We find that SHAPER shapes agents, whilst SHAPER w/o Context does not shape agents as indicated by better rewards, 0.5 vs 0.0 (Fig. 12). Interestingly, we also found that with a longer inner-episode length ($E = 100$), SHAPER w/o Context uses *history* to shape its co-player (Fig.12c). This shows that history can encode co-players' learning dynamics in some environments.

In the "Average Challenge", we find that averaging across the batch only helps M-FOS in the IPD, as it improves convergence speed. In all other scenarios, averaging across the batch did not significantly improve performance (see Figure 5b). Shaping agents must approximate, via observations, a co-players update rule. If this update is batched (such as with stochastic gradient descent), the batching mechanism should in theory provide a better estimate. If the batching mechanism is not required, this suggests experience in the update is not diverse. Comparing games, the diversity of co-player behaviours within the IMP is much less than IPD. Within the IPD, SHAPER sees no improvement with the batch mechanism compared to M-FOS (see Figure 5a - 5b). Here we postulate that given M-FOS has a limited context (1-step), batching provides M-FOS with greater context such that it can infer co-player learner. Shaper does not require averaging as it captures more context via its hidden than M-FOS does. This suggests that moving forward, OS methods should consider Context, History and Batching, as mechanisms for observing the experience / learning of co-players.

6 RELATED WORK

Opponent Shaping methods explicitly account for their opponent's learning. Just like SHAPER, these approaches recognise that the actions of any one agent influence their co-players policy and seek to use this mechanism to their advantage [12, 14, 20, 27, 44, 47]. However, in contrast to SHAPER, these approaches require privileged information to shape their opponents. These models are also myopic since anticipating many steps is intractable due to the difficulty of estimating higher-order gradients. Balaguer et al. [4] and Lu et al. [30] solve the issues above by framing opponent shaping as a meta reinforcement learning problem, which allows them to account for long-term shaping, where there is no need for higher-order gradients.

Algorithms for Social Dilemmas often achieve desirable outcomes in high-dimensional social dilemmas yet assume access to hand-crafted notions of adherence [46], social influence [3, 19], gifting [32] or social conventions [22]. While these approaches can achieve desirable outcomes, they change the agent's objectives and alter the dynamics of the underlying game.

Multi-Agent Meta-Learning methods have also shown success in general-sum games with other learners [1, 21, 45]. Similar to SHAPER, they take inspiration from meta-RL - their approach is to learn the optimal initial parameterisation for the meta-agent akin to Model-Agnostic Meta Learning [11]. In contrast, SHAPER uses an approach similar to RL^2 [9], which trains an RNN-based agent to implement efficient learning for its next task. Finally, SHAPER is optimised using ES, which empirically performs better with long-time horizons than policy-gradient methods [29–31].

7 CONCLUSION

When agents interact, the actions of each agent influence the rewards and observations of others and, through their learning, ultimately affect their behaviour. Leveraging this connection is called opponent shaping, and has received considerable attention recently.

This paper introduces SHAPER, a shaping method suitable for high-dimensional games. We are the first to scale shaping successfully to long-time horizon general-sum games with temporally-extended actions, and we provide extensive performance analysis in these settings. We formalise the concept of history and context for shaping and analyse their respective roles empirically. Next, we formalise the previously implicit concept of averaging across the batch and show that it's helpful for previous methods to learn. Future work might investigate scenarios where averaging across a batch is also necessary for SHAPER. Finally, we identify a fundamental problem in the widely-used CoinGame.

ACKNOWLEDGMENTS

We'd like to thank members of the UCL DARK lab, members of the Oxford FLAIR, members of Deepmind Gamma, Mika Semyalven and Joel Liebo for fruitful discussions and comments on an earlier draft of this paper. AK was supported by the EPSRC Grant EP/S021566/1, UCL International Scholar Award for Doctoral Training Centres and Cooperative AI Foundation Grant (1201294). This work was supported by an Oracle for Research Cloud Grant (19158657).

8 ETHICS STATEMENT*

Shaping can be used for good and bad. Empirically, shaping has lead learning agents to find more prosocial solutions in mixed-incentive settings. However, one can imagine scenarios where shaping is used with a negative impact on society. Assuming that learning agents will be deployed in the real world, e.g., online learning self-driving cars, it is important we understand how such agents interact. Early opponent shaping research has already shown that two naive agents mutually defect in the iterated prisoner's dilemma and that opponent shaping leads to the more prosocial tit-for-tat strategy. It is important that we develop these methods further, investigate if they keep leading to more prosocial outcomes even in more difficult environments, and if not, what improvements can we make such that they do. In our work, we show that in grid-worlds with temporally-extended actions and long-time horizons, opponent shaping tends to find more prosocial solutions than Naive Learners. Investigating shaping is important to prevent misuse of the paradigm. We are at the beginning of fundamental research in shaping and better understanding the necessary components to achieve shaping will help us to better control shaping agents. Opponent Shaping is still in an early phase of development and practical implications are limited so immediate negative societal influence is unlikely.

REFERENCES

[1] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. 2018. Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments. In *International Conference on Learning Representations*.

[2] Robert Axelrod and William D Hamilton. 1981. The evolution of cooperation. *science* 211, 4489 (1981), 1390–1396.

[3] Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob, Mojtaba Komeili, Karthik Konath, Minae Kwon, Adam Lerer, Mike Lewis, Alexander H. Miller, Sasha Mitts, Adithya Renduchintala, Stephen Roller, Dirk Rowe, Weiyan Shi, Joe Spisak, Alexander Wei, David Wu, Hugh Zhang, and Markus Zijlstra. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science* 378, 6624 (2022), 1067–1074. https://doi.org/10.1126/science.ade9097

[4] Jan Balaguer, Raphael Koster, Christopher Summerfield, and Andrea Tacchetti. 2022. The Good Shepherd: An Oracle Agent for Mechanism Design. arXiv preprint arXiv:2202.10135.

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

[6] Noam Brown and Tuomas Sandholm. 2017. Libratus: the superhuman AI for no-limit poker. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.

[7] Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin R McKee, Joel Z Leibo, Kate Larson, and Thore Graepel. 2021. Open Problems in Cooperative AI. In *Cooperative AI workshop*.

[8] Robyn M Dawes. 1980. Social dilemmas. *Annual review of psychology* 31, 1 (1980), 169–193.

[9] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. RL$ˆ2$: Fast Reinforcement Learning via Slow Reinforcement Learning. arXiv preprint arXiv:1611.02779.

[10] Benjamin Ellis, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. 2022. SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2212.07489* (2022).

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. 1126–1135.

[12] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. 2018. Learning with Opponent-Learning Awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 122–130.

[13] Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. 2019. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 1942–1951.

[14] Kitty Fung, Qizhen Zhang, Chris Lu, Timon Willi, and Jakob Nicolaus Foerster. 2023. Analyzing the Sample Complexity of Model-Free Opponent Shaping. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*. https://openreview.net/forum?id=Dm2fbPpU6v

[15] Marc Harper, Vincent Knight, Martin Jones, Georgios Koutsovoulos, Nikoleta E. Glynatsi, and Owen Campbell. 2017. Reinforcement learning produces dominant strategies for the Iterated Prisoner's Dilemma. *PLOS ONE* 12, 12 (2017), e0188046.

[16] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. 2020. *Haiku: Sonnet for JAX.* http://github.com/deepmind/dm-haiku

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[18] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. 2019. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science* 364, 6443 (2019), 859–865. https://doi.org/10.1126/science.aau6249

[19] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. 2019. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*. PMLR, 3040–3049.

[20] Dong-Ki Kim, Miao Liu, Matthew Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan P. How. 2021. A Policy Gradient Algorithm for Learning to Learn in Multiagent Reinforcement Learning. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*. 5541–5550.

[21] Dong Ki Kim, Miao Liu, Matthew D Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan How. 2021. A policy gradient algorithm for learning to learn in multiagent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5541–5550.

[22] Raphael Köster, Kevin R McKee, Richard Everett, Laura Weidinger, William S Isaac, Edward Hughes, Edgar A Duéñez-Guzmán, Thore Graepel, Matthew Botvinick, and Joel Z Leibo. 2020. Model-free conventions in multi-agent reinforcement learning with heterogeneous preferences. *arXiv preprint arXiv:2010.09054* (2020).

[23] Robert Tjarko Lange. 2022. *evosax: JAX-based Evolution Strategies.* http://github.com/RobertTLange/evosax

[24] Robert Tjarko Lange. 2022. *gymnax: A JAX-based Reinforcement Learning Environment Library.* http://github.com/RobertTLange/gymnax

[25] Joel Z. Leibo, Edgar A. Duéñez-Guzmán, Alexander Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. 2021. Scalable Evaluation of Multi-Agent Reinforcement Learning with Melting Pot. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 6187–6199.

[26] Adam Lerer and Alexander Peysakhovich. 2017. Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *CoRR* abs/1707.01068 (2017).

[27] Alistair Letcher, David Balduzzi, Sébastien Racanière, James Martens, Jakob N. Foerster, Karl Tuyls, and Thore Graepel. 2019. Differentiable Game Mechanics. *J. Mach. Learn. Res.* 20 (2019), 84:1–84:40.

[28] Alistair Letcher, Jakob N. Foerster, David Balduzzi, Tim Rocktäschel, and Shimon Whiteson. 2019. Stable Opponent Shaping in Differentiable Games. In *7th International Conference on Learning Representations*.

[29] Chris Lu, Jakub Grudzien Kuba, Alistair Letcher, Luke Metz, Christian Schröder de Witt, and Jakob N. Foerster. 2022. Discovered Policy Optimisation. *CoRR* abs/2210.05639 (2022). https://doi.org/10.48550/arXiv.2210.05639 arXiv:2210.05639

[30] Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. 2022. Model-Free Opponent Shaping. In *International Conference on Machine Learning*. PMLR, 14398–14411.

[31] Chris Lu, Timon Willi, Alistair Letcher, and Jakob Nicolaus Foerster. 2022. Adversarial Cheap Talk. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*.

[32] Andrei Lupu and Doina Precup. 2020. Gifting in Multi-Agent Reinforcement Learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (Auckland, New Zealand) *(AAMAS '20)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 789–797.

[33] William H. Press and Freeman J. Dyson. 2012. Iterated Prisoner's Dilemma contains strategies that dominate any evolutionary opponent. *Proceedings of the National Academy of Sciences* 109, 26 (2012), 10409–10413.

[34] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 4295–4304.

[35] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv preprint arXiv:1703.03864.

[36] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. *CoRR* abs/1902.04043 (2019).

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

[38] L. S. Shapley. 1953. Stochastic Games. *Proceedings of the National Academy of Sciences* 39, 10 (1953), 1095–1100.

[39] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489.

[40] Glenn H Snyder. 1971. " Prisoner's Dilemma" and" Chicken" Models in International Politics. *International Studies Quarterly* 15, 1 (1971), 66–103.

[41] Alexander Vezhnevets, Yuhuai Wu, Maria Eckstein, Rémi Leblond, and Joel Z Leibo. 2020. Options as responses: Grounding behavioural hierarchies in multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR.

[42] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.* 575, 7782 (2019), 350–354.

[43] Timon Willi, Akbir Khan, Newton Kwan, Mikayel Samvelyan, Chris Lu, and Jakob Foerster. 2023. Pax: Multi-Agent Learning in JAX. https://github.com/ucl-dark/pax.

[44] Timon Willi, Alistair Letcher, Johannes Treutlein, and Jakob N. Foerster. 2022. COLA: Consistent Learning with Opponent-Learning Awareness. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*. 23804–23831.

[45] Zhe Wu, Kai Li, Enmin Zhao, Hang Xu, Meng Zhang, Haobo Fu, Bo An, and Junliang Xing. 2021. L2E: Learning to Exploit Your Opponent. arXiv preprint arXiv:2102.09381.

[46] Yuyu Yuan, Ting Guo, Pengqian Zhao, and Hongpu Jiang. 2022. Adherence Improves Cooperation in Sequential Social Dilemmas. *Applied Sciences* 12, 16 (2022), 8004.

[47] Stephen Zhao, Chris Lu, Roger Baker Grosse, and Jakob Nicolaus Foerster. 2022. Proximal Learning With Opponent-Learning Awareness. *arXiv preprint arXiv:2210.10125* (2022).

4.1 Limitations and Future Work

The research presented in this chapter successfully demonstrates the scaling of Opponent Shaping, via the proposed Shaper algorithm, to general-sum environments with high-dimensional states, partial observability, and temporally-extended actions; features absent from the simple matrix games where OS was previously evaluated. By analysing the roles of history and context, and formalising batch averaging, we provide insights into the mechanisms enabling scalable shaping and introduce a simpler, effective algorithm.

Furthermore, this work provides directions for future research. Firstly, while the "in-the-Matrix" grid-world environments are significantly more complex than matrix games, they are still structured, discrete domains. Scaling Shaper or similar OS methods to problems involving continuous state-action spaces, pixel-based observations (e.g., MARL in simulators like CARLA (Dosovitskiy et al., 2017) or Isaac Gym (Makoviychuk et al., 2021)) remains an open challenge. Extending Shaper to more than two agents has been explored in Souly et al. (2023). The effectiveness of the RNN-based architecture for capturing relevant history and context in such domains needs further validation. The computational demands motivate the need for accelerated simulation tools, such as the JaxMARL library presented in Chapter 5.

Secondly, Shaper operates in the meta-learning framework introduced by M-FOS. While it does not require access to co-player parameters or gradients for shaping, we assume we have access to co-players at training time, or at least that a test-time co-player has similar learning dynamics to our training-time co-players. Essentially, LOLA and COLA can shape online but require white-box access to the opponents. Shaper does not require white-box access, but needs to approximate the co-players' learning dynamics beforehand. Future work could investigate the distribution of co-players at training time, allowing the generalisation of shaping strategies to a wide range of co-players and different game dynamics at test time. Integrating the "in-the-Matrix" games into JaxMARL (Rutherford et al., 2023), where they are referred to as STORM (Spatial-Temporal Representations of Matrix Games), computationally allows for training over a wide range of agents and environments.

Thirdly, our evaluations focused on Naive Learners (NLs) as co-players. While isolating the shaping effect is standard practice, future work should investigate Shaper's robustness and effectiveness when interacting with other adaptive agents, including other Shaper agents in self-play, agents using different MARL algorithms, or potentially humans.

Finally, shaping can be explored for Large Language Models (LLMs). For example, linear self-attention has been shown to learn gradient descent in-context (von Oswald et al., 2023). One could potentially construct a linear self-attention layer that learns a LOLA-like update. More empirically, LLMs can be used as policies or meta-policies in general-sum games, especially text-based environments like in debate settings (Khan et al., 2024a). The knowledge encoded in LLMs might enable them to generate extortion-like meta-policies zero-shot for a diverse range of co-players and environments.

5

JaxMARL: Multi-Agent RL Environments and Algorithms in JAX

The development and rigorous assessment of advanced Multi-Agent Reinforcement Learning algorithms, such as the Opponent Shaping techniques explored in Part I (COLA, Shaper), and scalable architectures like Mixture of Experts (MoEs) detailed in Part II, depend on extensive empirical validation across diverse and complex scenarios. However, traditional MARL research pipelines are bottlenecked by the computational demands of simulating multi-agent interactions, often relying on CPU-bound execution, which limits parallelism and makes large-scale, statistically robust experimentation expensive, contributing to an evaluation crisis (Gorsane et al., 2022). While libraries like JAX (Bradbury et al., 2018) have revolutionised single-agent RL by enabling end-to-end hardware acceleration on GPUs (Lu et al., 2022a), these performance benefits have not been extended to the multi-agent domain. Addressing this gap, this chapter, based on the work published in Rutherford et al. (2023), introduces JaxMARL as a crucial tool. It stands as the first open-source library offering efficient, JAX-native implementations of a wide array of standard MARL environments—including MPE (Lowe et al., 2017), Overcooked (Carroll et al., 2019), Hanabi (Bard et al., 2020), the StarCraft variant SMAX, and the STORM suite highly relevant to Chapter 4, alongside popular MARL algorithms. JaxMARL's

core contribution is a high-performance platform reducing the computational cost of MARL experimentation by leveraging JAX's `jit` compilation and `vmap` vectorisation capabilities. It facilitates the evaluation and exploration of computationally intensive methods and architectures central to this thesis.

Among the environments included in JaxMARL, the STORM (Spatial-Temporal Representations of Matrix Games) suite, previously referred to as "in-the-Matrix" games in Chapter 4, is particularly significant for evaluating the scalability of OS techniques. Developed to bridge the gap between the analytically tractable but simplistic matrix games used in foundational OS studies (like those in Chapter 3) and the complexity of realistic MARL benchmarks, STORM introduces spatial and temporal dimensions to classic game-theoretic interactions. Agents in STORM navigate a partially observable grid-world, collecting "Cooperate" or "Defect" resources (represented as coins) which dynamically determine their behavioural strategy (policy) in an underlying matrix game, such as the Iterated Prisoner's Dilemma or Matching Pennies. Interaction is triggered only when two "ready" agents (who have collected at least one coin) successfully target each other with an "interact" beam, at which point payoffs are determined based on their current inventory-defined policies and the rules of the embedded matrix game. STORM was inspired by Leibo et al. (2021).

The design choices within STORM make it an ideal testbed for assessing OS methods like Shaper. Unlike atomic actions in matrix games, cooperation and defection become temporally-extended behaviours requiring navigation and resource management. The partial observability demands memory and strategic exploration. Crucially for shaping, agents are temporarily frozen after an interaction. Their inventories, which represent their current policies, are revealed to each other. This provides a clearer signal than trying to infer policies only from observation trajectories. Furthermore, in contrast to the original MeltingPot environments (Leibo et al., 2021), the random placement of coins adds necessary stochasticity, preventing simple memorisation. These elements collectively create a challenging yet structured environment where the effectiveness of shaping strategies in complex,

partially observable settings with delayed consequences can be evaluated, enabled by the computational efficiency afforded by JaxMARL's implementation.

JaxMARL: Multi-Agent RL Environments and Algorithms in JAX

Alexander Rutherford[1*†] **Benjamin Ellis**[1*†] **Matteo Gallici**[2*†] **Jonathan Cook**[1†]
Andrei Lupu[1†] **Garðar Ingvarsson**[3†] **Timon Willi**[1†] **Ravi Hammond**[1†]
Akbir Khan[3] **Christian Schroeder de Witt**[1] **Alexandra Souly**[3]
Saptarashmi Bandyopadhyay[4] **Mikayel Samvelyan**[3] **Minqi Jiang**[3] **Robert Lange**[5]
Shimon Whiteson[1] **Bruno Lacerda**[1] **Nick Hawes**[1] **Tim Rocktäschel**[3]
Chris Lu[1*†] **Jakob Foerster**[1]
[1]University of Oxford, [2]Universitat Politècnica de Catalunya, [3]University College London,
[4]University of Maryland, [5]Technical University Berlin

Abstract

Benchmarks are crucial in the development of machine learning algorithms, with available environments significantly influencing reinforcement learning (RL) research. Traditionally, RL environments run on the CPU, which limits their scalability with typical academic compute. However, recent advancements in JAX have enabled the wider use of hardware acceleration, enabling massively parallel RL training pipelines and environments. While this has been successfully applied to single-agent RL, it has not yet been widely adopted for multi-agent scenarios. In this paper, we present JaxMARL, the first open-source, Python-based library that combines GPU-enabled efficiency with support for a large number of commonly used MARL environments and popular baseline algorithms. Our experiments show that, in terms of wall clock time, our JAX-based training pipeline is around 14 times faster than existing approaches, and up to 12500x when multiple training runs are vectorized. This enables efficient and thorough evaluations, potentially alleviating the evaluation crisis in the field. We also introduce and benchmark SMAX, a JAX-based approximate reimplementation of the popular StarCraft Multi-Agent Challenge, which removes the need to run the StarCraft II game engine. This not only enables GPU acceleration, but also provides a more flexible MARL environment, unlocking the potential for self-play, meta-learning, and other future applications in MARL. The code is available at https://github.com/flairox/jaxmarl.

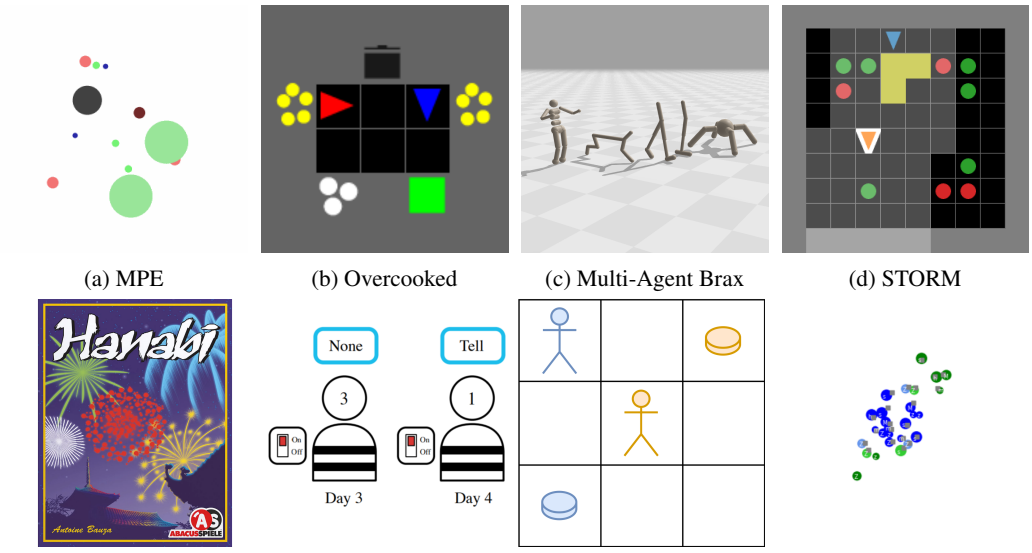
1 Introduction

Benchmarks are crucial for developing new single and multi-agent reinforcement learning (MARL) algorithms. They define problems, enable comparisons, and focus research efforts. For example, the development of MuZero was driven by the challenges presented by Go and Chess [54]. Similarly, decentralised StarCraft Micromanagement tasks [17] led to the creation of algorithms like QMIX [52], a popular MARL technique.

In RL research, the runtime of simulations and algorithms is a critical factor affecting the efficiency, thoroughness, and feasibility of experiments. RL training pipelines often require a large number of environment interactions and long, expensive, experimental runs significantly impede research progress. Hardware acceleration and parallelization is an approach to address this: by running

[*]Equal Contribution
[†]Core Contributor

38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks.



(a) MPE (b) Overcooked (c) Multi-Agent Brax (d) STORM

(e) Hanabi (f) Switch Riddle (g) Coin Game (h) SMAX

Figure 1: JaxMARL environments. We provide JAX-based implementations of a wide range of customizable MARL environments, covering continuous and discrete dynamics, variable number of agents, full and partial observability, and cooperative, competitive and mixed-incentive settings.

environments on hardware accelerators (e.g. GPUs), we can use many more environment instances in parallel than is feasible with a CPU, drastically improving runtime. However, such an approach typically requires significant engineering effort and often relies on non-Python codebases [56], which reduces its accessibility for many Machine Learning researchers where Python is the lingua franca. That said, recent releases, such as the JAX [7] library and PyTorch's `functorch` module [22], have improved accessibility by enabling Python code to be parallelized and just-in-time compiled on hardware accelerators. This laid the foundation for PureJaxRL [38], which leveraged JAX to implement a parallelized approach, demonstrating that running both the environment and the model training on the same GPU yields a 10x speedup over a traditional pipeline with a GPU-trained policy but a CPU-based environment, and 4000x when multiple training runs are vectorized. This speedup enables new research directions [39, 28] and makes large-scale RL research more accessible [46].

We introduce JaxMARL, which brings these benefits to multi-agent learning. To the best of our knowledge, JaxMARL is the first open-source, Python-based library which leverages JAX for GPU acceleration and supports a wide range of popular MARL environments (as shown in Figure 1) as well as algorithms. We show that MARL greatly benefits from this approach as under the traditional approach, of using CPU-based environments, experiments tend to be particularly slow due to the increased computational burden of training multiple agents simultaneously and the higher sample complexity arising from challenges like non-stationarity and decentralized partial observability. Utilizing an end-to-end JAX-based pipeline for MARL significantly accelerates these experiments, opening up new possibilities for research in this field.

Alongside computational issues, MARL research also struggles with thorough evaluation standards [21]. In particular, MARL papers typically only test on a few domains. Of the 75 recent MARL papers analysed by [21], 50% used only one evaluation environment and a further 30% used only two. While SMAC [53] and MPE [36], the two most used environments, have various tasks or maps, the lack of a standard set raises the risk of biased comparisons and incorrect conclusions. This leads to environment overfitting and unclear progress markers. By alleviating computational constraints, JaxMARL allows for rapid evaluations across a broad set of environments and hence is a powerful tool to address MARL's current evaluation crisis.

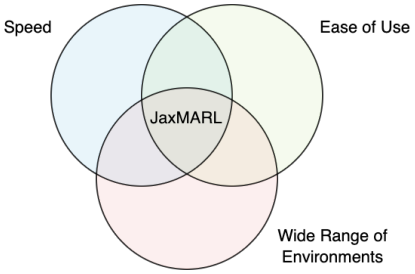


Figure 2: Our philosophy. JaxMARL combines a wide range of environments with ease of use and evaluation speed.

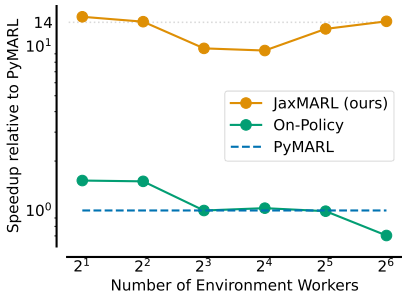


Figure 3: Speed of training an RNN agent using IPPO on a multi-particle environment in JaxMARL compared to two popular MARL libraries, see the Appendix for details.

More specifically, in this paper, our contributions are as follows:

- **JAX Implementations of Popular MARL Environments:** We implement a wide range of popular MARL environments in JAX, enabling fast experimentation across diverse environments. Taking advantage of large scale parallelism, many of our environments run over three orders of magnitude faster on a GPU than their CPU-based counterparts. They are implemented in Python ensuring ease-of-use, following our philosophy set out in Figure 2.
- **New MARL Environment Suites:** We introduce two new MARL environment suites: SMAX and STORM. SMAX is an approximate reimplementation of the popular SMAC(v2) [15] benchmark entirely in JAX, rather than using the StarCraft II game engine like SMAC. It is therefore more customizable and significantly faster: SMAX training is 40,000x faster than the equivalent SMAC implementation on a single NVIDIA 2080 when multiple training runs are vectorized. STORM is a general-sum environment suite inspired by the Melting Pot [34] environment suite that features temporally extended actions within social dilemmas.
- **Implementation of Popular MARL Algorithms in JAX:** We implement many popular MARL algorithms in JAX, such as IPPO, MAPPO and QMIX. As outlined in Figure 3, our training pipeline is up to 14x faster than current popular approaches, and up to 12500x when multiple training runs are vectorized.
- **Comprehensive Benchmarking:** We thoroughly benchmark the speed and correctness of our environments and algorithms, comparing them to existing popular repositories. Generally, our end-to-end JAX implementations run several thousand times faster than their CPU-based counterparts while maintaining equivalent agent performance.
- **Environment Evaluation Recommendations and Best Practice:** Finally, we provide environment evaluation recommendations for different MARL research settings, such as centralized training with decentralized execution and zero-shot coordination. We also provide scripts for large scale evaluation and plotting based on best-practice in the field.

2 Background

Our work brings the benefits of hardware acceleration to multi-agent reinforcement learning.

Hardware Accelerated Environments JAX enables the use of Python code with any hardware accelerator, allowing researchers to write hardware-accelerated code easily. Within the RL community, writing environment code in JAX has gained recent popularity. This brings three chief advantages. Firstly, environments written in JAX can be very easily parallelised by using JAX's `vmap` operation, which vectorises a function across an input dimension. Secondly writing the environment in JAX allows the agent and environment to be co-located on the GPU, which eliminates the time taken to copy between CPU and GPU memory. Finally, the code written can be compiled just-in-time, thereby improving performance. Combined, these factors bring significant increases in training speed,

with PureJaxRL [38] achieving a 4000x speedup with vectorized training over traditional training in single-agent settings.

Multi-Agent Reinforcement Learning Settings Multi-Agent Reinforcement Learning is a subfield of reinforcement learning that focuses on environments where multiple agents interact and learn simultaneously. MARL encompasses various settings that address interactions between multiple agents. The most widely-studied MARL setting is the fully-cooperative one, in which agents work together to achieve a common goal. One key framework is centralized training with decentralized execution (CTDE) [37], which allows agents to share information during the learning phase or use additional environment information. During execution, however, agents act based on their independent and partial observations of the environment. Another is zero-shot coordination, which focuses on training agents to coordinate successfully with unseen partners or in new environments without additional training [30]. Beyond fully-cooperative settings, there are zero-sum and general-sum [34] benchmarks that study competitive and mixed incentive interactions.

3 JaxMARL

We present JaxMARL, a library containing simple and accessible JAX implementations of popular MARL environments and algorithms. JaxMARL enables significant acceleration and parallelisation over existing implementations. To the best of our knowledge, JaxMARL is the first open-source library that provides JAX-based implementations of a wide range of both MARL environments and baselines. JaxMARL's *interface* is inspired by PettingZoo [61] and Gymnax [32], while the *design philosophy* is based on PureJaxRL [38] and CleanRL [26]. We designed it to be a simple and easy-to-use interface for a wide range of MARL problems. A full specification is provided in the Appendix.

3.1 Environments

JaxMARL contains a diverse range of JAX reimplementations of *existing* environments. It also *introduces* SMAX, a novel SMAC-like JAX environment, and STORM, an expansion of matrix games to grid-world scenarios. In this section, we introduce our environments while further details on their implementations can be found in the Appendix.

We measure the speed of our environments in steps per second when using random actions and compare our speed to that of the original environments in Table 3, see the Appendix for details.

3.1.1 New Environments

SMAX The StarCraft Multi-Agent Challenge (SMAC) is a popular benchmark in cooperative multi-agent reinforcement learning (MARL) but has several limitations. SMAC's environment lacks sufficient stochasticity for complex policies [15], and its reliance on the StarCraft II engine makes it slow and memory-intensive [43]. Additionally, StarCraft II's constraints limit scenario variety and do not support competitive self-play without significant engineering. To address these issues, we introduce SMAX, a SMAC-like, hardware-accelerated, customizable environment. SMAX features more lightweight dynamics and a less exploitable AI. SMAX incorporates original SMAC scenarios and scenarios similar to those in SMACv2, but is also far more customizable. We provide more details on SMAX and how it improves on SMAC and SMACv2 in the Appendix.

Spatial-Temporal Representations of Matrix Games (STORM) Inspired by the "in the Matrix" games in Melting Pot 2.0 [1], the STORM [29] environment expands on matrix games by representing them as grid-world scenarios. Agents collect resources which define their strategy during interactions and are rewarded based on a pre-specified payoff matrix. STORM can represent cooperative, competitive or general-sum games, like the prisoner's dilemma [57]. Thus, STORM can be used for studying paradigms such as *opponent shaping*, where agents act with the intent to change other agents' learning dynamics, which has been empirically shown to lead to more prosocial outcomes [18, 65, 41, 29, 69]. Compared to the Coin Game or simple matrix games, the grid-world setting presents a variety of new challenges such as partial observability, multi-step agent interactions, temporally-extended actions, and longer time horizons. Unlike the "in the Matrix" games from Melting Pot, STORM features stochasticity, increasing the difficulty [15].

3.1.2 Existing Environments

We have also provided JAX-based implementations of several existing environments.

Hanabi [3] is a fully-cooperative partially-observable multiplayer card game, where players can observe others' cards but not their own. It is a common benchmark for zero-shot coordination, theory of mind, and ad-hoc teamplay research [23, 24, 9, 39].

Overcooked is commonly used for assessing fully-cooperative and fully-observable Human-AI task performance. Our implementation mimics the original from Overcooked-AI [10]. For a discussion on this environment's limitations see [33].

MABrax is a derivative of Multi-Agent MuJoCo [49], an extension of the MuJoCo Gym environment [62] that is commonly used for benchmarking continuous multi-agent robotic control.

Multi-Agent Particle Environment (MPE) tasks feature a 2D world with simple physics where particle agents can move, communicate, and interact with fixed landmarks [36].

Coin Game is a two-player grid-world environment which emulates social dilemmas such as the iterated prisoner's dilemma [57]. While this is a common benchmark for the general-sum setting, previous work [29] has illustrated issues which STORM corrects.

Switch Riddle [16] is a simple cooperative communication task included as a debugging tool.

3.2 Algorithms

In this section, we present our re-implementation of five well-known MARL baseline algorithms using JAX. All of our training pipelines are fully compatible with JAX's `jit` and `vmap` functions, resulting in significant acceleration of the training processes, as outlined in Figure 3. It also enables parallelisation of training across many seeds and hyperparameters on a single GPU. We follow CleanRL's philosophy of providing clear, single-file implementations [26] and provide a brief overview of the implemented baselines in the Appendix.

PPO We implement both Independent PPO (IPPO) [55, 14] and Multi-Agent PPO (MAPPO) [67], with both implementations based on PureJaxRL [38]. We utilise parameter sharing across homogeneous agents and provide both feed-forward and RNN policies.

Q-learning Our Q-Learning baselines, including Independent Q-Learning (IQL) [60], Value Decomposition Networks (VDN) [59], and QMIX [51], have been implemented in accordance with the PyMARL codebase [51] to ensure consistency with published results and enable direct comparisons with PyTorch.

4 Evaluation Recommendations

Previous work [21] has found significant differences in the evaluation protocols between MARL research works. We identify four main research areas that would benefit from our library: cooperative centralised training with decentralised execution (CTDE) [16], zero-shot coordination [23], general-sum games, and cooperative continuous action methods.

To aid comparisons between methods, we recommend standard *minimal* sets of evaluation environments for each of these settings in Table 1. It's important to note that these are *minimal* and we encourage as broad an evaluation as possible. For example, in the zero-shot coordination setting, all methods should be able to evaluate on Hanabi and Overcooked. However, it may also be possible to evaluate such methods on the SMACv2 settings of SMAX. Similarly, SMAX could be used to evaluate two-player zero-sum methods by training in self-play. For some settings, such as continuous action environments and general-sum games, there is only one difficult environment. We encourage further development of JAX-based environments in these settings to improve the quality of evaluation.

To compute aggregate performance statistics, we follow the recommendations of [2] and evaluate the inter-quartile mean across the different classes of environments. To do this, we recommend normalising performance of the algorithms on the relevant classes of environment (for example via looking at the maximum and minimum performance across algorithms as discussed in [21]), and

Table 1: Recommended minimal environment evaluation sets for different research settings

Setting	Recommended Environments
CTDE	SMAX (all scenarios), Hanabi (2-5 players), Overcooked
Zero-shot Coordination	Hanabi (2 players), Overcooked (5 basic scenarios)
General-Sum	STORM (iterated prisoner's dilemma), STORM (matching pennies)
Cooperative Continuous Actions	MABrax

computing a mean *per seed*. Then compute the inter-quartile mean across aggregated statistics in each environment. We provide code for performing this calculation. This allows environment classes to be compared fairly, without over-weighting those with more individual scenarios.

5 Results

To demonstrate the utility of our library, we evaluate PPO against Q-Learning algorithms on a range of cooperative environments. We discover that not only does it have improved performance, but also that it is more practical to use for end-to-end GPU training.

We then evaluate the speed of our library. We compare algorithm and environment runtimes with similar CPU-based environments. We find that when training PPO, JaxMARL is 31x quicker on SMAX when compared to training in SMAC and 14x quicker on MPE for a single run, and 12,500x for vectorized training runs. Finally, we verify the correctness of our implementations by performing thorough comparisons with prior work.

5.1 Multi-Environment Comparison

We provide a preliminary comparison of our PPO and Q-Learning baselines in Figure 4. The IQM and mean were aggregated across 9 SMAX tasks, excluding the two maps with more than 10 units, all 5 Overcooked maps, and the 2 cooperative scenarios of MPE, running 10 seeds per task. We did not evaluate on Hanabi or all SMAX tasks because of the large memory overhead of storing the replay buffer for the Q-Learning methods on the GPU. We normalize the scores of each run on each task against the highest score obtained by any algorithm in that task, and then average the scores in each environment to avoid bias towards SMAX (which contributes more tasks).[1]

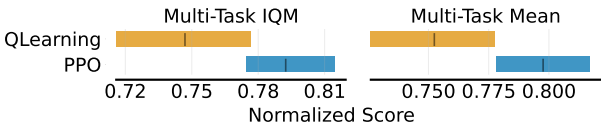


Figure 4: Normalised scores aggregated over SMAX, MPE and Overcooked. PPO shows a clear advantage.

Environment	PPO	Q-Learning
SMAX	10	60
Overcooked	1	10
MPE	1.5	2
Hanabi	450	-

Table 2: Mean training time (in minutes) of a single run. PPO is much faster.

The aggregated IQM and Mean scores in Figure 4 show a clear advantage of PPO baselines over Q-Learning. Furthermore, in Table 2 we find that PPO is 6 times faster in SMAX and 10 times faster in Overcooked. We provide additional analysis of this in the Appendix.

5.2 Speed Benchmarking

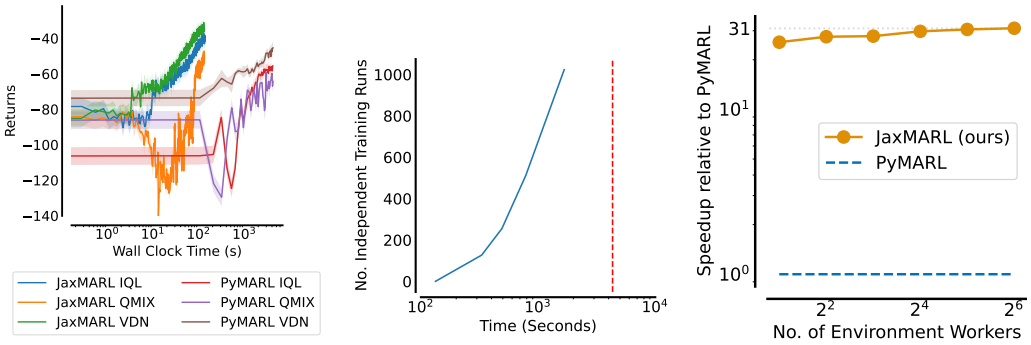
We compare the performance of our environments in steps per second when using random actions to the original environments in Table 3, with details of this test provided in the Appendix.

We next compare the speed of our training pipeline to that of PyMARL. As shown in Figure 5a, a single Q-Learning training run for MPE's simple spread task takes 130 seconds with JaxMARL while PyMARL requires over an hour. Furthermore, using JAX we can parallelise over the entire

[1]For PPO, we used MAPPO as the baseline algorithm, except for Overcooked where we used IPPO. For Q-Learning, we used VDN, except for SMAX where we use QMIX as it performs slightly better.

Table 3: Benchmark results for JAX-based MARL environments (steps-per-second) when taking random actions. All environments are significantly faster than existing CPU implementations.

Environment	Original, 1 Env	Jax, 1 Env	Jax, 100 Envs	Jax, 10k Envs
MPE Simple Spread	8.3×10^4	5.5×10^3	5.2×10^5	4.0×10^7
Switch Riddle	2.7×10^4	6.2×10^3	7.9×10^5	6.7×10^7
Hanabi	2.1×10^3	1.4×10^3	1.1×10^5	5.0×10^6
Overcooked	1.9×10^3	3.6×10^3	3.0×10^5	1.7×10^7
MABrax Ant 4x2	1.8×10^3	2.7×10^2	1.8×10^4	7.6×10^5
Starcraft 2s3z	8.3×10^1	5.4×10^2	4.5×10^4	2.7×10^6
Starcraft 27m vs 30m	2.7×10^1	1.5×10^2	1.1×10^4	1.9×10^5
STORM	–	2.5×10^3	1.8×10^5	1.5×10^7
Coin Game	2.0×10^4	4.7×10^3	4.1×10^5	4.0×10^7



(a) (b) (c)

Figure 5: JaxMARL speed benchmarking results. Figure 5a compares JaxMARL's returns in MPE over wall clock time with PyMARL's when using Q-Learning algorithms. Figure 5b demonstrates JaxMARL algorithms' ability to train many seeds in parallel. The figure compares training time (on the x-axis) for a varying number of training runs (on the y-axis) training using QMIX on MPE. The red dotted represents the time taken to train a single agent with PyMARL. Figure 5c illustrates the speedup of a JaxMARL IPPO training run using SMAX compared to PyMARL using SMAC across a varying number of environment rollout threads.

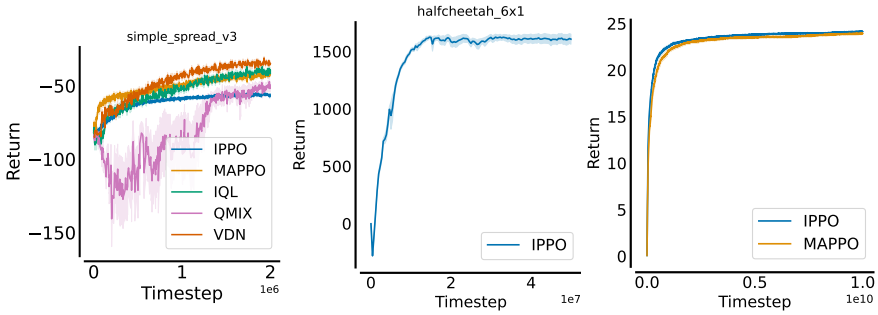
training process within a single hardware accelerator. For QMIX on MPE, this allows us to complete 1024 individual training runs in 198.4 seconds, compared to 1 hour and 10 minutes for a single training run with PyMARL, a speed up of 21,500x per agent. This analysis is repeated for IPPO in the Appendix and we find a speedup of 12,500x. Figure Figure 5c demonstrates the speedup gained from using SMAX with JaxMARL's IPPO implementation compared to training on SMAC with PyMARL. Across a varying number of environment rollout threads, JaxMARL gives a speedup of up to 31x.

5.3 Algorithm and Environment Correctness

In this section, we compare our environment and algorithm implementations with prior work and demonstrate equivalence where applicable.

Overcooked The transition dynamics of our Overcooked implementation match those of the Overcooked-AI implementation. We demonstrate this by training an IPPO policy on our implementation and evaluating the policy on both our implementation and the original at regular intervals. The performance is similar across the implementations. Results can be found in the Appendix.

SMAX SMAX and SMAC are different environments, as they have different opponent policies and dynamics. However, we demonstrate some similarity between them by comparing our IPPO and MAPPO implementations against MAPPO results on SMAC, using the implementation from [58]. We show this figure, along with a more in-depth description of their differences, in the appendix.



(a) MPE Simple Spread (b) MABrax (c) Hanabi

Figure 6: Training Curves for a range of JaxMARL environments. Performance is aggregated across 10 seeds and error bars show standard error.

We additionally present aggregate and detailed performance across SMAX in the Appendix. The PPO-based IPPO and MAPPO perform better than the Q-Learning methods, with the centralised information provided in the state helping MAPPO significantly outperform IPPO.

MPE Our MPE environment corresponds exactly to the PettingZoo implementation. We validate this for each environment using a uniform-random policy on 1000 rollouts, ensuring all observations and rewards are within a tolerance of 1×10^{-4} at each transition. We additionally compare the results of our Q-Learning and PPO implementations with existing libraries, the results of which, along with the performance of IQL on the remaining MPE environments, can be found in the Appendix. We compare Q-Learning and PPO on Simple Spread in Figure 6a.

MABrax As Brax differs subtly from MuJoCo, MABrax does not correspond to MAMuJoCo but the learning dynamics are qualitatively similar. Results therefore are not directly comparable across the two environments. We report mean training return across 10 seeds for IPPO on `halfcheetah_6x1` in Figure 6b. We additionally report the training curves for IPPO on `ant_4x2`, `hopper_3x1`, `walker2d_2x3` and `humanoid_9|8` in the Appendix.

Hanabi Our implementation matches the Hanabi Learning Environment. To verify the environment's accuracy, we obtained 10,000 action trajectories from the original C++ repository using their pretrained models. We confirmed that processing these action trajectories with JaxMARL produces the same states and returns as the C++ repository. In addition, we transferred the models trained with Pytorch/C++ to Jax and verified they obtain similar scores in JaxMARL. Finally, as shown in Figure 6c, our IPPO and MAPPO models attain scores $\mathbf{24.18 \pm 0.04}$ and $\mathbf{23.95 \pm 0.09}$ respectively, which are better than (for IPPO) or similar to (for MAPPO) the scores attained in [67].

6 Related Work

MARL Libraries and Algorithms Several open-source libraries exist for both MARL algorithms and environments. The popular library PyMARL [53] provides PyTorch implementations of QMIX, VDN and IQL and integrates easily with SMAC. E-PyMARL [48] extends this by adding the actor-critic algorithms MADDPG [36], MAA2C [44], IA2C [44], and MAPPO, and supports SMAC, Gym [8], Robot Warehouse [11], Level-Based Foraging [11], and MPE environments. Recently released MARLLib [25] is instead based on the open-source RL library RLLib [35] and combines a wide range of competitive, cooperative and mixed environments with a broad set of baseline algorithms. Meanwhile, MALib [70] focuses on population-based MARL across a wide range of environments. However, none of these frameworks feature hardware-accelerated environments and thus lack the associated performance benefits.

Hardware-Accelerated and JAX-Based RL There has also been a recent proliferation of hardware-accelerated and JAX-based RL environments. Isaac gym [42] provides a GPU-accelerated simulator for a range of robotics platforms and CuLE [12] is a CUDA reimplementation of the Atari Learning

Environment [4]. Both of these environments are GPU-specific and cannot be extended to other hardware accelerators. Madrona [56] is an extensible game-engine written in C++ that allows for GPU acceleration and parallelisation across environments. However, it requires environment code to be written in C++, limiting its accessibility. VMAS [5] provides a vectorized 2D physics engine written in PyTorch and a set of challenging multi-robot scenarios, including those from the MPE environment. For RL environments implemented in JAX, Jumanji [6] features mostly single-agent environments with a strong focus on combinatorial problems. The authors also provide an actor-critic baseline in addition to random actions. PGX [31] includes several board-game environments written in JAX. Gymnax [32] provides JAX implementations of the BSuite [47], classic continuous control, MinAtar [66] and other assorted environments. Gymnax's sister-library, gymnax-baselines, provides PPO and ES baselines. Further extensions to Gymnax [40] also include POPGym environments [45]. Brax [19] reimplements the MuJoCo simulator in JAX and also provides a PPO implementation as a baseline. Jax-LOB [20] implements a vectorized limit order book as an RL environment that runs on the accelerator. Perhaps the most similar to our work is Mava [50], which provides a MAPPO baseline, as well as integration with the Robot Warehouse environment. None of these libraries combine a range of JAX-based MARL environments with both value-based and actor-critic baselines.

7 Conclusion

Hardware acceleration offers important opportunities for MARL research by lowering computational barriers, increasing the speed at which ideas can be iterated, and allowing for more thorough evaluation. We present JaxMARL, an open-source library of popular MARL environments and baseline algorithms implemented in JAX. We combine ease of use with hardware accelerator enabled efficiency to give significant speed-ups compared to traditional CPU-based implementations. Furthermore, by bringing together a wide range of MARL environments under one codebase, we have the potential to help alleviate issues with MARL's evaluation standards. We hope that JaxMARL will help advance MARL by enabling researchers to conduct research with thorough, fast, and effective evaluations.

Limitations and Future Work. While our work provides significant advancements, several limitations remain. First, we observe that the speedups are less pronounced for off-policy, value-based methods. Additionally, there are inherent challenges with end-to-end JAX implementations, such as the difficulty in efficiently handling environments with a variable number of agents or those with massive observation sizes. Furthermore, our MARL environments largely re-implement or draw inspiration from existing environment suites, meaning they do not yet push the boundaries of MARL capabilities. Developing novel MARL environments that push the boundaries of current capabilities could provide new and challenging benchmarks for the community.

8 Acknowledgements

This work received funding from the EPSRC Programme Grant "From Sensing to Collaboration" (EP/V000748/1). MG was partially founded by the FPI-UPC Santander Scholarship FPI-UPC_93. JF is partially funded by the UKI grant EP/Y028481/1 (originally selected for funding by the ERC). JF is also supported by the JPMC Research Award and the Amazon Research Award.

References

[1] John P Agapiou, Alexander Sasha Vezhnevets, Edgar A Duéñez-Guzmán, Jayd Matyas, Yiran Mao, Peter Sunehag, Raphael Köster, Udari Madhushani, Kavya Kopparapu, Ramona Comanescu, et al. Melting pot 2.0. *arXiv preprint arXiv:2211.13746*, 2022.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

[3] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[5] Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. Vmas: A vectorized multi-agent simulator for collective robot learning. *The 16th International Symposium on Distributed Autonomous Robotic Systems*, 2022.

[6] Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphaël Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. In *ICLR*, 2024. URL https://openreview.net/forum?id=C4CxQmp9wc.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[9] Rodrigo Canaan, Xianbo Gao, Julian Togelius, Andy Nealen, and Stefan Menzel. Generating and adapting to diverse ad-hoc partners in hanabi. *IEEE Transactions on Games*, 2022.

[10] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

[11] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[12] Steven Dalton and iuri frosio. Accelerating reinforcement learning through gpu atari emulation. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19773–19782. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/e4d78a6b4d93e1d79241f7b282fa3413-Paper.pdf.

[13] Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

[14] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?, November 2020. URL http://arxiv.org/abs/2011.09533. arXiv:2011.09533 [cs].

[15] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[16] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf.

[17] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 1146–1155. PMLR, 2017.

[18] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, 2018.

[19] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

[20] Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. Jax-lob: A gpu-accelerated limit order book simulator to unlock large scale reinforcement learning for trading. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, ICAIF '23, page 583–591, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702402. doi: 10.1145/3604237.3626880. URL https://doi.org/10.1145/3604237.3626880.

[21] Rihab Gorsane, Omayma Mahjoub, Ruan John de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 35:5510–5521, 2022.

[22] Richard Zou Horace He. functorch: Jax-like composable function transforms for pytorch. https://github.com/pytorch/functorch, 2021.

[23] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[24] Hengyuan Hu, Samuel Sokota, David Wu, Anton Bakhtin, Andrei Lupu, Brandon Cui, and Jakob Foerster. Self-explaining deviations for coordination. *Advances in Neural Information Processing Systems*, 35:38400–38410, 2022.

[25] Siyi Hu, Yifan Zhong, Minquan Gao, Weixun Wang, Hao Dong, Xiaodan Liang, Zhihui Li, Xiaojun Chang, and Yaodong Yang. Marllib: A scalable and efficient multi-agent reinforcement learning library. *Journal of Machine Learning Research*, 2023.

[26] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

[27] Roberto Ierusalimschy. *Programming in lua*. Roberto Ierusalimschy, 2006.

[28] Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms. In *International Conference on Learning Representations*, volume 12, 2024.

[29] Akbir Khan, Timon Willi, Newton Kwan, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Scaling opponent shaping to high dimensional games. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 1001–1010, Richland, SC, 2024. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9798400704864.

[30] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76: 201–264, 2023.

[31] Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 45716–45743, 2023.

[32] Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

[33] Niklas Lauffer, Ameesh Shah, Micah Carroll, Michael D Dennis, and Stuart Russell. Who needs to know? minimal knowledge for optimal coordination. In *International Conference on Machine Learning*, pages 18599–18613. PMLR, 2023.

[34] Joel Z. Leibo, Edgar A. Duéñez-Guzmán, Alexander Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6187–6199. PMLR, 2021.

[35] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

[36] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[37] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[38] Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

[39] Chris Lu, Timon Willi, Alistair Letcher, and Jakob Nicolaus Foerster. Adversarial cheap talk. In *International Conference on Machine Learning*, pages 22917–22941. PMLR, 2023.

[40] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[41] Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. Model-free opponent shaping. In *International Conference on Machine Learning*, pages 14398–14411. PMLR, 2022.

[42] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu based physics simulation for robot learning. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *NeurIPS Datasets and Benchmarks*, 2021. URL http://dblp.uni-trier.de/db/conf/nips/neurips2021db.html#MakoviychukWGLS21.

[43] Adam Michalski, Filippos Christianos, and Stefano V Albrecht. Smaclite: A lightweight environment for multi-agent reinforcement learning. *arXiv preprint arXiv:2305.05566*, 2023.

[44] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[45] Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=chDrutUTs0K.

[46] Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Viacheslav Sinii, Artem Agarkov, and Sergey Kolesnikov. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop, NeurIPS2023*, 2023. URL https://openreview.net/forum?id=xALDC4aHGz.

[47] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

[48] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021. URL http://arxiv.org/abs/2006.07869.

[49] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

[50] Arnu Pretorius, Kale ab Tessera, Andries P. Smit, Kevin Eloff, Claude Formanek, St John Grimbly, Siphelele Danisa, Lawrence Francis, Jonathan Shock, Herman Kamper, Willie Brink, Herman Engelbrecht, Alexandre Laterre, and Karim Beguir. Mava: A research framework for distributed multi-agent reinforcement learning. *arXiv preprint arXiv:2107.01460*, 2021. URL https://arxiv.org/pdf/2107.01460.pdf.

[51] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

[52] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[53] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[54] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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

[56] Brennan Shacklett, Luc Guy Rosenzweig, Zhiqiang Xie, Bidipta Sarkar, Andrew Szot, Erik Wijmans, Vladlen Koltun, Dhruv Batra, and Kayvon Fatahalian. An extensible, data-oriented architecture for high-performance, many-world simulation. *ACM Trans. Graph.*, 42(4), 2023.

[57] Glenn H Snyder. " prisoner's dilemma" and" chicken" models in international politics. *International Studies Quarterly*, 15(1):66–103, 1971.

[58] Mingfei Sun, Sam Devlin, Jacob Beck, Katja Hofmann, and Shimon Whiteson. Trust region bounds for decentralized ppo under non-stationarity. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 5–13, 2023.

[59] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS 2018)*, volume 3, pages 2085–2087, 2018.

[60] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[61] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[62] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[63] Edan Toledo, Laurence Midgley, Donal Byrne, Callum Rhys Tilbury, Matthew Macfarlane, Cyprien Courtot, and Alexandre Laterre. Flashbax: Streamlining experience replay buffers for reinforcement learning with jax, 2023. URL https://github.com/instadeepai/flashbax/.

[64] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[65] Timon Willi, Alistair Letcher, Johannes Treutlein, and Jakob N. Foerster. COLA: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23804–23831, 2022.

[66] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[67] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

[68] Qizhen Zhang, Chris Lu, Animesh Garg, and Jakob Foerster. Centralized model and exploration policy for multi-agent rl. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1500–1508, 2022.

[69] Stephen Zhao, Chris Lu, Roger B Grosse, and Jakob Foerster. Proximal learning with opponent-learning awareness. *Advances in Neural Information Processing Systems*, 35:26324–26336, 2022.

[70] Ming Zhou, Ziyu Wan, Hanjing Wang, Muning Wen, Runzhe Wu, Ying Wen, Yaodong Yang, Yong Yu, Jun Wang, and Weinan Zhang. Malib: A parallel framework for population-based multi-agent reinforcement learning. *Journal of Machine Learning Research*, 24(150):1–12, 2023. URL http://jmlr.org/papers/v24/22-0169.html.

5.1 Limitations and Future Work

While valuable for bridging the gap between simple matrix games and complex MARL tasks, as demonstrated in Chapter 4, the STORM environments included within JaxMARL come with limitations. Currently, STORM environments operate on discrete grids with relatively simple agent capabilities and rely on predefined matrix games (like IPD or IMP) to structure the core interaction payoffs. Compared to many real-world scenarios with continuous state-action spaces or emergent, non-stationary game dynamics, this is still a simplification. Current implementations are also primarily focused on two-player interactions, though an n-player variant has since been added to JaxMARL. Future work on STORM could involve increasing its complexity by incorporating continuous elements and designing variants with more emergent strategic interactions rather than fixed matrix payoffs.

Beyond enhancing the STORM environments, their efficient implementation within JaxMARL unlocks significant avenues for future research previously prevented by computational costs. The speed-ups enable large-scale empirical studies investigating the performance and generalisation of various OS algorithms across a wide distribution of STORM configurations (varying grid sizes, observability, underlying matrix games, and co-player learning dynamics). This platform is ideal for studying the complex co-evolutionary dynamics that arise when multiple sophisticated OS agents interact in self-play and mutually shape each other's learning. Furthermore, STORM enables meta-MARL research, training agents to quickly adapt shaping strategies to new tasks or opponent types. Critically, linking back to the broader themes of this thesis, STORM allows deploying and evaluating agents that combine external opponent shaping capabilities with internal scalable architectures like MoEs, allowing investigation into how internal coordination and specialisation aid performance in these challenging interactive settings.

Part II

Mixture of Experts for Deep Reinforcement Learning

6

Mixtures of Experts Unlock Parameter Scaling for Deep RL

A central ambition in deep RL is to create agents capable of solving increasingly complex tasks, often requiring models with substantial representational capacity (Sutton and Barto, 1998; Mnih et al., 2013b). In supervised learning domains, particularly natural language processing and computer vision (Brown et al., 2020), increasing model size (parameter count) has proven to be a reliable path towards enhanced performance, governed by predictable scaling laws (Kaplan et al., 2020a).

However, scaling deep RL agents has been far less straightforward. As highlighted in Chapter 2, simply increasing the size of standard monolithic neural network architectures in RL frequently yields diminishing returns or even performance degradation (Ostrovski et al., 2021; Kumar et al., 2021; Lyle et al., 2022; Graesser et al., 2022; Nikishin et al., 2022b; Farebrother et al., 2022; Sokar et al., 2023b; Ceron et al., 2023; Schwarzer et al., 2023), contrasting starkly with observations in supervised learning.

This limitation is often attributed to challenges inherent in RL, such as learning instability arising from bootstrapping and non-stationary data distributions, compounded by architectural issues like parameter under-utilisation – evidenced by phenomena like dormant neurons (Sokar et al., 2023b) and loss of network

plasticity (Lyle et al., 2022). Effectively scaling deep RL necessitates architectures that utilise parameters efficiently.

Drawing inspiration from their success in enabling massive scaling in supervised learning (Shazeer et al., 2017a; Fedus et al., 2022; Puigcerver et al., 2023), this chapter *investigates whether Mixture of Expert architectures can overcome these limitations in deep RL.* Conceptually viewing the network as an internal cooperative system, the expert modules can be seen as specialised sub-networks, with the routing mechanism coordinating their activation and combining their outputs. This chapter's central question is: Can this modular, specialised architecture, which facilitates internal coordination, overcome the parameter scaling limitations observed in standard deep RL networks?

Based on the work published in Obando-Ceron et al. (2024), this chapter empirically evaluates the impact of incorporating MoE modules, specifically the widely-used sparse Top1-MoE and the fully-differentiable Soft MoE (Puigcerver et al., 2023) into standard value-based deep RL agents (DQN (Mnih et al., 2013a) and Rainbow (Hessel et al., 2018)). We replace the penultimate dense layer of these agents with MoE modules and systematically vary the number of experts, effectively increasing the parameter count in a structured manner. We evaluate performance on the Arcade Learning Environment (ALE; Mnih et al., 2013a) benchmark. Our methodology includes comparisons against baselines where the corresponding dense layer is widened to match the MoE parameter counts. To gain insight into the underlying mechanisms, we examine metrics related to parameter utilisation, such as dormant neuron ratios.

The presented results provide evidence that MoEs, particularly Soft MoE, unlock effective parameter scaling in deep RL. We show significant performance improvements over baseline architectures that scale with the number of experts, while simply widening dense layers often fails to do so. Our analyses suggest these benefits are linked to MoEs promoting better parameter utilisation and potentially stabilising learning dynamics. This chapter thus establishes MoEs as a promising architectural direction for building more capable and scalable deep RL agents.

Mixtures of Experts Unlock Parameter Scaling for Deep RL

Johan Obando-Ceron* [1 2 3] **Ghada Sokar*** [1] **Timon Willi*** [1 4] **Clare Lyle** [1] **Jesse Farebrother** [1 2 5]
Jakob Foerster [4] **Karolina Dziugaite** [1] **Doina Precup** [1 2 5] **Pablo Samuel Castro** [1 2 3]

Abstract

The recent rapid progress in (self) supervised learning models is in large part predicted by empirical scaling laws: a model's performance scales proportionally to its size. Analogous scaling laws remain elusive for reinforcement learning domains, however, where increasing the parameter count of a model often hurts its final performance. In this paper, we demonstrate that incorporating Mixture-of-Expert (MoE) modules, and in particular Soft MoEs (Puigcerver et al., 2023), into value-based networks results in more parameter-scalable models, evidenced by substantial performance increases across a variety of training regimes and model sizes. This work thus provides strong empirical evidence towards developing scaling laws for reinforcement learning. **We make our code publicly available.**

1. Introduction

Deep Reinforcement Learning (RL) – the combination of reinforcement learning algorithms with deep neural networks – has proven effective at producing agents that perform complex tasks at super-human levels (Mnih et al., 2015; Berner et al., 2019; Vinyals et al., 2019; Fawzi et al., 2022; Bellemare et al., 2020). While deep networks are critical to any successful application of RL in complex environments, their design and learning dynamics in RL remain a mystery. Indeed, recent work highlights some of the surprising phenomena that arise when using deep networks in RL, often going against the behaviours observed in supervised learning settings (Ostrovski et al., 2021; Kumar et al., 2021a; Lyle et al., 2022a; Graesser et al., 2022; Nikishin et al., 2022; Sokar et al., 2023; Ceron et al., 2023).

*Equal contribution [1]Google DeepMind [2]Mila - Québec AI Institute [3]Université de Montréal [4]University of Oxford [5]McGill University. Correspondence to: Johan Obando-Ceron <jobando0730@gmail.com>, Pablo Samuel Castro <psc@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

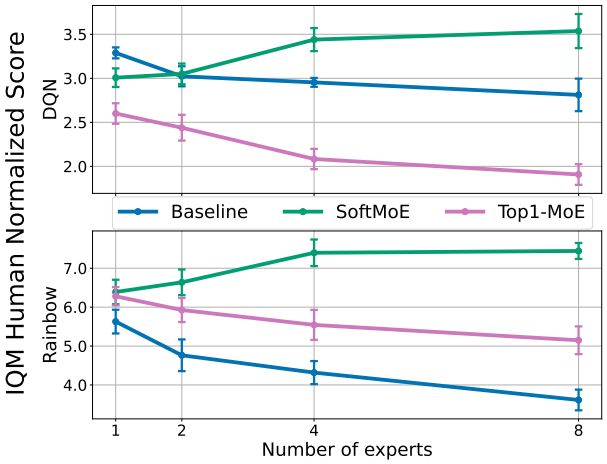


Figure 1. **The use of Mixture of Experts allows the performance of DQN (top) and Rainbow (bottom) to scale with an increased number of parameters**. While Soft MoE helps in both cases and improves with scale, Top1-MoE only helps in Rainbow, and does not improve with scale. The corresponding layer in the baseline is scaled by the number of experts to (approximately) match parameters. IQM scores computed over 200M environment steps over 20 games, with 5 independent runs each, and error bars showing 95% stratified bootstrap confidence intervals. The replay ratio is fixed to the standard 0.25.

The supervised learning community convincingly showed that larger networks result in improved performance, in particular for language models (Kaplan et al., 2020). In contrast, recent work demonstrates that scaling networks in RL is challenging and requires the use of sophisticated techniques to stabilize learning, such as supervised auxiliary losses, distillation, and pre-training (Farebrother et al., 2022; Taiga et al., 2022; Schwarzer et al., 2023). Furthermore, deep RL networks are *under-utilizing* their parameters, which may account for the observed difficulties in obtaining improved performance from scale (Kumar et al., 2021a; Lyle et al., 2022a; Sokar et al., 2023). Parameter count cannot be scaled efficiently if those parameters are not used effectively.

Architectural advances, such as transformers (Vaswani et al., 2017), adapters (Houlsby et al., 2019), and Mixtures of Experts (MoEs; Shazeer et al., 2017), have been central to the

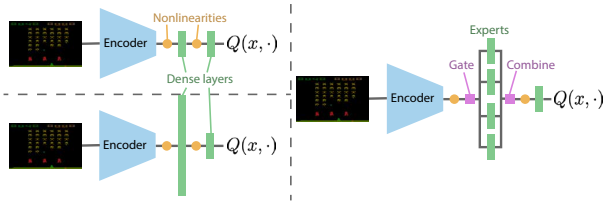


Figure 2. **Incorporating MoE modules into deep RL networks. Top left:** Baseline architecture; **bottom left:** Baseline with penultimate layer scaled up; **right:** Penultimate layer replaced with an MoE module.

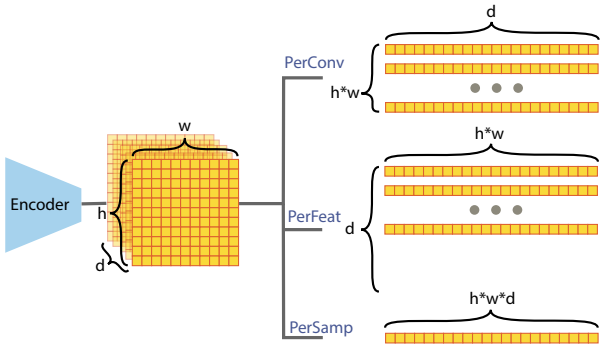


Figure 3. **Tokenization types considered:** PerConv (per convolution), PerFeat (per feature), and PerSamp (per sample).

scaling properties of supervised learning models, especially in natural language and computer vision problem settings. MoEs, in particular, are crucial to scaling networks to billions (and recently trillions) of parameters, because their modularity combines naturally with distributed computation approaches (Fedus et al., 2022). Additionally, MoEs induce *structured sparsity* in a network, and certain types of sparsity have been shown to improve network performance (Evci et al., 2020; Gale et al., 2019).

In this paper, we explore the effect of mixture of experts on the parameter scalability of value-based deep RL networks, i.e., does performance increase as we increase the number of parameters? We demonstrate that incorporating Soft MoEs (Puigcerver et al., 2023) strongly improves the performance of various deep RL agents, and performance improvements scale with the number of experts used. We complement our positive results with a series of analyses that help us understand the underlying causes for the results in Section 4. For example, we investigate different gating mechanisms, motivating the use of Soft MoE, as well as different tokenizations of inputs. Moreover, we analyse different properties of the experts hidden representations, such as dormant neurons (Sokar et al., 2023), which provide empirical evidence as to why Soft MoE improves performance over the baseline.

Finally, we present a series of promising results that pave the way for further research incorporating MoEs in deep RL networks in Section 5. For instance, in Section 5.1 we show preliminary results that Soft MoE outperforms the baseline on a set of Offline RL tasks; in Section 5.2, we evaluate Soft MoE's performance in low-data training regimes; and lastly, we show that exploring different architectural designs is a fruitful direction for future research in Section 5.3.

2. Preliminaries

2.1. Reinforcement Learning

Reinforcement learning methods are typically employed in sequential decision-making problems, with the goal of finding optimal behavior within a given environment (Sutton & Barto, 1998). These environments are typically formalized as Markov Decision Processes (MDPs), defined by the tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{X}$ represents the set of states, $\mathcal{A}$ the set of available actions, $\mathcal{P} : \mathcal{X} \times \mathcal{A} \to \Delta(\mathcal{X})$[1] the transition function, $\mathcal{R} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\gamma \in [0, 1)$ the discount factor.

An agent's behaviour, or policy, is expressed via a mapping $\pi : \mathcal{X} \to \Delta(\mathcal{A})$. Given a policy π, the *value* V^π of a state x is given by the expected sum of discounted rewards when starting from that state and following π from then on: $V^\pi(x) := \mathop{\mathbb{E}}_{\pi,\mathcal{P}} \left[\sum_{t=i}^{\infty} \gamma^t \mathcal{R}(x_t, a_t) \mid x_0 = x \right]$. The state-action function Q^π quantifies the value of first taking action a from state x, and then following π thereafter: $Q^\pi(x, a) := \mathcal{R}(x, a) + \gamma \mathop{\mathbb{E}}_{x' \sim \mathcal{P}(x,a)} V^\pi(x')$. Every MDP admits an optimal policy π^* in the sense that $V^{\pi^*} := V^* \geq V^\pi$ uniformly over $\mathcal{X}$ for all policies π.

When $\mathcal{X}$ is very large (or infinite), function approximators are used to express Q, e.g., DQN by Mnih et al. (2015) uses neural networks with parameters θ, denoted as $Q_\theta \approx Q$. The original architecture used in DQN, hereafter referred to as the CNN architecture, comprises of 3 convolutional layers followed by 2 dense layers, with ReLu nonlinearities (Fukushima, 1969) between each pair of layers. In this work, we mostly use the newer Impala architecture (Espeholt et al., 2018), but will provide a comparison with the original CNN architecture. DQN also used a *replay buffer*: a (finite) memory where an agent stores transitions received while interacting with the environment, and samples mini-batches from to compute gradient updates. The *replay ratio*[2] is defined as the ratio of gradient updates to environment

[1] $\Delta(X)$ denotes a distribution over the set X.

[2] In the hyperparameters established in (Mnih et al., 2015), the policy is updated every 4 environment steps collected, resulting in a replay ratio of 0.25.

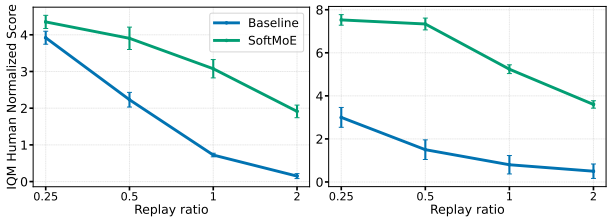


Figure 4. **Soft MoE yields performance gains even at high replay ratio values.** DQN (left) and Rainbow (right) with 8 experts. See Section 4 for training details.

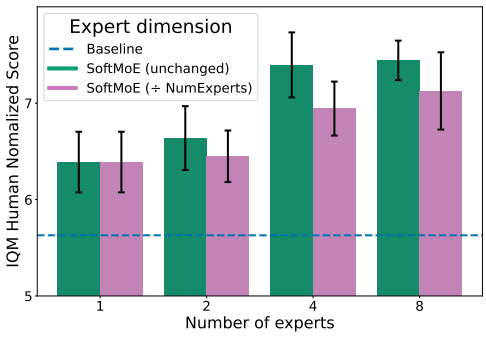


Figure 5. **Scaling down the dimensionality of Soft MoE experts has no significant impact on performance in Rainbow**. See Section 4 for training details.

interactions, and plays a role in our analyses below.

Rainbow (Hessel et al., 2018) extended the original DQN algorithm with multiple algorithmic components to improve learning stability, sample efficiency, and overall performance. Rainbow was shown to significantly outperform DQN and is an important baseline in deep RL research.

2.2. Mixtures of Experts

Mixtures of experts (MoEs) have become central to the architectures of most modern Large Language Models (LLMs). They consist of a set of n "expert" sub-networks activated by a gating network (typically learned and referred to as the *router*), which routes each incoming token to k experts (Shazeer et al., 2017). In most cases, k is smaller than the total number of experts (k=1 in our work), thereby inducing sparser activations (Fedus et al., 2022). These sparse activations enable faster inference and distributed computation, which has been the main appeal for LLM training. MoE modules typically replace the dense feed forward blocks in transformers (Vaswani et al., 2017). Their strong empirical results have rendered MoEs a very active area of research in the past few years (Shazeer et al., 2017; Lewis et al., 2021; Fedus et al., 2022; Zhou et al., 2022; Puigcerver et al., 2023; Lepikhin et al., 2020; Zoph et al., 2022; Gale et al., 2023).

Such hard assignments of tokens to experts introduce a number of challenges such as training instabilities, dropping of tokens, and difficulties in scaling the number of experts (Fedus et al., 2022; Puigcerver et al., 2023). To address some of these challenges, Puigcerver et al. (2023) introduced Soft MoE, which is a fully differentiable soft assignment of tokens-to-experts, replacing router-based hard token assignments.

Soft assignment is achieved by computing (learned) mixes of per-token weightings for each expert, and averaging their outputs. Following the notation of Puigcerver et al. (2023), let us define the input tokens as $\mathbf{X} \in \mathbb{R}^{m \times d}$, where m is the number of d-dimensional tokens. A Soft MoE layer applies a set of n experts on individual tokens, $\left\{f_i : \mathbb{R}^d \to \mathbb{R}^d\right\}_{1:n}$. Each expert has p input- and output-slots, represented respectively by a d-dimensional vector of parameters. We denote these parameters by $\mathbf{\Phi} \in \mathbb{R}^{d \times (n \cdot p)}$.

The input-slots $\tilde{\mathbf{X}} \in \mathbb{R}^{(n \cdot p) \times d}$ correspond to a weighted average of all tokens: $\tilde{\mathbf{X}} = \mathbf{D}^\top \mathbf{X}$, where

$$\mathbf{D}_{ij} = \frac{\exp\left((\mathbf{X}\mathbf{\Phi})_{ij}\right)}{\sum_{i'=1}^{m} \exp\left((\mathbf{X}\mathbf{\Phi})_{i'j}\right)}.$$

D is typically referred to as the *dispatch weights*. We then denote the expert outputs as $\tilde{\mathbf{Y}}_i = f_{\lfloor i/p \rfloor}\left(\tilde{\mathbf{X}}_i\right)$. The output of the Soft MoE layer $\mathbf{Y}$ is the combination of $\tilde{\mathbf{Y}}$ with the *combine weights* $\mathbf{C}$ according to $\mathbf{Y} = \mathbf{C}\tilde{\mathbf{Y}}$, where

$$\mathbf{C}_{ij} = \frac{\exp\left((\mathbf{X}\mathbf{\Phi})_{ij}\right)}{\sum_{j'=1}^{n \cdot p} \exp\left((\mathbf{X}\mathbf{\Phi})_{ij'}\right)}.$$

Note how $\mathbf{D}$ and $\mathbf{C}$ are learned only through $\mathbf{\Phi}$, which we will use in our analysis. The results of Puigcerver et al. (2023) suggest that Soft MoE achieves a better trade-off between accuracy and computational cost compared to other MoE methods.

3. Mixture of Experts for Deep RL

Below we discuss some important design choices in our incorporation of MoE modules into DQN-based architectures.

Where to place the MoEs? Following the predominant usage of MoEs to replace dense feed-forward layers (Shazeer et al., 2017; Fedus et al., 2022; Gale et al., 2023), we replace the penultimate layer in our networks with an MoE module, where each expert has the same dimensionality as the original dense layer. Thus, we are effectively widening the penultimate layer's dimensionality by a factor equal to the number of experts. Figure 2 illustrates our deep RL MoE architecture. We discuss some alternatives in Section 5.3.

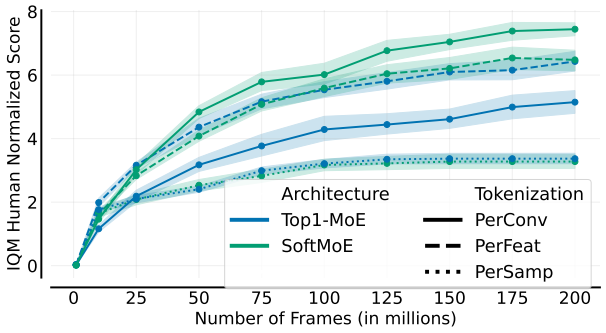


Figure 6. **Comparison of different tokenization methods on Rainbow with 8 experts.** Soft MoE with PerConv achieves the best results. Top1-MoE works best with PerFeat. PerSamp is worst over both architectures. See Section 4 for training details.

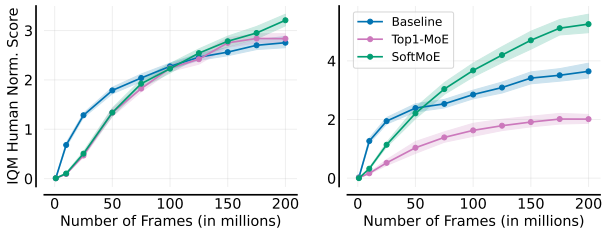


Figure 7. **In addition to the Impala encoder, MoEs also show performance gains for the standard CNN encoder on DQN (left) and Rainbow (right)**, with 8 experts. See Section 4 for training details.

What is a token? MoEs sparsely route inputs to a set of experts (Shazeer et al., 2017) and are mostly used in the context of transformer architectures where the inputs are *tokens* (Fedus et al., 2022). For the vast majority of supervised learning tasks where MoEs are used there is a well-defined notion of a token; except for a few works using Transformers (Chen et al., 2021), this is not the case for deep RL networks. Denoting by $C^{(h,w,d)} \in \mathbb{R}^3$ the output of the convolutional encoders, we define *tokens* as d-dimensional slices of this output; thus, we split C into $h \times w$ tokens of dimensionality d (PerConv in Figure 3). This approach to tokenization is taken from what is often used in vision tasks (see the Hybrid architecture of Dosovitskiy et al. (2021)). We did explore other tokenization approaches (discussed in Section 4.2), but found this one to be the best performing and most intuitive. Finally, a trainable linear projection is applied after each expert to maintain the token size d at the output.

What flavour of MoE to use? We explore the top-k gating architecture of Shazeer et al. (2017) following the simplified $k = 1$ strategy of Fedus et al. (2022), as well as the Soft MoE variant proposed by Puigcerver et al. (2023); for the rest of the paper we refer to the former as Top1-MoE and the latter as Soft MoE. We focus on these two, as Top1-MoE is the predominantly used approach, while Soft MoE is simpler and shows evidence of improving performance. Since Top1-MoEs activate one expert per token while Soft MoEs can activate multiple experts per token, Soft MoEs are arguably more directly comparable in terms of the number of parameters when widening the dense layers of the baseline.

4. Empirical evaluation

We focus our investigation on DQN (Mnih et al., 2015) and Rainbow (Hessel et al., 2018), two value-based agents that have formed the basis of a large swath of modern deep RL research. Recent work has demonstrated that using the ResNet architecture (Espeholt et al., 2018) instead of the original CNN architecture (Mnih et al., 2015) yields strong empirical improvements (Graesser et al., 2022; Schwarzer et al., 2023), so we conduct most of our experiments with this architecture. As in the original papers, we evaluate on 20 games from the Arcade Learning Environment (ALE), a collection of a diverse and challenging pixel-based environments (Bellemare et al., 2013b).

Our implementation, included with this submission, is built on the Dopamine library (Castro et al., 2018)[3], which adds stochasticity (via sticky actions) to the ALE (Machado et al., 2018). We use the recommendations from Agarwal et al. (2021) for statistically-robust performance evaluations, in particular focusing on interquartile mean (IQM). Every experiment was run for 200M environment steps, unless reported otherwise, with 5 independent seeds, and we report 95% stratified bootstrap confidence intervals. All experiments were run on NVIDIA Tesla P100 GPUs, and each took on average 4 days to complete.

4.1. Soft MoE Helps Parameter Scalability

To investigate the efficacy of Top1-MoE and Soft MoE on DQN and Rainbow, we replace the penultimate layer with the respective MoE module (see Figure 2) and vary the number of experts. Given that each expert is a copy of the original penultimate layer, we are effectively increasing the number of parameters of this layer by a factor equal to the number of experts. To compare more directly in terms of number of parameters, we evaluate simply widening the penultimate layer of the base architectures by a factor equal to the number of experts.

As Figure 1 demonstrates, Soft MoE provides clear performance gains, and these gains increase with the number of experts; for instance in Rainbow, increasing the number of experts from 1 to 8 results in a 20% performance improve-

[3]Dopamine code available at https://github.com/google/dopamine.

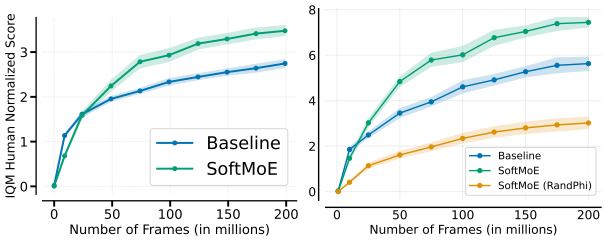


Figure 8. **(Left) Evaluating the performance on 60 Atari 2600 games; (Right) Evaluating the performance of Soft MoE with a random Φ matrix**. (Left) Even over 60 games, Soft MoE performs better than the baseline. (Right) Learning Φ is beneficial over a random phi, indicating that Soft MoE's perfomance gains are not only due to the distribution of tokens to experts. Both plots run with Rainbow using the Impala architecture and 8 experts. See Section 4 for training details.

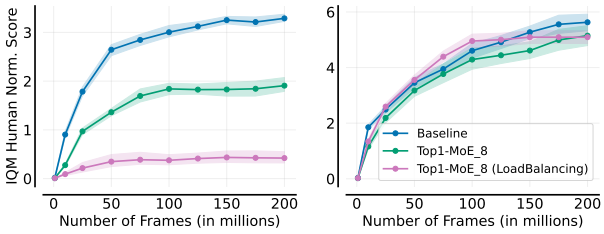


Figure 9. **Load-balancing losses from Ruiz et al. (2021) are unable to improve the performance of Top1-MoE** with 8 experts on neither DQN (left) nor Rainbow (right). See Section 4 for training details.

ment. In contrast, the performance of the base architectures *declines* as we widen its layer; for instance in Rainbow, as we increase the layer multiplier from 1 to 8 there is a performance decrease of around 40%. This is a finding consistent with prior work demonstrating the difficulty in scaling up deep RL networks (Farebrother et al., 2022; Taiga et al., 2022; Schwarzer et al., 2023). Top1-MoE seems to provide gains when incorporated into Rainbow, but fails to exhibit the parameter-scalability we observe with Soft MoE.

It is known that deep RL agents are unable to maintain performance when scaling the *replay ratio* without explicit interventions (D'Oro et al., 2023; Schwarzer et al., 2023). In Figure 4 we observe that the use of Soft MoEs maintains a strong advantage over the baseline even at high replay ratios, further confirming that they make RL networks more parameter efficient.

4.2. Impact of Design Choices

Number of experts Fedus et al. (2022) argued that the number of experts is the most efficient way to scale models in supervised learning settings. Our results in Figure 1 demonstrate that while Soft MoE does benefit from more experts, Top1-MoE does not.

Dimensionality of experts As Figure 1 confirms, scaling the corresponding layer in the base architectures does not match the performance obtained when using Soft MoEs (and in fact worsens it). We explored dividing the dimensionality of each expert by the number of experts, effectively bringing the number of parameters on-par with the original base architecture. Figure 5 demonstrates that Soft MoE maintains its performance, even with much smaller experts. This suggests the observed benefits come largely from the structured sparsity induced by Soft MoEs, and not necessarily the size of each expert.

Gating and combining Top1-MoEs and Soft MoEs use learned gating mechanisms, albeit different ones: the former uses a top-1 router, selecting an expert for each token, while the latter uses dispatch weights to assign weighted tokens to expert slots. The learned "combiner" component takes the output of the MoE modules and combines them to produce a single output (see Figure 2). If we use a single expert, the only difference between the base architectures and the MoE ones are then the extra learned parameters from the gating and combination components. Figure 1 suggests that most of the benefit of MoEs comes from the combination of the gating/combining components with multiple experts. Interestingly, Soft MoE with a single expert still provides performance gains for Rainbow, suggesting that the learned Φ matrix (see Section 2.2) has a beneficial role to play. Indeed, the right panel of Figure 8, where we replace the learned Φ with a random one, confirms this.

Tokenization As mentioned above, we focus most of our investigations on PerConv tokenization, but also explore two others: PerFeat is essentially a transpose of PerConv, producing d tokens of dimensionality $h \times w$; PerSamp uses the entire output of the encoder as a *single* token (see Figure 3). In Figure 6 we can observe that while PerConv works best with Soft MoE, Top1-MoE seems to benefit more from PerFeat.

Encoder Although we have mostly used the encoder from the Impala architecture (Espeholt et al., 2018), in Figure 7 we confirm that Soft MoE still provides benefits when used with the standard CNN architecture from Mnih et al. (2015).

Game selection To confirm that our findings are not limited to our choice of 20 games, we ran a study over all the 60 Atari 2600 games with 5 independent seeds, similar to previous works (Fedus et al., 2020; Ceron et al., 2023). In the left panel of Figure 8, we observe that Soft MoE results in improved performance over the full 60 games in the suite.

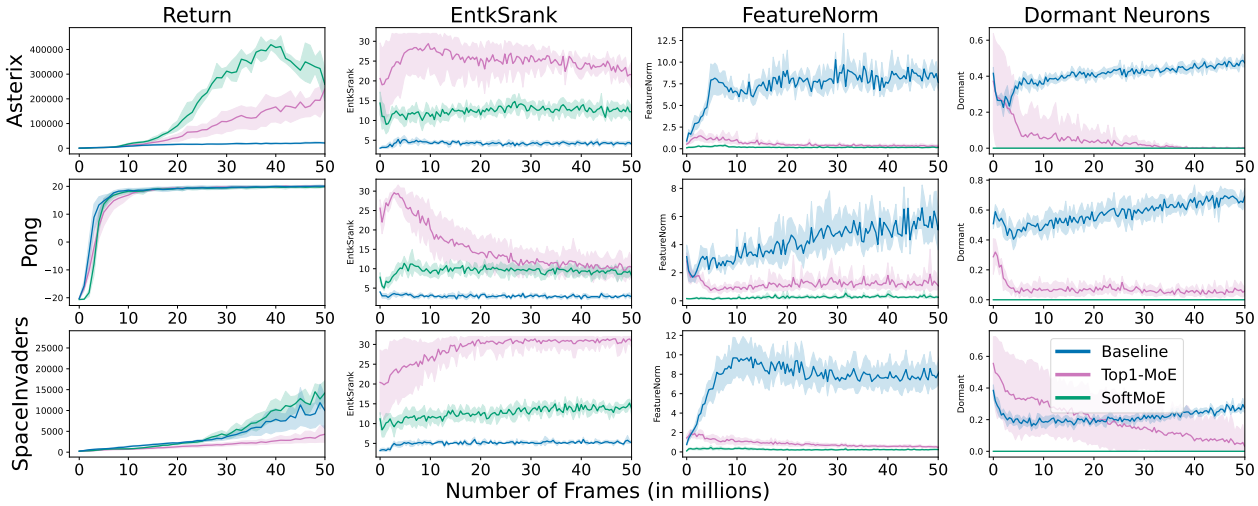


Figure 10. **Additional analyses:** both standard Top1-MoE and Soft MoE architectures exhibit similar properties of the hidden representation: both are more robust to dormant neurons and have higher effective rank of both the features and the gradients. We also see that the MoE architectures exhibit less feature norm growth than the baseline. See Section 4 for training details.

Number of active experts A crucial difference between the two flavors of MoEs considered here is in the activation of experts: Top1-MoE activates only *one* expert per forward pass (hard-gating), whereas Soft MoE activates all of them. Hard-gating is known to be a source of training difficulties, and many works have explored adding load-balancing losses (Shazeer et al., 2017; Ruiz et al., 2021; Fedus et al., 2022; Mustafa et al., 2022). To investigate whether Top1-MoE is underperforming due to improper load balancing, we added the load and importance losses proposed by Ruiz et al. (2021) (equation (7) in Appendix 2). Figure 9 suggests the addition of these load-balancing losses is insufficient to boost the performance of Top1-MoE. While it is possible other losses may result in better performance, these findings suggest that RL agents benefit from having a weighted combination of the tokens, as opposed to hard routing.

4.3. Additional Analysis

In the previous sections, we have shown that deep RL agents using MoE networks are better able to take advantage of network scaling, but it is not obvious a priori how they produce this effect. While a fine-grained analysis of the effects of MoE modules on network optimization dynamics lies outside the scope of this work, we zoom in on three properties known to correlate with training instability in deep RL agents: the rank of the features (Kumar et al., 2021a), interference between per-sample gradients (Lyle et al., 2022b), and dormant neurons (Sokar et al., 2023). We conduct a deeper investigation into the effect of MoE layers on learning dynamics by studying the Rainbow agent using the Impala architecture, and using 8 experts for the runs with the MoE modules. We track the norm of the features, the rank of the empirical neural tangent kernel (NTK) matrix (i.e. the matrix of dot products between per-transition gradients sampled from the replay buffer), and the number of dormant neurons – all using a batch size of 32 – and visualize these results in Figure 10.

We observe significant differences between the baseline architecture and the architectures that include MoE modules. The MoE architectures both exhibit higher numerical ranks of the empirical NTK matrices than the baseline network, and have negligible dormant neurons and feature norms. These findings suggest that the MoE modules have a stabilizing effect on optimization dynamics, though we refrain from claiming a direct causal link between improvements in these metrics and agent performance. For example, while the rank of the ENTK is higher for the MoE agents, the best-performing agent does not have the highest ENTK rank. The absence of pathological values of these statistics in the MoE networks does however, suggest that whatever the precise causal chain is, the MoE modules have a stabilizing effect on optimization.

5. Future directions

To provide an in-depth investigation into both the resulting performance and probable causes for the gains observed, we focused on evaluating DQN and Rainbow with Soft MoE and Top1-MoE on the standard ALE benchmark. The observed performance gains suggest that these ideas would also be beneficial in other training regimes. In this section

we provide strong empirical results in a number of different training regimes, which we hope will serve as indicators for promising future directions of research.

5.1. Offline RL

We begin by incorporating MoEs in offline RL, where agents are trained on a fixed dataset without environment interactions. Following prior work (Kumar et al., 2021b), we train the agents over 17 games and 5 seeds for 200 iterations, where 1 iteration corresponds to 62,500 gradient updates. We evaluated on datasets composed of 5%, 10%, 50% of the samples (drawn randomly) from the set of all environment interactions collected by a DQN agent trained for 200M steps (Agarwal et al., 2020). In Figure 11 we observe that combining Soft MoE with two modern offline RL algorithms (CQL (Kumar et al., 2020) and CQL+C51 (Kumar et al., 2022)) attains the best aggregate final performance. Top1-MoE provides performance improvements when used in conjunction with CQL+C51, but not with CQL. Similar improvements can be observed when using 10% of the samples (see Figure 17).

5.2. Agent Variants For Low-Data Regimes

DQN and Rainbow were both developed for a training regime where agents can take millions of steps in their environments. Kaiser et al. (2020) introduced the 100k[4] benchmark, which evaluates agents on a much smaller data regime (100k interactions) on 26 games. We evaluate the performance on two popular agents for this regime: DrQ(ϵ), an agent based on DQN (Yarats et al., 2021; Agarwal et al., 2021), and DER (Van Hasselt et al., 2019), an agent based on Rainbow. When trained for 100k environment interactions we saw no real difference between the different variants, suggesting that the benefits of MoEs, at least in our current setup, only arise when trained for a significant amount of interactions. However, when trained for 50M steps we do see gains in both agents, in particular with DER (Figure 12). The gains we observe in this setting are consistent with what we have observed so far, given that DrQ(ϵ) and DER are based on DQN and Rainbow, respectively.

5.3. Expert Variants

Our proposed MoE architecture replaces the feed-forward layer after the encoder with an MoE, where the experts consist of a single feed-forward layer (Figure 2). This is based on what is common practice when adding MoEs to transformer architectures, but is by no means the only way to utilize MoEs. Here we investigate three variants of using Soft MoE for DQN and Rainbow with the Impala architecture: **Big:** Each expert is a full network.The final linear layer is included in DQN (each expert has its own layer), but excluded from Rainbow (so the final linear layer is shared amongst experts).[5] **All:** A separate Soft MoE is applied at each layer of the network, excluding the last layer for Rainbow. **Regular:** the setting used in the rest of this paper. For **Big** and **All**, the routing is applied at the the input level, and we are using PerSamp tokenization (see Figure 3).

Puigcerver et al. (2023) propose using ℓ_2 normalization to each Soft MoE layer for large tokens, but mention that it makes little difference for smaller tokens. Since the tokens in our experiments above are small (relative to the large models typically using MoEs) we chose not to include this in the experiments run thus far. This may no longer be the case with **Big** and **All**, since we are using PerSamp tokenization on the full input; for this reason, we investigated the impact of ℓ_2 normalization when running these results.

Figure 13 summarizes our results, from which we can draw a number of conclusions. First, these experiments confirm our intuition that *not* including regularization in the setup used in the majority of this paper performs best. Second, consistent with Puigcerver et al. (2023), normalization seems to help with **Big** experts. This is particularly noticeable in DQN, where it even surpasses the performance of the Regular setup; the difference between the two is most likely due to the last layer being shared (as in Rainbow) or non-shared (as in DQN). Finally, using separate MoE modules for each layer (**All**) seems to not provide many gains, *especially* when coupled with normalization, where the agents are completely unable to learn.

In summary, our results suggest there may be promise in exploring alternative architectures for use with mixtures of experts.

6. Related Work

Mixture of Experts MoEs were first proposed by Jacobs et al. (1991) and have recently helped scaling language models up to trillions of parameters thanks to their modular nature, facilitating distributed training, and improved parameter efficiency at inference (Lepikhin et al., 2020; Fedus et al., 2022). MoEs are also widely studied in computer vision (Wang et al., 2020; Yang et al., 2019; Abbas & Andreopoulos, 2020; Pavlitskaya et al., 2020), where they enable scaling vision transformers to billions of parameters while reducing the inference computation cost by half (Riquelme et al., 2021), help with vision-based continual learning (Lee et al., 2019), and multi-task problems (Fan et al., 2022). MoEs also help performance in transfer- and

[4]Here, 100k refers to agent steps, or 400k environment frames, due to skipping frames in the standard training setup.

[5]This design choice was due to the use of C51 in Rainbow, which makes it non-trivial to maintain‘ the “token-preserving” property of MoEs.

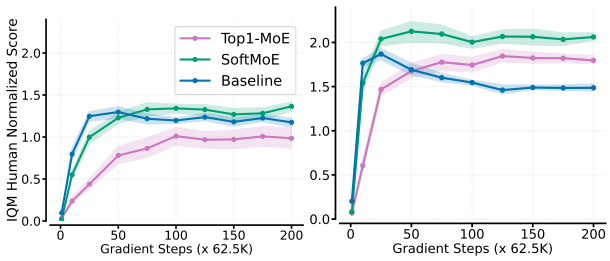


Figure 11. **Normalized performance across** 17 **Atari games for CQL (left) and CQL + C51 (right)**, with the ResNet (Espeholt et al., 2018) architecture and 8 experts trained on offline data. Soft MoE not only remains generally stable with more training, but also attains higher final performance. See Section 4 for training details.

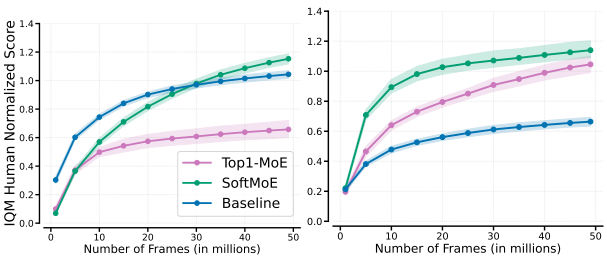


Figure 12. **Normalized performance across** 26 **Atari games for DrQ(ϵ) (left) and DER (right)**, with the ResNet architecture (Espeholt et al., 2018) and 8 experts (see Figure 16, for 4 experts). Soft MoE not only remains generally stable with more training, but also attains higher final performance. We report interquantile mean performance with error bars indicating 95% confidence intervals.

multi-task learning settings, e.g. by specializing experts to sub-problems (Puigcerver et al., 2020; Chen et al., 2023; Ye & Xu, 2023) or by addressing statistical performance issues of routers (Hazimeh et al., 2021). There have been few works exploring MoEs in RL for single (Ren et al., 2021; Akrour et al., 2022) and multi-task learning (Hendawy et al., 2024). However their definition and usage of "mixtures-of-experts" is somewhat different than ours, focusing more on orthogonal, probabilistic, and interpretable MoEs.

Parameter Scalability and Efficiency in Deep RL Lack of parameter scalability in deep RL can be partly explained by a lack of parameter efficiency. Parameter count cannot be scaled efficiently if those parameters are not used effectively. Recent work shows that networks in RL under-utilize their parameters. Sokar et al. (2023) demonstrates that networks suffer from an increasing number of inactive neurons throughout online training. Similar behavior is observed by Gulcehre et al. (2022) in offline RL. Arnob et al. (2021) shows that 95% of the network parameters can be pruned at initialization in offline RL without loss in performance. Numerous works demonstrate that periodic resetting of the network weights improves performance (Nikishin et al., 2022; Dohare et al., 2021; Sokar et al., 2023; D'Oro et al., 2022; Igl et al., 2020; Schwarzer et al., 2023).

Another line of research demonstrates that RL networks can be trained with a high sparsity level ($\sim$90%) without loss in performance (Tan et al., 2022; Sokar et al., 2022; Graesser et al., 2022; Ceron et al., 2024). These observations call for techniques to better utilize the network parameters in RL training, such as using MoEs, which we show decreases dormant neurons drastically over multiple tasks and architectures. To enable scaling networks in deep RL, prior works focus on algorithmic methods (Farebrother et al., 2022; Taiga et al., 2022; Schwarzer et al., 2023; Farebrother et al., 2024). In contrast, we focus on alternative *network topologies* to enable robustness towards scaling.

7. Discussion and Conclusion

As RL continues to be used for increasingly complex tasks, we will likely require larger networks. As recent research has shown (and which our results confirm), naïvely scaling up network parameters does not result in improved performance. Our work shows empirically that MoEs have a beneficial effect on the performance of value-based agents across a diverse set of training regimes.

Mixtures of Experts induce a form of *structured sparsity* in neural networks, prompting the question of whether the benefits we observe are simply a consequence of this sparsity rather than the MoE modules themselves. Our results suggest that it is likely a combination of both: Figure 1 demonstrates that in Rainbow, adding an MoE module with a *single* expert yields statistically significant performance improvements, while Figure 5 demonstrates that one can scale down expert dimensionality without sacrificing performance. The right panel of Figure 8 further confirms the necessity of the extra parameters in Soft MoE modules.

Recent findings in the literature demonstrate that while RL networks have a natural tendency towards neuron-level sparsity which can hurt performance (Sokar et al., 2023), they can benefit greatly from explicit parameter-level sparsity (Graesser et al., 2022). When taken in combination with our findings, they suggest that there is still much room for exploration, and understanding, of the role sparsity can play in training deep RL networks, especially for parameter scalability.

Even in the narrow setting we have focused on (replacing the penultimate layer of value-based agents with MoEs for off-policy learning in single-task settings), there are many open questions that can further increase the benefits of MoEs: different values of k for Top1-MoEs, different tokenization choices, using different learning rates (and perhaps optimizers) for routers, among others. Of course, expanding

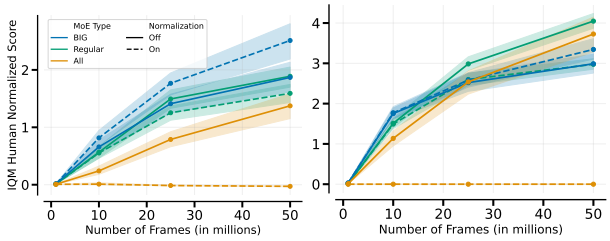


Figure 13. **Normalized performance over** 20 **games for expert variants** with DQN (left) and Rainbow (right), also investigating the use of the normalization of Puigcerver et al. (2023). We report interquantile mean performance with shaded areas indicating 95% confidence intervals.

beyond the ALE could provide more comprehensive results and insights, potentially at a fraction of the computational expense (Ceron & Castro, 2021).

The results presented in Section 5 suggest MoEs can play a more generally advantageous role in training deep RL agents. More broadly, our findings confirm the impact architectural design choices can have on the ultimate performance of RL agents. We hope our findings encourage more researchers to further investigate this – still relatively unexplored – research direction.

Acknowledgements

The authors would like to thank Gheorghe Comanici, Owen He, Alex Muzio, Adrien Ali Taïga, Rishabh Agarwal, Hugo Larochelle, Ayoub Echchahed, and the rest of the Google DeepMind Montreal team for valuable discussions during the preparation of this work; Gheorghe deserves a special mention for providing us valuable feedback on an early draft of the paper. We thank the anonymous reviewers for their valuable help in improving our manuscript. We would also like to thank the Python community (Van Rossum & Drake Jr, 1995; Oliphant, 2007) for developing tools that enabled this work, including NumPy (Harris et al., 2020), Matplotlib (Hunter, 2007), Jupyter (Kluyver et al., 2016), Pandas (McKinney, 2013) and JAX (Bradbury et al., 2018).

Impact statement

This paper presents work whose goal is to advance the field of Machine Learning, and reinforcement learning in particular. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbas, A. and Andreopoulos, Y. Biased mixtures of experts: Enabling computer vision inference under data transfer limitations. *IEEE Transactions on Image Processing*, 29: 7656–7667, 2020.

Agarwal, R., Schuurmans, D., and Norouzi, M. An optimistic perspective on offline reinforcement learning. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 104–114. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/agarwal20c.html.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Akrour, R., Tateo, D., and Peters, J. Continuous action reinforcement learning from a mixture of interpretable experts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(10): 6795–6806, oct 2022. ISSN 0162-8828. doi: 10.1109/TPAMI.2021.3103132. URL https://doi.org/10.1109/TPAMI.2021.3103132.

Arnob, S. Y., Ohib, R., Plis, S., and Precup, D. Single-shot pruning for offline reinforcement learning. *arXiv preprint arXiv:2112.15579*, 2021.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013a. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013b.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77 – 82, 2020.

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

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al. Jax: composable transformations of python+ numpy programs. 2018.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Ceron, J. S. O. and Castro, P. S. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pp. 1373–1383. PMLR, 2021.

Ceron, J. S. O., Bellemare, M. G., and Castro, P. S. Small batch deep reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=wPqEvmwFEh.

Ceron, J. S. O., Courville, A., and Castro, P. S. In value-based deep reinforcement learning, a pruned network is a good network. In *Forty-first International Conference on Machine Learning*. PMLR, 2024. URL https://openreview.net/forum?id=seo9V9QRZp.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Chen, Z., Shen, Y., Ding, M., Chen, Z., Zhao, H., Learned-Miller, E. G., and Gan, C. Mod-squad: Designing mixtures of experts as modular multi-task learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11828–11837, 2023.

Ding, X., Zhang, X., Han, J., and Ding, G. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11963–11975, 2022.

Dohare, S., Sutton, R. S., and Mahmood, A. R. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P.-L., Bellemare, M. G., and Courville, A. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022. URL https://openreview.net/forum?id=4GBGwVIEYJ.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P.-L., Bellemare, M. G., and Courville, A. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OpC-9aBBVJe.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/evci20a.html.

Fan, Z., Sarkar, R., Jiang, Z., Chen, T., Zou, K., Cheng, Y., Hao, C., Wang, Z., et al. M^3vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design. *Advances in Neural Information Processing Systems*, 35:28441–28457, 2022.

Farebrother, J., Greaves, J., Agarwal, R., Le Lan, C., Goroshin, R., Castro, P. S., and Bellemare, M. G. Proto-value networks: Scaling representation learning with auxiliary tasks. In *The Eleventh International Conference on Learning Representations*, 2022.

Farebrother, J., Orbay, J., Vuong, Q., Taïga, A. A., Chebotar, Y., Xiao, T., Irpan, A., Levine, S., Castro, P. S., Faust, A., Kumar, A., and Agarwal, R. Stop regressing: Training value functions via classification for scalable deep rl. In *Forty-first International Conference on Machine Learning*. PMLR, 2024.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pp. 3061–3071. PMLR, 2020.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Fukushima, K. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Trans. Syst. Sci. Cybern.*, 5(4):322–333, 1969. doi: 10.1109/TSSC.1969.300225. URL https://doi.org/10.1109/TSSC.1969.300225.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

Gale, T., Narayanan, D., Young, C., and Zaharia, M. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

Graesser, L., Evci, U., Elsen, E., and Castro, P. S. The state of sparse training in deep reinforcement learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7766–7792. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/graesser22a.html.

Gulcehre, C., Srinivasan, S., Sygnowski, J., Ostrovski, G., Farajtabar, M., Hoffman, M., Pascanu, R., and Doucet, A. An empirical study of implicit regularization in deep offline rl. *arXiv preprint arXiv:2207.02099*, 2022.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Hazimeh, H., Zhao, Z., Chowdhery, A., Sathiamoorthy, M., Chen, Y., Mazumder, R., Hong, L., and Chi, E. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

Hendawy, A., Peters, J., and D'Eramo, C. Multi-task reinforcement learning with mixture of orthogonal experts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aZH1dM3GOX.

Hessel, M., Modayil, J., Hasselt, H. V., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jesson, A., Lu, C., Gupta, G., Filos, A., Foerster, J. N., and Gal, Y. Relu to the rescue: Improve your on-policy actor-critic with positive advantages. *arXiv preprint arXiv:2306.01460*, 2023.

Kaiser, L., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *IOS Press*, pp. 87–90. 2016. doi: 10.3233/978-1-61499-649-1-87.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=O9bnihsFfXU.

Kumar, A., Agarwal, R., Ma, T., Courville, A., Tucker, G., and Levine, S. Dr3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2021b.

Kumar, A., Agarwal, R., Geng, X., Tucker, G., and Levine, S. Offline q-learning on diverse multi-task data both scales and generalizes. In *The Eleventh International Conference on Learning Representations*, 2022.

Lee, S., Ha, J., Zhang, D., and Kim, G. A neural dirichlet process mixture model for task-free continual learning. In *International Conference on Learning Representations*, 2019.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2020.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:232428341.

Lyle, C., Rowland, M., and Dabney, W. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=ZkC8wKoLbQ7.

Lyle, C., Rowland, M., Dabney, W., Kwiatkowska, M., and Gal, Y. Learning dynamics and generalization in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 14560–14581. PMLR, 2022b.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: evaluation protocols and open problems for general agents. *J. Artif. Int. Res.*, 61 (1):523–562, jan 2018. ISSN 1076-9757.

McKinney, W. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 1 edition, February 2013. ISBN 9789351100065. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1449319793.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015.

Mustafa, B., Riquelme, C., Puigcerver, J., Jenatton, R., and Houlsby, N. Multimodal contrastive learning with limoe: the language-image mixture of experts. *Advances in Neural Information Processing Systems*, 35:9564–9576, 2022.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. The primacy bias in deep reinforcement learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. doi: 10.1109/MCSE.2007.58.

Ostrovski, G., Castro, P. S., and Dabney, W. The difficulty of passive learning in deep reinforcement learning. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=nPHA8fGicZk.

Pavlitskaya, S., Hubschneider, C., Weber, M., Moritz, R., Huger, F., Schlicht, P., and Zollner, M. Using mixture of expert models to gain insights into semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 342–343, 2020.

Puigcerver, J., Ruiz, C. R., Mustafa, B., Renggli, C., Pinto, A. S., Gelly, S., Keysers, D., and Houlsby, N. Scalable transfer learning with expert models. In *International Conference on Learning Representations*, 2020.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts, 2023.

Ren, J., Li, Y., Ding, Z., Pan, W., and Dong, H. Probabilistic mixture-of-experts for efficient deep reinforcement learning. *CoRR*, abs/2104.09122, 2021. URL https://arxiv.org/abs/2104.09122.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595, 2021.

Ruiz, C. R., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A. S., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=FrIDgjDOH1u.

Schwarzer, M., Obando Ceron, J. S., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. Bigger,

better, faster: Human-level Atari with human-level efficiency. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 30365–30380. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/schwarzer23a.html.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Sokar, G., Mocanu, E., Mocanu, D. C., Pechenizkiy, M., and Stone, P. Dynamic sparse training for deep reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. The dormant neuron phenomenon in deep reinforcement learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32145–32168. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/sokar23a.html.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Taiga, A. A., Agarwal, R., Farebrother, J., Courville, A., and Bellemare, M. G. Investigating multi-task pretraining and generalization in reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Tan, Y., Hu, P., Pan, L., Huang, J., and Huang, L. Rlx2: Training a sparse deep reinforcement learning model from scratch. In *The Eleventh International Conference on Learning Representations*, 2022.

Van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

Van Rossum, G. and Drake Jr, F. L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

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

Wang, X., Yu, F., Dunlap, L., Ma, Y.-A., Wang, R., Mirhoseini, A., Darrell, T., and Gonzalez, J. E. Deep mixture of experts via shallow embedding. In *Uncertainty in artificial intelligence*, pp. 552–562. PMLR, 2020.

Yang, B., Bender, G., Le, Q. V., and Ngiam, J. Condconv: Conditionally parameterized convolutions for efficient inference. *Advances in neural information processing systems*, 32, 2019.

Yarats, D., Kostrikov, I., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Ye, H. and Xu, D. Taskexpert: Dynamically assembling multi-task representations with memorial mixture-of-experts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 21828–21837, 2023.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models, 2022.

6.1 Limitations and Future Work

This chapter provides strong empirical evidence that incorporating MoE modules, especially Soft MoE, into value-based deep RL agents like DQN and Rainbow improves performance that scales positively with the number of experts. This addresses a key limitation of standard architectures. Our analyses link these gains to better parameter utilisation.

However, there are several limitations which suggest avenues for future work. Firstly, our study focused exclusively on value-based RL algorithms (DQN, Rainbow) within the Atari Learning Environment. While a standard benchmark, generalising these findings to other algorithmic classes, such as actor-critic methods (PPO (Schulman et al., 2017), SAC (Haarnoja et al., 2018)) and other domains, including continuous control, robotics, or complex strategy games, is crucial. The interplay between MoEs and policy representations or actor-critic dynamics might differ significantly. Chapter 7 begins to explore actor-critic methods, but a broader investigation is needed.

Secondly, we adopted a specific MoE integration strategy: replacing the penultimate layer, even though it is common in transformers to have MoEs in all layers. The optimal placement and configuration of MoE modules within deep RL network architectures (e.g., earlier layers, multiple MoE layers, different tokenisation schemes) warrants further exploration. Our initial investigation in Chapter 7 touches on this, revealing challenges that deserve deeper study. Moreover, MoEs are particularly popular in supervised learning because they enable different types of parallelisation across data and GPUs (Huang et al., 2019; Shoeybi et al., 2019; Rajbhandari et al., 2020). These parallelisation paradigms were not explored in the above work.

Thirdly, while we favoured Soft MoE based on its differentiability and empirical success, the reasons for the comparatively poorer performance of Top1-MoE in our RL setting are not fully explained. Whether this is due to difficulties of hard routing in RL training dynamics or suboptimal router training/load-balancing (despite testing standard losses) remains an open question. Developing RL-specific routing mechanisms for MoEs could be a fruitful research direction.

Fourthly, the theoretical understanding of why MoEs benefit deep RL parameter scaling remains nascent. While our analyses of dormant neurons and representational rank provide useful indications, a deeper causal understanding of how MoEs interact with TD learning, function approximation stability, and exploration is lacking. Developing theoretical tools to analyse MoE dynamics in the context of RL optimisation could guide more principled architectural design (Gallici et al., 2024; Lyle et al., 2024).

Finally, from the "internal MARL" analogy, the routing mechanism employed is relatively simple compared to sophisticated MARL coordination strategies. Future work could explore explicitly training the router using MARL algorithms or concepts (e.g., rewarding routers for effective expert allocation or utilising communication protocols between experts), potentially leading to more adaptive and efficient internal specialisation and coordination.

In conclusion, this chapter establishes MoEs as a suitable tool for scaling deep RL agents. Still, work remains in generalising these findings across algorithms and domains, optimising MoE configurations for RL, deepening our theoretical understanding, and fully exploring the potential suggested by the internal multi-agent system perspective. Chapter 7 begins to explore some of these aspects by evaluating MoEs in actor-critic agents under multi-task and continual learning settings.

7

Mixture of Experts in a Mixture of RL settings

Chapter 6 established a crucial finding: Mixture of Expert architectures, especially Soft MoE, successfully unlock parameter scaling in standard single-task deep RL environments like the ALE. By enabling performance to improve significantly with the increased expert count, MoEs offer a promising architectural solution to the scaling limitations that plague monolithic deep RL networks (Section 2.4), seemingly by promoting better internal parameter utilisation and stability. This provides initial motivation for viewing MoEs through the lens of an internal cooperative system (Section 2.5), where expert specialisation and coordination contribute to overall capability.

However, the standard RL training protocol investigated in Chapter 6 represents only one facet of learning agents' challenges. Real-world scenarios often demand agents that handle multiple tasks concurrently or adapt sequentially to entirely new environments or objectives. These situations introduce explicitly amplified non-stationarity exceeding that typically encountered in single-task learning. This allows us to understand how MoE architectures, with their inherent modularity and specialisation, perform under these more demanding conditions.

This chapter investigates the efficacy and dynamics of MoE architectures when subjected to increased non-stationarity of Multi-Task Reinforcement Learning (MTRL) and Continual Reinforcement Learning (CRL) settings. These settings allow us to probe deeper into the internal coordination and adaptation capabilities of MoEs.

In MTRL, where multiple tasks are learned simultaneously, can the internal MoE router effectively learn to allocate different tasks or contexts to specialised expert agents? Does this internal specialisation lead to better performance and knowledge compartmentalisation than monolithic architectures facing diverse objectives concurrently?

In CRL, where tasks are learned sequentially, can the modularity of MoEs, by potentially isolating task-specific knowledge within experts, help mitigate the problem of catastrophic forgetting? How does the internal "coordination strategy" (routing) adapt as the agent transitions between distinct tasks?

Based on the work presented in Willi et al. (2024), this chapter extends the investigation from Chapter 6 by evaluating various MoE architectural configurations (including "Big", "All", "Middle", "Final" placements) and routing strategies (Hardcoded, Soft MoE, TopK). We utilise actor-critic agents (PPO) trained on MTRL and CRL benchmarks derived from MinAtar environments (Lange, 2022), chosen to amplify non-stationarity effects. By analysing agent performance, patterns of expert specialisation, and network plasticity metrics, this chapter provides insights into MoE behaviour under amplified non-stationarity. We aim to understand the critical MoE components and design choices for robust learning in dynamic settings, and derive best practices for integrating MoEs into agents facing diverse or changing task demands.

Mixture of Experts in a Mixture of RL settings

Timon Willi[*1,2,6], **Johan Obando-Ceron**[*3,4,6], **Jakob Foerster**[1,2], **Karolina Dziugaite**[5,6], **Pablo Samuel Castro**[3,4,6]

Foerster Lab for AI Research[1]
University of Oxford[2]
Mila - Québec AI Institute[3]
Université de Montréal[4]
McGill University[5]
Google DeepMind[6]

Abstract

Mixtures of Experts (MoEs) have gained prominence in (self-)supervised learning due to their enhanced inference efficiency, adaptability to distributed training, and modularity. Previous research has illustrated that MoEs can significantly boost Deep Reinforcement Learning (DRL) performance by expanding the network's parameter count while reducing dormant neurons[1], thereby enhancing the model's learning capacity and ability to deal with non-stationarity. In this work, we shed more light on MoEs' ability to deal with non-stationarity and investigate MoEs in DRL settings with "amplified" non-stationarity via multi-task training, providing further evidence that MoEs improve learning capacity. In contrast to previous work, our multi-task results allow us to better understand the underlying causes for the beneficial effect of MoE in DRL training, the impact of the various MoE components, and insights into how best to incorporate them in actor-critic-based DRL networks. Finally, we also confirm results from previous work.

1 Introduction

Deep Reinforcement Learning (RL), which integrates reinforcement learning algorithms with deep neural networks, has demonstrated remarkable success in enabling agents to achieve complex tasks beyond human capabilities in domains ranging from video games to strategic board games and beyond (Mnih et al., 2015; Berner et al., 2019; Vinyals et al., 2019; Fawzi et al., 2022; Bellemare et al., 2020). Despite the pivotal role of deep networks in these advanced RL applications, their learning dynamics within RL contexts still need to be fully understood. Recent research has uncovered unexpected behaviours and phenomena associated with the use of deep networks in RL, which often diverge from those observed in traditional supervised learning environments (Ostrovski et al., 2021; Kumar et al., 2021; Lyle et al., 2022; Graesser et al., 2022; Nikishin et al., 2022a; Sokar et al., 2023; Obando Ceron et al., 2023).

Transformers (Vaswani et al., 2017), adapters (Houlsby et al., 2019), and Mixture of Experts (MoEs; Shazeer et al., 2017), are crucial for the scalability of supervised learning models, particularly within the domains of natural language processing and computer vision. MoEs stand out by facilitating the scaling of networks to encompass trillions of parameters, a feat made possible through their modular design that seamlessly integrates with distributed computing techniques (Fedus et al., 2022). Moreover, MoEs introduce a form of structured sparsity into the network architecture, a characteristic

*Authors contributed equally. Correspondence to [timon.willi, jobando0730]@gmail.com,psc@google.com

[1] *Dormant neurons:* neurons that have become practically inactive through low activations.

associated with enhancements in network performance through various studies on network sparsity (Evci et al., 2020; Gale et al., 2019; Jin et al., 2022). Finally, there is growing evidence in the supervised learning literature that MoEs specialise on different problem characteristics in *multi-task* settings (Gupta et al., 2022). These settings are inherently non-stationary and may benefit from the modularity and sparsity induced by MoE-based architectures.

Recently, Ceron et al. (2024b) demonstrated that MoEs unlock scaling in DRL networks for single-task settings. However, they did so under a specific setting where only the penultimate layer was replaced by an MoE module. Their analyses suggest that incorporating MoEs makes networks less susceptible to loss of plasticity, as evidenced by measurements including the fraction of dormant neurons. Sokar et al. (2023), in exploring the phenomenon of dormant neurons in DRL, provided strong evidence that their growth is due mainly to the non-stationary nature of RL training.

In this work, we set out to better understand *how MoEs help training under non-stationarity* and *which aspects of MoEs yield these results*. To do so, we "amplify" the non-stationarity of DRL training by investigating settings where multiple tasks are learned concurrently by the same agent. Specifically, we investigate the incorporation of a variety of MoE architectures in Multi-Task Reinforcement Learning (MTRL) and Continual Reinforcement Learning (CRL) settings. Our results demonstrate that the induced sparsity of expert modules is critical to mitigating plasticity loss under amplified non-stationarity and highlight the difficulty and importance of properly training the router. While focusing on the MTRL and CRL settings, some insights below apply to more traditional single-task settings.

2 Background

2.1 Reinforcement Learning

A Markov Decision Process (MDP; Bellman, 1957; Puterman, 1990; Sutton & Barto, 2018) is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma \rangle$, where $\mathcal{S}$ denotes the set of all possible states, $\mathcal{A}$ denotes the set of possible actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition probability kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, ρ denotes the initial state distribution, and γ (where $0 < \gamma \leq 1$) is the discount factor that determines the present value of future rewards. In Reinforcement Learning (RL), a policy π assigns to each state s a probability distribution $\pi(s)$ over actions in $\mathcal{A}$. The objective in RL is to devise a policy π that maximises the expected sum of discounted rewards $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right)\right]$. The policy parameterised by θ is denoted as $\pi_\theta(a_t \mid s_t)$. The parameter θ is chosen via optimisation maximising $J(\theta)$, thereby achieving the highest possible cumulative reward.

In **Multi-Task Reinforcement Learning** (MTRL), an agent engages with a variety of tasks τ from a set $\mathcal{T}$, with each task τ constituting a distinct Markov Decision Process (MDP) denoted by $\mathcal{M}^\tau = \langle \mathcal{S}^\tau, \mathcal{A}^\tau, \mathcal{P}^\tau, r^\tau, \rho^\tau, \gamma^\tau \rangle$. The objective in MTRL is to devise a unified policy π that optimises the average expected cumulative discounted return across all tasks, expressed as $J(\theta) = \frac{1}{|\mathcal{T}|} \sum_\tau J_\tau(\theta)$. In our work, at each training step a single agent trains synchronously on multiple tasks. MTRL is effective at measuring an agent's capabilities at devising control policies from a highly varying set of inputs and environment dynamics.

Continual RL (Abbas et al., 2023) is a variant of MTRL where the agent trains on one task for an extended period before switching to a new task (Khetarpal et al., 2022); once all tasks have been trained on once, the agent once again trains on all the environments in the same order. This setting enables measuring an agent's ability to learn new tasks while retaining previously learned policies.

As a concrete example, imagine we have two MDPs ($\mathcal{M}^1$, $\mathcal{M}^2$). MTRL would train on both ($\mathcal{M}^1, \mathcal{M}^2$) at each step, whereas CRL would train on $\mathcal{M}^1$ for an extended number of steps, then $\mathcal{M}^2$, and so on: $\mathcal{M}^1 \to \mathcal{M}^2 \to \mathcal{M}^1 \to \mathcal{M}^2$.

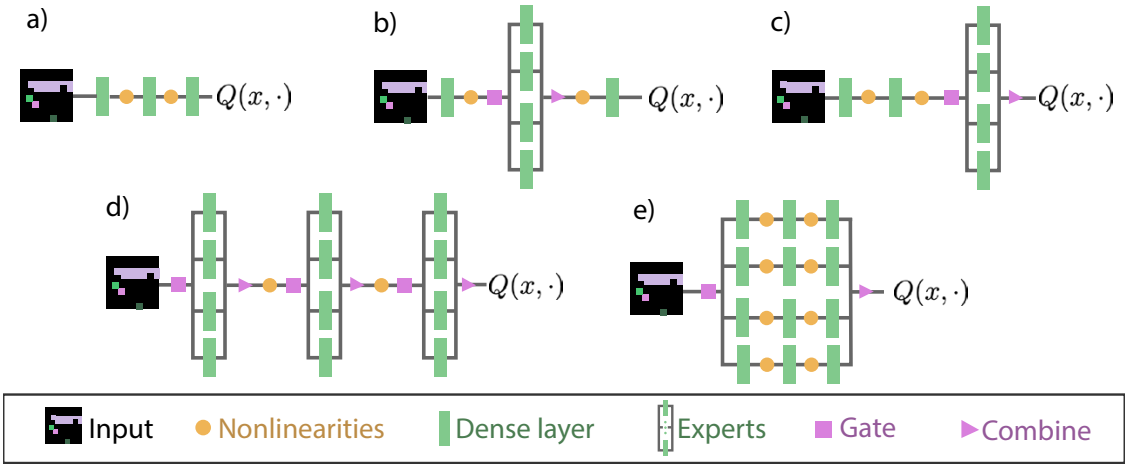


Figure 1: **Architectures considered: (a)** Baseline architecture; **(b)** Middle, used by Ceron et al. (2024b); **(c)** Final, where an MoE module replaces the final layer; **(d)** All, where all layers are replaced with an MoE module; **(e)** Big, with a single MoE module where an expert comprises the full original network.

2.2 MoEs

Mixtures of Experts (MoEs) emerged as a cornerstone in designing Large Language Models (LLMs), integrating a collection of n "expert" sub-networks. A gating mechanism, known as the router and usually learned during training, manages the experts by directing each incoming token to k selected experts (Shazeer et al., 2017). Typically, k is less than the total count of experts (in our case, $k = 1$). This sparsity is key for enhancing inference speed and facilitating distributed computing, making it a pivotal factor in training LLMs. In transformer architectures, MoE units substitute all dense feedforward layers (Vaswani et al., 2017). The impressive empirical performance of MoEs has sparked significant research interest (Shazeer et al., 2017; Lewis et al., 2021; Fedus et al., 2022; Zhou et al., 2022; Puigcerver et al., 2023; Lepikhin et al., 2020; Zoph et al., 2022; Gale et al., 2023).

The strict routing of tokens to specific experts, known as hard assignments, presents several issues, including training instability, token loss, and obstacles in expanding the number of experts (Fedus et al., 2022; Puigcerver et al., 2023). To mitigate these issues, Puigcerver et al. (2023) proposed the concept of **Soft MoE**, which utilises a soft, fully differentiable method for allocating tokens to experts, thereby circumventing the limitations associated with router-based hard assignments. This soft assignment method calculates a blend of weights for each token across the experts and aggregates their outputs accordingly. Adopting the terminology of Puigcerver et al. (2023), consider input tokens represented by $\mathbf{X} \in \mathbb{R}^{m \times d}$, with m indicating the count of d-dimensional tokens. A Soft MoE layer processes these tokens through n experts, each defined as $\{f_i : \mathbb{R}^d \to \mathbb{R}^d\}_{1:n}$. Every expert is associated with p slots for both input and output, each slot characterised by a d-dimensional vector of parameters. These parameters are collectively denoted as $\mathbf{\Phi} \in \mathbb{R}^{d \times (n \cdot p)}$.

The input slots $\tilde{\mathbf{X}} \in \mathbb{R}^{(n \cdot p) \times d}$ represent a weighted average of all tokens, given by $\tilde{\mathbf{X}} = \mathbf{D}^\top \mathbf{X}$, where $\mathbf{D}$ is commonly known as the dispatch weights. The outputs from the experts are expressed as $\tilde{\mathbf{Y}}_i = f_{\lfloor i/p \rfloor}(\tilde{\mathbf{X}}_i)$. For the Soft MoE layer, the overall output $\mathbf{Y}$ results from merging $\tilde{\mathbf{Y}}$ with the combined weights $\mathbf{C}$, described by $\mathbf{Y} = \mathbf{C}\tilde{\mathbf{Y}}$. $\mathbf{D}$ and $\mathbf{C}$ are represented by the following expressions:

$$\mathbf{D}_{ij} = \frac{\exp\left((\mathbf{X}\mathbf{\Phi})_{ij}\right)}{\sum_{i'=1}^{m} \exp\left((\mathbf{X}\mathbf{\Phi})_{i'j}\right)}, \qquad \mathbf{C}_{ij} = \frac{\exp\left((\mathbf{X}\mathbf{\Phi})_{ij}\right)}{\sum_{j'=1}^{n \cdot p} \exp\left((\mathbf{X}\mathbf{\Phi})_{ij'}\right)}.$$

The findings from Puigcerver et al. (2023) indicate that Soft MoE provides an improved balance between accuracy and computational expense relative to alternative MoE approaches.

3 Mixtures of Experts in a mixture of RL settings

Although shifting targets (due to bootstrapping) and dynamic data collection (from the agent's policy) already render single-task RL a non-stationary problem, MTRL and CRL take this non-stationarity to an extreme by changing the environments during training. A critical difference between the two is that in MTRL, at every step, the agent interacts and learns from each environment (e.g. regular task-switching). In contrast, in CRL tasks are switched very infrequently. Thus, both settings provide complementary perspectives when investigating the efficacy of MoEs under high levels of non-stationarity.

3.1 Experimental setup

As we investigate many settings in many scenarios, we wanted to maximise the number of runs per setting to ensure statistical robustness, while keeping the computational expense at bay. For this reason, we chose to run our experiments with the PureJaxRL codebase[2] (Lu et al., 2022b;a; 2023), which is a high-performance and parallelisable library including an implementation of Proximal Policy Optimisation (Schulman et al., 2017, PPO). Since Ceron et al. (2024b) focused on value-based methods, our use of PPO provides complementary insights and results. We rely on the Gymnax suite (Lange, 2022) to implement optimised versions of MinAtar environments (Young & Tian, 2019), which have been shown to provide insights comparable to the full ALE suite (Obando-Ceron & Castro, 2021). The hyper-parameters used are provided in Appendix H and were adapted from Jesson et al. (2023) (we deviate in the network size due to computational constraints). For all experiments, we evaluate on three environments: SpaceInvaders (SI), Breakout (BO), and Asterix (Ast). The input observations from Asterix differ substantially from SpaceInvaders and Breakout, whereas the latter are similar in observation and action space. This environment selection allows us to investigate whether MDP similarity encourages sharing representations between experts.

For **MTRL** we train simultaneously on SI, BO, and Ast; in practice, the PPO agent performs one update step per environment in sequence. For **CRL**, we train the agent on a fixed sequence of MDPs (Abbas et al., 2023) for $1e7$ environment steps ($\sim$ 80k update steps), specifically SI $\rightarrow$ BO $\rightarrow$ Ast $\rightarrow$ SI $\rightarrow$ BO $\rightarrow$ Ast. We present further analysis with different task orders in Section 4.

Ceron et al. (2024b) propose replacing the penultimate layer with an MoE module and sharing the other layers across the network. We term this variant Middle. We also evaluate a variant called Final, where the MoE module replaces the last layer. We also propose two new architectures: All, where MoE modules replace all three layers, and Big, where the network contains a single MoE module and each expert consists of a full network (see Figure 1). Since we are dealing with three distinct environments, all versions of MoEs have three experts. In all cases, we are using per-sample tokenization: one token – the state – per forward pass (Ceron et al., 2024b). All other hyper-parameters are reported in Appendix H.

We use a hardcoded routing strategy for many of our experiments to isolate the impact of *routing* versus *expert architecture*. This routing strategy will assign one expert for each task and route inputs accordingly. For the Big architecture, this effectively trains a separate network for each task and serves as a useful baseline. For all our results, we report the mean, averaged over 10 independent seeds, with shaded areas representing standard error. In most figures, we also present the average normalised performance across all tasks (in parentheses in the legend), where normalisation scores were taken from Jesson et al. (2023). Our experiments were run on a single Tesla P100 or A100 GPU, each taking 10 minutes. In total, we ran 870 distinct settings over 10 seeds each and are reported in Sections 3.2, 3.3 and 4 and Appendices A to G.

[2]PureJaxRL code available at: https://github.com/luchris429/purejaxrl

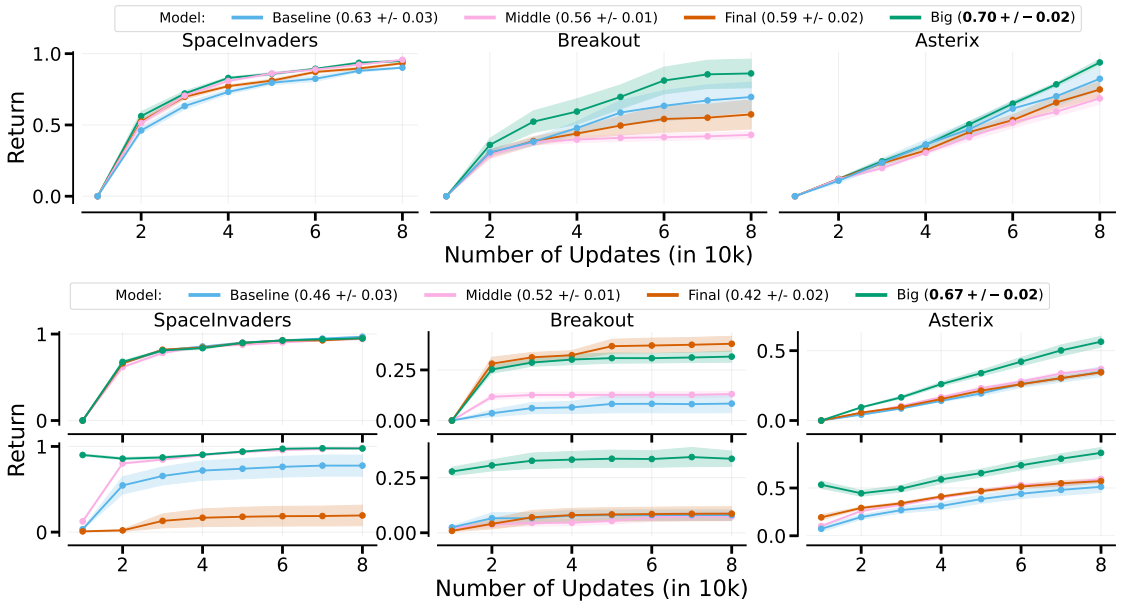


Figure 2: **Measuring the impact of MoE architectures** with hardcoded routing in MTRL (top). and CRL (bottom). In each legend, the numbers in parentheses indicate the average performance of each approach over all games. Big outperforms all other methods.

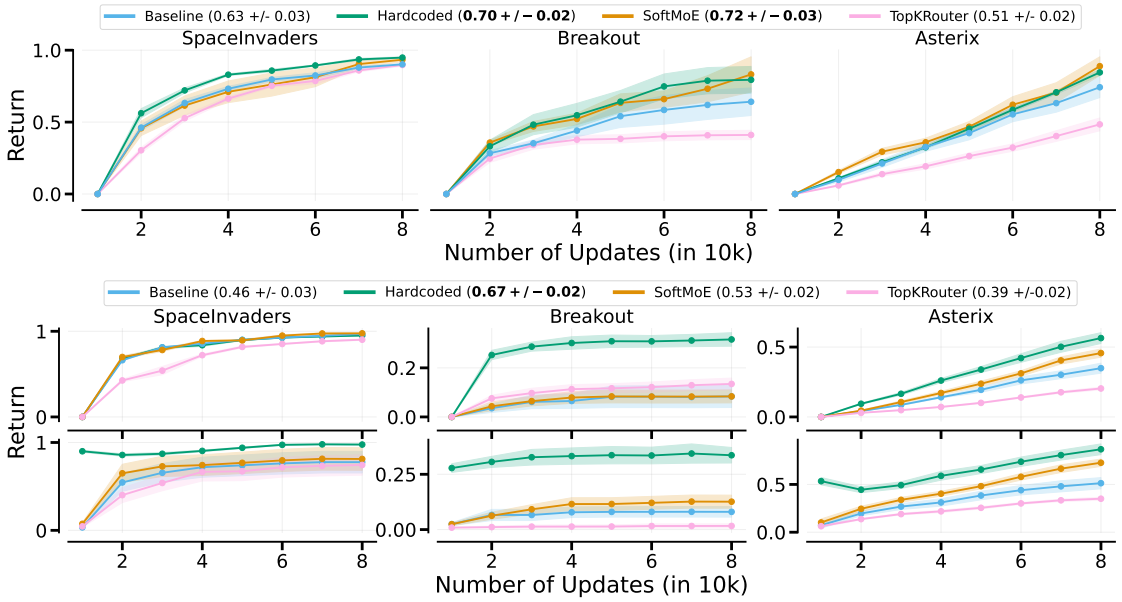


Figure 3: **Measuring the impact of routing** with Big architecture using different routing approaches under the MTRL (top row) and CRL (bottom row) settings. In each legend, the numbers in parentheses indicate the average performance of each approach across all games. SoftMoE and Hardcoded work best in MTRL, and Hardcoded works best in CRL, though SoftMoE still outperforms the baseline.

3.2 The impact of MoE architectures

To evaluate the impact of the choice of MoE architectures (Figure 1), we make use of the hardcoded router, which avoids potentially confounding factors due to also learning a routing strategy. In Figure 2, it is evident that the Big architecture performs best as expected since it is essentially a separate network per task. Still, it is promising to observe that all architectures outperform the baseline in the CRL setting, with All being the strongest performer. Surprisingly, in the MTRL setting, only Big outperforms the baseline. We hypothesise that All struggles due to suboptimal hyperparameters, as it was not computationally feasible to run a hyperparameter search over all possible settings. For Middle and Final, it is possible that gradient interference (Lyle et al., 2023) is complicating the learning process since there is parameter sharing outside of the MoE modules.

3.3 The impact of learned routers

The Big architecture provides a direct way to evaluate the impact of routing, as gating and combining are only done before and after the original network parameters, respectively. In Figure 3, we present the learning curves for Big architecture with varying routing strategies under CRL and MTRL.

In MTRL, we see little difference between the hardcoded router and SoftMoE. This is surprising since the hardcoded router trains separate networks for each task (performing as well as the baseline trained on each environment individually). This suggests that the gating used by SoftMoE is effective in situations where tasks are frequently changed. The rigidity of TopK routing appears to make it difficult for it to learn proper routing strategies, resulting in deteriorated performance.

In CRL, the hardcoded router performs best and retains previously learned policies (as evidenced by the second time the tasks are run). While SoftMoE ultimately outperforms the baseline in each task, it struggles in retaining previously learned policies; it is worth noting that the second time training on Ast, although its starting performance is essentially at zero, its final performance is higher than the first time training through, suggesting some policy retention (bottom right of Fig. 3).

A major learning challenge in the CRL setting is that no signal is provided to the network when the environment changes. Thus, a natural question is whether learned routers can effectively use task information. To investigate this, in Figure 4, we added the task ID as an input to the router (top row) and observed the surprising result that including task ID slightly hurts performance for Big-SoftMoE. Examining the gradient similarity from one update to the next (bottom left panel of Figure 4), it becomes evident that task-switching induces a discontinuity in the gradients used for learning. Interestingly, including this gradient similarity information as part of the input to the router does not hurt performance, but it does not improve either (bottom right).

In summary, our results suggest that SoftMoE routing is effective at dealing with high levels of non-stationarity, provided that discontinuous changes in environment dynamics (such as those arising from task switching) occur with relative frequency.

4 Extra Analyses

In the previous section, we provided empirical evidence suggesting that MoEs can improve DRL agents' performance in various non-stationary training regimes. Next, we conduct additional analyses to uncover the underlying causes of MoEs' benefits.

Impact on network plasticity. We measure the fraction of dormant neurons (Sokar et al., 2023) during training as a proxy for network plasticity. As Figs. 5 (top), 11, and 15 demonstrate, all MoE variants reduce the fraction of dormant neurons, suggesting MoEs help with maintaining network plasticity, consistent with the findings of Ceron et al. (2024b).

Expert specialisation. In Figs. 5 (bottom), 10, and 16 we measure the probabilities assigned to each expert during training; what these values indicate is the likelihood that inputs will be routed to each respective expert; observe that the hardcoded router has maximal specialisation, where each

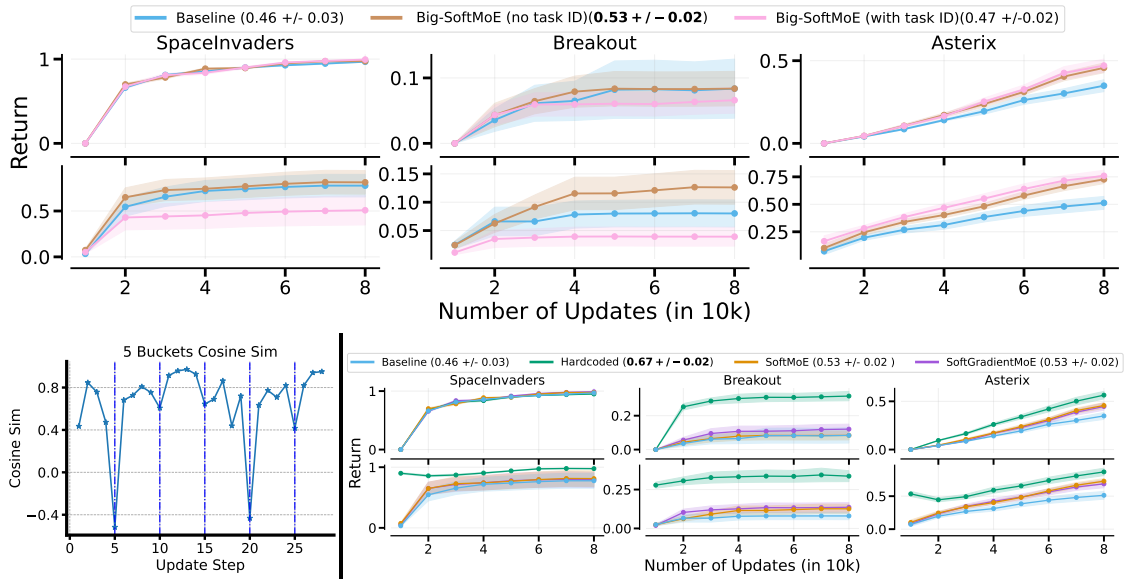


Figure 4: **Top:** Adding the task ID as an input to the router hurts performance for Big-SoftMoE. **Bottom left:** Sequential gradient similarity calculated throughout training, where dashed vertical lines represent when tasks switch. **Bottom right:** Adding gradient information as an input to the router does not improve performance.

expert is assigned to one task. We can also observe that both Big-SoftMoE and All-SoftMoE variants tend to specialise in all layers.

In supervised learning settings, it is common to use load-balancing losses to *avoid* this type of specialisation to maximise expert usage. We explored this idea by adding entropy regularisation during training and observed that, while we do see a decrease in expert specialisation (c.f. Figs. 4 (bottom), 10, and 16), this does not affect performance in any meaningful way (c.f. Fig. 14 and Table 8).

Impact on actor and critic networks. Ceron et al. (2024b) focused on value-based methods (where a single network serves as critic and actor), so using an actor-critic method like PPO provides a novel, complementary perspective. By default, we use MoE modules on the actor and critic networks, but in Figs. 6, 17 and 18 and Tables 9 and 10, we show that, in the two settings, it is best to use MoEs on both networks. However, the results suggest that MoEs have a greater impact on the actor than on the critic network. The fact that actor networks seem to benefit more from MoEs than critic networks is aligned with the findings of Graesser et al. (2022), where they found that actor networks could handle much higher levels of sparsity than critics without any degradation in performance.

Order of environments. To investigate the impact of environment ordering, we train using the ordering Ast → BO → SI to compare with the orderings we have used thus far; we present results for MTRL and CRL in Figs. 20 and 21, respectively. While conclusions do not change in MTRL, changing the order of environments affects CRL performance significantly (excluding the hardcoded router). We observe two interesting changes: (i) training BO after Ast (as opposed to after SI) causes all methods (excluding hardcoded) to collapse, and (ii) when training on Ast last, none of the agents were able to retain the learned policy (c.f. Fig. 3), whereas when training on Ast first there is some policy retention (as seen on the bottom left of Fig. 20). As mentioned previously, Ast differs substantially from the other two environments, so our findings in Fig. 20 suggest that the agents have overfit the input distribution of Ast, hindering its ability to adapt to the other environments but allowing the retention of the policy learned on Ast.

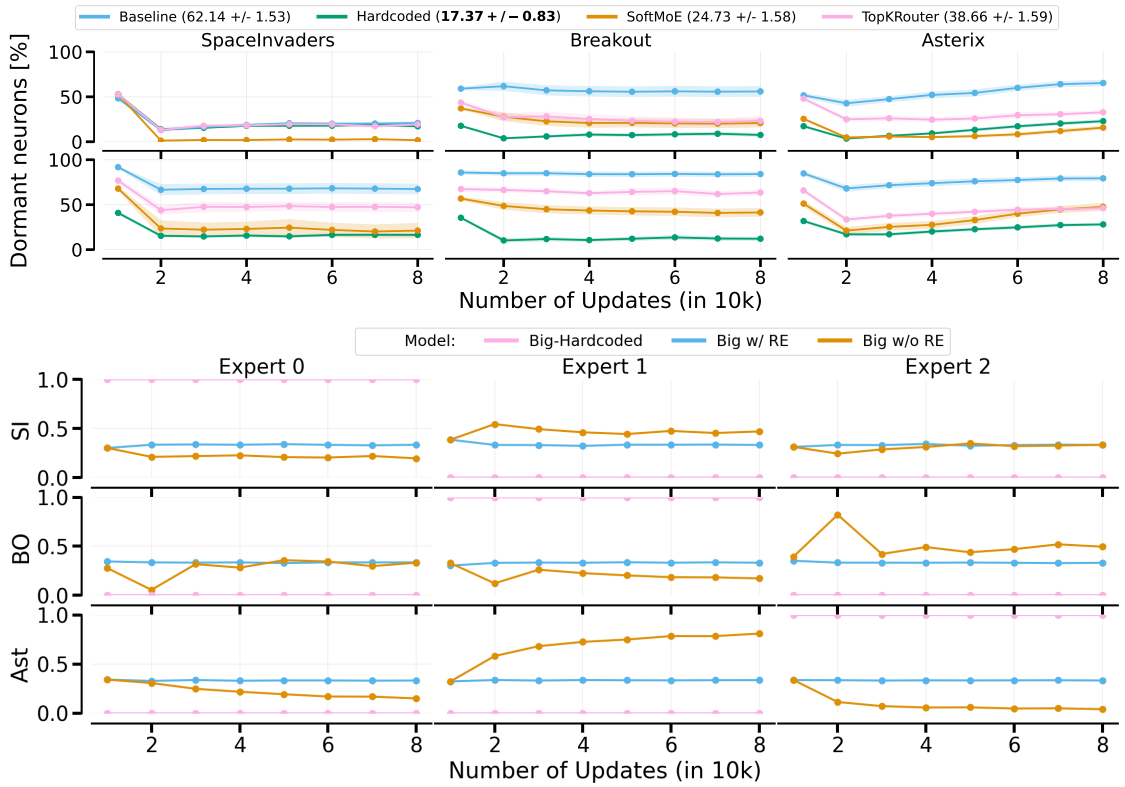


Figure 5: **Top:** presents the ratio of dormant neurons for CRL under different routing approaches using Big. The numbers in the legend represent average dormant neuron fractions across all games. MoE variants have lower dormant neurons than the baseline. **Bottom:** Regularising the entropy of the router makes the expert selection more uniform. Without regularisation, there is more specialisation. This shows one seed, as different seeds might choose different experts. See section 4 for more details.

Single Environment. Despite the clear improvements from CRL and MTRL, there are no significant performance improvements across all games in the single environment setting. However, Big improves over the baseline in Asterix while performing worse in Breakout, as shown in Figure 30 and Table 21, suggesting that MoEs might be beneficial in specific types of environments. Adding gradient information did not affect performance (see Figure 34).

5 Related Work

Parameter underutilisation is a roadblock to parameter efficiency in deep Reinforcement Learning (RL). The latter was highlighted by Sokar et al. (2023) in the form of dormant neurons. Arnob et al. (2021) demonstrate that in offline RL, up to 95% of network parameters can be pruned at initialisation without impacting performance. Further, several studies have shown that periodic network weight resets enhance performance (Igl et al., 2020; Dohare et al., 2021; Nikishin et al., 2022b; D'Oro et al., 2022; Sokar et al., 2023; Schwarzer et al., 2023) and that RL networks maintain performance when trained with a high degree of sparsity (Tan et al., 2022; Sokar et al., 2022; Graesser et al., 2022; Ceron et al., 2024a). These findings underscore the need for methods that more effectively leverage network parameters in RL training. Our work explores the use of Mixture of Experts (MoEs) for actor-critic methods, demonstrating significant reductions in dormant neurons across various tasks and network architectures.

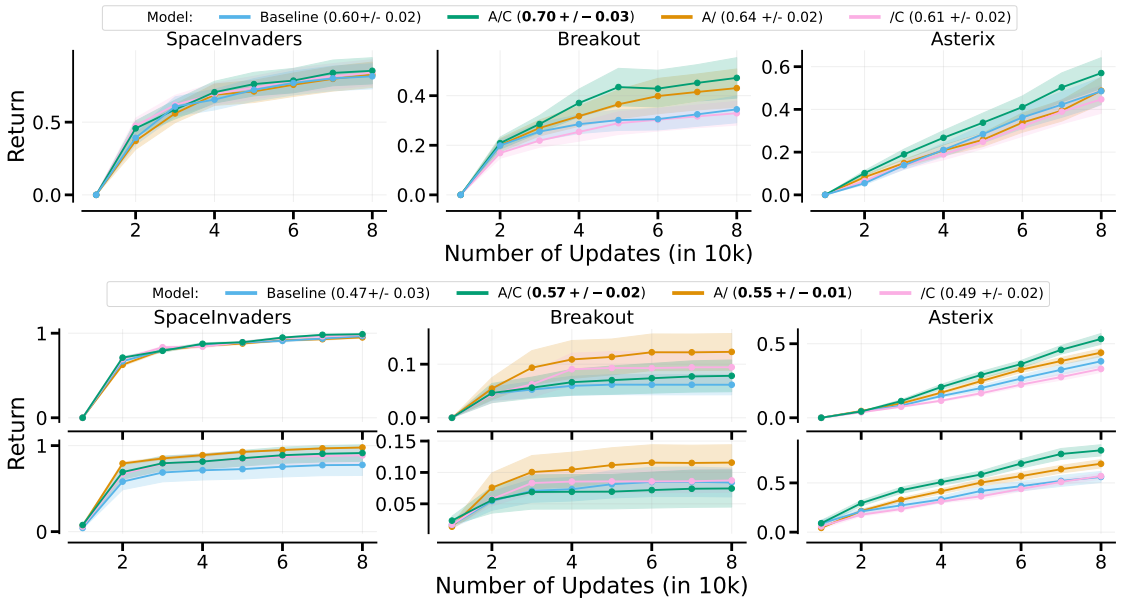


Figure 6: **Comparing the impact of MoE architectures on actor and critic networks** with the hardcoded router, under the MTRL (top row) and CRL (bottom row) settings. In each legend, the numbers in parentheses indicate the average performance of each approach across all games. It is best to use MoEs on both networks. However, the results suggest that MoEs have a greater impact when used on the actor network than on the critic. See section 4 for more details.

Mixtures of Experts (MoEs) revolutionised large-scale language/vision models primarily due to their modular design, which supports distributed training and enhances parameter efficiency during inference (Lepikhin et al., 2020; Fedus et al., 2022; Yang et al., 2019; Wang et al., 2020; Abbas & Andreopoulos, 2020; Pavlitskaya et al., 2020). MoEs show benefits in transfer and multi-task learning scenarios, e.g., by assigning experts to specific sub-problems (Puigcerver et al., 2023; Chen et al., 2023; Ye & Xu, 2023), or by improving the statistical performance of routers (Hazimeh et al., 2021).

MoEs have been studied in DRL (Ren et al., 2021; Hendawy et al., 2024; Akrour et al., 2021) but based on a previous definition of MoE (Jacobs et al., 1991), closely related to ensembling, and not the more recent interpretation of MoEs in LLMs. Ensembles are often used to represent the policy (Anschel et al., 2016; Lan et al., 2020; Agarwal et al., 2020; Peer et al., 2021; Chen et al., 2021; An et al., 2021; Wu et al., 2021; Liang et al., 2022) or to predict model dynamics (Shyam et al., 2019; Chua et al., 2018; Kurutach et al., 2018). Most closely related to ensembling is our Big architecture, where each expert is a full model. Fan et al. (2023) could be interpreted as using multiple meta-controllers as routers for Big and ensembling the resulting policy. In contrast to our work, they do not investigate different MoE architectures and rely on population-based training.

Two recent works have explored using MoEs (as used in LLMs) in DRL: the work of Ceron et al. (2024b) has already been referenced extensively above, as our work builds on their findings. More recently, Farebrother et al. (2024) argued that classification losses yield stabler learning dynamics than regression losses, which also applies to using MoEs.

6 Conclusion

Our work provides additional evidence of the effectiveness of MoEs in improving the training of DRL agents. Using MTRL and CRL grants us a novel perspective on evaluating and analysing MoEs under "extreme" non-stationarity. Consistent with the findings of Ceron et al. (2024b), DRL is most performant using SoftMoE, whereas it struggles with hard TopK routing.

Our use of the hardcoded router served as a useful baseline for our analyses and demonstrates much room for improvement in training DRL agents in multi-task settings. Indeed, in CRL, only mild policy retention was observed in Ast, and the retention amount was dependent on the order in which the environments were trained. An exciting avenue for future work is thus investigating what *task curricula* would lead to best agent performance and policy retention. As mentioned previously, the observations in Ast differ substantially from those of BO and SI (which are similar to each other); the fact that we only observed policy retention in Ast thus begs the question of whether the agent is over-fitting to the anomalous input distribution of Ast, at the expense of being able to generalise to the other environments.

Expert specialisation and whether load-balancing is desirable are also interesting questions for future research. The findings from the supervised learning community in this respect may not naturally carry over to DRL settings, largely due to training's inherent non-stationarity. Finally, MoEs could be investigated in multi-agent settings, where experts represent different agents in cooperative (Ellis et al., 2024) or general-sum settings (Lu et al., 2022b; Willi et al., 2022), where vectorised environments are widely available (Khan et al., 2023; Rutherford et al., 2023).

Acknowledgements

The authors would like to thank Gheorghe Comanici, Gopeshh Subbaraj, Doina Precup, Hugo Larochelle, and the rest of the Google DeepMind Montreal team for valuable discussions during the preparation of this work. Gheorghe Comanici deserves a special mention for providing us valuable feed-back on an early draft of the paper. We thank the anonymous reviewers for their valuable help in improving our manuscript. We would also like to thank the Python community Van Rossum & Drake Jr (1995); Oliphant (2007) for developing tools that enabled this work, including NumPy Harris et al. (2020), Matplotlib Hunter (2007), Jupyter Kluyver et al. (2016), Pandas McKinney (2013) and JAX Bradbury et al. (2018).

Broader Impact Statement

This paper introduces research aimed at pushing the boundaries of Machine Learning, with a particular focus on reinforcement learning. Our work holds various potential societal implications, although we refrain from singling out specific ones for emphasis in this context.

References

Alhabib Abbas and Yiannis Andreopoulos. Biased mixtures of experts: Enabling computer vision inference under data transfer limitations. *IEEE Transactions on Image Processing*, 29:7656–7667, 2020.

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. *arXiv preprint arXiv:2303.07507*, 2023.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 104–114. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/agarwal20c.html.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Riad Akrour, Davide Tateo, and Jan Peters. Continuous action reinforcement learning from a mixture of interpretable experts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6795–6806, 2021.

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

Oron Anschel, Nir Baram, and Nahum Shimkin. Deep reinforcement learning with averaged target dqn. *CoRR abs/1611.01929*, 2016.

Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *arXiv preprint arXiv:2112.15579*, 2021.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyun Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77 – 82, 2020.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

Johan Samir Obando Ceron, Aaron Courville, and Pablo Samuel Castro. In value-based deep reinforcement learning, a pruned network is a good network. In *Forty-first International Conference on Machine Learning*. PMLR, 2024a. URL https://openreview.net/forum?id=seo9V9QRZp.

Johan Samir Obando Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=X9VMhfFxwn.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

Zitian Chen, Yikang Shen, Mingyu Ding, Zhenfang Chen, Hengshuang Zhao, Erik G Learned-Miller, and Chuang Gan. Mod-squad: Designing mixtures of experts as modular multi-task learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11828–11837, 2023.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/evci20a.html.

Jiajun Fan, Yuzheng Zhuang, Yuecheng Liu, Jianye Hao, Bin Wang, Jiangcheng Zhu, Hao Wang, and Shu-Tao Xia. Learnable behavior control: Breaking atari human world records via sample-efficient behavior selection. *arXiv preprint arXiv:2305.05239*, 2023.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep rl. In *Forty-first International Conference on Machine Learning*. PMLR, 2024.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1): 5232–5270, 2022.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7766–7792. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/graesser22a.html.

Shashank Gupta, Subhabrata Mukherjee, Krishan Subudhi, Eduardo Gonzalez, Damien Jose, Ahmed H Awadallah, and Jianfeng Gao. Sparsely activated mixture-of-experts are robust multi-task learners. *arXiv preprint arXiv:2204.07689*, 2022.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

Ahmed Hendawy, Jan Peters, and Carlo D'Eramo. Multi-task reinforcement learning with mixture of orthogonal experts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aZH1dM3GOX.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9 (03):90–95, 2007.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Andrew Jesson, Chris Lu, Gunshi Gupta, Angelos Filos, Jakob Nicolaus Foerster, and Yarin Gal. Relu to the rescue: Improve your on-policy actor-critic with positive advantages. *arXiv preprint arXiv:2306.01460*, 2023.

Tian Jin, Michael Carbin, Dan Roy, Jonathan Frankle, and Gintare Karolina Dziugaite. Pruning's effect on generalization through the lens of training and regularization. *Advances in Neural Information Processing Systems*, 35:37947–37961, 2022.

Akbir Khan, Timon Willi, Newton Kwan, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Scaling opponent shaping to high dimensional games. *arXiv preprint arXiv:2312.12568*, 2023.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

Thomas Kluyver, Benjain Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *IOS Press*, pp. 87–90. 2016. doi: 10.3233/978-1-61499-649-1-87.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=O9bnihsFfXU.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2020.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:232428341.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*, pp. 13285–13301. PMLR, 2022.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, 2022a.

Chris Lu, Timon Willi, Alistair Letcher, and Jakob Nicolaus Foerster. Adversarial cheap talk. In *International Conference on Machine Learning*, pp. 22917–22941. PMLR, 2023.

Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. Model-free opponent shaping. In *International Conference on Machine Learning*, pp. 14398–14411. PMLR, 2022b.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZkC8wKoLbQ7.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Ávila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23190–23211. PMLR, 2023. URL https://proceedings.mlr.press/v202/lyle23b.html.

Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 1 edition, February 2013. ISBN 9789351100065. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1449319793.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 17–23 Jul 2022a. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022b.

Johan Obando Ceron, Marc Bellemare, and Pablo Samuel Castro. Small batch deep reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 26003–26024. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/528388f1ad3a481249a97cbb698d2fe6-Paper-Conference.pdf.

Johan Samir Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1373–1383. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/ceron21a.html.

Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3): 10–20, 2007. doi: 10.1109/MCSE.2007.58.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=nPHA8fGicZk.

Svetlana Pavlitskaya, Christian Hubschneider, Michael Weber, Ruby Moritz, Fabian Huger, Peter Schlicht, and Marius Zollner. Using mixture of expert models to gain insights into semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 342–343, 2020.

Oren Peer, Chen Tessler, Nadav Merlis, and Ron Meir. Ensemble bootstrapping for q-learning. In *International Conference on Machine Learning*, pp. 8454–8463. PMLR, 2021.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts, 2023.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Jie Ren, Yewen Li, Zihan Ding, Wei Pan, and Hao Dong. Probabilistic mixture-of-experts for efficient deep reinforcement learning. *arXiv preprint arXiv:2104.09122*, 2021.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

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

Max Schwarzer, Johan Samir Obando-Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level Atari with human-level efficiency. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 30365–30380. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/schwarzer23a.html.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pp. 5779–5788. PMLR, 2019.

Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32145–32168. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/sokar23a.html.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. Rlx2: Training a sparse deep reinforcement learning model from scratch. In *The Eleventh International Conference on Learning Representations*, 2022.

Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Xin Wang, Fisher Yu, Lisa Dunlap, Yi-An Ma, Ruth Wang, Azalia Mirhoseini, Trevor Darrell, and Joseph E Gonzalez. Deep mixture of experts via shallow embedding. In *Uncertainty in artificial intelligence*, pp. 552–562. PMLR, 2020.

Timon Willi, Alistair Hp Letcher, Johannes Treutlein, and Jakob Foerster. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pp. 23804–23831. PMLR, 2022.

Yanqiu Wu, Xinyue Chen, Che Wang, Yiming Zhang, and Keith W Ross. Aggressive q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance. *arXiv preprint arXiv:2111.09159*, 2021.

Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *Advances in neural information processing systems*, 32, 2019.

Hanrong Ye and Dan Xu. Taskexpert: Dynamically assembling multi-task representations with memorial mixture-of-experts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 21828–21837, 2023.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models, 2022.

7.1 Limitations and Future Work

This chapter explores the behaviour of MoE architectures within deep RL agents under the amplified non-stationarity presented by MTRL and CRL settings. We provide evidence that MoEs, particularly configurations involving large experts ("Big") or applying MoEs throughout the network ("All"), can enhance learning capacity compared to baseline models, especially in the MTRL context. The choice of routing strategy and MoE placement affects performance with the learning setting (MTRL vs. CRL), highlighting the importance of the specific type of non-stationarity when designing MoE-based agents. Our analyses also reinforce that MoEs help maintain network plasticity by reducing dormant neurons.

Despite these insights, this work possesses limitations that naturally lead to future research directions. Firstly, the experiments were conducted on a limited set of MinAtar environments. While useful for controlled comparisons, evaluating these architectures on more diverse MTRL and CRL benchmarks encompassing different state/action spaces (e.g., continuous control) and task relationships (e.g., tasks with varying degrees of overlap or interference) is needed to confirm the generality of our findings.

Secondly, the MTRL and CRL protocols were specific (synchronous updates across all tasks in MTRL, a fixed sequential order in CRL). Investigating MoE performance under other paradigms, such as asynchronous MTRL, different CRL task ordering schemes, or settings with unknown task boundaries, would provide a more comprehensive understanding of their adaptability.

Thirdly, while we compared different MoE placements and basic routing mechanisms, the exploration of architectural variants was not exhaustive. Further research could investigate hierarchical MoEs, experts of varying sizes, more sophisticated learned routers (potentially incorporating meta-learning or attention mechanisms), or dynamic expansion/contraction of the expert pool based on task demands. From the internal MARL perspective, the routers tested remain simple coordinators; exploring routers with more explicit coordination capabilities derived from MARL algorithms is an intriguing possibility.

Fourthly, although we observed differences in how architectures performed across MTRL and CRL (e.g., hardcoded routing excelling in CRL, Soft MoE in MTRL), the underlying reasons for these differences are not fully explained.

Finally, this work maintained a focus on intra-agent benefits. A crucial next step is to bridge this back to the inter-agent challenges discussed in Part I. Future work should investigate deploying agents equipped with these scalable and robust MoE architectures within actual multi-agent environments (e.g., using JaxMARL, Chapter 5) and potentially combine them with OS techniques (Chapter 3 & 4). This would allow studying whether improved internal capacity and adaptation translate to better performance in complex MARL tasks and how internal expert specialisation might interact with external factors like co-player strategies or emergent team roles. Combining the opponent shaping techniques from Part I with the MoE architectures from Part II within a single agent represents a particularly promising direction for future research into truly scalable and adaptable MARL agents.

8 Conclusion

The challenge of creating intelligent agents capable of effective learning and interaction in complex, often multi-agent, settings requires progress on multiple fronts. Standard reinforcement learning algorithms struggle with the non-stationarity inherent in MARL, and naive approaches fail to achieve cooperation. Deep learning approaches face challenges in efficiently scaling network capacity without sacrificing performance or stability. This thesis presents algorithmic and architectural advances in developing more scalable, robust, and cooperative learning agents.

8.1 Summary of Contributions

This thesis is divided into two main parts. Both parts tackle key challenges in developing scalable and effective learning agents for complex interactive settings from inter-agent and intra-agent perspectives.

Part I focuses on improving inter-agent interaction dynamics through advancing Opponent Shaping, particularly in general-sum scenarios. Recognising the limitations of prior work, Chapter 3 first addresses the issue of theoretical inconsistency by introducing Consistent LOLA (COLA). This work provides a formal definition of consistency, clarifies misconceptions regarding methods like CGD, and presents an algorithm capable of learning theoretically sound, consistent

update rules, demonstrating improved cooperation in benchmark matrix games. Building on this, Chapter 4 confronts the challenge of scaling OS beyond these simple settings. It introduces Shaper, a memory-based meta-learning algorithm for high-dimensional, partially observable environments with temporally-extended actions. This chapter demonstrates the first successful application of OS in complex grid-world social dilemmas by analysing the roles of history and context and formalising batch averaging, broadening the applicability of shaping techniques. Supporting these algorithmic advancements, Chapter 5 presents JaxMARL, a high-performance, JAX-based library designed to accelerate MARL research. By efficiently implementing diverse environments and baseline algorithms, JaxMARL is a toolkit for the empirical evaluation required for developing and testing sophisticated methods like COLA and Shaper.

Part II shifts the focus to intra-agent complexity and the fundamental problem of parameter scaling in deep RL. Chapter 6 provides strong empirical evidence that MoEs, particularly Soft MoE, unlock effective parameter scaling in value-based deep RL. We demonstrate that performance scales positively with the number of experts, overcoming the stagnation observed in standard architectures, and link this to improved parameter utilisation and network stability. Chapter 7 further probes the adaptability of MoEs by evaluating various architectural configurations under the amplified non-stationarity of Multi-Task and Continual RL. The results provide insights into how different MoE designs handle task switching and concurrent learning, highlighting the capacity for internal specialisation and validating the potential of MoEs for building adaptable deep RL agents.

8.2 Overall Significance & Limitations

The field of artificial intelligence is undergoing a paradigm shift, driven by foundation models, particularly Large Language Models (LLMs), which requires a re-evaluation of established work, including that explored in this thesis. Much of the work in RL and MARL, including the OS methods developed here, has traditionally operated under a *tabula rasa* assumption, i.e., agents learning complex behaviours

through environmental interaction from a blank slate. In contrast, LLMs are increasingly used as policies directly in RL environments, endowed with vast pre-trained knowledge and emergent capabilities for reasoning, planning, and even theory of mind (Paglieri et al., 2024; Hu et al., 2024; Lupu et al., 2025).

How do the contributions of this thesis resonate in this new era?

1. **The Enduring Relevance of Strategic Interaction (Opponent Shaping):** While the *mechanisms* of learning and adaptation in LLM-based agents may differ from traditional RL agents (e.g., relying more on in-context learning or targeted fine-tuning (von Oswald et al., 2023; Rafailov et al., 2023; Shao et al., 2024)), the fundamental challenge of strategic interaction persists (Wu et al., 2025). When multiple LLM-based agents interact, they still form a dynamic system that influences others' behaviour and learning dynamics. The core idea of OS remains crucial for achieving desired individual or collective outcomes. The *principles* of anticipating and shaping opponent adaptation, explored through COLA's consistency and Shaper's scalability, are likely to be essential, albeit potentially manifested through different means, such as strategic prompting, meta-learning adaptation strategies, or influencing the fine-tuning process of co-players. The need for robust, theoretically grounded methods to manage these high-stakes interactions, as pursued in Part I, will only grow.

2. **Architectural Insights for Scalable Agency (Mixture of Experts):** The success of MoEs in scaling LLMs to trillions of parameters finds a direct echo in the findings of Part II. This thesis provides critical *empirical validation* that MoE architectures, particularly Soft MoE, can overcome parameter scaling bottlenecks *specifically within the domain of deep RL*, linking improved performance to better parameter utilisation and network stability. Whether future agents are sophisticated RL systems trained end-to-end, LLMs fine-tuned for specific tasks or hybrid versions, they will require architectures capable of managing increased complexity. MoEs offer a proven

path towards modularity, conditional computation, and internal specialisation. The conceptualisation of MoEs as an internal multi-agent system, with experts coordinated by a router, offers a useful lens for designing and understanding these complex internal dynamics, bridging the gap between inter-agent and intra-agent coordination challenges (Zhuge et al., 2024).

Future Trajectories:

The most exciting frontiers lie at the synthesis of these ideas. Can we design agents that leverage the reasoning and world knowledge of LLMs while retaining the adaptive, optimising capabilities of RL? How can OS be adapted to shape the behaviour of pre-trained models like LLMs interacting with each other or with humans? Can the architectural lessons from MoEs in LLMs and RL lead to scalable agents capable of sophisticated individual reasoning *and* complex multi-agent strategy?

Developing agents that can effectively cooperate, compete, and coordinate in open-ended, dynamic multi-agent systems remains a grand challenge for AI. While the rise of LLMs introduces new possibilities and complexities, the foundational issues of learning, adaptation, strategic influence, and scalable architecture addressed in this thesis remain central. The work presented here on Opponent Shaping and Mixture-of-Experts provides essential building blocks and insights, paving the way for the next generation of intelligent interactive systems.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

David Abel, André Barreto, Michael Bowling, Will Dabney, Shi Dong, Steven Hansen, Anna Harutyunyan, Khimya Khetarpal, Clare Lyle, Razvan Pascanu, et al. Agency is frame-dependent. *arXiv preprint arXiv:2502.04403*, 2025.

Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

Jan Balaguer, Raphael Koster, Christopher Summerfield, and Andrea Tacchetti. The good shepherd: An oracle agent for mechanism design. arXiv preprint arXiv:2202.10135, 2022.

Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyun Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77 – 82, 2020.

Richard Bellman. On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, 38(8):716–719, 1952.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

Noam Brown and Tuomas Sandholm. Libratus: the superhuman ai for no-limit poker. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

Johan Samir Obando Ceron, Marc G Bellemare, and Pablo Samuel Castro. Small batch deep reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=wPqEvmwFEh`.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017. URL `https://arxiv.org/abs/1711.03938`.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts, 2022.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

Jesse Farebrother, Joshua Greaves, Rishabh Agarwal, Charline Le Lan, Ross Goroshin, Pablo Samuel Castro, and Marc G Bellemare. Proto-value networks: Scaling representation learning with auxiliary tasks. In *The Eleventh International Conference on Learning Representations*, 2022.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning, 2022. URL `https://arxiv.org/abs/2209.01667`.

Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, 2018.

Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951. PMLR, 2019.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

Rihab Gorsane, Omayma Mahjoub, Ruan de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *arXiv preprint arXiv:2209.10485*, 2022.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7766–7792. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/graesser22a.html`.

Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artif. Intell. Rev.*, 2022.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL `https://arxiv.org/abs/1801.01290`.

Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

Sihao Hu, Tiansheng Huang, and Ling Liu. Pokéllmon: A human-parity agent for pokémon battles with large language models. *arXiv preprint arXiv:2402.01118*, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019. doi: 10.1126/science.aau6249.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020a.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020b. URL `https://arxiv.org/abs/2001.08361`.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more persuasive llms leads to more truthful answers, 2024a. URL https://arxiv.org/abs/2402.06782.

Akbir Khan, Timon Willi, Newton Kwan, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Scaling opponent shaping to high dimensional games. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, page 1001–1010, 2024b.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=O9bnihsFfXU.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Joel Z. Leibo, Edgar Dué nez Guzmán, Alexander Sasha Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charles Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. PMLR, 2021.

Adam Lerer and Alexander Peysakhovich. Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *CoRR*, abs/1707.01068, 2017.

Alistair Letcher, David Balduzzi, Sébastien Racanière, James Martens, Jakob N. Foerster, Karl Tuyls, and Thore Graepel. Differentiable game mechanics. *J. Mach. Learn. Res.*, 20:84:1–84:40, 2019a.

Alistair Letcher, Jakob N. Foerster, David Balduzzi, Tim Rocktäschel, and Shimon Whiteson. Stable opponent shaping in differentiable games. In *7th International Conference on Learning Representations*, 2019b.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

Chris Lu, Jakub Grudzien Kuba, Alistair Letcher, Luke Metz, Christian Schröder de Witt, and Jakob N. Foerster. Discovered policy optimisation. *CoRR*, abs/2210.05639, 2022a. doi: 10.48550/arXiv.2210.05639. URL https://doi.org/10.48550/arXiv.2210.05639.

Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. Model-free opponent shaping. In *International Conference on Machine Learning*, pages 14398–14411. PMLR, 2022b.

Andrei Lupu, Timon Willi, and Jakob Nicolaus Foerster. The decrypto benchmark for multi-agent reasoning and theory of mind. 2025.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZkC8wKoLbQ7.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado P van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 106440–106473, 2024.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. URL https://arxiv.org/abs/2108.10470.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013a.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015a. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015b.

John F Nash. Non-cooperative games. In *The Foundations of Price Theory Vol 4*, pages 329–340. Routledge, 1951.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR, 17–23 Jul 2022a. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR, 17–23 Jul 2022b. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep rl. In *Proceedings of the 41st International Conference on Machine Learning*, pages 38520–38540, 2024.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=nPHA8fGicZk.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024.

A Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktaschel, Chris Lu, and Jakob Nicolaus Foerster. JaxMARL: Multi-Agent RL Environments in JAX. *Being Submitted to Neurips Datasets and Benchmarks*, 2023. URL `https://arxiv.org/abs/2311.10090`.

Florian Schäfer and Anima Anandkumar. Competitive gradient descent. In *Advances in Neural Information Processing Systems*, volume 32, pages 7623–7633, 2019.

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

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level Atari with human-level efficiency. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 30365–30380, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

L. S. Shapley. Stochastic Games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. ISSN 0027-8424.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017a. URL `https://openreview.net/forum?id=B1ckMDqlg`.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017b. URL https://openreview.net/forum?id=B1ckMDqlg.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023a.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32145–32168. PMLR, 23–29 Jul 2023b. URL https://proceedings.mlr.press/v202/sokar23a.html.

Alexandra Souly, Timon Willi, Akbir Khan, Robert Kirk, Chris Lu, Edward Grefenstette, and Tim Rocktäschel. Leading the pack: N-player opponent shaping. In *Multi-Agent Security Workshop @ NeurIPS'23*, 2023. URL https://openreview.net/forum?id=3b8hfpqtlM.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

J. R. R. Tolkien. *The Fellowship of the Ring*. George Allen & Unwin, London, 1954.

J. R. R. Tolkien. *The Return of the King*. George Allen & Unwin, London, 1955.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019a.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019b.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent, 2023. URL `https://arxiv.org/abs/2212.07677`.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8: 279–292, 1992.

Timon Willi, Alistair Letcher, Johannes Treutlein, and Jakob N. Foerster. COLA: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23804–23831, 2022.

Timon Willi, Johan Samir Obando Ceron, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, and Pablo Samuel Castro. Mixture of experts in a mixture of rl settings. In *Reinforcement Learning Conference*, 2024.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Chen Henry Wu, Rishi Rajesh Shah, Jing Yu Koh, Russ Salakhutdinov, Daniel Fried, and Aditi Raghunathan. Dissecting adversarial robustness of multimodal lm agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

Stephen Zhao, Chris Lu, Roger Baker Grosse, and Jakob Nicolaus Foerster. Proximal learning with opponent-learning awareness. *arXiv preprint arXiv:2210.10125*, 2022.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

Appendices

A

Consistent Learning with Opponent-Learning Awareness - Appendices

A. Nonconvergence of HOLA in the Tandem Game

In the following, we show that for the choice of look-ahead rate $\alpha = 1$, HOLA does not converge in the Tandem game. This shows that given a large enough look-ahead rate, even in a simple quadratic game, HOLA need not converge.

Proposition A.1. *Let L^1, L^2 be the two players' loss functions in the Tandem game as defined in Section 5:*

$$L^1(x, y) = (x + y)^2 - 2x \quad \textit{and} \quad L^2(x, y) = (x + y)^2 - 2y, \tag{3}$$

and let h_i^n denote the n-th order exact LOLA update for player i (where $n = 0$ denotes naive learning). Consider the look-ahead rate $\alpha := 1$. Then the functions $(h_i^n)_{n\in\mathbb{N}}$ for $i = 1, 2$ do not converge pointwise.

Proof. We will prove the auxiliary statement that

$$h_i^n(x, y) = 2^{n+2} - 2(1 + x + y)$$

for $i = 1, 2$. It then follows trivially that the h_i^n cannot converge.

The auxiliary result can be proven by induction. The base case $n = 0$ follows from

$$\nabla_i L^i(x, y) = 2 - 2(x + y) = 2^2 - 2(1 + x + y)$$

for $i = 1, 2$. Next, for the inductive step, we have to show that

$$h_1^n(x, y) = -\nabla_1(L^1(x, y + h_2^{n-1}(x, y))) \tag{4}$$
$$h_2^n(x, y) = -\nabla_2(L^2(x + h_1^{n-1}(x, y), y)) \tag{5}$$

for any $n > 0$. Substituting the inductive hypothesis in the second step, we have

$$\begin{aligned}
&- \nabla_1(L^1(x, y + h_2^{n-1}(x, y))) && (6)\\
&= -\nabla_1(L^1(x, y + 2^{n+1} - 2(1 + x + y))) && (7)\\
&= -\nabla_1\left((x + y + 2^{n+1} - 2 - 2x - 2y)^2 - 2x\right) && (8)\\
&= -\nabla_1\left((-x - y + 2^{n+1} - 2)^2 - 2x\right) && (9)\\
&= 2(-x - y + 2^{n+1} - 2) + 2 && (10)\\
&= 2^{n+2} - 2(1 + x + y) && (11)\\
&= h_1^n(x, y). && (12)
\end{aligned}$$

The derivation for $h_2^n(x, y)$ is exactly analogous. This shows the inductive step and thus finishes the proof.

□

B. Proof of Proposition 4.3

To begin, recall that some differentiable game with continuously differentiable loss functions L^1, L^2 is given, and that $h^n = (h_1^n, h_2^n)$ denotes the n-th order exact LOLA update function. We assume that the iLOLA update function h exists, defined via

$$h_i(\theta) := \lim_{n\to\infty} h_i^n(\theta),$$

for all $\theta \in \mathbb{R}^d$.

To prove Proposition 4.3, we need to show that h_1, h_2 are consistent, i.e., satisfy Definition 4.2, under the assumption that

$$\lim_{n\to\infty} \nabla_i h_{-i}^n(\theta) = \nabla_i h_{-i}(\theta)$$

for $i = 1, 2$ and any θ.

To that end, define the *(exact) LOLA operator* Ψ as the function mapping a pair of update functions $f := (f_1, f_2)$ to the RHS of Equations 1 and 2,

$$\Psi_1(f)(\theta) := -\alpha\nabla_1(L^1(\theta_1, \theta_2 + f_2(\theta_1, \theta_2))) \tag{13}$$
$$\Psi_2(f)(\theta) := -\alpha\nabla_2(L^2(\theta_1 + f_1(\theta_1, \theta_2), \theta_2)) \tag{14}$$

for any θ. Note that then we have $h_i^{n+1} = \Psi_i(h^n)$, i.e., Ψ maps n-th order LOLA to $n+1$-order LOLA.

In the following, we show that iLOLA is a fixed point of the LOLA operator, i.e., $\Psi(h) = h$. It follows from the definition of Ψ that then h is consistent. We denote by $\|\cdot\|$ the Euclidean norm or the induced operator norm for matrices. We focus on showing $\Psi_1(h) = h_1$. The case $i = 2$ is exactly analogous.

For arbitrary θ and n, define $\hat{\theta}_2 := \theta_2 + h_2(\theta)$ and $\hat{\theta}_2^n := \theta_2 + h_n^2(\theta)$ as the updated parameter of player 2. First, it is helpful to show that $\Psi_1(h^n)(\theta)$ converges to $\Psi_1(h)(\theta)$:

$$0 \leq \|\Psi_1(h)(\theta) - \Psi_1(h^n)(\theta)\| \tag{15}$$
$$= \alpha\|\nabla_1(L^1(\theta_1, \theta_2 + h_2(\theta))) - \nabla_1(L^1(\theta_1, \theta_2 + h_2^n(\theta))\| \tag{16}$$
$$= \alpha\|(\nabla_1 h_2(\theta))^\top \nabla_2 L^1(\theta_1, \hat{\theta}_2) - (\nabla_1 h_2^n(\theta))^\top \nabla_2 L^1(\theta_1, \hat{\theta}_2^n) + \nabla_1 L^1(\theta_1, \hat{\theta}_2) - \nabla_1 L^1(\theta_1, \hat{\theta}_2^n)\| \tag{17}$$
$$\leq \alpha\|(\nabla_1 h_2(\theta))^\top \nabla_2 L^1(\theta_1, \hat{\theta}_2) - (\nabla_1 h_2^n(\theta))^\top \nabla_2 L^1(\theta_1, \hat{\theta}_2^n)\| + \alpha\|\nabla_1 L^1(\theta_1, \hat{\theta}_2) - \nabla_1 L^1(\theta_1, \hat{\theta}_2^n)\| \tag{18}$$
$$= \alpha\|(\nabla_1 h_2(\theta))^\top (\nabla_2 L^1(\theta_1, \hat{\theta}_2) - \nabla_2 L^1(\theta_1, \hat{\theta}_2^n)) \tag{19}$$
$$+ (\nabla_1 h_2(\theta) - \nabla_1 h_2^n(\theta))^\top \nabla_2 L^1(\theta_1, \hat{\theta}_2^n)\| + \alpha\|\nabla_1 L^1(\theta_1, \hat{\theta}_2) - \nabla_1 L^1(\theta_1, \hat{\theta}_2^n)\|$$
$$\leq \alpha\|(\nabla_1 h_2(\theta))^\top\|\|\nabla_2 L^1(\theta_1, \hat{\theta}_2) - \nabla_2 L^1(\theta_1, \hat{\theta}_2^n)\|$$
$$+ \alpha\|(\nabla_1 h_2(\theta) - \nabla_1 h_2^n(\theta))^\top\|\|\nabla_2 L^1(\theta_1, \hat{\theta}_2^n)\| + \alpha\|\nabla_1 L^1(\theta_1, \hat{\theta}_2) - \nabla_1 L^1(\theta_1, \hat{\theta}_2^n)\| \tag{20}$$
$$\overset{n\to\infty}{\longrightarrow} 0. \tag{21}$$

In the last step, we used the following two facts. First, since $\nabla_i L^1(\theta)$ is assumed to be continuous in θ_2, and $\lim_{n\to\infty} \hat{\theta}_2^n = \theta_2 + \lim_{n\to\infty} h_2^n(\theta) = \theta_2 + h_2(\theta) = \hat{\theta}_2$ by assumption, it follows that $\lim_{n\to\infty} \nabla_2 L^1(\theta_1, \hat{\theta}_2^n) = \nabla_2 L^1(\theta_1, \hat{\theta}_2)$ and $\lim_{n\to\infty} \nabla_1 L^1(\theta_1, \hat{\theta}_2^n) = \nabla_1 L^1(\theta_1, \hat{\theta}_2)$. Second, by assumption, $\lim_{n\to\infty} \nabla h_2^n(\theta) = \nabla h_2(\theta)$. In particular, $\|\nabla_2 L^1(\theta_1, \hat{\theta}_2^n)\|$ must be bounded, and thus the three terms in (20) must all converge to 0 as $n \to \infty$. It follows by the sandwich theorem that $\lim_{n\to\infty} \Psi_1(h^n)(\theta) = \Psi_1(h)(\theta)$.

Now we can directly prove that $\Psi_1(h)(\theta) = h_1(\theta)$. It is

$$0 \leq \|\Psi_1(h)(\theta) - h_1(\theta)\| \tag{22}$$
$$= \|\Psi_1(h)(\theta) - \Psi_1(h^n)(\theta) + \Psi_1(h^n)(\theta) - h_1^n(\theta) + h_1^n(\theta) - h_1(\theta)\| \tag{23}$$
$$\leq \|\Psi_1(h)(\theta) - \Psi_1(h^n)(\theta)\| + \|\Psi_1(h^n)(\theta) - h_1^n(\theta)\| + \|h_1^n(\theta) - h_1(\theta)\| \tag{24}$$
$$= \|\Psi_1(h)(\theta) - \Psi_1(h^n)(\theta)\| + \|h_1^{n+1}(\theta) - h_1^n(\theta)\| + \|h_1^n(\theta) - h_1(\theta)\| \tag{25}$$
$$\overset{n\to\infty}{\longrightarrow} 0, \tag{26}$$

where in the last step we have used the above result, as well as the assumption that $h_1^n(\theta)$ converges pointwise, and thus must also be a Cauchy sequence, so the last and the middle term both converge to zero as well.

It follows by the sandwich theorem that $\Psi_1(h)(\theta) = h_1(\theta)$. Since θ was arbitrary, this concludes the proof. □

C. Infinite-order Taylor LOLA

In this Section, we repeat the analysis of iLOLA from Section 4.1 for infinite-order Taylor LOLA (Taylor iLOLA). I.e., we define Taylor consistency, and show that Taylor iLOLA satisfies this consistency equation under certain assumptions. This result will be needed for our proof of Proposition 4.4.

To begin, assume that some differentiable game with continuously differentiable loss functions L^1, L^2 is given. Define the Taylor LOLA operator Φ that maps pairs of update functions (f_1, f_2) to the associated Taylor LOLA update

$$\Phi_i(f) := -\alpha\nabla_i(L^i + (\nabla_{-i} L^i)^\top f_{-i}) \tag{27}$$

for $i = 1, 2$.

We then have the following definition.

Definition C.1 (Taylor consistency)**.** Two update functions f_1, f_2 are called Taylor consistent if for any $i = 1, 2$, we have

$$\Phi(f_1, f_2) = (f_1, f_2).$$

Next, let h_i^n denote i's n-th order Taylor LOLA update. I.e., $h_i^n := \Phi_i(h^{n-1})$ for $n \geq 0$, where we let $h_i^{-1} := 0$. Then we define

Definition C.2 (Taylor iLOLA)**.** If (h_1^n, h_2^n) converges pointwise as $n \to \infty$, define Taylor iLOLA as the limiting update

$$h := \lim_{n\to\infty} \begin{pmatrix} h_1^n \\ h_2^n. \end{pmatrix}$$

Finally, we provide a proof that Taylor iLOLA is Taylor consistent; i.e., we give a Taylor version of Proposition 4.3.

Proposition C.3. *Let h_i^n denote player i's n-th order Taylor LOLA update. Assume that $\lim_{n\to\infty} h_i^n(\theta) = h_i(\theta)$ and $\lim_{n\to\infty} \nabla_i h_{-i}^n(\theta) = \nabla_i h_{-i}(\theta)$ for all θ and $i \in \{1, 2\}$. Then Taylor iLOLA is Taylor consistent.*

Proof. The proof is exactly analogous to that of Proposition 4.3, but easier. We show $\Phi(h) = h$. It follows from the definition of Φ in Equation 27 that then h is Taylor consistent. We focus on showing $\Phi_1(h) = h_1$, and the case $i = 2$ is exactly analogous.

First, we show that $\Phi_1(h^n)(\theta)$ converges to $\Phi_1(h)(\theta)$ for all θ. Letting n be arbitrary and omitting θ in the following for clarity, it is

$$\begin{aligned}
0 &\leq \|\Phi_1(h) - \Phi_1(h^n)\| && (28)\\
&= \| - \alpha\nabla_1(L^1 + (\nabla_2 L^1)^\top h_2) + \alpha\nabla_1(L^1 + (\nabla_2 L^1)^\top h_2^n)\| && (29)\\
&= \alpha\| - \nabla_{12} L^1 h_2 - (\nabla_2 L^1)^\top (\nabla_1 h_2)^\top + \nabla_{12} L^1 h_2^n + (\nabla_2 L^1)^\top (\nabla_1 h_2^n)^\top\| && (30)\\
&\leq \alpha\|\nabla_{12} L^1 (h_2^n - h_2)\| + \alpha\|(\nabla_2 L^1)^\top (\nabla_1 h_2^n - \nabla_1 h_2)^\top\| && (31)\\
&\leq \alpha\|\nabla_{12} L^1\|\|h_2^n - h_2\| + \alpha\|\nabla_2 L^1\|\|\nabla_1 h_2^n - \nabla_1 h_2\| && (32)\\
&\overset{n\to\infty}{\longrightarrow} 0. && (33)
\end{aligned}$$

In the last step, we used the assumptions that $\lim_{n\to\infty} h_2^n = h_2$ and $\lim_{n\to\infty} \nabla h_2^n = \nabla h_2$. It follows by the sandwich theorem that $\lim_{n\to\infty} \Phi_1(h^n)(\theta) = \Phi_1(h)(\theta)$.

It follows from the above that $\Phi_1(h)(\theta) = h_1(\theta)$, using exactly the same argument as in Equations 22-26 with Φ instead of Ψ. Since θ was arbitrary, this concludes the proof.

□

D. Proof of Proposition 4.4

We begin by proving that LCGD does not coincide with Taylor LOLA and CGD neither coincides with exact nor Taylor iLOLA. It is sufficient to manifest a single counter-example: we consider the Tandem game given by $L^1 = (x + y)^2 - 2x$ and $L^2 = (x + y)^2 - 2y$ (using x, y instead of θ_1, θ_2 for simplicity). Throughout this proof we use the notation introduced by Balduzzi et al. (2018) and Letcher et al. (2019b) including the *simultaneous gradient*, the *off-diagonal Hessian* and the *shaping term* of the game as

$$\xi = \begin{pmatrix} \nabla_1 L^1 \\ \nabla_2 L^2 \end{pmatrix} \qquad \text{and} \qquad H_o = \begin{pmatrix} 0 & \nabla_{12} L^2 \\ \nabla_{21} L^2 & 0 \end{pmatrix} \qquad \text{and} \qquad \chi = \operatorname{diag}\left(H_o^T \nabla L\right)$$

respectively. Note that in two-player games, Taylor LOLA's shaping term reduces to

$$\chi = \begin{pmatrix} \nabla_{12} L^2 \nabla_2 L^1 \\ \nabla_{21} L^1 \nabla_1 L^2 \end{pmatrix}.$$

LCGD $\neq$ LOLA. Following Schäfer & Anandkumar (2019), LCGD is given by

$$\text{LCGD} = -\alpha \begin{pmatrix} \nabla_x f - \alpha D^2_{xy} f \nabla_y g \\ \nabla_y g - \alpha D^2_{yx} g \nabla_x f \end{pmatrix} = -\alpha \begin{pmatrix} I & -\alpha D^2_{xy} f \\ -\alpha D^2_{yx} g & I \end{pmatrix} \begin{pmatrix} \nabla_x f \\ \nabla_y g \end{pmatrix} = -\alpha (I - \alpha H_o)\xi$$

while Taylor LOLA is given (Letcher et al., 2019b) by

$$\text{LOLA} = -\alpha(I - \alpha H_o)\xi + \alpha^2 \chi \,.$$

Any game with $\chi \neq 0$ will yield a difference between LCGD and LOLA; in particular,

$$\chi = 4(x+y)\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

in the Tandem game implies that LCGD $\neq$ LOLA whenever parameters lie outside the measure-zero set $\{x + y = 0\} \subset \mathbb{R}^2$.

CGD does not recover HOLA. Since CGD is obtained through a bilinear approximation (Taylor expansion) of the loss functions, one would expect that the authors' claim of recovering HOLA is with regards to Taylor (not exact) HOLA. For completeness, and to avoid any doubts for the reader, we prove that CGD neither corresponds to exact nor Taylor HOLA.

Following Schäfer & Anandkumar (2019), the series-expansion of CGD is given by

$$\text{CGD}n = -\alpha \sum_{i=0}^{n} \begin{pmatrix} 0 & -\alpha D^2_{xy} f \\ -\alpha D^2_{yx} g & 0 \end{pmatrix}^i \begin{pmatrix} D_x f \\ D_y g \end{pmatrix} = -\alpha \sum_{i=0}^{n} (-\alpha H_o)^i \xi$$

and converges to CGD whenever $\alpha < 1/\|H_o\|$ (where $\|\cdot\|$ denotes the operator norm induced by the Euclidean norm on the space). Assume for contradiction that the series-expansion of CGD recovers HOLA, i.e. that CGDn = HOLAn for all n. In particular, we must have

$$\text{CGD} = \lim_{n\to\infty} \text{CGD}n = \lim_{n\to\infty} \text{HOLA}n = \text{iLOLA}$$

whenever $\alpha < 1/\|H_o\|$. In the tandem game, we have

$$H_o = 2\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

with $\|H_o\| = 2$, so CGD = iLOLA whenever $\alpha < 1/2$. Moreover, H_o being constant implies that

$$\nabla\text{HOLA}n = \nabla\text{CGD}n = -\alpha \sum_{i=0}^{n} (-\alpha H_o)^i \nabla\xi \,,$$

so gradients of HOLA also converge pointwise for all $\alpha < 1/2$. In particular, CGD = iLOLA must satisfy the (exact or Taylor) consistency equations by Proposition 4.3 or Proposition C.3. However, the update for CGD is given by

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = -\alpha(I + \alpha H_o)^{-1}\xi = -2\alpha(x+y-1)\begin{pmatrix} 1 & 2\alpha \\ 2\alpha & 1 \end{pmatrix}^{-1}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{-2\alpha(x+y-1)}{1+2\alpha}\begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

For the exact case, the RHS of the first consistency equation is

$$\begin{aligned} -\alpha\nabla_x\left((x+y+f_2)^2 - 2x\right) &= -2\alpha\left((1+\nabla_x f_2)(x+y+f_2) - 1\right) \\ &= \frac{-2\alpha}{1+2\alpha}\left(x+y+\frac{-2\alpha(x+y-1)}{1+2\alpha} - 1 - 2\alpha\right) \\ &= f_1 + \frac{4\alpha^2(x+y+2\alpha)}{(1+2\alpha)^2} \end{aligned}$$

which does not coincide with the LHS of the consistency equation ($= f_1$) whenever parameters lie outside the measure-zero set $\{x + y + 2\alpha = 0\} \subset \mathbb{R}^2$. Similarly for Taylor iLOLA, the RHS of the first consistency equation is

$$\begin{aligned}-\alpha\nabla_x\left((x+y)^2 - 2x + 2(x+y)f_2\right) &= -2\alpha\left(x + y - 1 + \frac{-2\alpha(x+y-1)}{1+2\alpha} + \frac{-2\alpha(x+y)}{1+2\alpha}\right) \\ &= f_1 + \frac{4\alpha^2(x+y)}{(1+2\alpha)^2}\end{aligned}$$

which does not coincide with the LHS of the consistency equation ($= f_1$) whenever parameters lie outside the measure-zero set $\{x + y = 0\} \subset \mathbb{R}^2$. This is a contradiction to consistency; we are done.

LCGD = LookAhead. We have already shown that LCGD is given by $-\alpha(I - \alpha H_o)\xi$ in the proof that LCGD $\neq$ LOLA. This coincides exactly with LookAhead following Letcher et al. (2019b).

CGD recovers higher-order LookAhead. The series expansion of CGD is given by

$$\text{CGD}n = -\alpha\sum_{i=0}^{n}(-\alpha H_o)^i\xi\,.$$

Also, higher-order (Taylor) LookAhead is defined recursively by expanding

$$\begin{aligned}f_1^{n+1} &= -\alpha\nabla_1\left(L^1(\theta^1, \theta^2 + \perp f_2^n)\right) \approx -\alpha\left(\nabla_1 L^1 + \nabla_{12}L^1 f_2^n\right) \\ f_2^{n+1} &= -\alpha\nabla_2\left(L^2(\theta^1 + \perp f_1^n, \theta^2)\right) \approx -\alpha\left(\nabla_2 L^2 + \nabla_{21}L^2 f_1^n\right)\,,\end{aligned}$$

where $\perp$ is the stop-gradient operator (see (Balduzzi et al., 2018) for details on this operator) and $f_1^{-1} = f_2^{-1} = 0$. This can be written more succinctly as

$$\begin{pmatrix}f_1^{n+1}\\ f_2^{n+1}\end{pmatrix} = -\alpha\begin{pmatrix}\nabla_1 L^1 + \nabla_{12}L^1 f_2^n\\ \nabla_2 L^2 + \nabla_{21}L^2 f_1^n\end{pmatrix} = -\alpha\xi - \alpha H_o\begin{pmatrix}f_1^n\\ f_2^n\end{pmatrix}\,.$$

We prove by induction that

$$\begin{pmatrix}f_1^n\\ f_2^n\end{pmatrix} = -\alpha\sum_{i=0}^{n}(-\alpha H_o)^i\xi$$

for all $n \geq 0$. The base case is trivial; assume the statement holds for any fixed $n \geq 0$. Then

$$\begin{pmatrix}f_1^{n+1}\\ f_2^{n+1}\end{pmatrix} = -\alpha\xi - \alpha H_o\left(-\alpha\sum_{i=0}^{n}(-\alpha H_o)^i\xi\right) = -\alpha\xi - \alpha\sum_{i=1}^{n+1}(-\alpha H_o)^i\xi = -\alpha\sum_{i=0}^{n+1}(-\alpha H_o)^i\xi$$

as required. Finally we conclude

$$\text{LookAhead}n = \begin{pmatrix}f_1^n\\ f_2^n\end{pmatrix} = -\alpha\sum_{i=0}^{n}(-\alpha H_o)^i\xi = \text{CGD}n$$

as required. □

E. Proof of Proposition 4.5

We prove that the two pairs of linear functions

$$f_1 = f_2 = -2(x + y + 1)$$

and

$$f_1 = f_2 = -\frac{1}{2}(x + y - 2)$$

are solutions to the consistency equations in the Tandem game with $\alpha = 1$. (See below for a generalization to any $\alpha > 0$.) For the first pair of functions, we have

$$-\nabla_x \left(L^1(x, y + f_2)\right) = -\nabla_x \left((x + y + 2)^2 - 2x\right) = -2(x + y + 1) = f_1$$

for the first consistency equation and similarly

$$-\nabla_y \left(L^2(x + f_1, y)\right) = -\nabla_x \left((x + y + 2)^2 - 2y\right) = -2(x + y + 1) = f_2$$

for the second. For the second pair of functions we similarly obtain

$$-\nabla_x \left(L^1(x, y + f_2)\right) = -\nabla_x \left(\frac{1}{4}(x + y + 2)^2 - 2x\right) = -\frac{1}{2}(x + y - 2) = f_1$$

for the first consistency equation and

$$-\nabla_y \left(L^2(x + f_1, y)\right) = -\nabla_x \left(\frac{1}{4}(x + y + 2)^2 - 2y\right) = -\frac{1}{2}(x + y - 2) = f_2$$

for the second. This shows that both functions are solutions to the consistency equations. For general $\alpha > 0$, we can similarly show that $f_1 = f_2 = ax + by + c$ with

$$a = \frac{\pm\sqrt{1 + 8\alpha} - 1 - 4\alpha}{4\alpha} \quad ; \quad b = \frac{-2\alpha(1 + a)}{1 + 2\alpha(1 + a)} \quad ; \quad c = \frac{2\alpha}{1 + 2\alpha(1 + a)}$$

are two distinct solutions (depending on $\pm$) to the consistency equations in the Tandem game, noting that the denominators cannot be 0 for $\alpha > 0$ (otherwise leading to a contradiction in the expression for a). This is left to the reader, noting that the proof for $\alpha = 1$ is sufficient to establish that consistent solutions are not always unique. □

F. Proof of Proposition 4.6

Recall from the proof of Proposition 4.5 that the linear functions

$$f_1 = f_2 = -2(x + y + 1)$$

are consistent solutions to the Tandem game with $\alpha = 1$. The SFPs of the Tandem game are $(x, 1 - x)$ for each $x \in \mathbb{R}$, but none of these are preserved by the consistent solutions above since

$$f_1(x, 1 - x) = f_2(x, 1 - x) = -4 \neq 0\,.$$

We conclude that consistency does not imply preservation of SFPs.

Moreover, we prove that any (non-zero) linear solution to the consistency equations cannot preserve more than one SFP in the Tandem game, for any opponent shaping rate α. Assuming it did, we must have linear functions

$$f_1 = ax + by + c$$
$$f_2 = a'x + b'y + c'$$

satisfying

$$f_1(x, 1 - x) = 0 = f_1(x', 1 - x')$$

for some $x \neq x' \in \mathbb{R}$. Subtracting RHS from LHS we obtain $(x - x')(a - b) = 0$ hence $a = b$, which substituted again into the LHS yields $b = -c$. Applying the same method for f_2 we obtain $a' = b' = -c'$ and so f_1, f_2 take the form

$$f_1 = a(x + y - 1)$$
$$f_2 = a'(x + y - 1).$$

Note that since f_1, f_2 were assumed to be nonzero, it follows that $a, a' \neq 0$. Plugging these into the first consistency equation, we obtain

$$a(x + y - 1) = -2\alpha\left[(1 + a')\left((x + y)(1 + a') - a'\right) - 1\right]\,.$$

Comparing x terms and constant terms yields

$$a = -2\alpha(1 + a')^2 \qquad \text{and} \qquad a = -2\alpha\left(1 + a' + a'^2\right)$$

which concludes the contradiction $a' = 0$. □

G. Proof of Proposition 4.7

LOLA and SOS diverge. Assume $(x_0, y_0) \neq 0$ and $\alpha > 1$. We prove the more general claim that p-LOLA diverges for any $0 \leq p \leq 1$ (where p may take a different value at each learning step), recalling that LOLA and SOS are both special cases of p-LOLA (Letcher et al., 2019b). Indeed, the p-LOLA gradient update is given by

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = -\alpha(I - \alpha H_o)\xi + p\alpha^2\chi = -\alpha \begin{pmatrix} y + \alpha x(1+p) \\ -x + \alpha y(1+p) \end{pmatrix}$$

and we show that each update leads to increasing distance from the origin as follows:

$$\begin{aligned} \|(x + h_1, y + h_2)\|^2 &= x^2 - 2x\alpha(y + \alpha x(1+p)) + \alpha^2\left(y^2 + \alpha^2 x^2(1+p)^2 + 2\alpha xy(1+p)\right) + \\ &\quad y^2 - 2y\alpha(-x + \alpha y(1+p)) + \alpha^2\left(x^2 + \alpha^2 y^2(1+p)^2 - 2\alpha xy(1+p)\right) \\ &= \left(x^2 + y^2\right)\left(1 - \alpha^2(2p+1) + \alpha^4(1+p)^2\right) \\ &\geq \left(x^2 + y^2\right)\left(1 - \alpha^2 + \alpha^4\right) \coloneqq \|(x,y)\|^2 \lambda \end{aligned}$$

where the inequality follows because the final expression in p has positive derivative for $\alpha > 1$, hence minimized at $p = 0$. Now $\lambda > 1$ for any $\alpha > 1$, so we conclude by induction that

$$\|(x_n, y_n)\|^2 \geq \lambda^n \|(x_0, y_0)\|^2 \to \infty$$

as $n \to \infty$, provided $(x_0, y_0) \neq 0$, as required.

Consistent solution converges. We begin by showing that the following linear functions satisfy the consistency equations for the Hamiltonian game:

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \frac{-\alpha}{1 + 2\alpha^2} \begin{pmatrix} y + 2\alpha x \\ -x + 2\alpha y \end{pmatrix}.$$

Indeed, the RHS of the first consistency equation is

$$\begin{aligned} -\alpha\nabla_x \left(x \left(y - \alpha\frac{-x + 2\alpha y}{1 + 2\alpha^2}\right)\right) &= \frac{-\alpha}{1 + 2\alpha^2}\Big(y(1 + 2\alpha^2) - \alpha(-x + 2\alpha y) + \alpha x\Big) \\ &= \frac{-\alpha}{1 + 2\alpha^2}\Big(y + 2\alpha x\Big) = f_1 \end{aligned}$$

and similarly for the second equation.

To prove uniqueness, assume there is a second pair of linear update functions $\hat{f}_1, \hat{f}_2$ also satisfying consistency. Let $a, b, c \in \mathbb{R}$ such that $\hat{f}_1(x, y) = ax + by + c$. Note that substituting the second equation into the first yields

$$\begin{aligned} \hat{f}_1(x,y) &= -\alpha\nabla_x \left(L^1(x, y - \alpha\nabla_y \left(L^2(x + \hat{f}_1(x,y), y)\right))\right) \\ &= -\alpha\left(y + \alpha\left(2x + \hat{f}_1(x,y) + x\nabla_x\hat{f}_1(x,y) + y\nabla_y\hat{f}_1(x,y) + xy\nabla_{xy}\hat{f}_1(x,y)\right)\right) \end{aligned}$$

Expanding the above and substituting the equation for $\hat{f}_1$, we obtain

$$ax + by + c = -2\alpha^2 x(1 + a) - \alpha y(1 + 2\alpha b) - \alpha^2 c$$

for all $x, y \in \mathbb{R}$, which yields (by comparing coefficients)

$$a(1 + 2\alpha^2) = -2\alpha^2 \quad ; \quad b(1 + 2\alpha^2) = -\alpha \quad ; \quad c(1 + \alpha^2) = 0.$$

It follows that

$$\hat{f}_1(x,y) = \frac{-\alpha}{1 + 2\alpha^2}\Big(y + 2\alpha x\Big) = f_1(x,y),$$

proving the uniqueness of f_1. Since f_2 is directly determined by f_1 via the second consistency equation, this concludes the proof.

Finally we prove that this linear update leads to decreasing distance from the origin as follows:

$$\begin{aligned}\|(x + f_1, y + f_2)\|^2 &= x^2 - \frac{2x\alpha}{1+2\alpha^2}(y + 2\alpha x) + \frac{\alpha^2}{(1+2\alpha^2)^2}\left(y^2 + 4\alpha^2 x^2 + 4\alpha xy\right) + \\ &\quad y^2 - \frac{2y\alpha}{1+2\alpha^2}\alpha(-x + 2\alpha y) + \frac{\alpha^2}{(1+2\alpha^2)^2}\left(x^2 + 4\alpha^2 y^2 - 4\alpha xy\right) \\ &= \left(x^2 + y^2\right)\left(1 - \frac{\alpha^2(3+4\alpha^2)}{(1+2\alpha^2)^2}\right) := \|(x, y)\|^2 \lambda\,.\end{aligned}$$

Notice that the derivative of λ is strictly negative in α while its limit as $\alpha \to \infty$ is 0, with value 1 at $\alpha = 0$, hence $|\lambda| = \lambda < 1$ for any $\alpha > 0$. We conclude by induction that

$$\|(x_n, y_n)\|^2 = \lambda^n \|(x_0, y_0)\|^2 \to 0$$

as $n \to \infty$, with λ decreasing (hence the speed of convergence increasing) as α increases. □

H. Training Details COLA

All code was implemented using Python. The code relies on the PyTorch library for autodifferentiability (Paszke et al., 2019).

H.1. Polynomial games

For the polynomial games, COLA uses a neural network with 1 non-linear layer for both $h_1(\theta^1, \theta^2)$ and $h_2(\theta^1, \theta^2)$. The non-linearity is a ReLU function. The layer has 8 nodes. For training, we randomly sample pairs of parameters on a [-1, 1] parameter region. In general, the size of the region is a hyperparameter. We use a batch size of 8. We found that training is improved with a learning rate scheduler. For the learning rate scheduling we use a γ of 0.9. We train the neural network for 120,000 steps. To compute the consistency loss we use the squared distance measure. The optimizer used is Adam (Kingma & Ba, 2015).

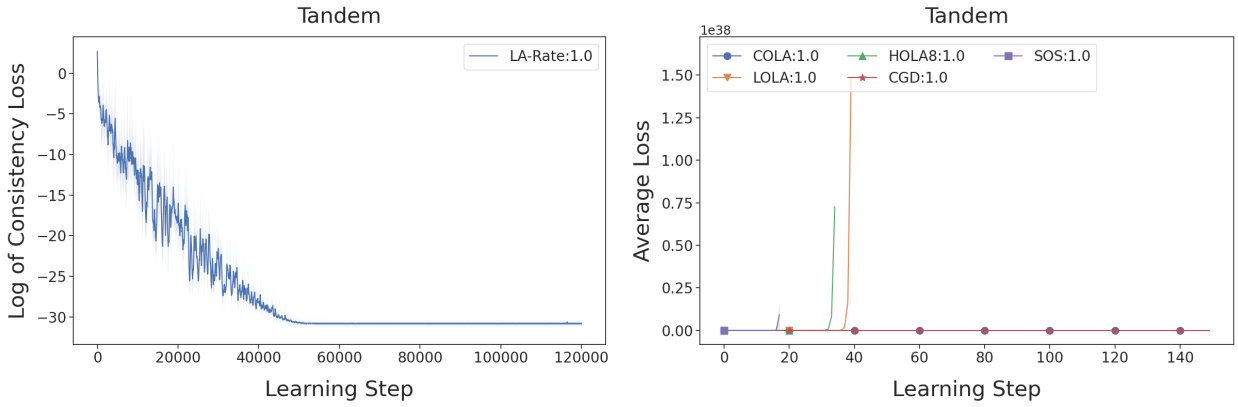
H.2. Non-polynomial games

For the non-polynomial games, we deploy a neural network with 3 non-linear layers using Tanh activation functions. Each layer has 16 nodes. For this type of game, the parameter region is set to [-7, 7], because the parameters will be squished into probability space, allowing us to explore the full probability space. During training, we used a batch size of 64. The optimizer used is Adam (Kingma & Ba, 2015).

I. Further experimental results

I.1. Tandem game

In this section, we will provide more empirical results on the Tandem game. First, in Figure 4, we display a look-ahead regime where SOS, LOLA and HOLA8 diverge in training, whereas COLA and CGD do not. Second, in Figure 5 we compare the gradient fields of HOLA4 and COLA at a low and high look-ahead rate of 0.1 and 1.0 respectively. The updates are shown on the parameter region of interesting Θ for the Tandem game, which is [-1,1]. Figure 5a and 5c show that the solutions are very similar at low look-ahead rates but become dissimilar at high look-ahead rates, shown in Figure 5b and 5d. This also confirms our quantitative observation in Table 1b.



(a) (b)

Figure 4. (a): Consistency loss of COLA at a look-ahead rate of 1.0. (b): Solutions on the Tandem game by COLA, LOLA, HOLA8, CGD, and SOS with a look-ahead rate of 1.0. The standard deviation for the initialization of parameters used here is 0.1, which is standard in the literature (Letcher et al., 2019b).

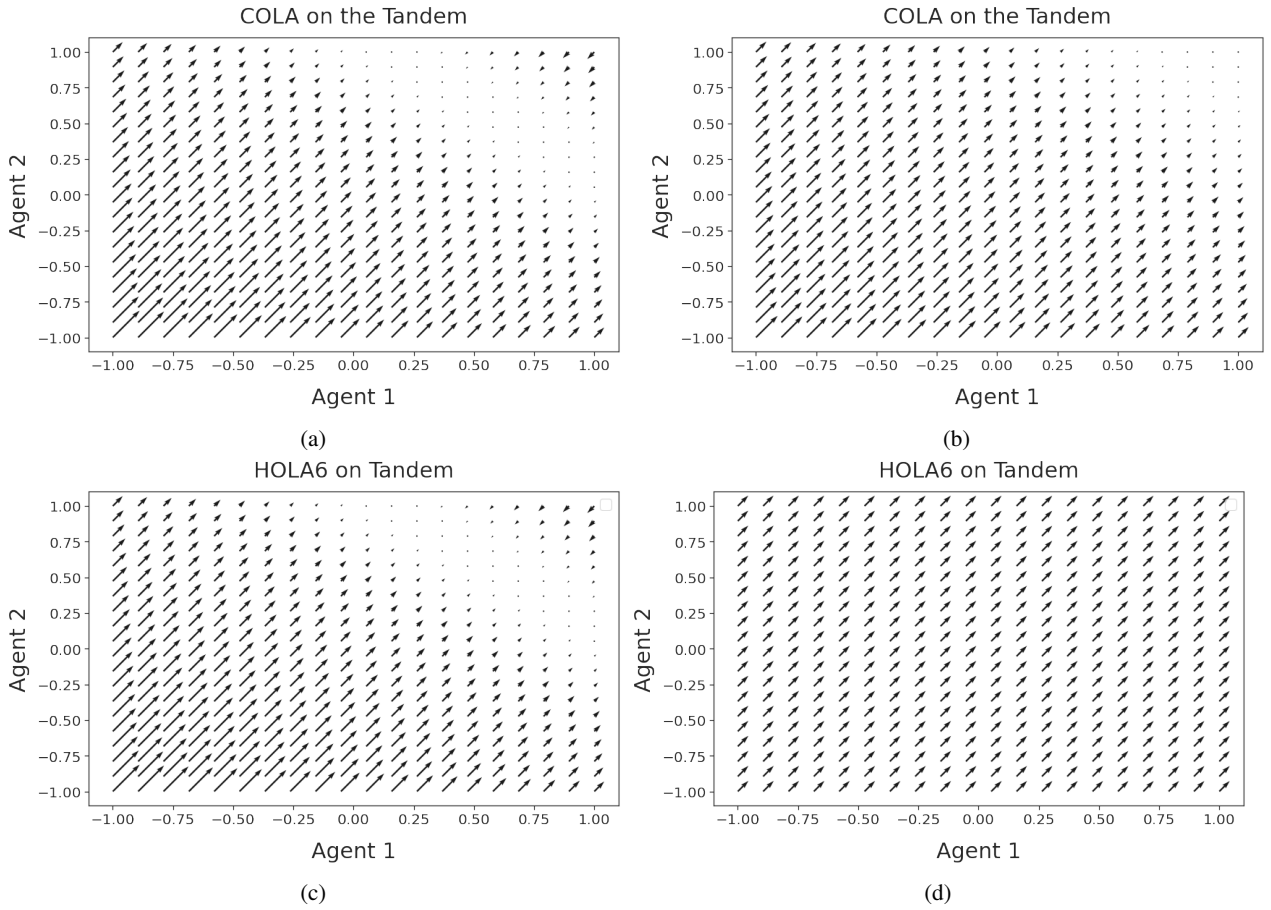


Figure 5. Gradients field of the Tandem game at two different look-ahead rates, 0.1 and 1.0.

I.2. Balduzzi and Hamiltonian game

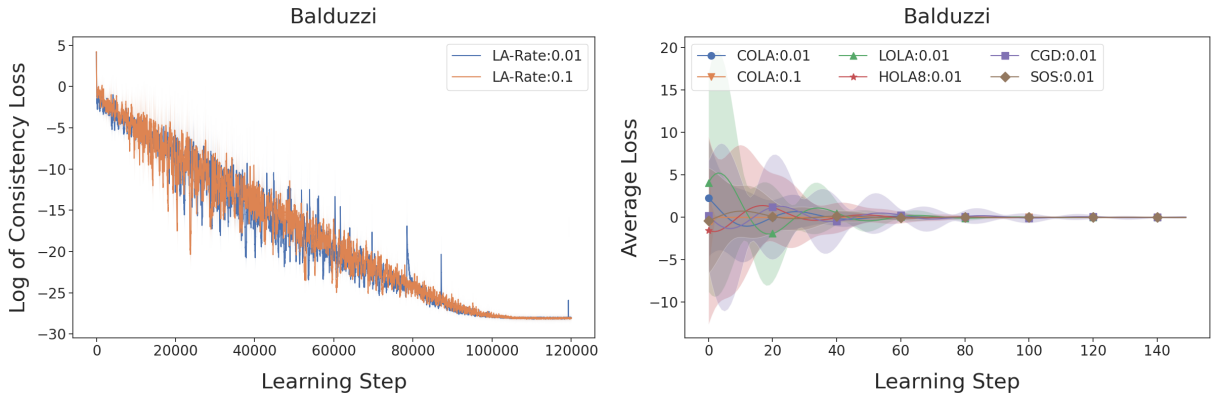
The Hamiltonian game was originally introduced in (Balduzzi et al., 2018) as a minimal example of Hamiltonian dynamics. Recall that its loss function is

$$L^1(x, y) = xy \quad \text{and} \quad L^2(x, y) = -xy \tag{34}$$

The Balduzzi game was introduced to investigate the behaviour of differentiable game algorithms when a weak attractor is coupled with strong rotational forces in the Hamiltonian dynamics (Balduzzi et al., 2018), captured by the losses

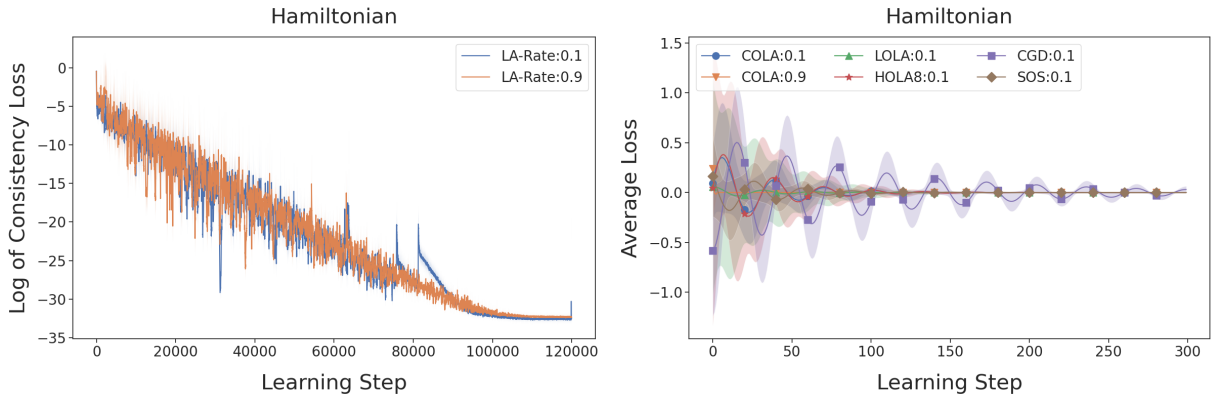
$$L^1(x, y) = \frac{1}{2}x^2 + 10xy \quad \text{and} \quad L^2(x, y) = \frac{1}{2}y^2 - 10xy \tag{35}$$

Results for both games are displayed in Figures 6 and 7. COLA at high look-ahead rates converges considerably faster than the other methods on both games. This supports Proposition 4.7 empirically. In Table 4a, 4b and 5a we find similar consistency loss behaviour as we do for Tandem. Note however, that the look-ahead thresholds are at significantly different levels. On the Balduzzi game, the threshold is much lower around 0.02, whereas for the Hamiltonian game it is between 0.1 and 0.4.



(a) (b)

Figure 6. (a): Consistency losses of COLA at different look-ahead rates. (b): Solutions on the Balduzzi game by COLA, LOLA, HOLA8, CGD, and SOS with a look-ahead rate of 0.01 (and 0.1 additionally for COLA). The standard deviation for the initialization of parameters used here is 1.0



(a) (b)

Figure 7. (a): Consistency losses of COLA at different look-ahead rates. (b): Solutions on the Hamiltonian game by COLA, LOLA, HOLA8, CGD, and SOS with a look-ahead rate of 0.1 (and 0.9 additionally for COLA). The standard deviation for the initialization of parameters used here is 1.0

I.3. Matching Pennies

Matching Pennies (MP) is a single-shot, zero-sum game, where two players, A and B, each flip a biased coin (Lee & K, 1967). Player A wins if the outcomes of both flips are the same and player B wins if they are different.

The visual difference between the updates found by COLA and HOLA4 is shown in Figure 8. Whereas they are very similar at a relatively low look-ahead rate, at a high look-ahead rate HOLA4's gradient field shows signs of high variance, especially around the origin, whereas COLA's gradient field appears to learn a robust update function. Note that HOLA4's update function results in the learning behaviour shown in Figure 2b.

Table 4. On the Hamiltonian game: (a) Log of the squared consistency loss. (b) Cosine similarity between COLA and LOLA, HOLA3, and HOLA6 over different look-ahead rates. The values represent the mean of a 1,000 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA3	HOLA6	COLA
0.9	6.99	4.63	13.68	1e-14±2e-15
0.5	12.67	13.77	13.00	1e-14±2e-15
0.4	5.78	7.14	6.38	1e-14±2e-15
0.1	0.08	0.01	2e-6	7e-15±1e-15
0.05	2e-5	4e-10	3e-15	1e-14±2e-14

(b)

α	LOLA	HOLA3	HOLA6
0.9	1.00±0.00	-1.00±0.00	0.50±0.00
0.5	1.00±0.00	1.00±0.00	1.00±0.00
0.4	1.00±0.00	1.00±0.00	1.00±0.00
0.1	1.00±0.00	1.00±0.00	1.00±0.00
0.05	1.00±0.00	1.00±0.00	1.00±0.00

Table 5. On the Balduzzi game: (a) Log of the squared consistency loss. (b) Cosine similarity between COLA and LOLA, HOLA3, and HOLA6 over different look-ahead rates. The values represent the mean of a 1,000 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA3	HOLA6	COLA
0.9	2e+6	5e+10	4e+17	5e-13±1e-13
0.1	3e+2	1.e+3	2e+4	7e-13±1e-13
0.05	2e+1	4.01	1.03	7e-13±9e-14
0.03	2.16	0.07	0.01	8e-13±1e-13
0.01	0.03	1e-5	2e-10	7e-13±1e-13

(b)

α	LOLA	HOLA3	HOLA6
0.9	1.00±0.00	-1.00±0.00	0.01±4e-9
0.1	1.00±0.00	1.00±0.00	0.40±1e-8
0.05	1.00±0.00	1.00±0.00	1.00±0.00
0.03	1.00±0.00	1.00±0.00	1.00±0.00
0.01	1.00±0.00	1.00±0.00	1.00±0.00

Table 6. Payoff Matrix for the Matching Pennies game.

	Head	Tail
Head	(+1, -1)	(-1, +1)
Tail	(-1, +1)	(+1, -1)

Table 7. Cosine similarities over multiple COLA training runs on the MP and Tandem game for different look-ahead rates.

Game@LR	Cosine Sim
MP@10	0.97 ± 0.01
MP@0.5	0.99 ± 0.01
Tandem@0.1	1.00 ± 0.00
Tandem@1.0	0.98 ± 0.01

I.4. Ultimatum game

Here we provide more empirical results on the Ultimatum game. First, in Table 8a and 8b we display the consistency losses of COLA and HOLA at different look-ahead rates. In comparison to the polynomial games, here we observe that COLA's consistency losses are not as low as HOLA*n*'s losses at low look-ahead rates. Nonetheless, they are low enought to constitute a consistent solution. Moreover, the COLA and HOLA*n* solutions are very similar according to the cosine similarity score. Qualitatively, we compare the updates in Figure 9. Similarly to the MP game, we observe that HOLA4's update shows higher variance around the origin that COLA's update. This variance is reflected in HOLA4's learning behaviour shown in Figure 10d, where HOLA4 do not converge to the fair solution consistently.

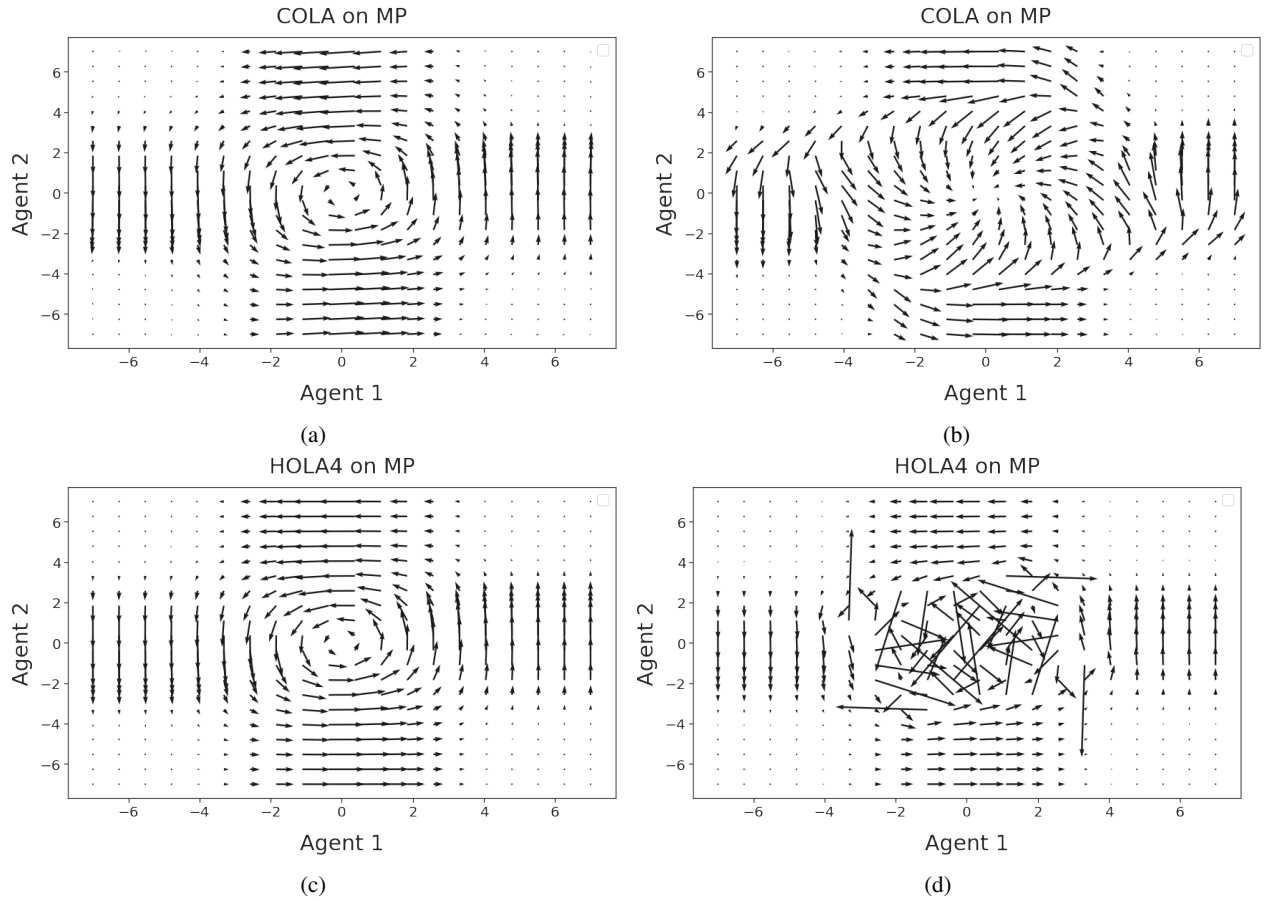


Figure 8. Gradients field of the MP game at two different look-ahead rates: 0.5 (RHS) and 10 (LHS). COLA is on the upper row, HOLA4 is on the lower row.

Table 8. On the Ultimatum game: Over multiple look-ahead rates we compare (a) the consistency losses and (b) the cosine similarity between COLA and LOLA, HOLA2, and HOLA4. The values represent the mean of a 1,000 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA2	HOLA4	COLA
1.1	2e-3	5e-3	0.01	4e-4±5e-5
0.7	4e-4	2e-4	2e-4	7e-5±1e-5
0.3	3e-5	2e-7	5e-8	3e-6±2e-6
0.1	2e-7	1e-11	6e-13	4e-6±4e-6
0.001	3e-15	4e-17	9e-17	4e-6±3e-6

(b)

α	LOLA	HOLA2	HOLA4
1.1	0.96±0.01	0.96±0.01	0.95±0.01
0.7	0.98±0.01	0.98±0.01	0.98±0.01
0.3	0.99±0.01	0.99±0.01	0.99±0.01
0.1	0.99±0.01	0.99±0.01	0.99±0.01
0.001	0.99±0.01	0.99±0.01	0.99±0.01

I.5. Chicken game

In the Chicken game, an agent can either choose to yield to avoid a catastrophic payoff but face a small punishment if they are the only agent to yield. Imagine a game where two agents drive towards each other in their cars. If both never swerve, they frontally crash into each other, an obviously catastrophic outcome. If any of the agents "chicken out", e.g. swerve, they do not crash but receive a small punishment for having chickened out. At the same time, the other agent is being rewarded for staying on track, as quantified in Table 9.

Next we report the consistency losses on the Chicken game in Table 10a. We identify a look-ahead rate threshold where HOLA*n*'s consistency loss becomes increasingly bigger with increasing order. For the Chicken game, this threshold is fairly

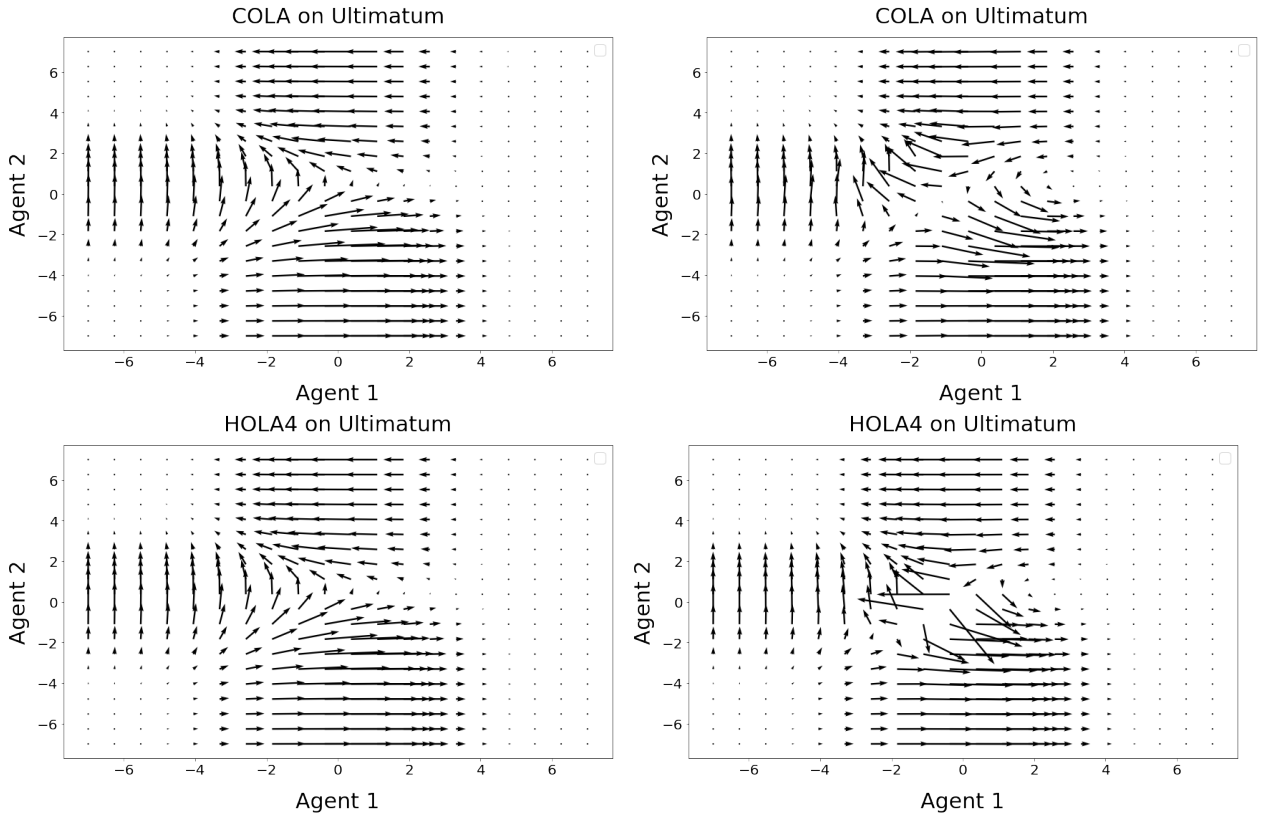


Figure 9. Gradients field of the ultimatum game at two different look-ahead rates: 0.2 (RHS) and 1.1 (LHS). COLA is on the upper row, HOLA4 is on the lower row.

Table 9. Payoff Matrix for the Chicken game.

	C (swerve)	D (straight)
C (swerve)	0, 0	-1, +1
D (straight)	+1 ,-1	-100, -100

low, between 0.01 and 0.05. Interestingly, we note that around a look-ahead rate of 0.05 and 0.1 it becomes harder to find a consistent solution for COLA.

Table 10. On the Chicken game: Over multiple look-ahead rates we compare (a) the consistency losses and (b) the cosine similarity between COLA and LOLA, HOLA2, and HOLA4. The values represent the mean of a 1,000 samples, uniformly sampled from the parameter space Θ. The error bars represent one standard deviation and capture the variance over 10 different COLA training runs.

(a)

α	LOLA	HOLA2	HOLA4	SOS	CGD	COLA
1.0	2429	3892	46637	1494	1677	0.01±0.01
0.5	643	484	4320	475	2330	0.03±0.01
0.1	11.99	7.69	73.28	2.73	8.46	0.70±0.10
0.05	0.84	0.17	0.47	0.37	1.31	0.06±0.01
0.01	9e-4	3e-6	6e-9	2e-4	0.04	5e-4±3e-4

(b)

α	LOLA	HOLA2	HOLA4
1.0	0.39±0.03	0.57±0.02	0.57±0.03
0.5	0.54±0.04	0.62±0.03	0.64±0.04
0.1	0.88±0.03	0.88±0.03	0.86±0.03
0.05	0.93±0.03	0.93±0.03	0.93±0.03
0.01	0.96±0.02	0.96±0.02	0.96±0.01

We also perform a qualitative comparison on the gradient fields of COLA and HOLA4 on the Chicken game (see Figure 11). The difference at high look-ahead rates is pronounced, as HOLA4 shows high variance around the origin. Nonetheless, the gradient field leads to swerving in the actual game (see Figure 12d, which is a preferable outcome. Moreover, HOLA4

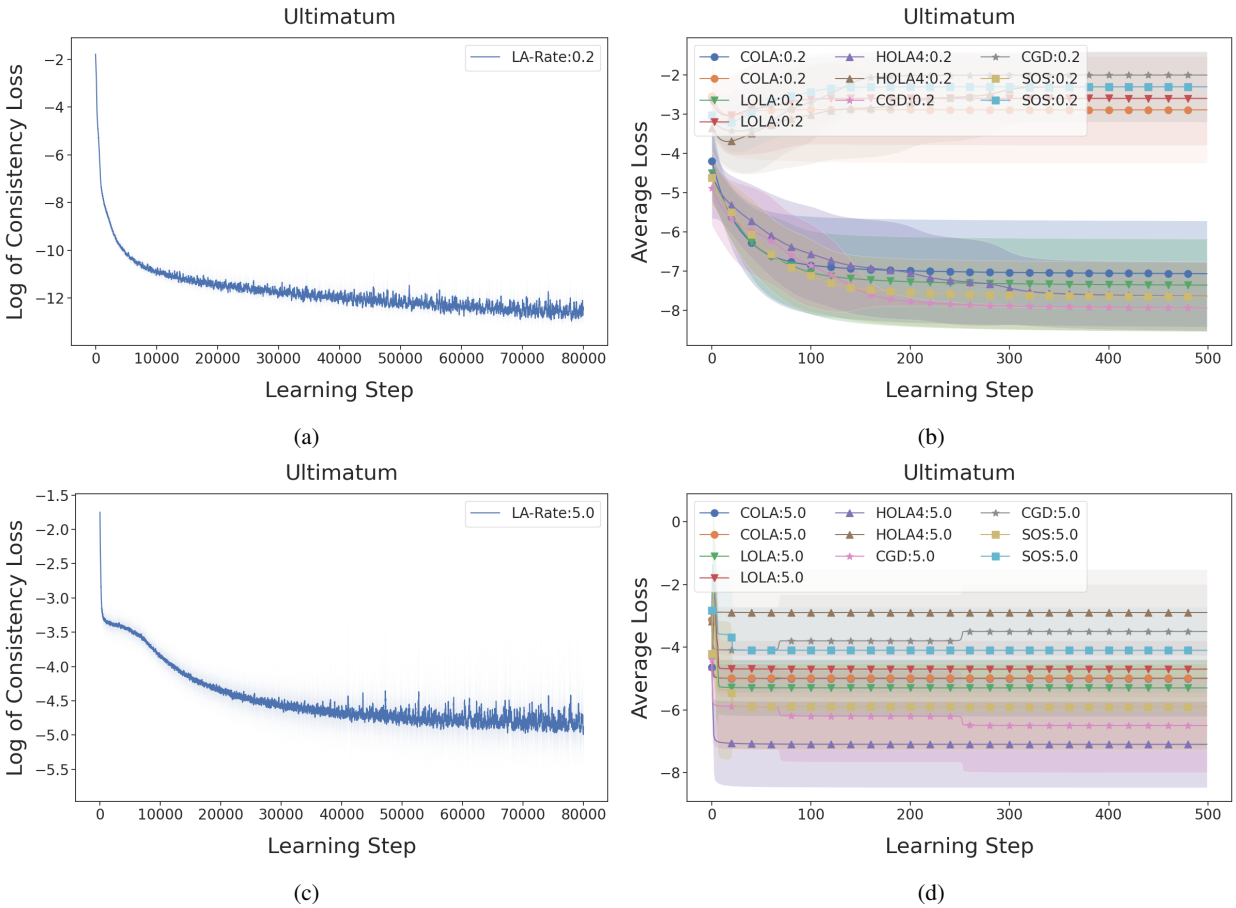


Figure 10. (a) and (c): Consistency losses of COLA at different look-ahead rates. (b) and (d): Solutions on the Ultimatum game by COLA, LOLA, HOLA4, CGD, and SOS with a look-ahead rate of 0.2 and 5.0. The standard deviation for the initialization of parameters used here is 1.0.

appears to converge to Swerving consistently despite the chaotic gradient field, showcasing that the behaviour in the game is not necessarily correlated with the qualitative analysis of the gradient fields.

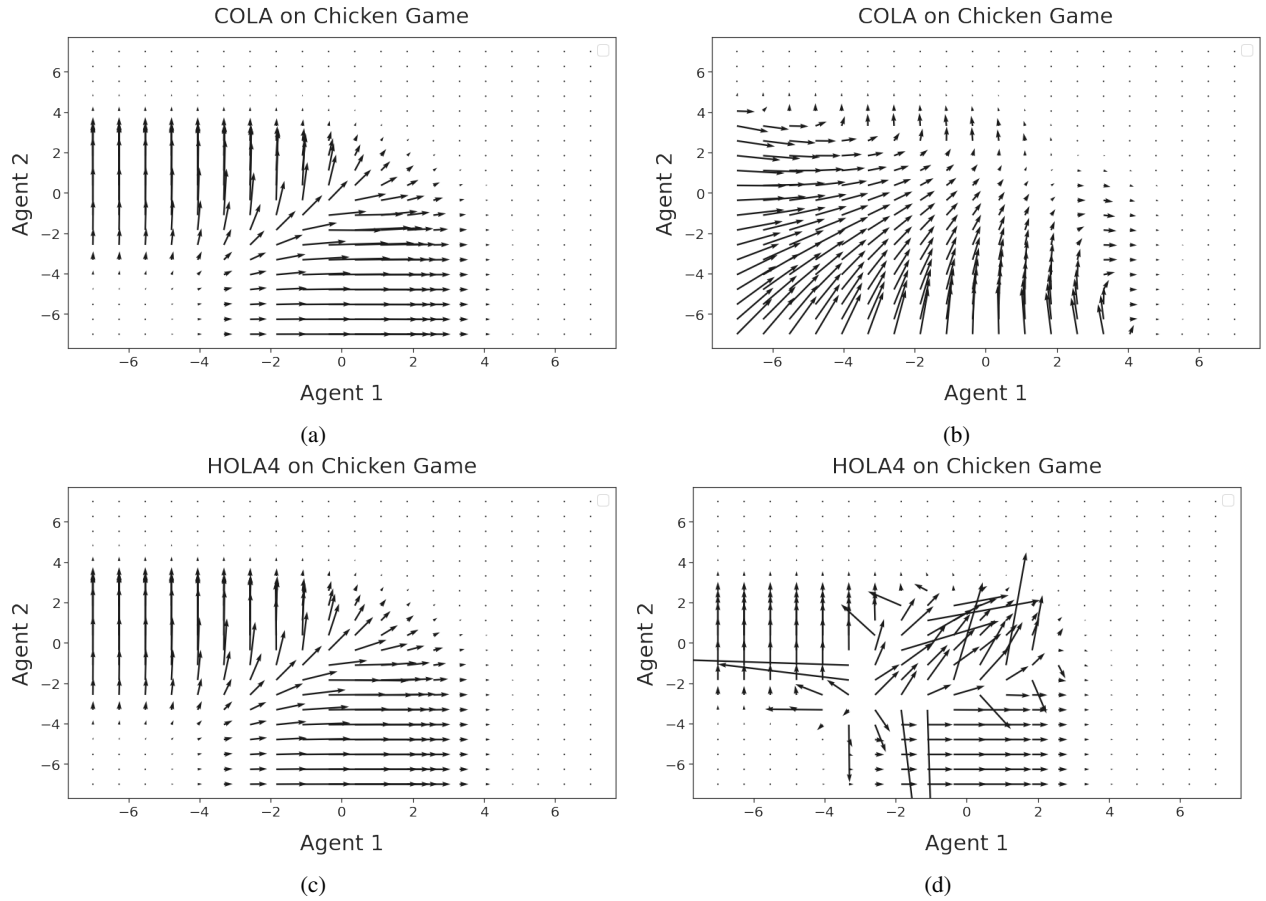


Figure 11. Gradients field of the Chicken game for COLA and HOLA4 at two different look-ahead rates, 0.01 (LHS) and 1.0. (RHS)

I.6. IPD

As a baseline comparison, we show the performance of different state-of-the-art algorithms in Figure 13. CGD does not recover the tit-for-tat policy whereas exact LOLA, Taylor LOLA and SOS do.

Table 11. Payoff Matrix for the IPD game.

	C	D
C	(-1, -1)	(0, -3)
D	(0, -3)	(-2, -2)

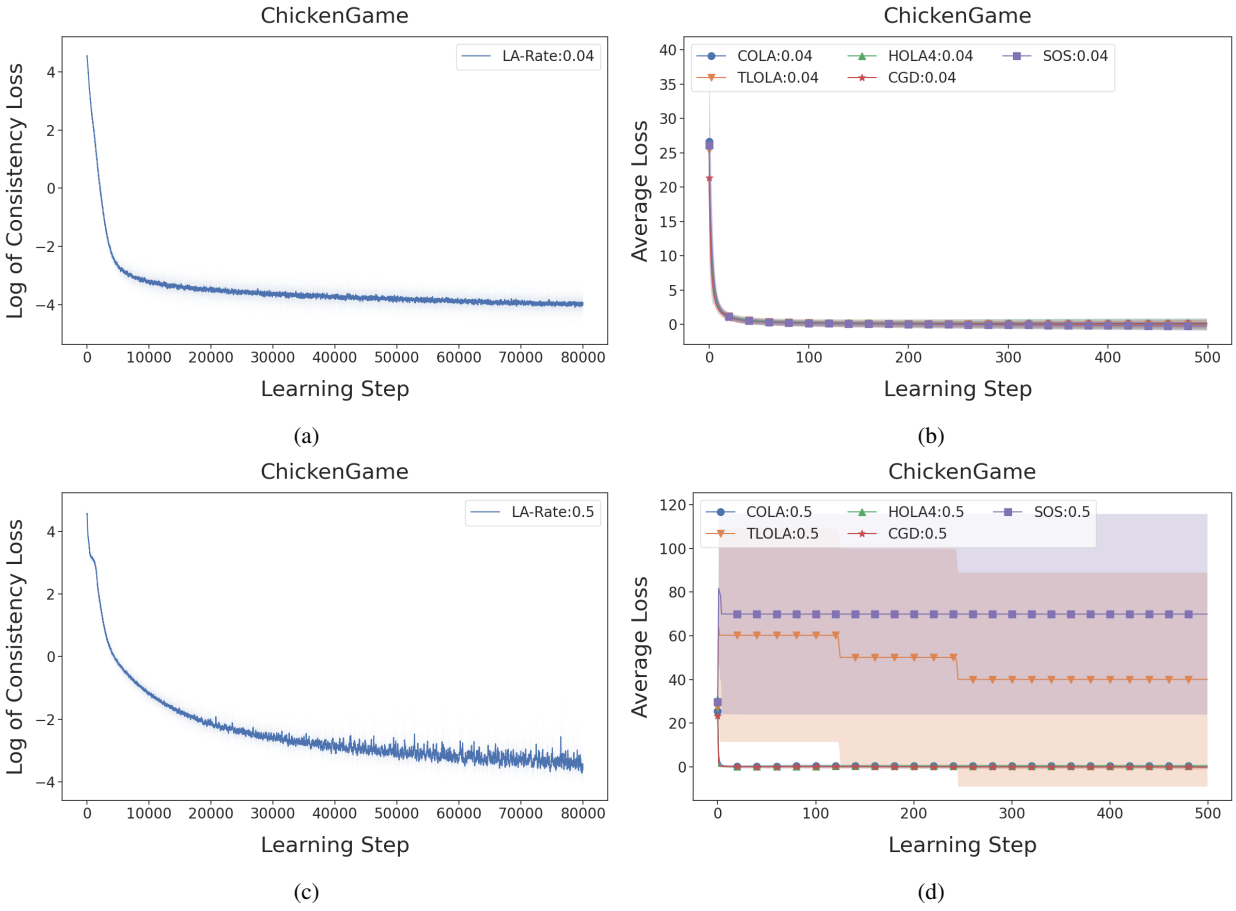


Figure 12. (a) and (c): Consistency losses of COLA at different look-ahead rates. (b) and (d): Solutions on the Ultimatum game by COLA, TLOLA, HOLA4, CGD, and SOS with a look-ahead rate of 0.04 and 0.5. We used a standard deviation of 1.0 to initialize the parameters.

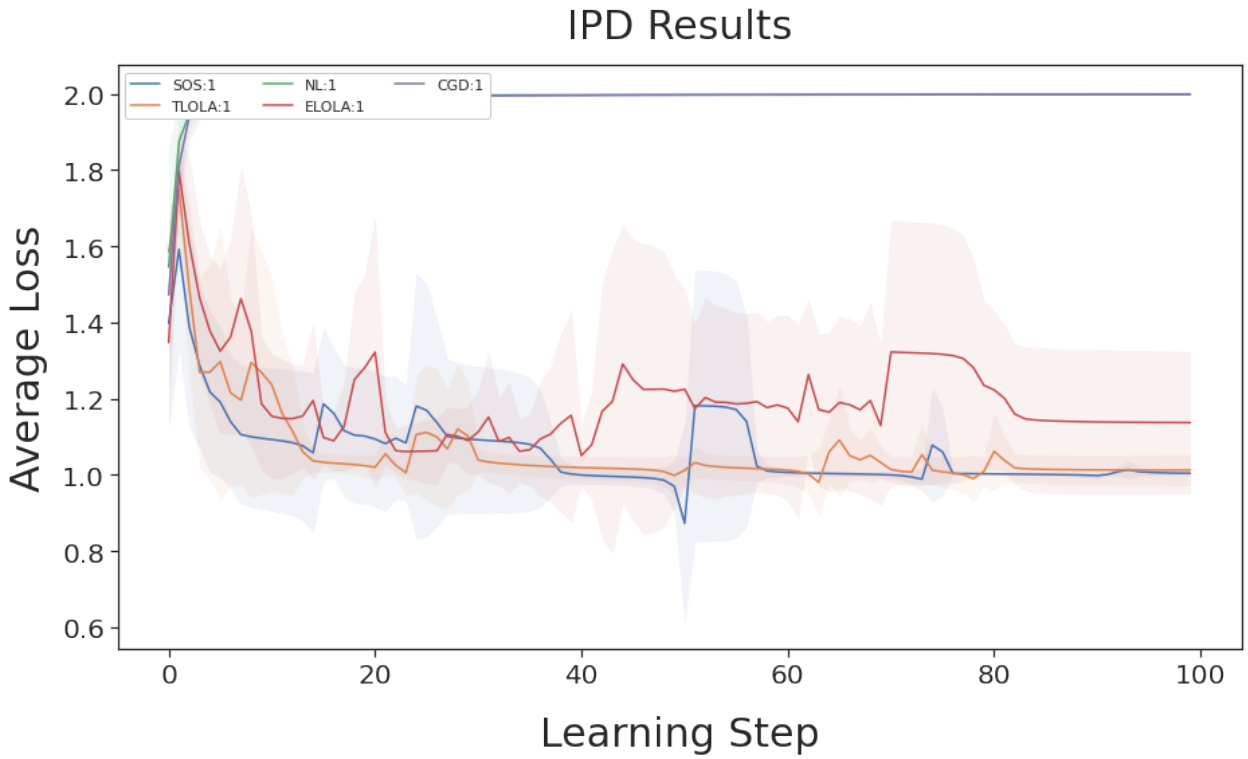


Figure 13. CGD, SOS, Taylor LOLA (TLOLA), Exact LOLA (ELOLA) and Naive Learning (NL) on the IPD at a look-ahead rate of 1.0. We used a standard deviation of 1.0 to initialize the parameters.

B

Scaling Opponent Shaping to High Dimensional Games - Appendices

A SHAPER DETAILS

Below we list the SHAPER algorithm for both batching and un-batched version.

Algorithm 2 SHAPER Update: Given a POSG $\mathcal{M}$, policies $\pi_{\phi_i}, \pi_{\phi_{-i}}$ and their respective initial hidden states h_i, h_{-i} and a distribution of initial co-players ρ_ϕ, this algorithm updates a meta-agent policy ϕ_i over T trials consisting of E episodes.

Require: $\mathcal{M}, \phi_i, \rho_\phi, E, T, f$

for $t = 0$ **to** T **do**
  Initialise trial reward $\bar{J} = 0$
  Initialise meta-agent hidden states $h_i = \mathbf{0}$
  Sample co-players $\phi_{-i} \sim \rho_\phi$
  for $e = 0$ **to** E **do**
    Initialise co-players' $h_{-i} = \mathbf{0}$
    $J_i, J_{-i}, h'_i, h'_{-i} = \mathcal{M}(\phi_i, \phi_{-i}, h_i, h_{-i})$
    Update ϕ_{-i} according to co-players' update rule.
    $h_i \leftarrow f(h'_i)$
    $\bar{J} \leftarrow \bar{J} + J_i$
  end for
  Update ϕ_i with respect to $\bar{J}$
end for

Here we also provide a diagram of the batching method.

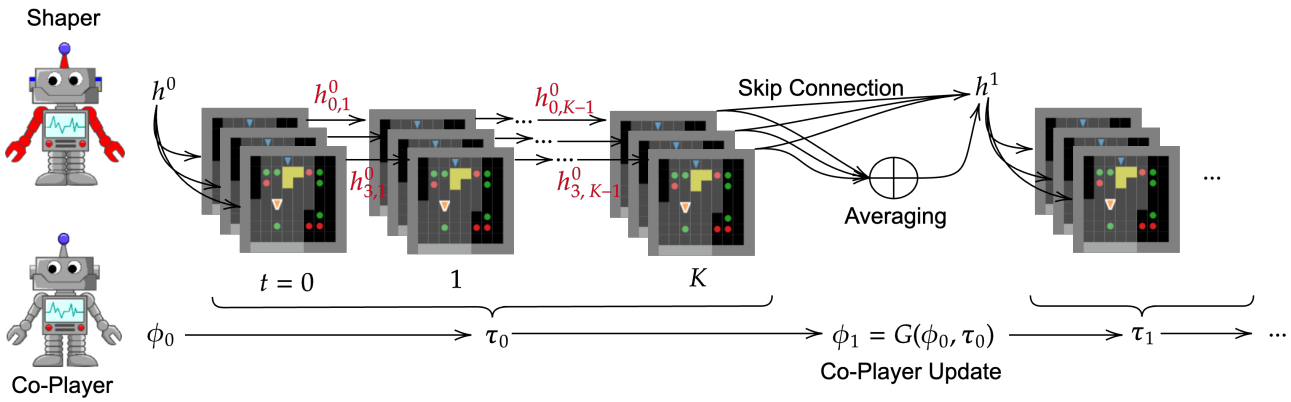


Figure 6: Typically, co-players are trained over vectorized environments, depicted as superimposed in-game frames. The co-players update their parameters after an episode of K steps. The co-players parameter update depends on the trajectories from the whole batch of vectorized environments. However, the shaping method deploys one hidden state per environment. Without averaging, the respective hidden states miss context information, which could be important to ensure proper shaping.

B MATRIX GAME DETAILS

Here we present details of training of shaping agents in Iterated Matrix Games.

B.1 Payoff Matrices

Table 5: Payoff Matrices

	C	D
C	(-1,-1)	(0, -3)
D	(-3, 0)	(-2, -2)

(a) Iterated Prisoners Dilemma (IPD)

	H	T
H	(1,-1)	(-1, 1)
T	(-1, 1)	(1, 1)

(b) Iterated Matching Pennies (IMP)

B.2 Training Details

We present training curves for both IPD and IMP below.

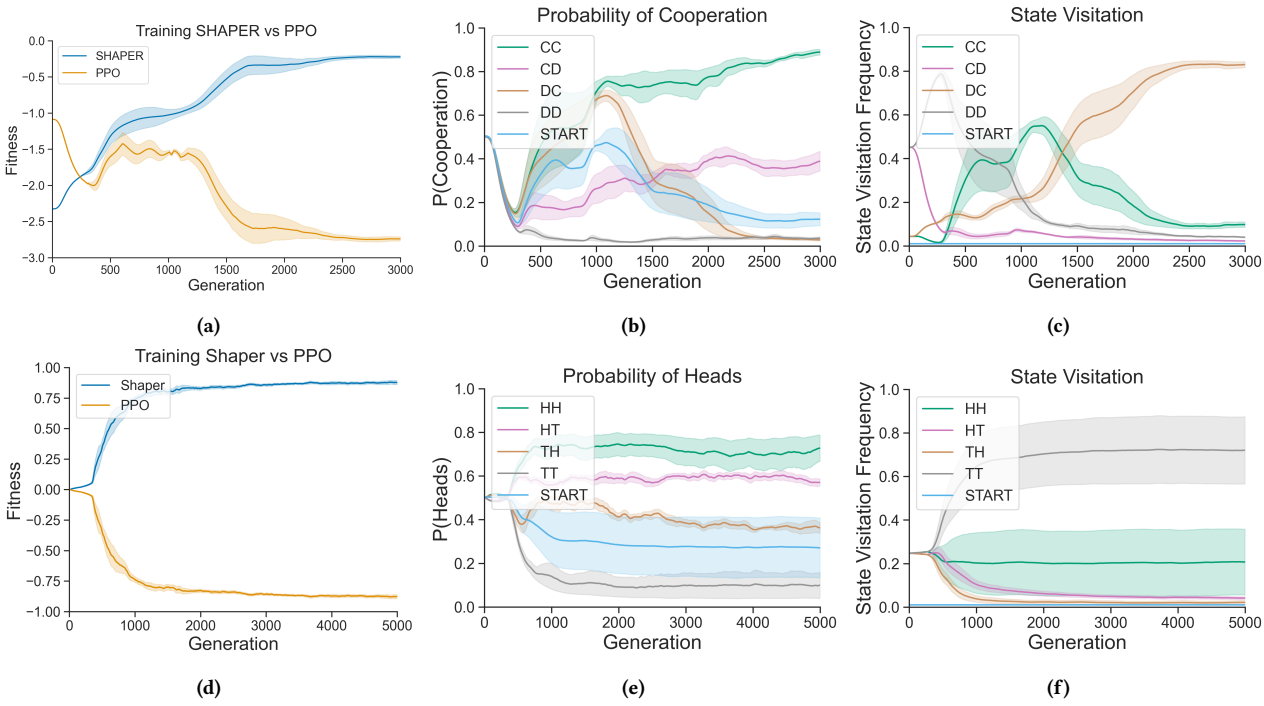
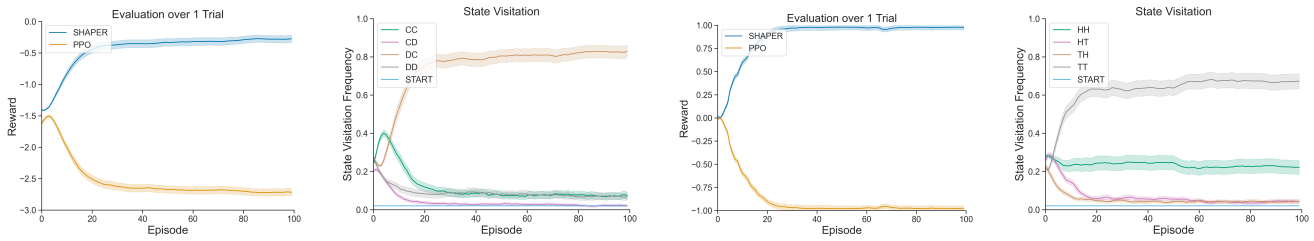


Figure 7: Training results in the finite IPD over 5 seeds for SHAPER. Here we display the (a) fitness, (b) conditional probability of cooperation, and (c) state visitation. Training results in the IMP over 5 seeds for SHAPER. (d) Fitness (e) Empirical probability of Cooperative action conditioned by state and (f) state visitation.

B.3 Evaluation



(a) (b) (c) (d)

Figure 8: SHAPER finds extortion-like strategies in finite matrix games - evaluation trials composed of 100 episodes and over 20 seeds. In the IPD, we show reward (a) and state visitation (b) to demonstrate shaping by the high proportion of DC states. In the IMP, we evaluate (c) reward and (d) state visitation to demonstrate shaping by the high proportion of matching states HH and TT.

C MATRIX GAME RESULTS

Table 6: Converged reward per step (meta-agent, co-player) for agents against Naive Learners in finite matrix games. SHAPER can shape co-players to exploitative equilibria. We report mean and standard deviation over 20 randomised co-players.

	IPD	IMP
SHAPER	**-0.1 ± 0.02, -2.8 ± 0.05**	**0.9 ± 0.02, -0.9 ± 0.02**
M-FOS	-0.6 ± 0.14, -2.3 ± 0.14	**0.8 ± 0.09, -0.8 ± 0.09**
GS	-1.0 ± 0.03, -1.3 ± 0.10	0.0 ± 0.01, 0.0 ± 0.01
CT-NL	-2.0 ± 0.00, -2.0 ± 0.00	0.0 ± 0.00, 0.0 ± 0.00

D GENERALISABILITY OVER LONG TIME PERIOD

We also present the results for allowing the co-player to adapt longer to the meta-agent. This is in aims to understand what an RL agents best response to a meta-agent looks like. We report the scores below

Table 7: Converged Reward (meta-agent, co-player) for agents trained with Naive Learners on the CoinGame, IPDitM and IMPitM. The reward is averaged per episode, mean is reported across 100 seeds with standard deviations. Trial continued until episodic reward converged (for CoinGame=5000 episodes, for * in the Matrix = 1000).

	CoinGame	IPD in the Matrix	IMP in the Matrix
SHAPER	**5.21 ± 0.66,−4.84 ± 0.81**	**21.94 ± 1.12,22.40 ± 0.92**	**0.14 ± 0.06,−0.14 ± 0.06**
M-FOS (ES)	3.12 ± 0.42,2.27 ± 0.40	14.69 ± 1.28,24.93 ± 1.14	**0.11 ± 0.70,−0.11 ± 0.07**
M-FOS (RL)	0.85 ± 0.60,−0.23 ± 0.44	7.58 ± 0.21,7.26 ± 0.17	0.04 ± 0.00,−0.04 ± 0.00
GS	**5.38 ± 0.42, −3.31 ± 0.59**	15.43 ± 1.39,25.43 ± 1.00	0.00 ± 0.00,0.00 ± 0.00
PT-NL	0.56 ± 0.59,0.26 ± 0.48	6.33 ± 0.33,6.96 ± 0.31	−0.17 ± 0.10,0.17 ± 0.10
CT-NL	0.44 ± 0.65,0.31 ± 0.61	5.56 ± 0.02,5.56 ± 0.02	−0.10 ± 0.06,0.10 ± 0.06

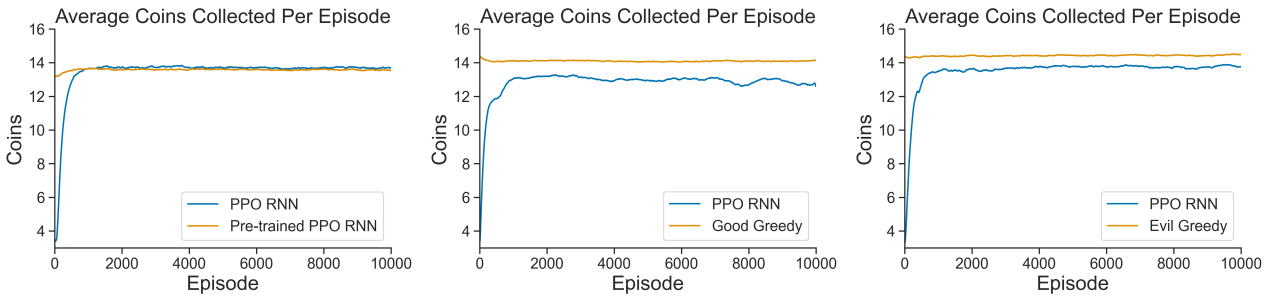
E COINGAME DETAILS

E.1 Evaluating Player's Competency

In the CoinGame, agents struggle to learn (via RL) when trained against a pre-trained opponent. On inspection of trajectories, we found that competent agents removed a sufficient amount of coins to restrict reinforcement learners ability to capture signal from the game.

To address this, we show that adjusting the observations such that an agent receives to an *egocentric* viewpoint (i.e. an agent always observes that it is in the centre of the grid) leads to competency against a competent opponent. In this case, we measured competency as an agent's ability to pick up coins. *Competent* agents were those who picked up a similar number of coins to those trained against a stationary agent. Throughout the rest of the paper, we refer to the original version of CoinGame as *non-egocentric* CoinGame and the

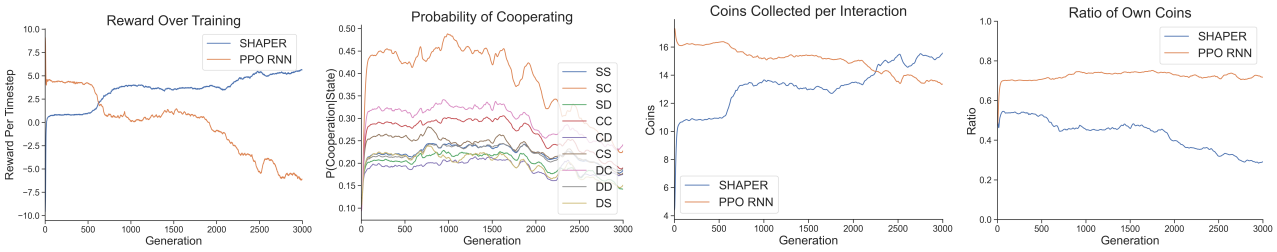
modified observation version as *egocentric* CoinGame. In addition, we deviate from the original 5 by 5 version of CoinGame to a 3 by 3 version, following the setting used in [30].



(a) (b) (c)

Figure 9: Results of the experimental protocol verifying that PPO RNN learns to play the egocentric CoinGame against (a) pre-trained PPO RNN (b) Good Greedy (c) Evil Greedy. Notice that the agent learns to pick up roughly the same number of coins per episode as its competent opponents.

E.2 Training Details



(a) (b) (c) (d)

Figure 10: Training results of SHAPER vs. PPO RNN in the egocentric CoinGame. (a) Fitness, (b) the meta-agent's frequency of picking up its own colour coin depending on existing convention, (c) the number of coins picked up per episode, (d) both agent's frequency of picking up its own colour coin over a full episode.

F ABLATION STUDIES RESULTS & DETAILS

Table 8: Ablations highlighting the importance of context and history for Shaping. We report converged reward per step (meta-agent, co-player) for agents against Naive Learners.

Context Challenge: IPD	
SHAPER	-0.8, -2.0
SHAPER w/o Context	-1.25, -1.75
History Challenge: IMP (Length=2)	
SHAPER	0.5, -0.5
SHAPER w/o History	0.0, 0.0
History Challenge: IMP (Length=100)	
SHAPER	0.5, -0.5
SHAPER w/o History	0.5, -0.5

We also include additional state visitation for the hardstop challenge. This helps validate that SHAPER reacts to agents becoming exploitable after a hardstop has occurred.

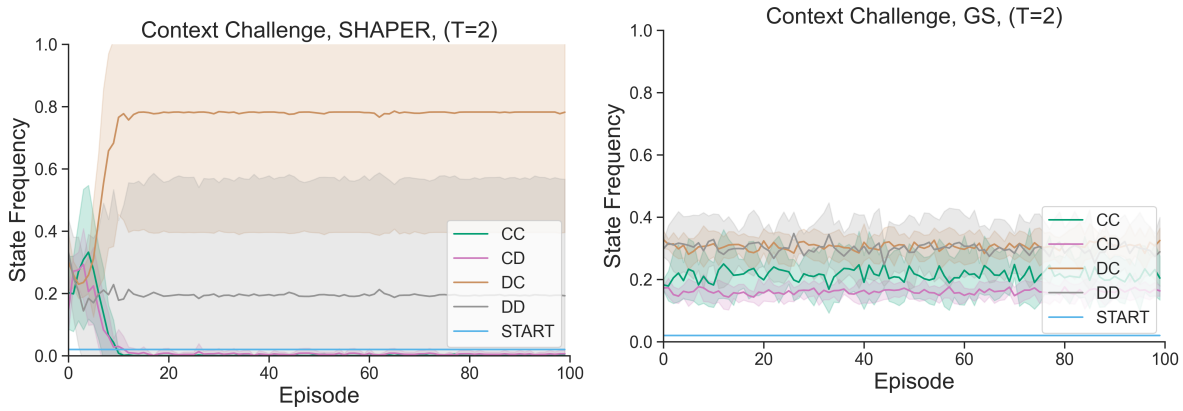
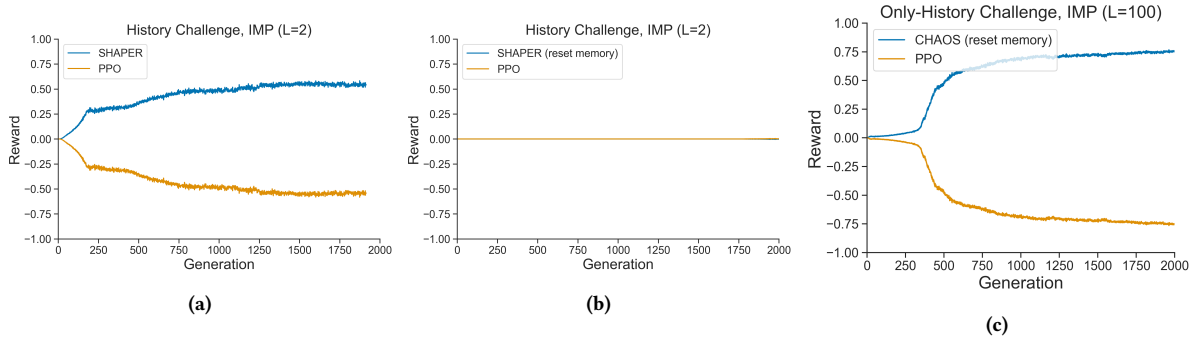


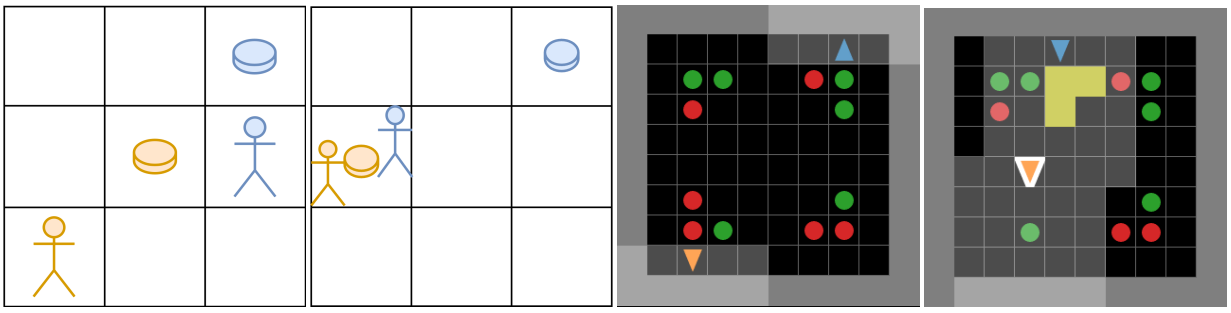
Figure 11: State visitation for the Hardstop Challenge with meta-agents (a) SHAPER and (b) GS. Here we see SHAPER responds to co-players frozen stationary policy by moving into either DD (the best response to a defective agent) or DC (the best response to a fully cooperative agent), whilst GS is unable to adjust its approach after the co-player stop learning, continuing to plays the sub-optimal shaping strategy)



(a) (b) (c)

Figure 12: The Only-History Challenge: Training curves in the IMP with episode length = 2 for (a) SHAPER and (b) Shaper without context. Note that in short time-spans, history cannot enable shaping. Additionally (c) SHAPER without context in IMP with episode length = 100 shows with sufficient timespan, history can enable shaping.

G GRIDWORLD DETAILS



(a) (b) (c) (d)

Figure 13: Illustration of the CoinGame, a multi-step general-sum game. (a) shows a typical state, where agents gain +1 for collecting any coin and, if the coin is not theirs, inflict a penalty of −2 to the co-player. (b) demonstrates a degenerate state of the game, where memory-less agents infer co-player behaviour (in this case, the blue agent defects). (c-d) Render of the IPDitM games. Agents with restricted visibility and orientation traverse a grid picking up either *Defect* or *Cooperate* coins. (c) shows an initial state of the game before either agent has a coin. Once agents pick up a coin, their appearance changes, and they can interact. (d) shows agent one having collected a coin and agent 2 firing the interact beam.

H IPD IN THE MATRIX DETAILS

H.1 Environment

We first present a typical rollout from the environments below.

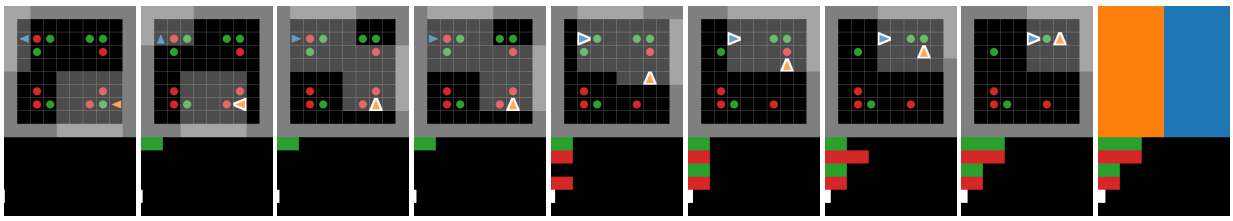


Figure 14: Rollout of a typical episode. Player 1 collects a green coin first then two reds before a final green coin; Player 2 collects a red then green coin. Both agents collect an equal ratio of coins and when interacting both play a mixed strategy of $(0.5, 0.5)$ resulting in them receiving equal reward when interacting. This state is presented for 5 frames (whilst frozen) after which coins are restored (in same positions) and agents respawn (in new positions).

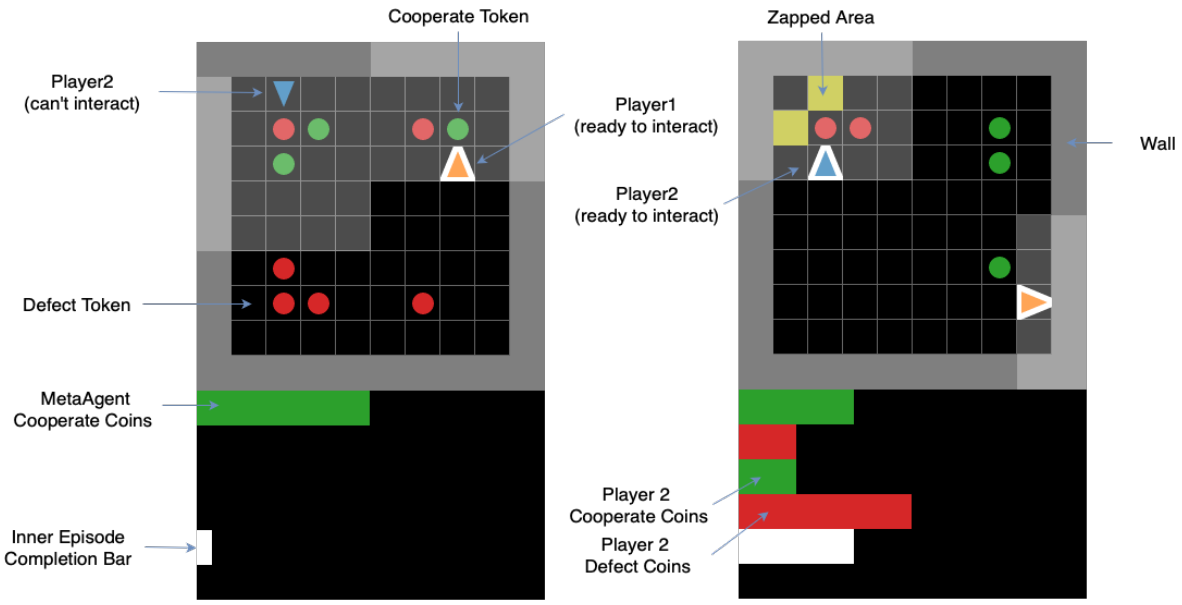


Figure 15: Annotated Image of IPDiTM renders, demonstrating the objects within the game

*** in the Matrix** extends matrix games to gridworld environments [41], where * is any normal-form game. For visual descriptions, see Figures 13(c,d), 14, and15. Agents collect two types of resources into their inventory: *Cooperate* and *Defect* coins. Once an agent has collected any coin, the agent's colour changes, representing that the agent is "ready" for interaction. Agents can fire an 'interact' beam to an area in front of them. If an agent's interact beam catches a "ready" agent, both receive rewards equivalent to playing a matrix game *, where their inventory represents their policy. For example, when agent 1's inventory is 1 *Cooperate* coin and 3 *Defect* coins, agent 1's probability to cooperate is 25%. Upon a successful interaction, agents are frozen for five steps whilst their inventories are displayed to one another. After the freeze, agents respawn in new locations, and the current coins are reset. Agents have orientation, limited directed visibility and can step forward. Agents cannot occupy the same spot, with collisions giving preference to the stationary player; the agent who moves is randomised in the case both moved. Agents only observe their own inventory. On the completion of a full episode, coin locations are randomised.

Difference to Melting Pot: The meltingpot environment is 23x15, whereas our grid is 8x8. A size we chose to be as large as possible whilst still being able to optimise the methods given compute limitations. Meltingpot has walls placed within the environment, whereas ours

does not. While our environment is smaller, we add additional stochasticity by randomising the coin positions, which are fixed in Meltingpot. Finally, unlike the meltingpot environment, agents spawned with no coins (as opposed to one of each), this made it easier for agents to choose pure cooperate or defect strategies. We found that randomising coin positions important as even large environments representing POMDPs with insufficient stochasticity, such as the The Starcraft Multi-Agent Challenge, can be solved without memory[10, 36]. We found similar evidence in the IPDitM. In IMPitM, the same differences hold.

H.2 Training Details

Below we present training curves for the three major shaping baselines.

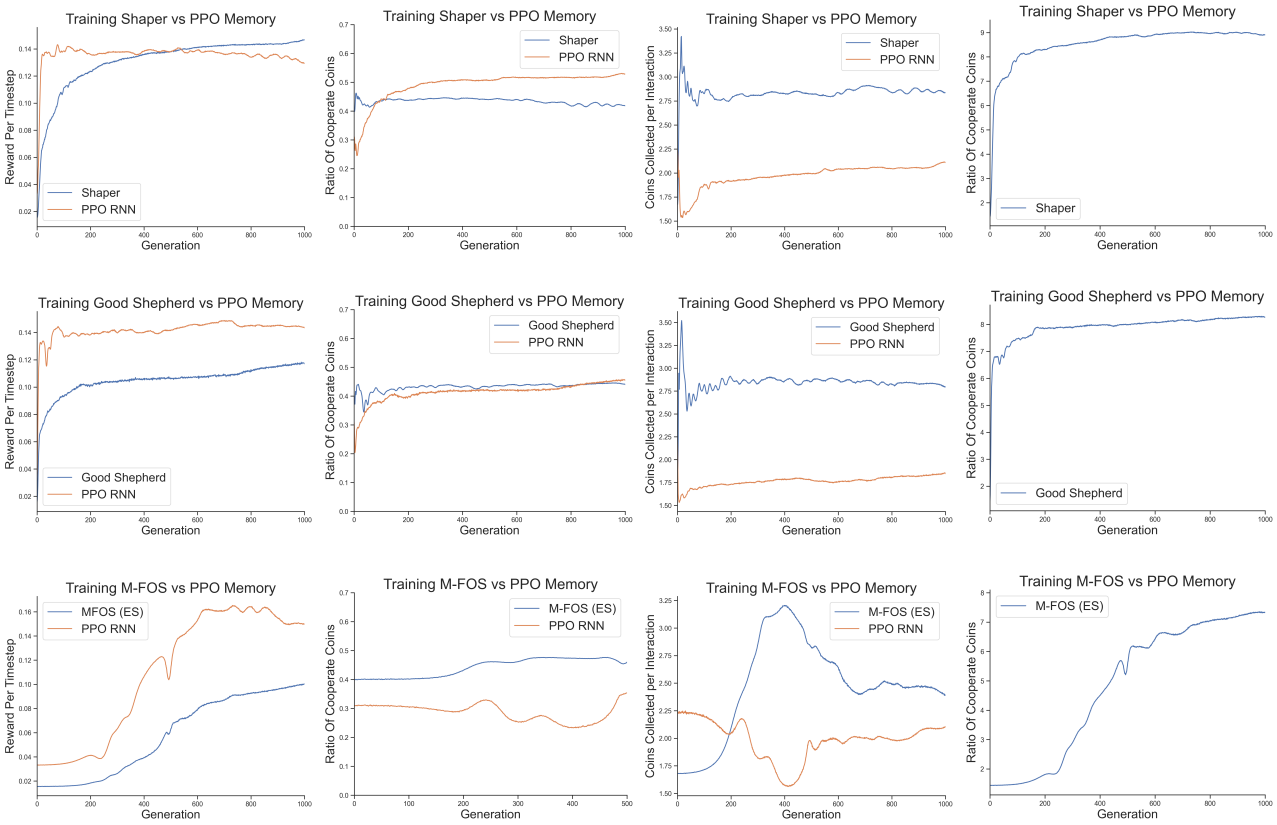
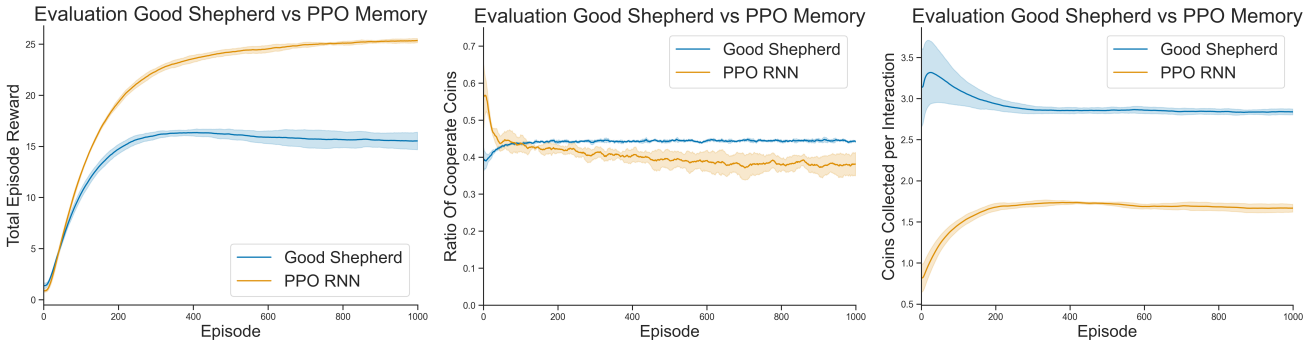
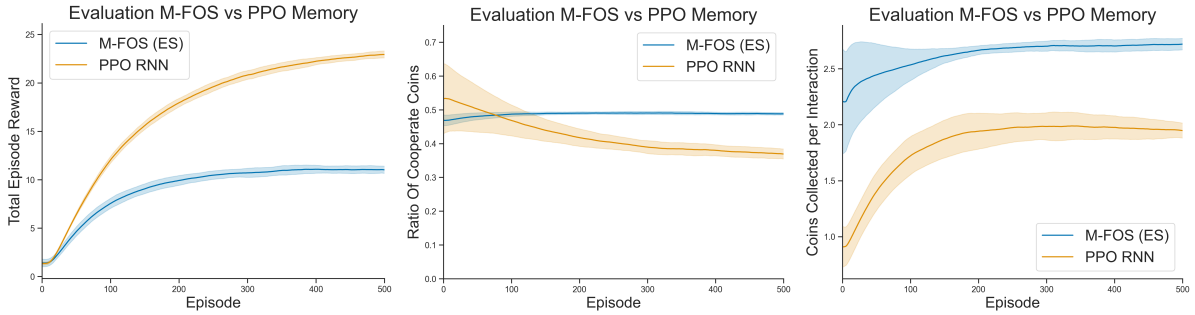


Figure 16: Training results for OS methods vs. PPO RNN in the IPDitM. (Column 1) Reward Per Timestep, (Column 2) the meta-agents's frequency of picking up its own colour coin depending on existing convention, (Column 3) the number of coins picked up per episode, (Column 4) the number of soft resets / successful interactions.



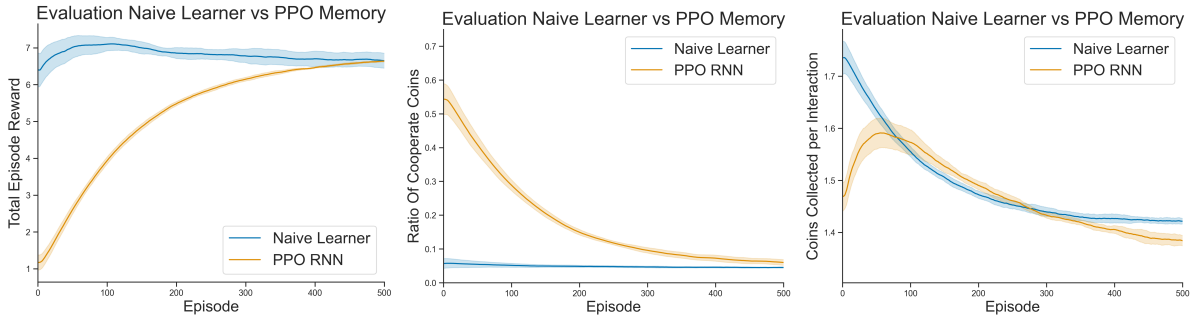
(a) (b) (c)

Figure 17: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.



(a) (b) (c)

Figure 18: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.



(a) (b) (c)

Figure 19: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.

I IMP IN THE MATRIX DETAILS

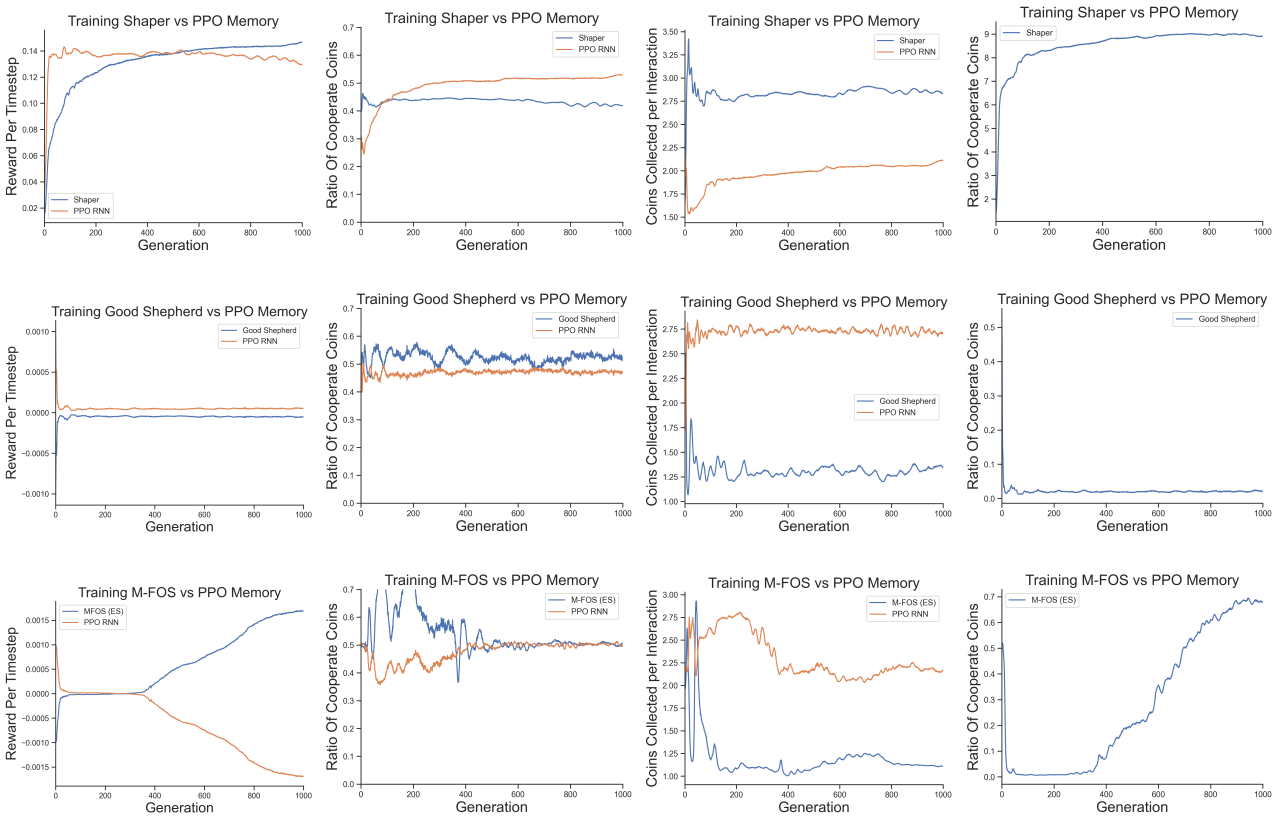
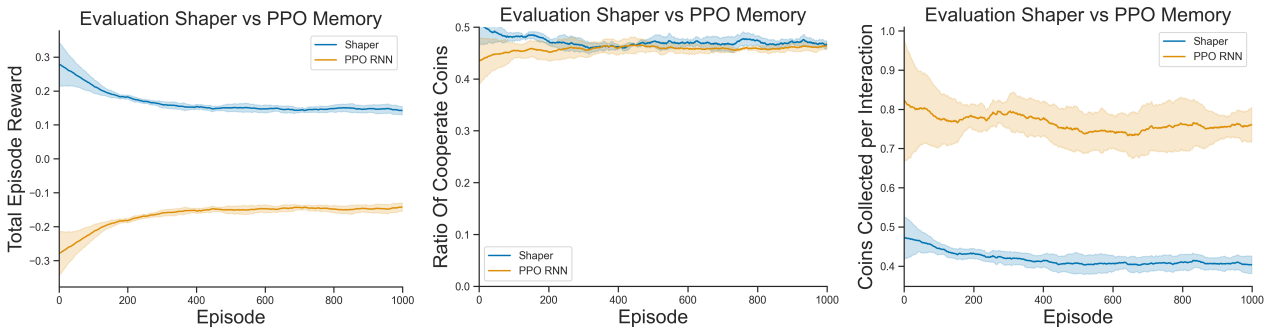
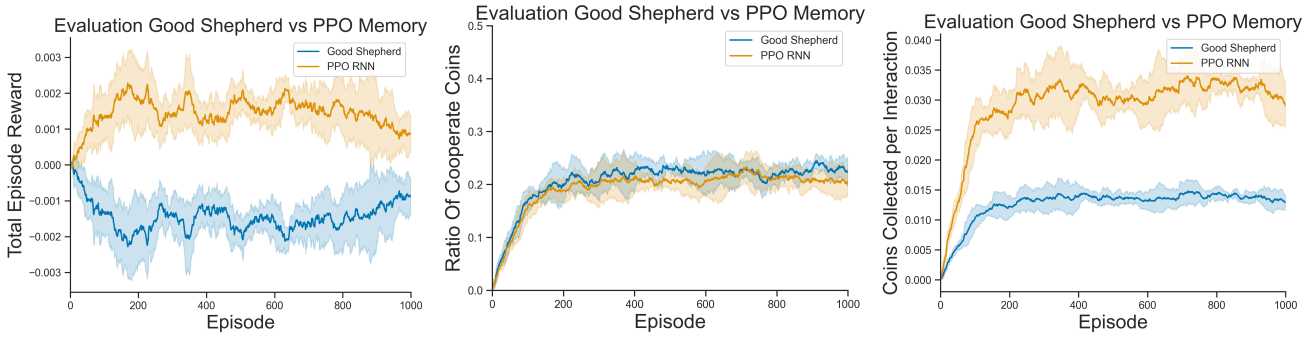


Figure 20: Training results for Shapers vs. PPO RNN in the IMPitM. (Column 1) Reward Per Timestep, (Column 2) the meta-agent's frequency of picking up its own colour coin depending on existing convention, (Column 3) the number of coins picked up per episode, (Column 4) the number of soft resets / successful interactions.



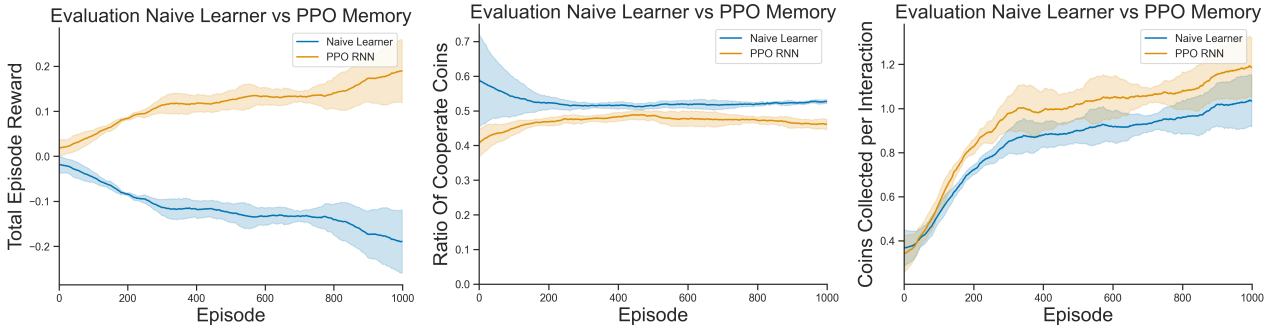
(a) (b) (c)

Figure 21: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.



(a) (b) (c)

Figure 22: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.



(a) (b) (c)

Figure 23: Evaluation results over a single trial (with new co-player) compromising over five seeds for the IPDitM. (a) Mean reward per timestep, (b) mean ratio of picking up cooperate coins per soft-reset, (c) total number of coins picked up per soft-reset.

J CROSS-PLAY RESULTS

We also present cross-play for shaping algorithms against each other on the IPD in the Matrix game.

Table 9: Episode reward for a single evaluation trial against different OS shaping methods. Neither agent takes gradient updates, but those with memory SHAPER and M-FOS are able to use memory to change their policy during the trial. We report mean and std over 5 seeds.

	SHAPER	GS	M-FOS
SHAPER	16.48 ± 0.88	28.61 ± 1.82	7.32 ± 0.34
GS	20.23 ± 1.27	0 ± 0	1.91 ± 0.27
M-FOS	5.08 ± 0.36	1.35 ± 0.28	16.25 ± 0.95

K VARIANCE OVER SEEDS

Here we also report the scores for each game by median.

Table 10: Converged episode reward per episode (meta-agent, co-player) for agents trained with Naive Learners on the CoinGame, IPDitM and IMPitM. The median is reported across 100 seeds with standard error of mean.

	CoinGame		IPD in the Matrix		IMP in the Matrix	
SHAPER	**3.46 ± 0.66**,	**−1.73 ± 0.09**	**22.29 ± 0.11**,	**21.99 ± 0.11**	**0.09 ± 0.03**,	**−0.09 ± 0.03**
M-FOS (ES)	3.19 ± 0.09,	3.28 ± 0.11	10.47 ± 0.35,	25.50 ± 0.33	**0.11 ± 0.02**,	**−0.11 ± 0.02**
M-FOS (RL)	0.24 ± 0.14,	0.819 ± 0.08	7.39 ± 0.08,	7.29 ± 0.05	0.07 ± 0.02,	−0.07 ± 0.02
GS	**4.48 ± 0.14**,	**−2.63 ± 0.12**	15.24 ± 0.19,	6.84 ± 0.11	0.00 ± 0.00,	0.00 ± 0.00
PT-NL	0.07 ± 0.15,	1.48 ± 0.26	6.41 ± 0.12,	6.89 ± 0.13	−0.13 ± 0.07,	0.13 ± 0.07
CT-NL	0.34 ± 0.66,	0.09 ± 0.93	6.03 ± 0.02,	5.07 ± 0.18	−0.11 ± 0.02,	0.11 ± 0.02

L HYPER-PARAMETERS

We used the Jax library [5] with the Haiku framework [16] to implement our neural networks. For the Evolution strategies, we relied on the Evosax library [23]. Our experiments were performed on NVIDIA A100, A40 and V100 GPUs.

L.1 Implementation Details

We performed hyperparameter optimisation over GS, M-FOS and Shaper - evaluating network sizes (8, 16, 32), co-player learning rates ($2.5e^{-3}$, $2e^{-2}$, 1^{e-1}, 1), co-player discount (0.96, 0.98, 0.99), and population size (128, 256, 512, 1000). We report best parameters in the Appendix L.

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
Torso GRU Size	[25]
Length of Meta-Episode	100
Length of Inner Episode	100
Number of Generations	5000
Batch Size	100
Population Size	1000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 11: Hyperparameters for Shaper in Iterated Prisoner's Dilemma

Hyperparameter	Value
Number of Minibatches	4
Number of Epochs	2
Gamma	0.96
GAE Lambda	0.95
PPP clipping epsilon	0.2
Value Coefficient	0.5
Clip Value	True
Max Gradient Norm	0.5
Entropy Coefficient Start	0.02
Entropy Coefficient Horizon	2000000
Entropy Coefficient End	0.001
Learning rate	1
ADAM epsilon	1e-5

Table 12: Hyperparameters for Tabular-PPO in Iterated Prisoner's Dilemma

Hyperparameter	Value
Number of Actor Hidden Layers	2
Number of Critic Hidden Layers	2
Network Hidden Size	[16, 16]
Length of Meta-Episode	100
Length of Inner Episode	100
Number of Generations	5000
Batch Size	100
Population Size	1000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 13: Hyperparameters for GS in Iterated Prisoner's Dilemma

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
Actor GRU Hidden Size	16
Critic GRU Hidden Size	16
Meta-Agent Gru Hidden Size	16
Hidden Layer Size	16
Length of Meta-Episode	100
Length of Inner Episode	100
Number of Generations	5000
Batch Size	100
Population Size	1000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 14: Hyperparameters for M-FOS in Iterated Prisoner's Dilemma

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
Torso Gru Size	[16]
Length of Meta-Episode	600
Length of Inner Episode	16
Number of Generations	3000
Batch Size	100
Population Size	4000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 15: Hyperparameters for SHAPER in Iterated Matching Pennies

Hyperparameter	Value
Number of Minibatches	8
Number of Epochs	2
Gamma	0.96
GAE Lambda	0.95
PPO clipping epsilon	0.2
Value Coefficient	0.5
Clip Value	True
Max Gradient Norm	0.5
Anneal Entropy	False
Entropy Coefficient Start	0.02
Entropy Coefficient Horizon	2000000
Entropy Coefficient End	0.001
LR Scheduling	False
Learning Rate	0.005
ADAM Epsilon	1e-5
With CNN	False

Table 16: Hyperparameters for PPO Memory and Tabular in the CoinGame

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
Hidden Size	[16]
Length of Meta-Episode	600
Length of Inner Episode	16
Number of Generations	3000
Batch Size	100
Population Size	4000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
Actor GRU Hidden Size	16
Critic GRU Hidden Size	16
Meta-Agent Gru Hidden Size	16
Hidden Layer Size	16
Length of Meta-Episode	100
Length of Inner Episode	100
Number of Generations	5000
Batch Size	100
Population Size	1000
OpenES sigma init	0.04
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.01
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 17: Hyperparameters for M-FOS in CoinGame

Hyperparameter	Value
Number of Actor Hidden Layers	1
Number of Critic Hidden Layers	1
GRU Hidden Size	32
Kernel Shape	[3,3]
Hidden Layer Size	16
Length of Meta-Episode	500
Length of Inner Episode	128
Number of Generations	1000
Batch Size	50
Population Size	1000
OpenES sigma init	0.075
OpenES sigma decay	0.999
OpenES sigma limit	0.01
OpenES init min	0.0
OpenES init max	0.0
OpenES clip min	-1e10
OpenES clip max	1e10
OpenES lrate init	0.05
OpenES lrate decay	0.9999
OpenES lrate limit	0.001
OpenES beta 1	0.99
OpenES beta 2	0.999
OpenES eps	1e-8

Table 18: Hyperparameters for SHAPER in * in the Matrix games

Hyperparameter	Value
Number of Minibatches	8
Number of Epochs	2
Gamma	0.96
GAE Lambda	0.95
PPO clipping epsilon	0.2
Value Coefficient	0.5
Clip Value	True
Max Gradient Norm	0.5
Anneal Entropy	False
Entropy Coefficient Start	0.02
Entropy Coefficient Horizon	2000000
Entropy Coefficient End	0.001
LR Scheduling	False
Learning Rate	0.005
ADAM Epsilon	1e-5
With CNN	True
Output Channels	16
Kernel Shape	[3,3]

Table 19: Hyperparameters for PPO Memory in * in the Matrix games

Table 20: Number of Parameters for each environment

	Matrix Games	Coin Game	*in The Matrix
SHAPER	1104	2272	21896
GS	416	688	3892
M-FOS	5360	6896	29024

M FAQ

Why didn't you use meltingpot directly?

We rely heavily on using Jax and its computational efficiencies. Therefore, we need our environments vectorised, which is not the case for meltingpot.

What are implementation differences between your implementation and meltingpot?

Generally, meltingpot, though a gridworld, has pixel-based observations whereas our environment provides access to the grid directly. In IPDitM, the meltingpot environment is 23x15, whereas our grid is 8x8. A size we chose to be as large as possible whilst still being able to optimise the methods given compute limitations. Meltingpot has walls placed within the environment, whereas ours does not. While our environment is smaller, we add additional stochasticity by randomising the coin positions, which are fixed in Meltingpot. Finally, unlike the meltingpot environment, agents spawned with no coins (as opposed to one of each), this made it easier for agents to choose pure cooperate or defect strategies. We found that randomising coin positions important as even large environments representing POMDPs with insufficient stochasticity, such as the The Starcraft Multi-Agent Challenge, can be solved without memory[10, 36]. We found similar evidence in the IPDitM. In IMPitM, the same differences hold.

What compute resources did you use?

We had access to 32 A40s, 32 A100s, distributed over 8-GPU machines. An experiment is distributed over 8 GPUs. Training GS, M-FOS or SHAPER on IPDitM takes approximately 5 days respectively.

What frameworks did you use?

We used the Jax library (Bradbury et al., 2018) with the Haiku framework (Hennigan et al., 2020) to implement our neural networks. We use the Evosax library [23] for the Evolution Strategies method and have adapted the interface of gymnax [24] for our environment implementations.

Are there any other differences between Shaper and M-FOS?

In the algorithm definition of M-FOS, there are two action spaces: the meta-action space $\bar{\mathcal{A}}$ and the underlying action space $\mathcal{A}$. The meta-action space consists of the policy parameters of the underlying agent or a conditional vector parameterising this, and the conventional action space is the action space of the game. In SHAPER, the only action space is that of the underlying game, meaning *there is only one agent in SHAPER, whereas there are two agents in M-FOS.* Consequently, SHAPER is a special case of M-FOS.

Moreover, SHAPER's architecture is different from the architectures proposed in M-FOS. M-FOS proposes *two* significantly different architectures [30]. For the CoinGame, in which a simple table cannot represent policies, M-FOS proposes an architecture akin to Hierarchical RL, consisting of the meta-agent and agent. Both the meta-agent and the underlying agent are recurrent neural networks. The underlying agent resets their hidden state after each episode, whereas the meta-agent does not. In this architecture, the meta-agent does not output the full parameterisation of the underlying agent but instead outputs a conditioning variable, which the underlying agent uses as input. M-FOS assumes that outputting a conditioning variable is equivalent to outputting a policy parameterisation. The conditioning variable is fixed during an episode. In contrast, SHAPER only uses one recurrent neural network that does not reset its hidden state after an episode. *Both M-FOS and SHAPER capture context and history; however, SHAPER only needs one agent. Originally, M-FOS only conditions on the past meta-episode. To ensure the fairest comparison, our M-FOS conditions on all past meta-episodes.*

How do the full network sizes of these methods compare, and how does their performance compare when you control for it?

We control for the network sizes in our experiments as much as possible. Note, however, that we performed hyperoptimization over our hidden layer size for M-FOS and we chose the best performing hyperparameters. More specifically, M-FOS and Shaper use the same network architectures. M-FOS has two networks with a hidden size of 16, respectively. Shaper has one network with a hidden size of 32. We report total number of parameters below

Are there any other differences between Shaper and GS?

GS does not use a recurrent agent, and there is no discussion about notions of history or context. Thus GS cannot capture history or context. Even though it is not discussed, the framework is extensible to a recurrent meta-agent. However, that meta-agent only captures history as the hidden state is reset after each episode. By not capturing context, GS fails to shape in zero-sum games, such as the Matching Pennies, which we show in Section 4.

C

JaxMARL: Multi-Agent RL Environments and Algorithms in JAX - Appendices

Appendix

We structure the appendix as follows. Appendix A provides further background on SMAC, Appendix B describes the environments included within JaxMARL, and Appendix C sets out JaxMARL's API. Appendix D explores JaxMARL's Value-Based MARL methods, including relevant implementation details. Appendix E reports our speed comparisons while Appendix F details training and correctness results not included in the main text. Our hyperparameter values are listed in Appendix G.

A Further Background on SMAC

StarCraft is a popular environment for testing RL algorithms. It typically features a centralised controller issuing commands to balance *micromanagement*, the low-level control of individual units, and *macromanagement*, the high level plans for economy and resource management.

SMAC [53], instead, focuses on *decentralised* unit micromanagement across a range of scenarios divided into three broad categories: *symmetric*, where each side has the same units, *asymmetric*, where the enemy team has more units, and *micro-trick*, which are scenarios designed specifically to feature a particular StarCraft micromanagement strategy. SMACv2 [15] demonstrates that open-loop policies can be effective on SMAC and adds additional randomly generated scenarios to rectify SMAC's lack of stochasticity. However, both of these environments rely on running the full game of StarCraft II, which severely increases their CPU and memory requirements. SMAClite [43] attempts to alleviate this computational burden by recreating the SMAC environment primarily in NumPy, with some core components written in C++. While this is much more lightweight than SMAC, it cannot be run on a GPU and therefore cannot be parallelised effectively with typical academic hardware, which commonly has very few CPU cores compared to industry clusters.

B Further Details on Environments

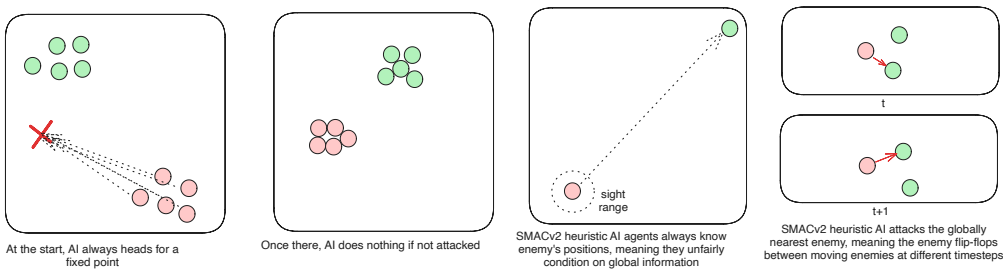
B.1 SMAX

The StarCraft Multi-Agent Challenge (SMAC) is a popular benchmark but has a number of shortcomings. First, as noted and addressed in SMACv2, SMAC is not particularly stochastic. This means that non-trivial win-rates are possible on many SMAC maps by conditioning a policy only on the timestep and agent ID. Additionally, SMAC relies on StarCraft II as a simulator. While this allows SMAC to use the wide range of units, objects and terrain available in StarCraft II, running an entire instance of StarCraft II is slow and memory intensive. StarCraft II runs on the CPU and therefore SMAC's parallelisation is severely limited with typical academic compute.

Using the StarCraft II game engine also constrains environment design. For example, StarCraft II groups units into three races and does not allow units of different races on the same team, limiting the variety of scenarios that can be generated. Secondly, SMAC does not support a competitive self-play setting without significant engineering work. The purpose of SMAX is to address these limitations. It provides access to a simplified SMAC-like, hardware-accelerated, customisable environment that supports self-play and custom unit types. SMAX models units as discs in a continuous 2D space. As listed in Table 4, we include all SMAC(v1) scenarios alongside three inspired by SMAC(v2).

Observations in SMAX are structured similarly to SMAC. Each agent observes the health, previous action, position, weapon cooldown and unit type of all allies and enemies in its sight range. Like SMACv2[15], we use the sight and attack ranges as prescribed by StarCraft II rather than the fixed values used in SMAC.

SMAX and SMAC have different returns. SMAC's reward function, like SMAX's, is split into two parts: one part for depleting enemy health, and another for winning the episode. However, in SMAC, the part which rewards depleting enemy health scales with the number of agents. This is most clearly demonstrated in `27m_vs_30m`, where a random policy gets a return of around 10 out of a maximum of 20 because almost all the reward is for depleting enemy health or killing agents, rather than winning the episode. In SMAX, however, 50% of the total return is always for depleting enemy health, and 50% for winning.



(a) SMAC heuristic AI operation (b) SMACv2 heuristic AI operation

Figure 7: As shown in Figure 7a, SMAC heuristic AI is decentralised, but does not generalise to new start positions. SMACv2 heuristic AI solves the problem of not being able to locate enemies on the map, but does so via conditioning on the global state, which means that some scenarios might be unwinnable. Additionally, the SMACv2 heuristic AI targets the closest enemy, which can lead to flip-flopping between targets. This is shown in Figure 7b

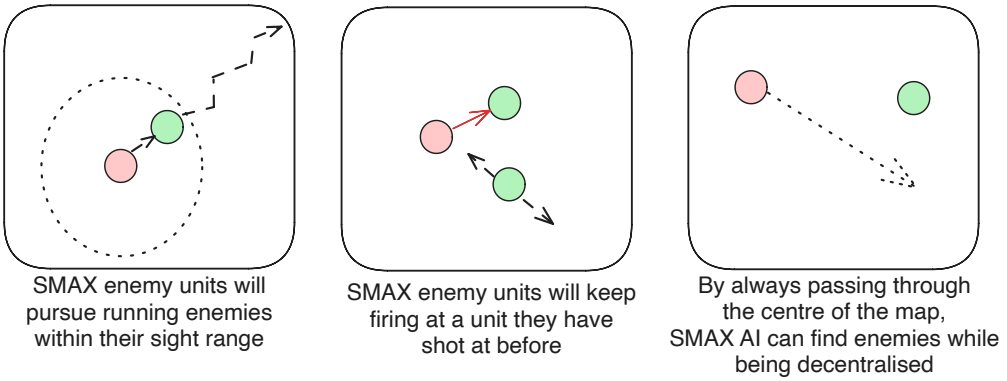


Figure 8: Explanation of the operation of the SMAX heuristic AI.

SMAX also features a different, and more sophisticated, heuristic AI. The heuristic in SMAC simply moves to a fixed location, attacking any enemies it encounters along the way, and the heuristic in SMACv2 globally pursues the nearest agent. Thus the SMAC AI often does not aggressively pursue enemies that run away, and cannot generalise to the SMACv2 start positions, whereas the SMACv2 heuristic AI conditions on global information and is exploitable because of its tendency to flip-flop between two similarly close enemies. SMAC's heuristic AI must be coded in the map editor, which does not provide a simple coding interface. Figure 7 demonstrates these limitations.

In contrast, SMAX features a decentralised heuristic AI that can effectively find enemies without requiring the global information of the SMACv2 heuristic. This guarantees that in principle a 50% win rate is always achievable by copying the decentralised heuristic policy exactly. This means any win-rate below 50% represents a concrete failure to learn. Some of the capabilities of the SMAX heuristic AI are illustrated in the Figure below.

Unlike StarCraft II, where all actions happen in a randomised order in the game loop, some actions in SMAX are simultaneous, meaning draws are possible. In this case both teams get 0 reward.

Like SMAC, each environment step in SMAX consists of eight individual time ticks. SMAX uses a discrete action space, consisting of movement in the four cardinal directions, a stop action, and a shoot action per enemy.

SMAX makes three notable simplifications of the StarCraft II dynamics to reduce complexity. First, zerg units do not regenerate health. This health regeneration is slow at 0.38 health per second, and so likely has little impact on the game. Protoss units also do not have shields. Shields only recharge after 10 seconds out of combat, and therefore are unlikely to recharge during a single micromanagement

task. Protoss units have additional health to compensate for their lost shields. Finally, the available unit types are reduced compared to SMAC. SMAX has no medivac, colossus or baneling units. Each of these unit types has special mechanics that were left out for the sake of simplicity. For the SMACv2 scenarios, the start positions are generated as in SMACv2, with the small difference that the 'surrounded' start positions now treat allies and enemies identically, rather than always spawning allies in the middle of the map. This symmetry guarantees that a 50% win rate is always achievable.

Collisions are handled by moving agents to their desired location first and then pushing them out from one another.

Table 4: SMAX scenarios. The first section corresponds to SMAC scenarios, while the second corresponds to SMACv2.

Scenario	Ally Units	Enemy Units	Start Positions
2s3z	2 stalkers and 3 zealots	2 stalkers and 3 zealots	Fixed
3s5z	3 stalkers and 5 zealots	3 stalkers and 5 zealots	Fixed
5m_vs_6m	5 marines	6 marines	Fixed
10m_vs_11m	10 marines	11 marines	Fixed
27m_vs_30m	27 marines	30 marines	Fixed
3s5z_vs_3s6z	3 stalkers and 5 zealots	3 stalkers and 6 zealots	Fixed
3s_vs_5z	3 stalkers	5 zealots	Fixed
6h_vs_8z	6 hydralisks	8 zealots	Fixed
smacv2_5_units	5 uniformly randomly chosen	5 uniformly randomly chosen	SMACv2-style
smacv2_10_units	10 uniformly randomly chosen	10 uniformly randomly chosen	SMACv2-style
smacv2_20_units	20 uniformly randomly chosen	20 uniformly randomly chosen	SMACv2-style

B.2 Spatial-Temporal Representations of Matrix Games (STORM)

This environment features directional agents within an 8x8 grid world with a restricted field of view. For a visual description, see Figure 9. Agents cannot move backwards or share the same location. Collisions are resolved by either giving priority to the stationary agent or randomly if both are moving. Agents collect two unique resources: *cooperate* and *defect* coins. Once an agent picks up any coin, the agent's colour shifts, indicating its readiness to interact. The agents can then release an *interact* beam directly ahead; when this beam intersects with another ready agent, both are rewarded based on the specific matrix game payoff matrix. The agents' coin collections determine their strategies. For instance, if an agent has 1 *cooperate* coin and 3 *defect* coins, there is a 25% likelihood of the agent choosing to cooperate. After an interaction, the two agents involved are frozen for five steps, revealing their coin collections to surrounding agents. After five steps, they respawn in a new location, with their coin count set back to zero. Once an episode concludes, the coin placements are shuffled. This grid-based approach to matrix games can be adapted for n-player versions. While STORM is inspired by MeltingPot 2.0, there are noteworthy differences:

- Meltingpot uses pixel-based observations while we allow for direct grid access.
- Meltingpot's grid size is typically 23x15, while ours is 8x8.
- Meltingpot features walls within its layout, ours does not.
- Our environment introduces stochasticity by shuffling the coin placements, which remain static in Meltingpot.
- Our agents begin with an empty coin inventory, making it easier for them to adopt pure cooperate or defect tactics, unlike in Meltingpot where they start with one of each coin.
- MeltingPot is implemented in Lua [27] where as ours is a vectorized implementation in JAX.

We deem the coin shuffling especially crucial because even large environments representing POMDPs, such as SMAC, can be solved without the need for memory if they lack sufficient randomness [15].

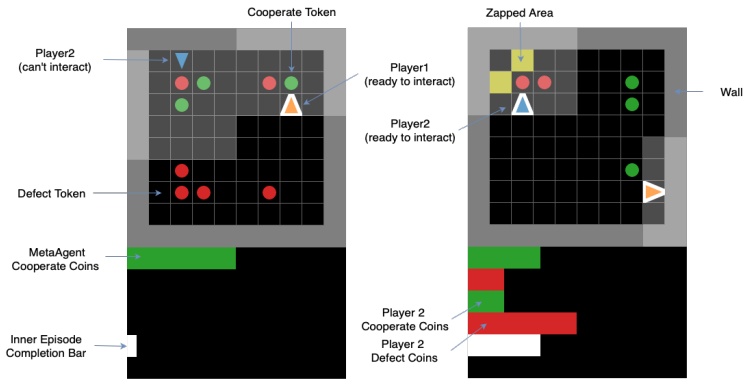


Figure 9: Annotated Image of IPDiTM renders, demonstrating the objects within the game

B.3 Coin Game

Coin Game is a two-player grid-world environment which emulates social dilemmas such as the iterated prisoner's dilemma [57]. Used as a benchmark for the general-sum setting, it expands on simpler social dilemmas by adding a high-dimensional state. Two players, 'red' and 'blue' move in a grid world and are each awarded 1 point for collecting any coin. However, 'red' loses 2 points if 'blue' collects a red coin and vice versa. Thus, if both agents ignore colour when collecting coins their expected reward is 0.

Two agents, 'red' and 'blue', move in a wrap-around grid and collect red and blue coloured coins. When an agent collects any coin, the agent receives a reward of 1. However, when 'red' collects a blue coin, 'blue' receives a reward of -2 and vice versa. Once a coin is collected, a new coin of the same colour appears at a random location within the grid. If a coin is collected by both agents simultaneously, the coin is duplicated and both agents collect it. Episodes are of a set length.

B.4 Switch Riddle

Originally used to illustrate the Differentiable Inter-Agent Learning algorithm [16], Switch Riddle is a simple cooperative communication environment that we include as a debugging tool. n prisoners held by a warden can secure their release by collectively ensuring that each has passed through a room with a light bulb and a switch. Each day, a prisoner is chosen at random to enter this room. They have three choices: do nothing, signal to the next prisoner by toggling the light, or inform the warden they think all prisoners have been in the room. The game ends when a prisoner informs the warden or the maximum time steps are reached. The rewards are +1 if the prisoner informs the warden, and all prisoners have been in the room, -1 if the prisoner informs the warden before all prisoners have taken their turn, and 0 otherwise, including when the maximum time steps are reached. We benchmark using the implementation from [68].

B.5 Hanabi

Hanabi is a fully cooperative partially observable multiplayer card game, where players can observe other players' cards but not their own. To win, the team must play a series of cards in a specific order while sharing only a limited amount of information between players. As reasoning about the beliefs and intentions of other agents is central to performance, it is a common benchmark for ZSC and ad-hoc teamplay research. Our implementation is inspired by the Hanabi Learning Environment [3] and includes custom configurations for varying game settings, such as the number of colours/ranks, number of players, and number of hint tokens. Compared to the Hanabi Learning Environment, which is written in C++ and split over dozens of files, our implementation is a single easy-to-read Python file, which simplifies interfacing with the library and running experiments.

```python
import jax
from jaxmarl import make

key = jax.random.PRNGKey(0)
key, key_reset, key_act, key_step = jax.random.split(key, 4)

# Initialise and reset the environment.
env = make('MPE_simple_world_comm_v3')
obs, state = env.reset(key_reset)

# Sample random actions.
key_act = jax.random.split(key_act, env.num_agents)
actions = {agent: env.action_space(agent).sample(key_act[i]) \
            for i, agent in enumerate(env.agents)}

# Perform the step transition.
obs, state, reward, done, infos = env.step(key_step, state, actions)
```

Figure 10: An example of JaxMARL's API, which is flexible and easy-to-use.

B.6 Overcooked

Inspired by the popular videogame of the same name, Overcooked is commonly used for assessing fully cooperative and fully observable Human-AI task performance. The aim is to quickly prepare and deliver soup, which involves putting three onions in a pot, cooking the soup, and serving it into bowls. Two agents, or *cooks*, must coordinate to effectively divide the tasks to maximise their common reward signal. Our implementation mimics the original from Overcooked-AI [10], including all five original layouts and a simple method for creating additional ones. For a discussion on the limitations of the Overcooked-AI environment, see [33].

B.7 Multi-Agent Particle Environments (MPE)

The multi-agent particle environments feature a 2D world with simple physics where particle agents can move, communicate, and interact with fixed landmarks. Each specific environment varies the format of the world and the agents' abilities, creating a diverse set of tasks that include both competitive and cooperative settings. We implement all the MPE scenarios featured in the PettingZoo library and the transitions of our implementation map exactly to theirs. We additionally include a fully cooperative predator-prey variant of *simple tag*, presented in [49]. The code is structured to allow for straightforward extensions, enabling further tasks to be added.

B.8 Multi-Agent Brax (MABrax)

MABrax is a derivative of Multi-Agent MuJoCo [49], an extension of the MuJoCo Gym environment [62] that is commonly used for benchmarking continuous multi-agent robotic control. Our implementation utilises Brax[19] as the underlying physics engine and includes five of *Multi-Agent MuJoCo*'s multi-agent factorisation tasks, where each agent controls a subset of the joints and only observes the local state. The included tasks, illustrated in Figure 1, are: `ant_4x2`, `halfcheetah_6x1`, `hopper_3x1`, `humanoid_9|8`, and `walker2d_2x3`. The task descriptions mirror those from Gymnasium-Robotics [13].

C JaxMARL's API

The interface of JaxMARL is inspired by PettingZoo [61] and Gymnax. We designed it to be a simple and easy-to-use interface for a wide-range of MARL problems. An example of instantiating an environment from JaxMARL's registry and executing one transition is presented in Figure 10. As JAX's JIT compilation requires pure functions, our `step` method has two additional inputs compared to PettingZoo's. The `state` object stores the environment's internal state and is updated with each call to `step`, before being passed to subsequent calls. Meanwhile, `key_step` is a pseudo-random key, consumed by JAX functions that require stochasticity. This key is separated from the internal state for clarity.

Similar to PettingZoo, the remaining inputs and outputs are dictionaries keyed by agent names, allowing for differing action and observation spaces. However, as JAX's JIT compilation requires arrays to have static shapes, the total number of agents in an environment cannot vary during an

episode. Thus, we do not use PettingZoo's *agent iterator*. Instead, the maximum number of agents is set upon environment instantiation and any agents that terminate before the end of an episode pass dummy actions thereafter. As asynchronous termination is possible, we signal the end of an episode using a special `"__all__"` key within `done`. The same dummy action approach is taken for environments where agents act asynchronously (e.g. turn-based games).

To ensure clarity and reproducibility, we keep strict registration of environments with suffixed version numbers, for example "MPE Simple Spread V3". Whenever JaxMARL environments correspond to existing CPU-based implementations, the version numbers match.

D Value-Based MARL Methods and Implementation details

Key features of our framework include parameter sharing, a recurrent neural network (RNN) for agents, an epsilon-greedy exploration strategy with linear decay, a uniform experience replay buffer, and the incorporation of Double Deep Q-Learning (DDQN) [64] techniques to enhance training stability. We stored the replay buffer in GPU memory using Flashbax [63].

Unlike PyMARL, we use the Adam optimizer as the default optimization algorithm. Below is an introduction to common value-based MARL methods.

IQL (Independent Q-Learners) is a straightforward adaptation of Deep Q-Learning to multi-agent scenarios. It features multiple Q-Learner agents that operate independently, optimizing their individual returns. This approach follows a decentralized learning and decentralized execution pipeline.

VDN (Value Decomposition Networks) extends Q-Learning to multi-agent scenarios with a centralized-learning-decentralized-execution framework. Individual agents approximate their own action's Q-Value, which is then summed during training to compute a jointed Q_{tot} for the global state-action pair. Back-propagation of the global DDQN loss in respect to a global team reward optimizes the factorization of the jointed Q-Value.

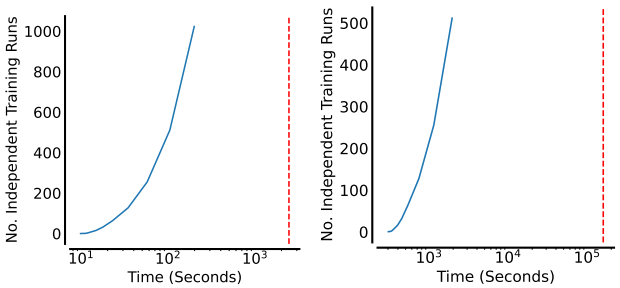
QMIX improves upon VDN by relaxing the full factorization requirement. It ensures that a global $argmax$ operation on the total Q-Value (Q_{tot}) is equivalent to individual $argmax$ operations on each agent's Q-Value. This is achieved using a feed-forward neural network as the mixing network, which combines agent network outputs to produce Q_{tot} values. The global DDQN loss is computed using a single shared reward function and is back-propagated through the mixer network to the agents' parameters. Hypernetworks generate the mixing network's weights and biases, ensuring non-negativity using an absolute activation function. These hypernetworks are two-layered multi-layer perceptrons with ReLU non-linearity.

Issues found when using Q-Learning in an end-to-end GPU setting. As discussed in the paper's results section, PPO demonstrates a clear advantage over Q-Learning for our benchmarked environments, both in agent performance and training runtime. The speed differential is caused by the optimal sampling/replay ratio for Q-Learning methods becoming rapidly unbalanced as the number of parallel environments increases, which requires us to use fewer parallel environments than we use with PPO. PPO also has a major advantage over Q-Learning in that it does not use a replay buffer, which can occupy a significant amount of GPU memory. Secondly, our experiments empirically showed PPO to be more stable during training.

A possible workaround is to increase the replay ratio by performing multiple update steps per training episode, which nevertheless affects computational efficiency. A better solution is to implement a distributed framework, separating the learning and sampling process, which is also out-of-scope for this work.

E Speed Comparison

The runs reported in Figures 3 and 5(c) were all run on the same system featuring two NVIDIA GeForce RTX 4090s (although only one was used for training), an Intel(R) Xeon(R) Silver 4316 CPU (20 cores with 40 threads), and 132 GB of RAM. We report the average environment steps per second over the entire RL training process, which for JaxMARL includes any compilation time. For Table 3, all results were collected on a single NVIDIA A100 GPU and AMD EPYC 7763 64-core processor. Environments were rolled out for 1000 sequential steps.



(a) MPE Simple Spread (b) SMAX `2s3z`

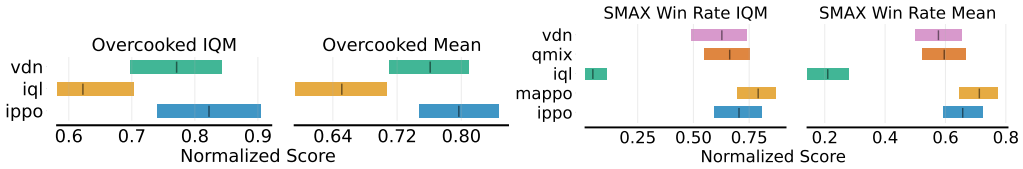
Figure 11: Time taken to train a varying number of seeds in parallel on the same device for JaxMARL IPPO (in blue) compared to the time taken to train one seed with MARLLIB (shown as the red dashed line)

In Figure 11 we repeat the analysis, reported in the main paper for QMIX, of JaxMARL's ability to train multiple seeds in parallel for IPPO. Training this way allows training agents many thousands of times faster, with a 12500x speed up in the MPE simple spread environment.

F Training & Correctness Results

F.1 Overcooked

We train IPPO, VDN and IQL agents in Overcooked and present their aggregate performance in Figure 12a. IPPO performs better than the Q-Learning methods in inter-quartile mean and mean, in line with our more general findings. During training, we use the same shaped reward as stated in the original Overooked paper [10], which is added to the score of the game with a factor that is decayed from 1 to 0 during the first half of training. We don't train MAPPO and QMIX for this task because, in Overcooked, agents can observe the entire state of the map. Therefore, there is no partial observability that can be improved through centralized training. We demonstrate correspondence by training an IPPO policy with JaxMARL on our implementation and evaluating the policy over 10 rollouts for both our Overcooked implementation and the original. Results are shown in Figure 14 with the similarity in performance demonstrating their equivalence.



(a) Overcooked aggregate performance (b) SMAX aggregate performance

Figure 12: Aggregate performance in Overcooked and SMAX for a range of algorithms. Performance is aggregated across 10 seeds and error bars represent 95% bootstrapped confidence intervals as recommended in [2].

F.2 MABrax

The performance of IPPO on `ant_4x2`, `humanoid_9|8`, `hopper_3x1` and `walker2d_2x3` is reported in Figure 15, with hyperparameters reported in Table 7.

F.3 MPE

Performance of Q-Learning baselines in all the MPE scenarios are reported in Figure 17 and Figure 18. The upper row represents cooperative scenarios, with results for all our Q-learning baselines reported.

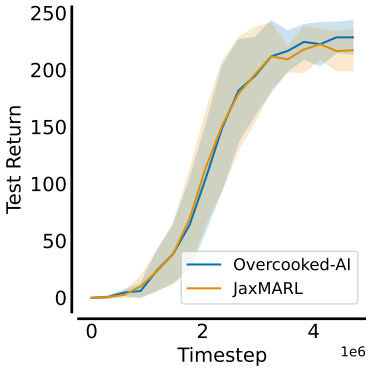


Figure 13: Evaluation performance throughout training of an IPPO policy trained with JaxMARL on our Overcooked Cramped Room scenario implementation and the original [10]. The similarity in performance demonstrates correspondence.

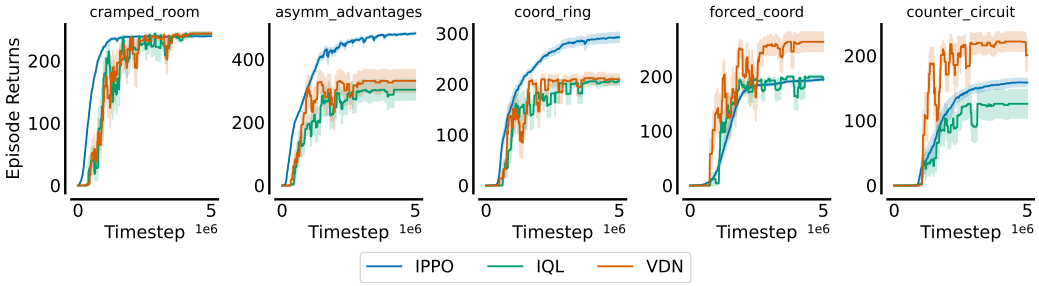


Figure 14: Evaluation of all algorithms in Overcooked scenarios. These scores are obtained after our own hypeparameter tuning, which held to better performances than the using original hyperparameters from Overcooked paper.

The bottom row refers to competitive scenarios, and results for IQL are divided by agent types. Hyperparameters are given in Table 8.

F.4 SMAX

The performance of different algorithms in SMAX versus MAPPO in SMAC is shown in Figure 19. Hyperparameters for IPPO and the Q-learning methods are given in Table 6 and Table 8 respectively. Some maps are significantly more difficult in SMAX, such as `10m_vs_11m`, whereas some are much easier such as `3s_vs_5z`.

F.5 Hanabi

The performances of our implementation of IPPO are JaxMARL's Hanabi for 2-3 players are reported in Table 5, together with the results provided for IPPO in the original Hanabi environment reported by [67].

# Players	Metric	IPPO [67]	IPPO (JaxMARL)
2	Avg.	24.00	23.95
	Best	24.19	24.12
3	Avg.	23.25	23.83
	Best	23.87	24.16

Table 5: Best and Average evaluation scores of the original IPPO implementation [67] and IPPO in JaxMARL's Hanabi.

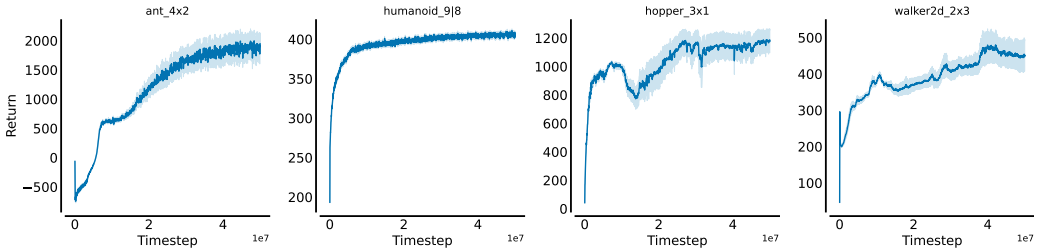


Figure 15: Performance of IPPO on MABrax Tasks

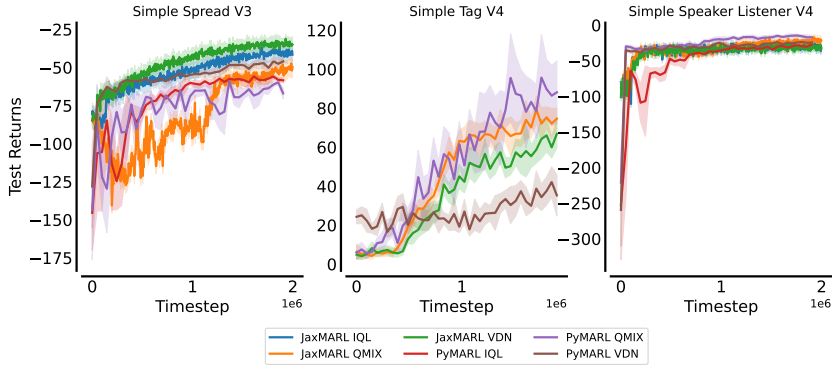


Figure 16: Comparison of the performances of Q-Learning baselines in PyMARL and JaxMARL in two cooperative scenarios of MPE (Spread and Speaker Listener) and one competitive scenario (Simple Tag). For Simple Tag, we pre-trained a prey in JaxMARL and then trained agents to compete with it in both PyMARL and JaxMARL. Despite the small differences in the obtained returns in the two frameworks, the algorithms show similar learning dynamics, and the final ordering is preserved, validating our environment and algorithm implementations.

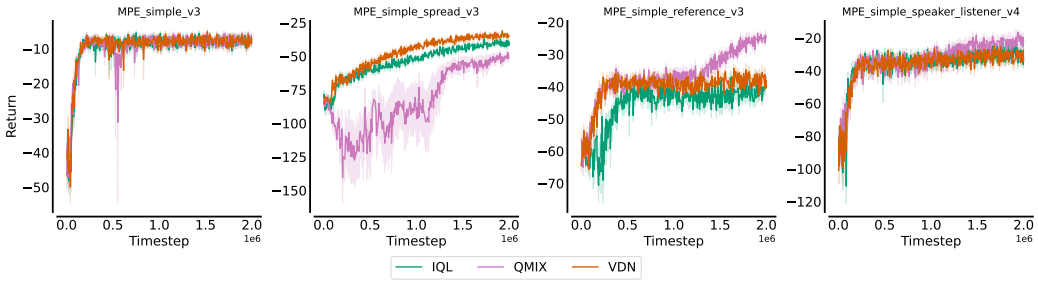


Figure 17: Evaluation of performances of QLearning in all the MPE cooperative scenarios.

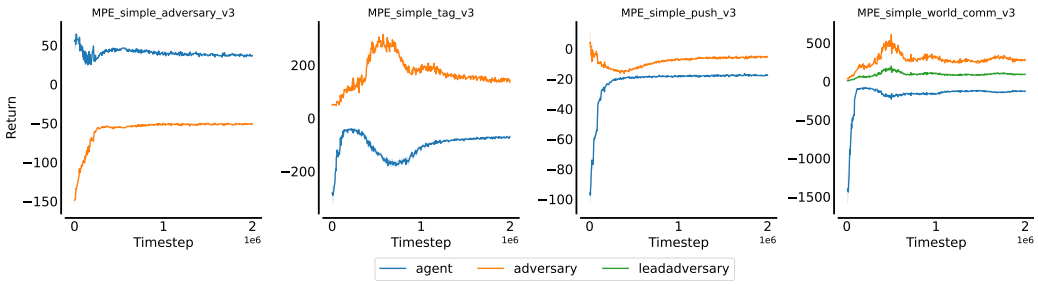


Figure 18: Evaluation of performances of IQL in all the MPE competetive scenarios. All the competetive agents are trained independently together. "Agent" and "Adversary" are teams, not single agents.

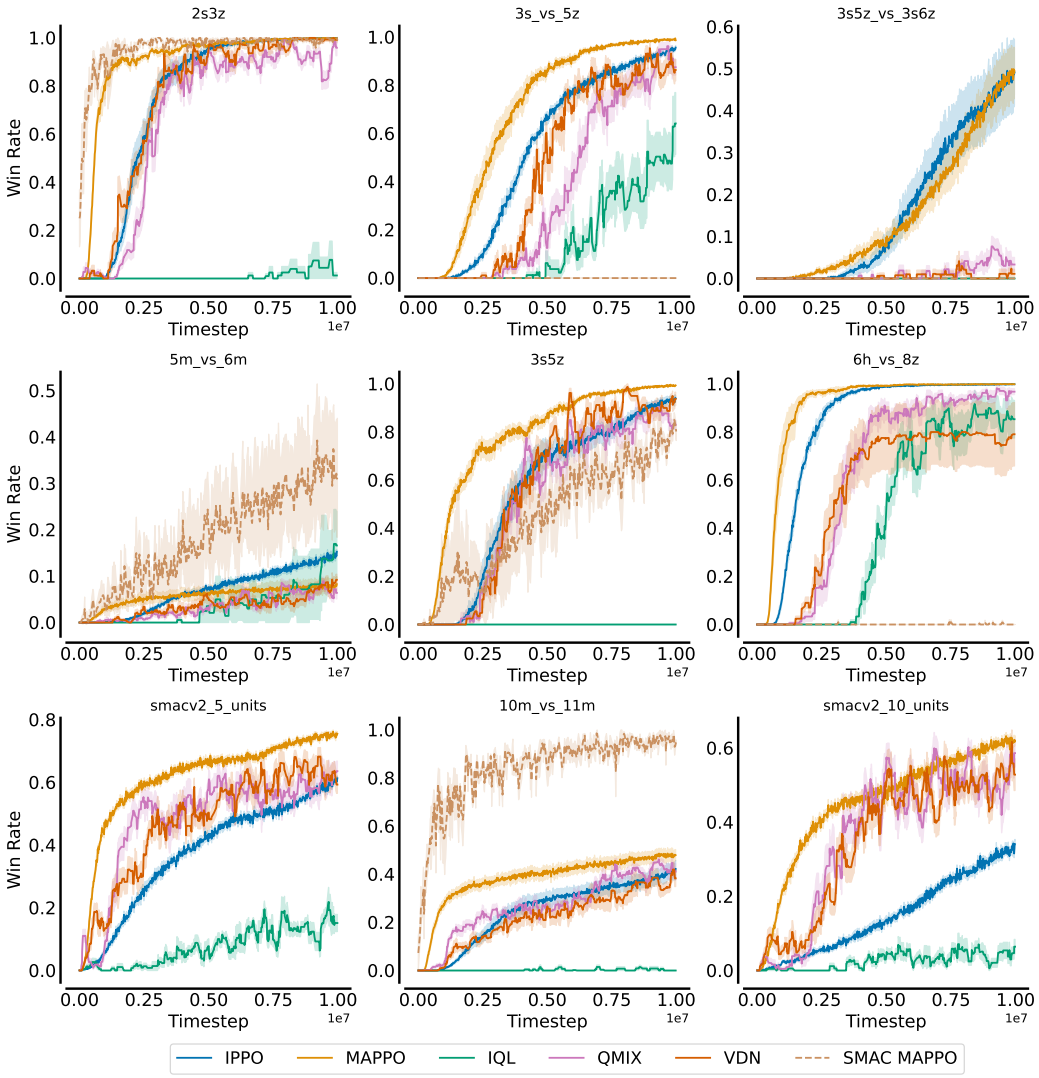


Figure 19: Comparison of IPPO, MAPPO, IQL, QMIX, VDN in SMAX with MAPPO in SMAC.

G Hyperparameters

Table 6: IPPO hyperparameters for MPE, SMAX, Hanabi and Overcooked.

Parameter	MPE	SMAX	Hanabi	Overcooked
PPO				
# training timesteps	1×10^6	1×10^7	1×10^10	5×10^6
# parallel environments	16	64	1024	64
# rollout steps	128	128	128	256
Adam learning rate	5×10^{-4}	4×10^{-3}	5×10^{-4}	5×10^{-4}
Anneal learning rate	True	True	True	True
Update epochs	5	2	4	4
Minibatches per epoch	2	2	4	16
γ	0.99	0.99	0.99	0.99
λ_{GAE}	1.0	0.95	0.95	0.95
Clip range	0.3	0.2	0.2	0.2
Entropy coefficient	0.01	0.0	0.01	0.01
Value loss coefficient	1.0	0.5	0.5	0.5
Maximum gradient norm	0.5	0.5	0.5	0.5
Activation	tanh	relu	relu	relu
Feed-forward network				
Number of layers	2	-	2	2
Fully-connected width	64	-	512	64
Recurrent network				
Hidden width	128	128	128	-
Number of full-connected layers	2	2	2	-
Fully-connected width	128	128	128	-

Table 7: IPPO hyperparameters for MABrax settings.

Parameter	Ant	HalfCheetah	Walker
PPO			
# training timesteps	1×10^8		
# parallel environments	64		
# rollout steps	300		
Adam learning rate	1×10^{-3}	6×10^{-4}	7×10^{-3}
Anneal learning rate	True		
Update Epochs	4		
Minibatches per epoch	4		
γ	0.99		
λ_{GAE}	1.0		
Clip range	0.2		
Entropy coefficient	2×10^{-6}	4.5×10^{-3}	1×10^{-3}
Value loss coefficient	4.5	0.14	1.9
Maximum gradient norm	0.5		
Activation	tanh		
Feed-forward network			
Number of layers	2		
Fully-connected width	64		

Table 8: Q-Learning hyperparameters for MPE, SMAX and Overcooked

Parameter	MPE	SMAX	Overcooked
Q-Learning			
# training timesteps	2×10^6	1×10^7	1×10^5
# parallel environments	8	16	32
# rollout steps	26	128	1
Adam learning rate	1×10^{-3}	5×10^{-5}	7.5×10^{-5}
Anneal learning rate	True	False	True
Buffer size	5×10^3	5×10^3	1×10^5
Buffer batch size	32	32	128
ϵ start	1.0	1.0	1.0
ϵ finish	0.05	0.05	0.05
ϵ decay	0.1	0.1	0.2
Hidden size	64	512	64
Maximum gradient norm	25	10	1
τ	1.0	1.0	1.0
Update epochs	1	8	4
Learning starts at timestep	1×10^4	1×10^4	1×10^3
γ	0.9	0.99	0.99
QMIX specific			
Mixed embedding width	32	64	-
Mixer hypernet width	128	256	-
Mixer initial scale	1×10^{-3}	1×10^{-3}	-

D

Mixtures of Experts Unlock Parameter Scaling for Deep RL - Appendices

A. Experimental details

Unless otherwise specified, in all experiments below we report the interquantile mean after 40 million environment steps; error bars indicate 95% stratified bootstrap confidence intervals (Agarwal et al., 2021). Most of our experiments were run with 20 games from the ALE suite (Bellemare et al., 2013a), as suggested by Fedus et al. (2020). However, for the Atari $100k$ agents (subsection 5.2), we used the standard set of 26 games (Kaiser et al., 2020) to be consistent with the benchmark. Finally, we also ran some experiments with the full set of 60 games. The specific games are detailed below.

20 game subset: AirRaid, Asterix, Asteroids, Bowling, Breakout, DemonAttack, Freeway, Gravitar, Jamesbond, MontezumaRevenge, MsPacman, Pong, PrivateEye, Qbert, Seaquest, SpaceInvaders, Venture, WizardOfWor, YarsRevenge, Zaxxon.

26 game subset: Alien, Amidar, Assault, Asterix, BankHeist, BattleZone, Boxing, Breakout, ChopperCommand, CrazyClimber, DemonAttack, Freeway, Frostbite, Gopher, Hero, Jamesbond, Kangaroo, Krull, KungFuMaster, MsPacman, Pong, PrivateEye, Qbert, RoadRunner, Seaquest, UpNDown.

60 game set: The 26 games above in addition to: AirRaid, Asteroids, Atlantis, BeamRider, Berzerk, Bowling, Carnival, Centipede, DoubleDunk, ElevatorAction, Enduro, FishingDerby, Gravitar, IceHockey, JourneyEscape, MontezumaRevenge, NameThisGame, Phoenix, Pitfall, Pooyan, Riverraid, Robotank, Skiing, Solaris, SpaceInvaders, StarGunner, Tennis, TimePilot, Tutankham, Venture, VideoPinball, WizardOfWor, YarsRevenge, Zaxxon.

B. Hyper-parameters list

Default hyper-parameter settings for DER (Van Hasselt et al., 2019) and DrQ(ϵ) (Kaiser et al., 2020; Agarwal et al., 2021) agents. Table 1 shows the default values for each hyper-parameter across all the Atari games.

Table 1. Default hyper-parameters setting for DER and DrQ(ϵ) agents.

	Atari	
Hyper-parameter	DER	DrQ(ϵ)
Adam's(ϵ)	0.00015	0.00015
Adam's learning rate	0.0001	0.0001
Batch Size	32	32
Conv. Activation Function	ReLU	ReLU
Convolutional Width	1	1
Dense Activation Function	ReLU	ReLU
Dense Width	512	512
Normalization	None	None
Discount Factor	0.99	0.99
Exploration ϵ	0.01	0.01
Exploration ϵ decay	2000	5000
Minimum Replay History	1600	1600
Number of Atoms	51	0
Number of Convolutional Layers	3	3
Number of Dense Layers	2	2
Replay Capacity	1000000	1000000
Reward Clipping	True	True
Update Horizon	10	10
Update Period	1	1
Weight Decay	0	0
Sticky Actions	False	False

Default hyper-parameter settings for DQN (Mnih et al., 2015) and Rainbow (Hessel et al., 2018) agents. Table 2 shows the default values for each hyper-parameter across all the Atari games.

Table 2. Default hyper-parameters setting for DQN and Rainbow agents.

	Atari	
Hyper-parameter	DQN	Rainbow
Adam's (ϵ)	1.5e-4	1.5e-4
Adam's learning rate	6.25e-5	6.25e-5
Batch Size	32	32
Conv. Activation Function	ReLU	ReLU
Convolutional Width	1	1
Dense Activation Function	ReLU	ReLU
Dense Width	512	512
Normalization	None	None
Discount Factor	0.99	0.99
Exploration ϵ	0.01	0.01
Exploration ϵ decay	250000	250000
Minimum Replay History	20000	20000
Number of Atoms	0	51
Number of Convolutional Layers	3	3
Number of Dense Layers	2	2
Replay Capacity	1000000	1000000
Reward Clipping	True	True
Update Horizon	1	3
Update Period	4	4
Weight Decay	0	0
Sticky Actions	True	True

Default hyper-parameter settings for CQL (Kumar et al., 2020) and CQL+C51 (Kumar et al., 2022) offline agents. Table 3 shows the default values for each hyper-parameter across all the Atari games.

Table 3. Default hyper-parameters setting for CQL and CQL+C51 agents.

	Atari	
Hyper-parameter	CQL	CQL+C51
Adam's(ϵ)	0.0003125	0.00015
Batch Size	32	32
Conv. Activation Function	ReLU	ReLU
Convolutional Width	1	1
Dense Activation Function	ReLU	ReLU
Normalization	None	None
Dense Width	512	512
Discount Factor	0.99	0.99
Learning Rate	0.00005	0.0000625
Number of Atoms	0	51
Number of Convolutional Layers	3	3
Number of Dense Layers	2	2
Fixed Replay Capacity	2,500,000 steps	2,500,000 steps
Reward Clipping	True	True
Update Horizon	1	3
Update Period	1	1
Weight Decay	0	0
Replay Scheme	Uniform	Uniform
Dueling	False	True
Double DQN	False	True
CQL coef	0.1	0.1

Default hyper-parameter settings for CNN architecture (Mnih et al., 2015) and Impala-based ResNet (Espeholt et al., 2018) Table 4 shows the default values for each hyper-parameter across all the Atari games.

Table 4. Default hyper-parameters for neural networks.

	Atari	
Hyper-parameter	CNN architecture (Mnih et al., 2015)	Impala-based ResNet (Espeholt et al., 2018)
Observation down-sampling	(84, 84)	(84, 84)
Frames stacked	4	4
Q-network (channels)	32, 64, 64	32, 64, 64
Q-network (filter size)	8 x 8, 4 x 4, 3 x 3	8 x 8, 4 x 4, 3 x 3
Q-network (stride)	4, 2, 1	4, 2, 1
Num blocks	-	2
Use max pooling	False	True
Skip connections	False	True
Hardware	Tesla P100 GPU	Tesla P100 GPU

C. Extra results

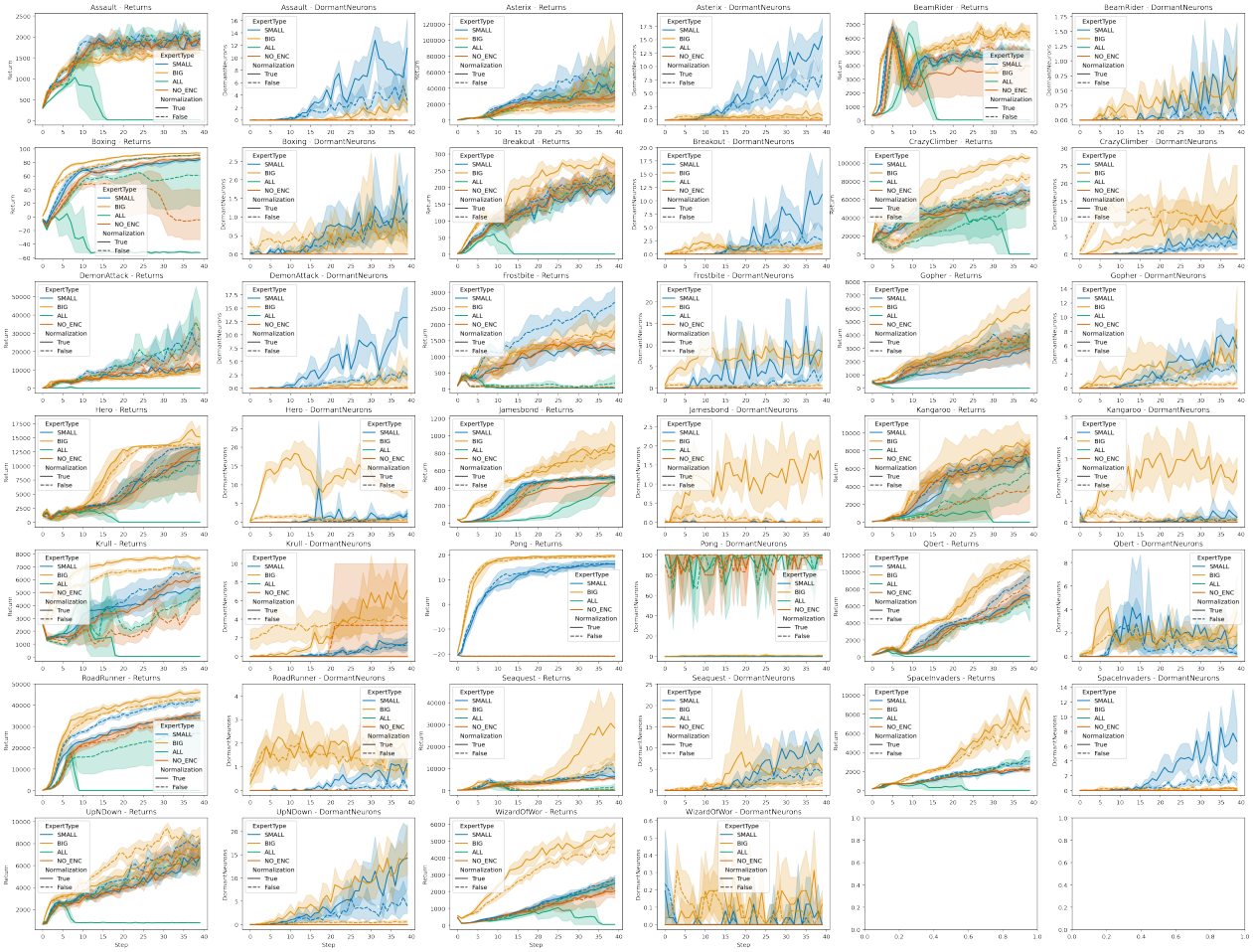


Figure 14. Results for architectural ablations as described in Section 5.3 on DQN. Additionally, we investigate the effect of the normalization that was proposed in the original Soft MoE paper.

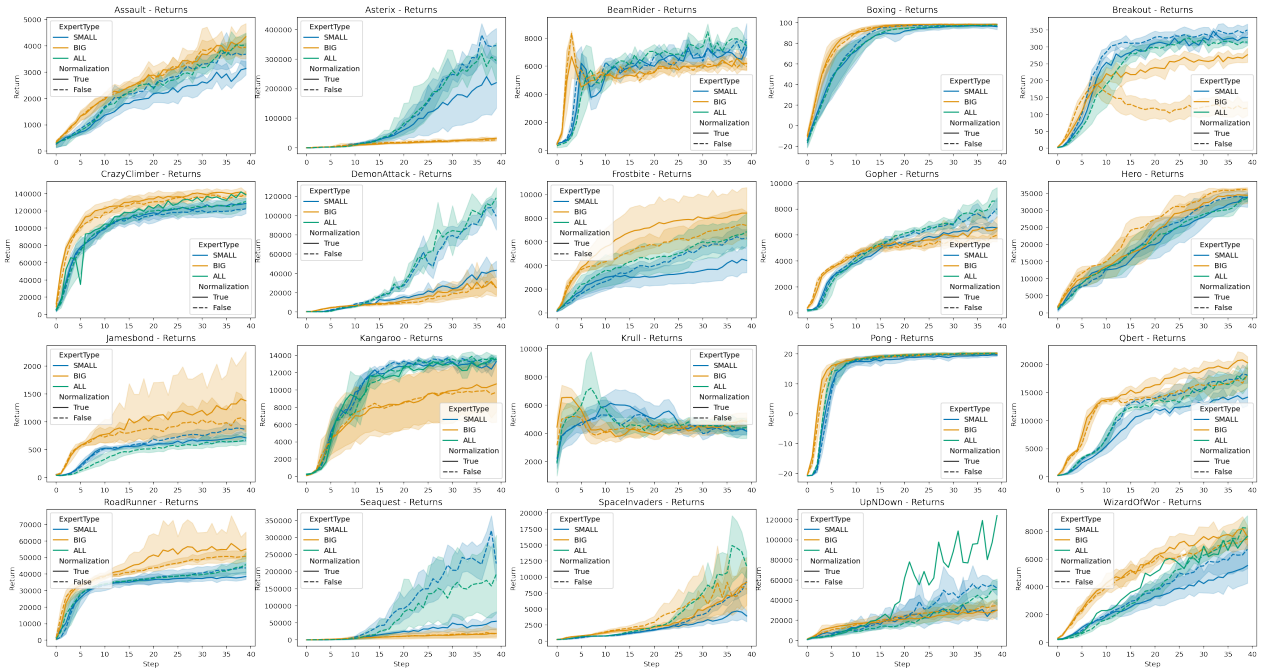


Figure 15. Results for architectural exploration as described in Section 5.3 on Rainbow. Additionally, we investigate the effect of the normalization that was proposed in the original Soft MoE paper.

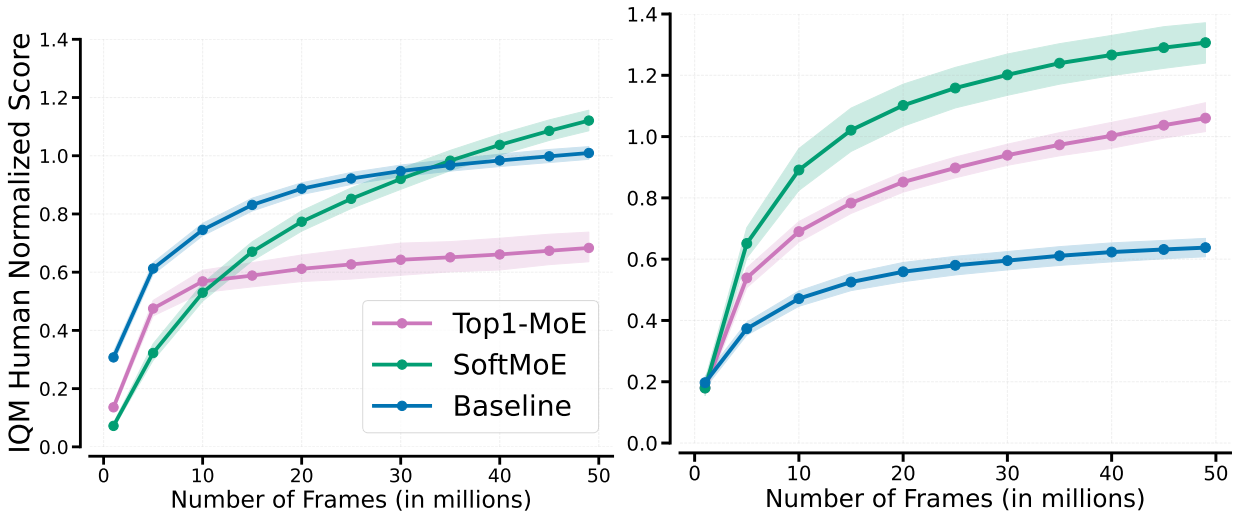


Figure 16. **Normalized performance across** 26 **Atari games for DrQ(ϵ) (left) and DER (right)**, with the ResNet architecture (Espeholt et al., 2018) and 4 experts. Soft MoE not only remains generally stable with more training, but also attains higher final performance. We report interquantile mean performance with error bars indicating 95% confidence intervals.

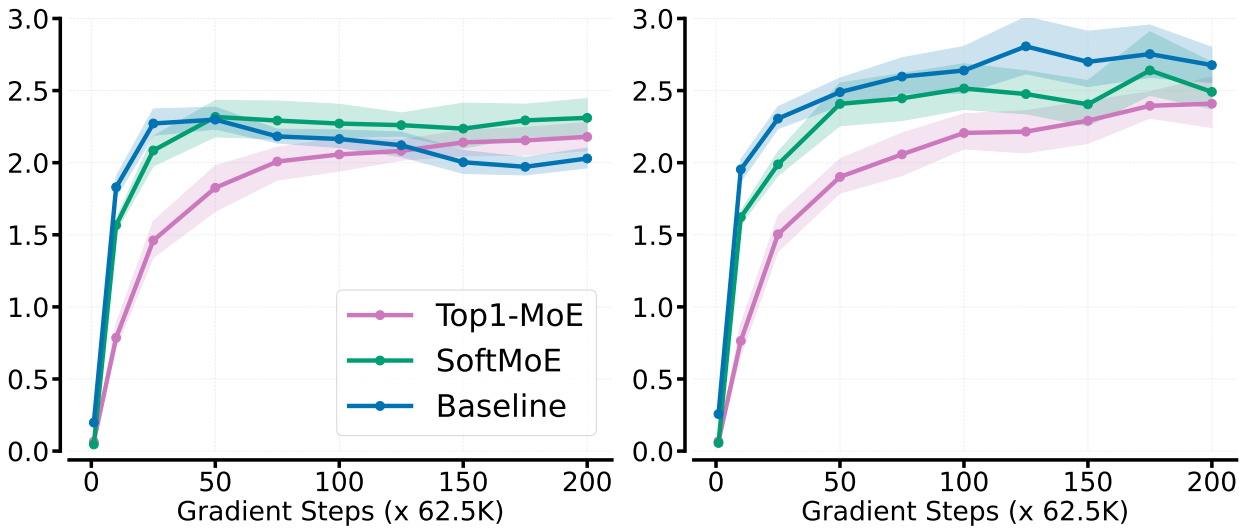


Figure 17. **Normalized performance across** 17 **Atari games for CQL+C51**. x-axis represents gradient steps; no new data is collected. **Left:** 10% and **Right:** 50% uniform replay. We report IQM with 95% stratified bootstrap CIs (Agarwal et al., 2021)

.

D. Varying Impala filter sizes

When dealing with small models, it's common to scale them up to enhance performance. This makes the scaling strategy crucial for balancing accuracy and efficiency. For Convolutional Neural Networks (CNNs), traditional scaling methods usually emphasize model depth, width, and input resolution (Ding et al., 2022), as well as the *filter*. The default filter size is 3x3 for the Impala CNN, and we ran experiments with and without SoftMoE using 4x4 and 6x6 filters to investigate the filter size scaling benefits. In both cases, SoftMoE outperforms the baseline.

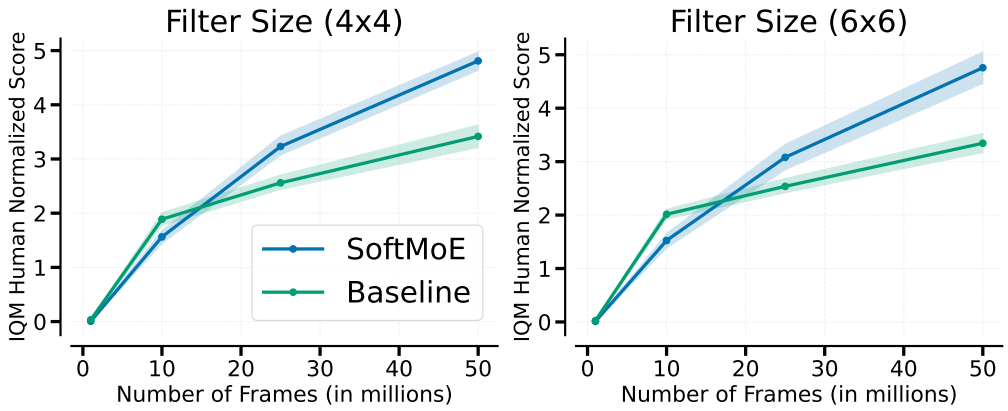


Figure 18. **Normalized performance across** 20 **Atari games with the ResNet architecture.** SoftMoE achieves the best results in both scenarios; default filter size (3x3) is increased to (4x4) and (6x6).

E. Measuring runtime

We plotted IQM performance against wall time, instead of the standard environment frames. SoftMoE and baseline have no noticeable difference in running time, whereas Top1-MoE is slightly faster than both.

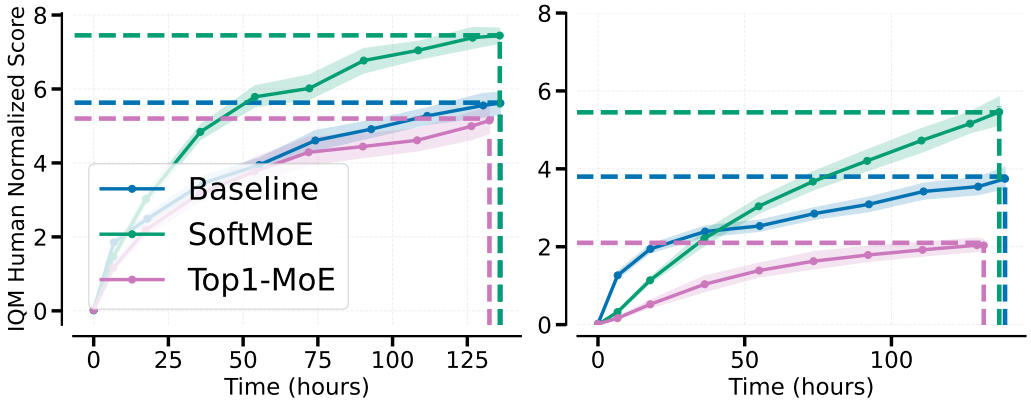


Figure 19. **Measuring wall-time versus IQM of human-normalized scores** in Rainbow over 20 games. **Left:** ImpalaCNN and **Right:** CNN network. Each experiment had 3 independent runs, and the confidence intervals show 95% confidence intervals.

F. Experiments with PPO

Based on reviewer suggestions, we have run some initial experiments with PPO and SAC on MuJoCo. We have not observed significant performance gains nor degradation with SoftMoE; with Top1-MoE we see a degradation in performance, similar to what we observed in our submission. We see a few possible reasons for the lack of improvement with SoftMoE:

1. For ALE experiments, all agents use Convolutional layers, whereas for the MuJoCo experiments (where we ran SAC and PPO) the networks only use dense layers. It is possible the induced sparsity provided by MoEs is most effective when combined with convolutional layers.
2. The suite of environments in MuJoCo are perhaps less complex than the set of experiments in the ALE, so performance with agents like SAC and PPO is somewhat saturated.

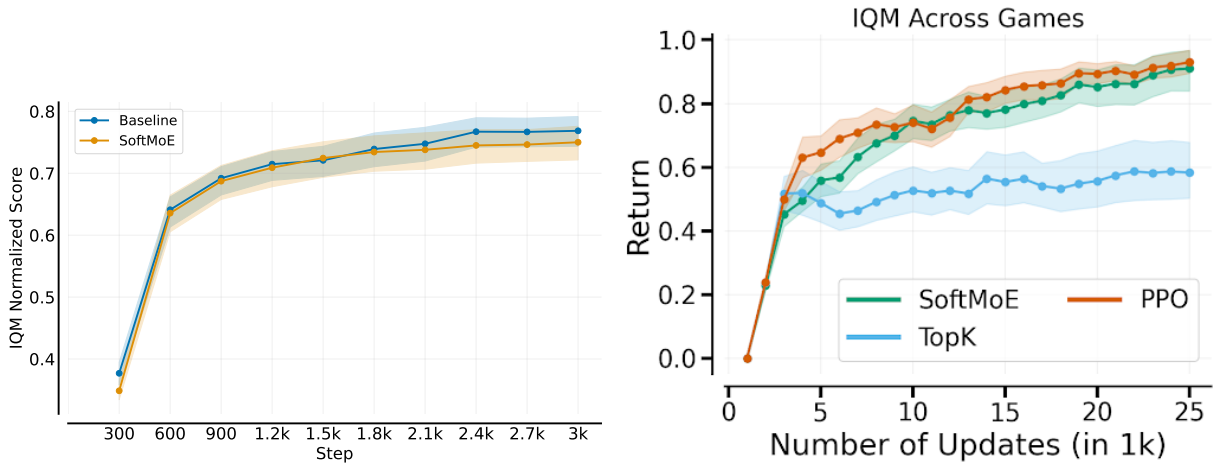


Figure 20. **Left:** Evaluating SAC with SoftMoE on 28 MuJoCo environments and **Right:** Evaluatin PPO on 9 MuJoCo-Brax environments. SoftMoEs seems to provide no gains nor degradation, whereas TopK seems to degrade performance (consistent with paper's findings). MuJoCo scores are normalized between 0 and 1000, with 5 seeds each; error bars indicate 95% stratified bootstrap confidence intervals. MuJoCo-Brax scores are normalized with respect to Jesson et al. (2023).

E

Mixture of Experts in a Mixture of RL settings - Appendices

A Continual RL

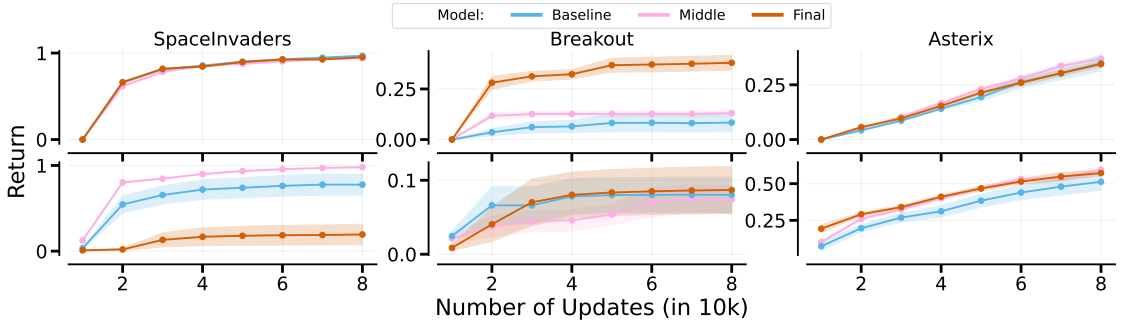


Figure 7: **CRL: Variants with shared parameters across task** do not necessarily improve performance over the baseline. Middle performs better than the baseline, whereas Final does not.

Game	Baseline	Final-Hardcoded	Middle-Hardcoded
SI	0.97 ± 0.01	0.95 ± 0.01	0.94 ± 0.01
BO	0.08 ± 0.05	0.38 ± 0.04	0.13 ± 0.01
Ast	0.35 ± 0.04	0.34 ± 0.01	0.37 ± 0.01
SI-2	0.78 ± 0.12	0.19 ± 0.12	0.98 ± 0.01
BO-2	0.08 ± 0.02	0.09 ± 0.03	0.07 ± 0.02
Ast-2	0.51 ± 0.06	0.57 ± 0.02	0.60 ± 0.02
Total	0.46 ± 0.03	0.42 ± 0.02	$\mathbf{0.52 \pm 0.01}$

Table 1: **CRL: Variants with shared parameters across task** do not necessarily improve performance over the baseline. Middle performs better than the baseline, whereas Final does not.

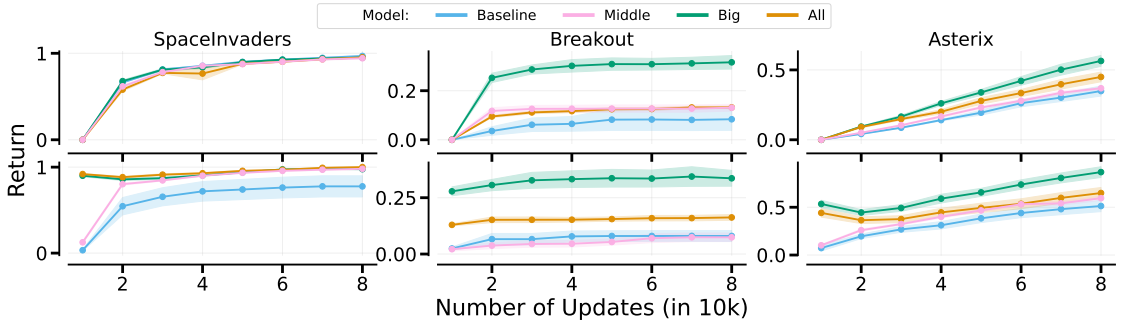


Figure 8: **CRL: Isolated Params: Variants with isolated parameters across task** improve performance over the baseline. Big-Hardcoded works the best.

Game	Baseline	All-Hardcoded	Big-Hardcoded	Middle-Hardcoded
Game	Baseline	Final-Hardcoded	Middle-Hardcoded	Additional-Column
SI	0.97 ± 0.01	0.95 ± 0.01	0.95 ± 0.00	0.94 ± 0.01
BO	0.08 ± 0.05	0.13 ± 0.00	0.32 ± 0.03	0.13 ± 0.01
Ast	0.35 ± 0.04	0.45 ± 0.03	0.56 ± 0.04	0.37 ± 0.01
SI-2	0.78 ± 0.12	1.00 ± 0.00	0.98 ± 0.01	0.98 ± 0.01
BO-2	0.08 ± 0.02	0.16 ± 0.01	0.34 ± 0.04	0.07 ± 0.02
Ast-2	0.51 ± 0.06	0.65 ± 0.06	0.87 ± 0.06	0.60 ± 0.02
Total	0.46 ± 0.03	0.56 ± 0.02	$\mathbf{0.67 \pm 0.02}$	0.52 ± 0.01

Table 2: **CRL: Isolated Params: Variants with isolated parameters across task** improve performance over the baseline. Big-Hardcoded works the best.

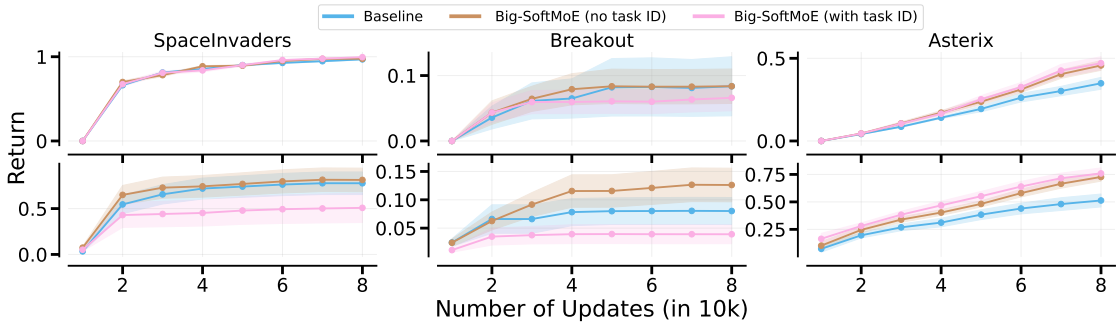


Figure 9: **CRL: Adding Task ID** as input slightly hurts performance for Big-SoftMoE.

Game	Baseline	w/ Task-ID	w/o Task-ID
SI	0.97 ± 0.01	0.98 ± 0.01	0.99 ± 0.01
BO	0.08 ± 0.05	0.08 ± 0.03	0.07 ± 0.02
Ast	0.35 ± 0.04	0.46 ± 0.03	0.47 ± 0.04
SI-2	0.78 ± 0.12	0.81 ± 0.13	0.51 ± 0.16
BO-2	0.08 ± 0.02	0.13 ± 0.03	0.04 ± 0.02
Ast-2	0.51 ± 0.06	0.73 ± 0.04	0.76 ± 0.05
Total	0.46 ± 0.03	$\mathbf{0.53 \pm 0.02}$	0.47 ± 0.02

Table 3: **CRL: Adding Task ID** as input slightly hurts performance for Big-SoftMoE.

Game	Baseline	Big-Hardcoded	Big-SoftGradMoE	Big-SoftMoE
SI	0.97 ± 0.01	0.95 ± 0.00	0.99 ± 0.01	0.98 ± 0.01
BO	0.08 ± 0.05	0.32 ± 0.03	0.12 ± 0.03	0.08 ± 0.03
Ast	0.35 ± 0.04	0.56 ± 0.04	0.45 ± 0.04	0.46 ± 0.03
SI-2	0.78 ± 0.12	0.98 ± 0.01	0.79 ± 0.13	0.81 ± 0.13
BO-2	0.08 ± 0.02	0.34 ± 0.04	0.14 ± 0.03	0.13 ± 0.03
Ast-2	0.51 ± 0.06	0.87 ± 0.06	0.69 ± 0.06	0.73 ± 0.04
Total	0.46 ± 0.03	$\mathbf{0.67 \pm 0.02}$	0.53 ± 0.02	0.53 ± 0.02

Table 4: **Performance of algorithms across games** with total performance.

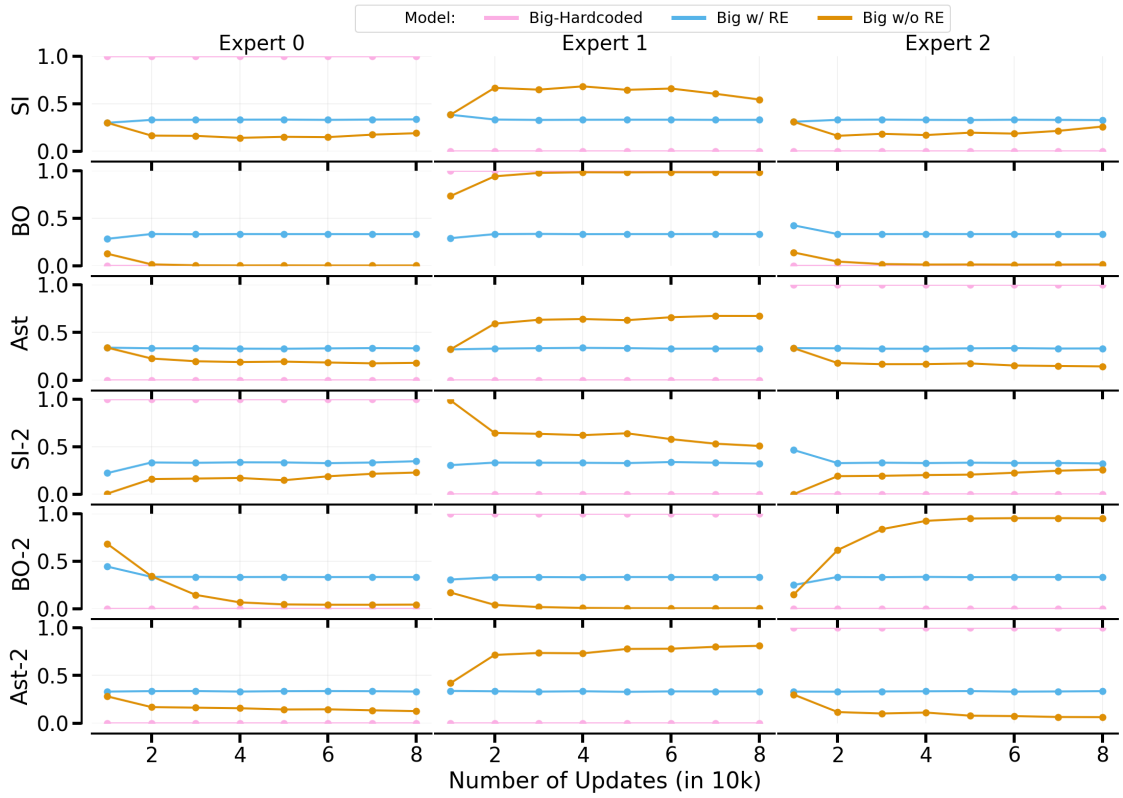


Figure 10: CRL: Big-SoftMoE also specialises in the CRL setting.

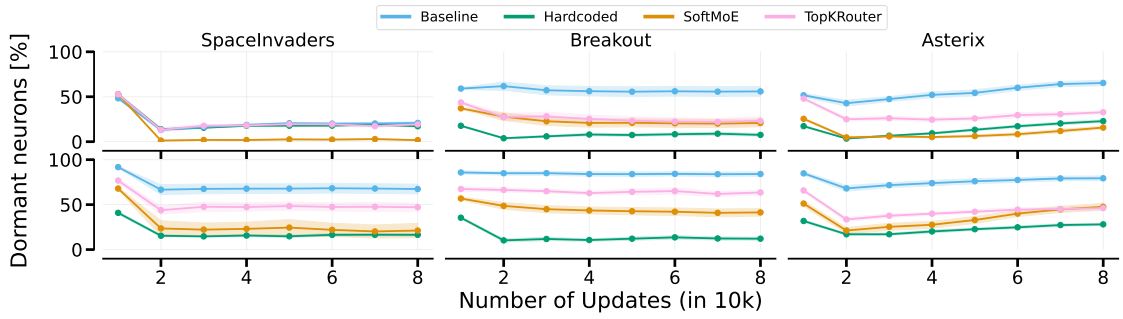


Figure 11: **CRL: MoE variants have less dormant neurons** than the baseline without MoE modules.

Game	Baseline	Big-Hardcoded	Big-TopK	Big-SoftMoE
SI	20.86 ± 1.41	17.19 ± 0.94	19.22 ± 1.78	1.72 ± 0.45
BO	55.94 ± 5.38	7.58 ± 1.44	23.20 ± 4.71	20.86 ± 4.34
Ast	65.31 ± 3.13	22.89 ± 1.49	32.73 ± 1.72	15.70 ± 1.65
SI-2	67.34 ± 5.38	16.41 ± 1.22	47.11 ± 4.49	21.17 ± 7.97
BO-2	84.06 ± 2.41	12.11 ± 1.61	63.52 ± 2.47	41.33 ± 4.19
Ast-2	79.30 ± 2.65	28.05 ± 1.27	46.17 ± 1.76	47.58 ± 3.94
Total	62.14 ± 1.53	$\mathbf{17.37 \pm 0.83}$	38.66 ± 1.59	24.73 ± 1.58

Table 5: **CRL Dormant Neurons for Big router variants**. The hardcoded variant has the least dormant neurons

A.1 Hard-switching based on Gradient Similarity

We also attempt to route when the gradient similarity drops below a threshold. However, this proved difficult as the thresholds depend on the architecture and expert might switch too early, as shown in Figure 12.

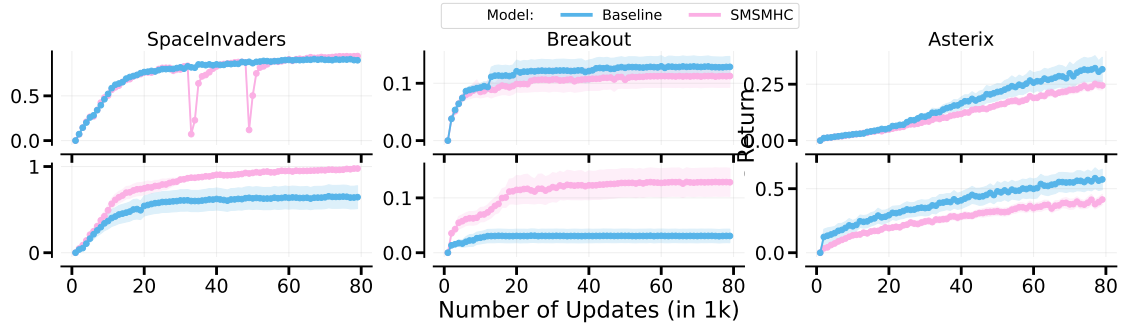


Figure 12: CRL: As shown here with SMSMHC, finding a correct threshold for gradient switching proves difficult, as the experts might switch too early, as in this case it already switches twice during SpaceInvaders (see the dips)

Game	Baseline	Big-Hardcoded	Big-TopK	Big-SoftMoE
SI	21.95 ± 1.15	22.34 ± 1.12	19.61 ± 2.10	2.89 ± 0.40
BO	22.50 ± 3.02	7.89 ± 1.21	20.39 ± 1.36	3.91 ± 1.23
Ast	72.66 ± 1.59	29.14 ± 1.11	44.30 ± 2.35	26.02 ± 1.21
Total	39.04 ± 0.88	19.79 ± 0.92	28.10 ± 1.81	$\mathbf{10.94 \pm 0.77}$

Table 6: **MTRL Dormant Neurons for Big router variants**. Big-SoftMoE has the least dormant neurons.

B Multi-Task RL

Combining router learning with some task-specialized layers via Hardcoding. We perform preliminary tests how enforced task-specialization in the final layer affects performance. For this, we introduce a new architecture termed SMSMHC (SoftMoE, SoftMoE, Hardcoded Router). This architecture consists of two initial layers of learned SoftMoE and a final layer with a Hardcoded Router. Contrary to expectations, SMSMHC does not yield a performance improvement (0.56 ± 0.02 and 0.53 ± 0.01), as shown in Figure 13, leading to questions about the value of specialization in this context.

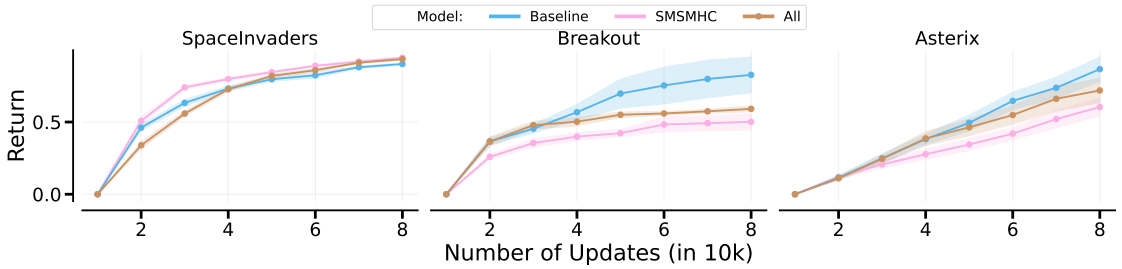


Figure 13: **MTRL: SMSMHC does not improve performance over All**, suggesting that extreme specialisation in the last layer is not necessarily helpful.

Game	Baseline	All	SMSMHC
SI	0.90 ± 0.01	0.94 ± 0.01	0.95 ± 0.00
BO	0.46 ± 0.07	0.33 ± 0.01	0.28 ± 0.03
Ast	0.51 ± 0.05	0.43 ± 0.05	0.36 ± 0.04
Total	$\mathbf{0.63 \pm 0.03}$	0.56 ± 0.02	0.53 ± 0.01

Table 7: **MTRL: SMSMHC does not improve performance over All**, suggesting that extreme specialisation in the last layer is not necessarily helpful.

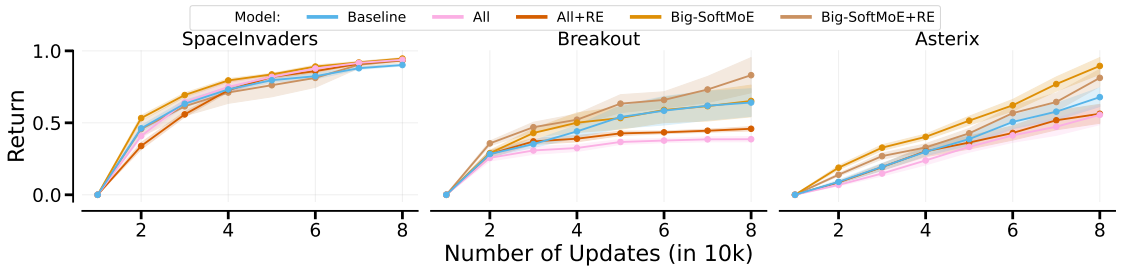


Figure 14: **MTRL: Regularising the entropy does not affect performance significantly**, suggesting that specialisation only plays a limited role for performance.

Game	Baseline	All w/o RE	All w/ RE	Big w/o RE	Big w/ RE
SI	0.90 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.95 ± 0.01	0.93 ± 0.01
BO	0.46 ± 0.07	0.28 ± 0.01	0.33 ± 0.01	0.47 ± 0.08	0.60 ± 0.09
Ast	0.51 ± 0.05	0.42 ± 0.05	0.43 ± 0.05	0.68 ± 0.04	0.62 ± 0.05
Total	0.63 ± 0.03	0.55 ± 0.02	0.56 ± 0.02	$\mathbf{0.70 \pm 0.02}$	$\mathbf{0.72 \pm 0.03}$

Table 8: MTRL: Regularising the entropy does not affect performance significantly, suggesting that specialisation only plays a limited role for performance.

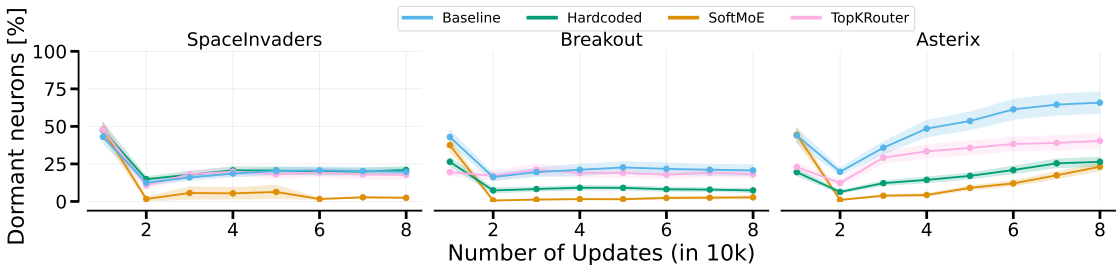


Figure 15: **MTRL: Generally, dormant neurons are lower** when using Big variants.

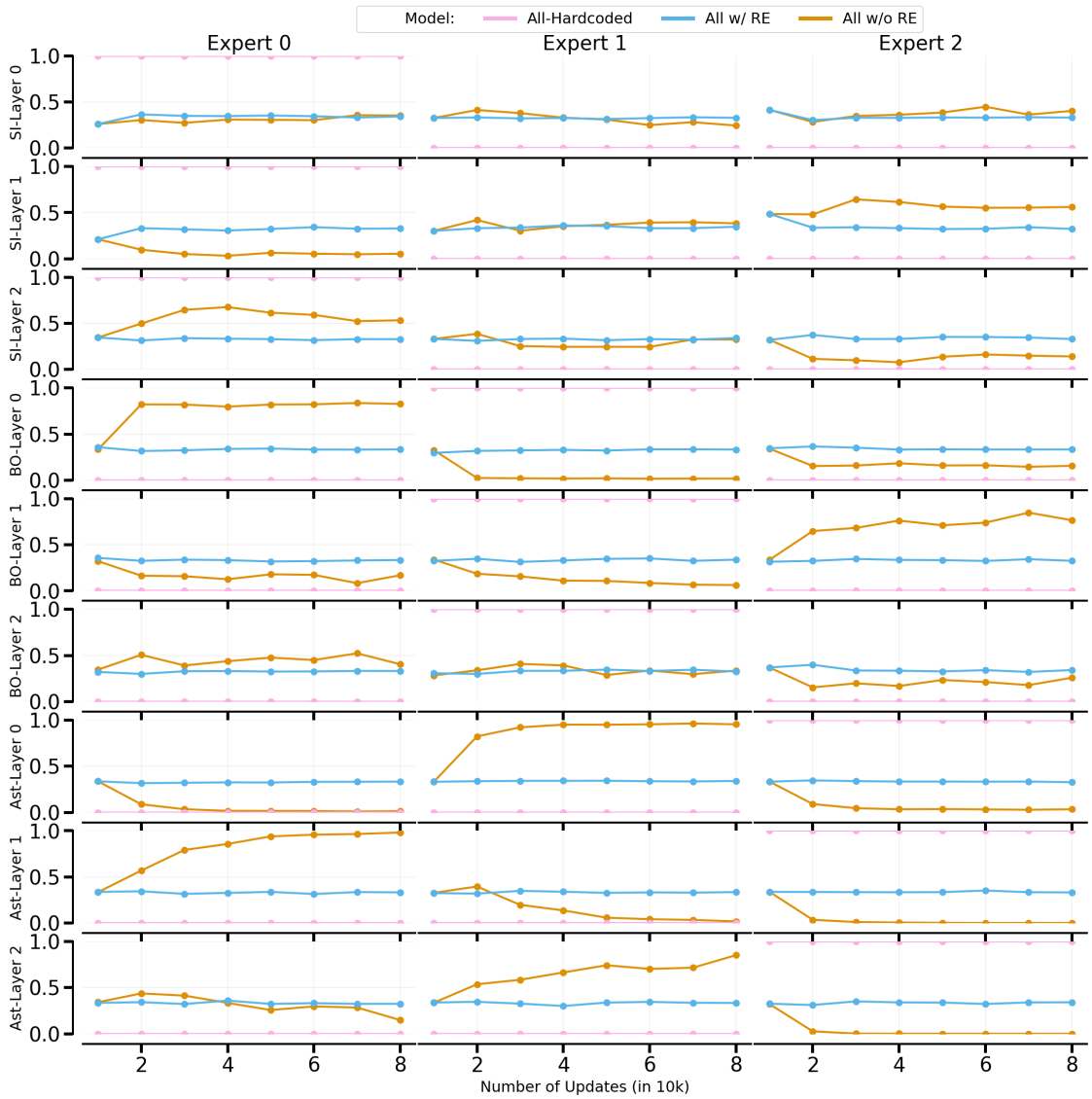


Figure 16: **MTRL:** Row 1-3 is Layer 1-3 when playing SpaveInvaders, row 4-6, is layer 1-3 when playing Breakout, row 7-9 is layer 1-3 when playing Asterix.

C Actor/Critic Ablation

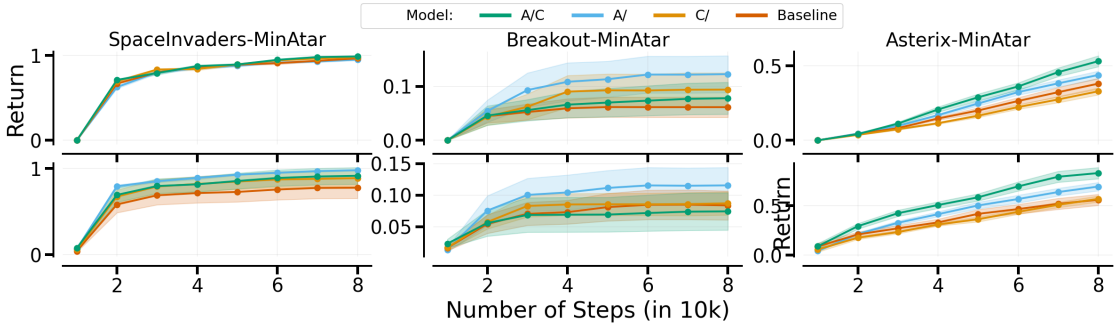


Figure 17: **CRL: In general, having an MoE module in the actor seems to be most helpful for performance**, whereas the critic MoE module does not improve performance significantly. We used the Big variant for the ablation.

Game	Baseline	/C	A/	A/C
SI	0.96 ± 0.00	0.98 ± 0.01	0.95 ± 0.01	0.99 ± 0.01
BO	0.06 ± 0.02	0.09 ± 0.03	0.12 ± 0.03	0.08 ± 0.03
Ast	0.38 ± 0.04	0.33 ± 0.02	0.44 ± 0.03	0.53 ± 0.04
SI-2	0.78 ± 0.12	0.88 ± 0.09	0.97 ± 0.01	0.91 ± 0.10
BO-2	0.08 ± 0.02	0.09 ± 0.02	0.12 ± 0.03	0.07 ± 0.03
Ast-2	0.56 ± 0.06	0.56 ± 0.03	0.71 ± 0.05	0.81 ± 0.05

Table 9: **CRL: In general, having an MoE module in the actor seems to be most helpful for performance**, whereas the critic MoE module does not improve performance significantly. We used the Big variant for the ablation.

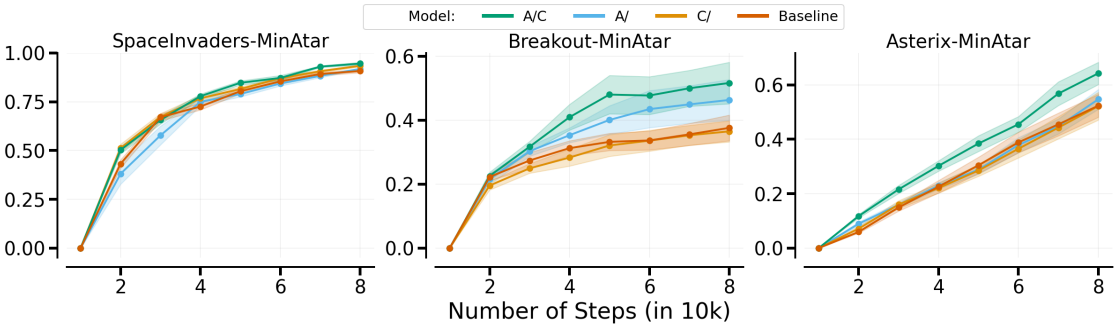


Figure 18: **MTRL: The combination of Actor and Critic MoE modules appears most beneficial**. We used the Big variant for the ablation.

Game	Baseline	/C	A/	A/C
SI	0.91 ± 0.01	0.93 ± 0.01	0.92 ± 0.00	0.95 ± 0.01
BO	0.38 ± 0.04	0.37 ± 0.03	0.46 ± 0.06	0.52 ± 0.07
Ast	0.52 ± 0.04	0.52 ± 0.05	0.55 ± 0.04	0.64 ± 0.04
Total	0.60 ± 0.02	0.61 ± 0.02	0.64 ± 0.02	$\mathbf{0.70 \pm 0.03}$

Table 10: **MTRL: The combination of Actor and Critic MoE modules appears most beneficial**. We used the Big variant for the ablation.

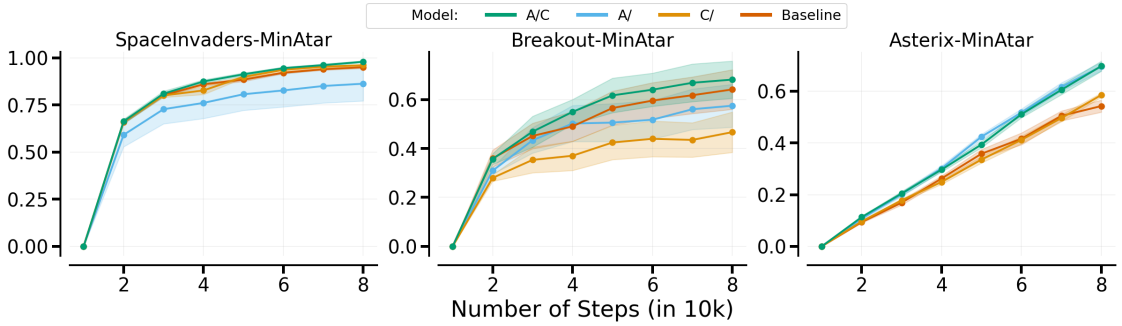


Figure 19: There is no significant difference in using either Actor or Critic, though the combination of both works significantly better than using only the Critic.

Game	Baseline	A/C	A/	C/
SI	0.95 ± 0.01	0.98 ± 0.00	0.86 ± 0.09	0.96 ± 0.00
BO	0.64 ± 0.08	0.68 ± 0.08	0.57 ± 0.09	0.47 ± 0.08
Ast	0.54 ± 0.02	0.70 ± 0.02	0.70 ± 0.02	0.59 ± 0.01
Total	$\mathbf{0.71 \pm 0.05}$	$\mathbf{0.79 \pm 0.05}$	$\mathbf{0.71 \pm 0.07}$	0.67 ± 0.05

Table 11: **There is no significant difference in using either Actor or Critic**, though the combination of both works significantly better than using only the Critic.

D Order Ablation

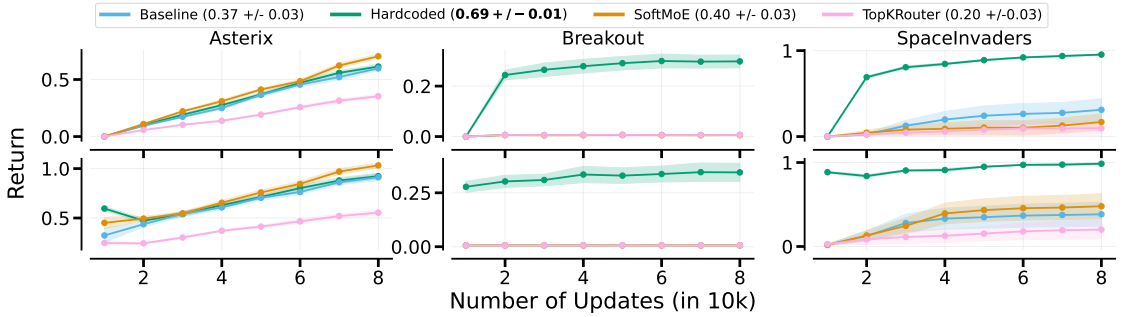


Figure 20: CRL: The order does affect the conclusion for CRL, especially because Breakout performance completely collapses if trained first on Asterix, then Breakout, then SpaceInvaders. Learned routers now do not perform better than the baseline. Big-Hardcoded still works as expected.

Game	Baseline	Big-Hardcoded	Big-TopK	Big-SoftMoE
SI	0.31 ± 0.13	0.95 ± 0.01	0.10 ± 0.08	0.17 ± 0.10
BO	0.01 ± 0.00	0.30 ± 0.03	0.01 ± 0.00	0.01 ± 0.00
Ast	0.60 ± 0.02	0.61 ± 0.02	0.35 ± 0.02	0.70 ± 0.02
SI-2	0.38 ± 0.14	0.98 ± 0.01	0.20 ± 0.11	0.48 ± 0.15
BO-2	0.01 ± 0.00	0.34 ± 0.04	0.01 ± 0.00	0.01 ± 0.00
Ast-2	0.91 ± 0.03	0.93 ± 0.03	0.55 ± 0.02	1.04 ± 0.03
Total	0.37 ± 0.03	$\mathbf{0.69 \pm 0.01}$	0.20 ± 0.03	0.40 ± 0.03

Table 12: CRL: The order does affect the conclusion for CRL, especially because Breakout performance completely collapses if trained first on Asterix, then Breakout, then SpaceInvaders. Learned routers now do not perform better than the baseline. Big-Hardcoded still works as expected.

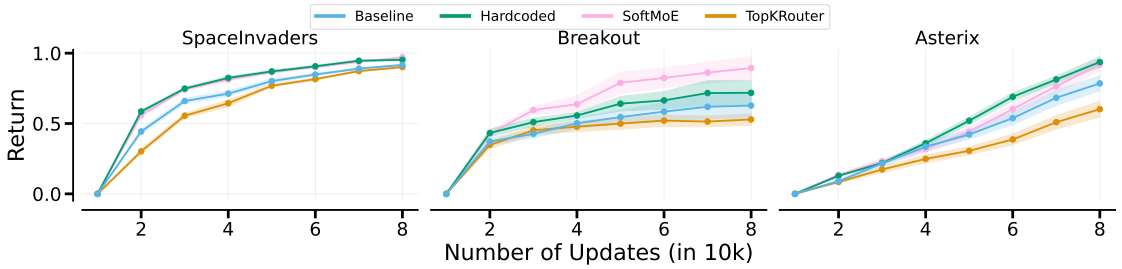


Figure 21: **MTRL: The conclusions do not change when changing the order of training for MTRL**. Learned routers and hardcoded routers perform on par and better than the baseline.

Game	Baseline	Big-Hardcoded	Big-TopK	Big-SoftMOE
SI	0.92 ± 0.01	0.95 ± 0.00	0.90 ± 0.01	0.97 ± 0.00
BO	0.34 ± 0.05	0.39 ± 0.05	0.29 ± 0.02	0.49 ± 0.04
Ast	0.51 ± 0.04	0.61 ± 0.02	0.39 ± 0.04	0.60 ± 0.03
Total	0.59 ± 0.02	0.65 ± 0.02	0.53 ± 0.01	0.68 ± 0.02

Table 13: **MTRL: The conclusions do not change when changing the order of training for MTRL**. Learned routers and hardcoded routers perform on par and better than the baseline.

E MTRL - More Results

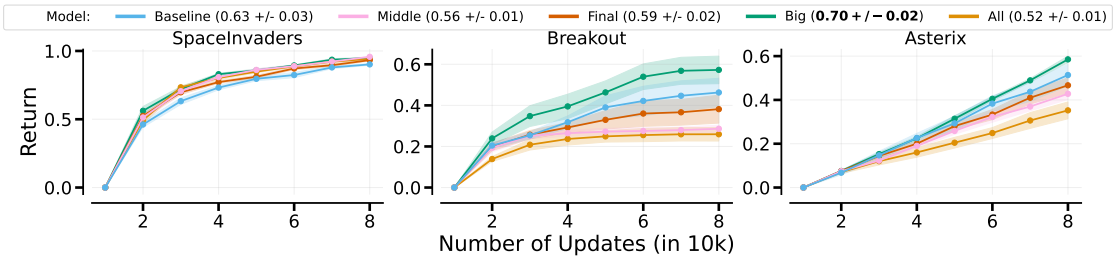


Figure 22: **MTRL: All Hardcoded architectures, Big-Hardcoded works best**. It is unclear why All does not perform as well as Big, though we hypothesise it is due to suboptimal hyperparameters

Game	Baseline	All	Big	Final	Middle
SI	0.90 ± 0.01	0.94 ± 0.01	0.95 ± 0.01	0.93 ± 0.01	0.96 ± 0.01
BO	0.46 ± 0.07	0.26 ± 0.03	0.57 ± 0.07	0.38 ± 0.07	0.29 ± 0.02
Ast	0.51 ± 0.05	0.35 ± 0.04	0.59 ± 0.01	0.47 ± 0.03	0.43 ± 0.03
Total	0.63 ± 0.03	0.52 ± 0.01	0.70 ± 0.02	0.59 ± 0.02	0.56 ± 0.01

Table 14: **MTRL: All Hardcoded architectures: Big-Hardcoded works best**. It is unclear why All does not perform as well as Big, though we hypothesise it is due to suboptimal hyperparameters.

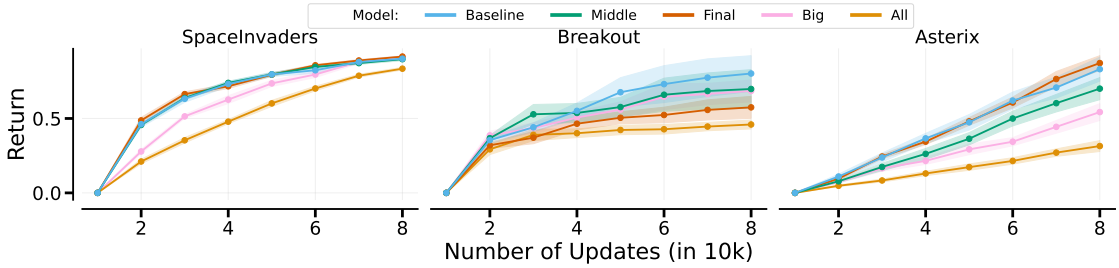


Figure 23: **MTRL: All TopKRouter architectures:** Generally, TopKRouters perform worse than the baseline.

Game	Baseline	All	Big	Final	Middle
SI	0.90 ± 0.01	0.83 ± 0.01	0.91 ± 0.01	0.92 ± 0.00	0.90 ± 0.01
BO	0.46 ± 0.07	0.27 ± 0.02	0.40 ± 0.04	0.33 ± 0.04	0.40 ± 0.07
Ast	0.51 ± 0.05	0.19 ± 0.02	0.34 ± 0.03	0.54 ± 0.03	0.43 ± 0.05
Total	$\mathbf{0.63 \pm 0.03}$	0.43 ± 0.01	0.55 ± 0.02	$\mathbf{0.60 \pm 0.01}$	$\mathbf{0.58 \pm 0.03}$

Table 15: **MTRL: All TopKRouter architectures:** Generally, TopKRouters perform worse than the baseline.

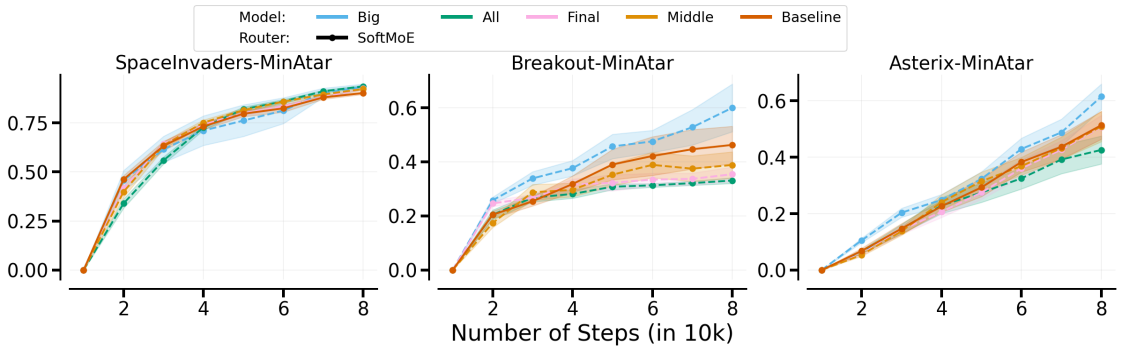


Figure 24: **MTRL: All SoftMoE Architectures:** Only Big-SoftMoE performs better than the baseline.

Game	Baseline	All	Big	Final	Middle
SI	0.90 ± 0.01	0.94 ± 0.01	0.93 ± 0.01	0.92 ± 0.01	0.92 ± 0.01
BO	0.46 ± 0.07	0.33 ± 0.01	0.60 ± 0.09	0.35 ± 0.03	0.39 ± 0.05
Ast	0.51 ± 0.05	0.43 ± 0.05	0.62 ± 0.05	0.51 ± 0.04	0.51 ± 0.05
Total	0.63 ± 0.03	0.56 ± 0.02	$\mathbf{0.72 \pm 0.03}$	0.60 ± 0.02	0.61 ± 0.02

Table 16: **MTRL: All SoftMoE Architectures:** Only Big-SoftMoE performs better than the baseline.

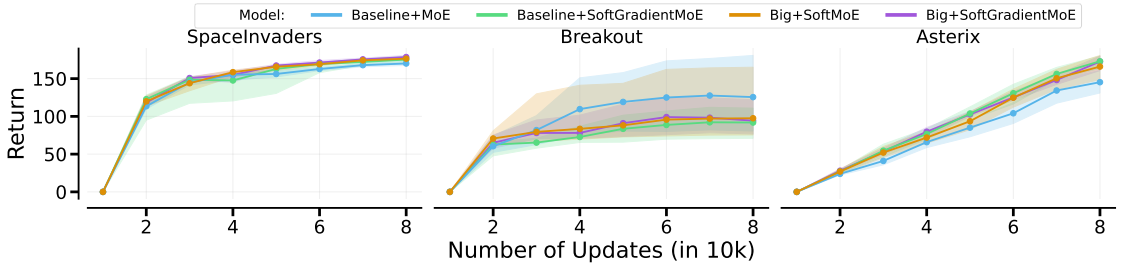


Figure 25: **MTRL: Big-SoftGradientMoE vs. Big-SoftMoE**, adding gradient information does not improve performance.

Game	Baseline	Big-Hardcoded	Big-SoftGradientMoE	Big-SoftMoE
SI	0.90 ± 0.01	0.95 ± 0.01	0.95 ± 0.00	0.93 ± 0.01
BO	0.46 ± 0.07	0.57 ± 0.07	0.51 ± 0.07	0.60 ± 0.09
Ast	0.51 ± 0.05	0.59 ± 0.01	0.66 ± 0.03	0.62 ± 0.05
Total	0.63 ± 0.03	$\mathbf{0.70 \pm 0.02}$	$\mathbf{0.70 \pm 0.03}$	$\mathbf{0.72 \pm 0.03}$

Table 17: **MTRL: Big-SoftGradientMoE vs. Big-SoftMoE**, adding gradient information does not improve performance.

F CRL - More Results

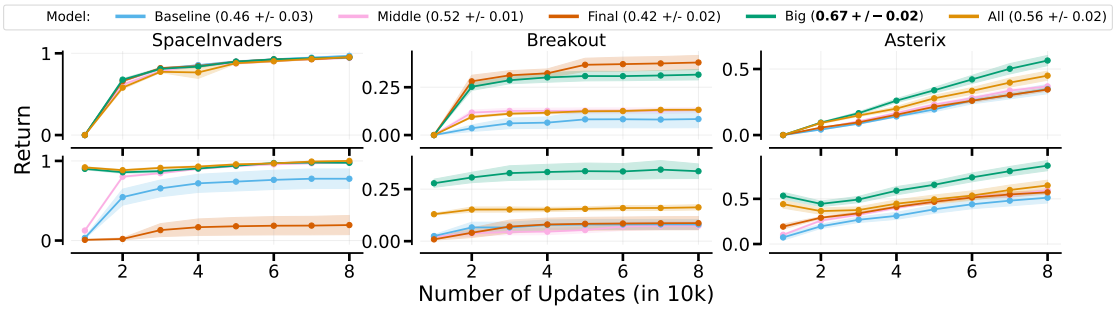


Figure 26: **CRL: All HardcodedRouter Architectures**. Big-Hardcoded works best. It is unclear why AllLayers performs significantly worse on Breakout, though we hypothesise it is due to suboptimal hyperparameters

Game	Baseline	All	Big	Final	Middle
SI	0.97 ± 0.01	0.95 ± 0.01	0.95 ± 0.00	0.95 ± 0.01	0.94 ± 0.01
BO	0.08 ± 0.05	0.13 ± 0.00	0.32 ± 0.03	0.38 ± 0.04	0.13 ± 0.01
Ast	0.35 ± 0.04	0.45 ± 0.03	0.56 ± 0.04	0.34 ± 0.01	0.37 ± 0.01
SI-2	0.78 ± 0.12	1.00 ± 0.00	0.98 ± 0.01	0.19 ± 0.12	0.98 ± 0.01
BO-2	0.08 ± 0.02	0.16 ± 0.01	0.34 ± 0.04	0.09 ± 0.03	0.07 ± 0.02
Ast-2	0.51 ± 0.06	0.65 ± 0.06	0.87 ± 0.06	0.57 ± 0.02	0.60 ± 0.02
Total	0.46 ± 0.03	0.56 ± 0.02	$\mathbf{0.67 \pm 0.02}$	0.42 ± 0.02	0.52 ± 0.01

Table 18: **CRL: All HardcodedRouter Architectures**. Big-Hardcoded works best. It is unclear why AllLayers performs significantly worse on Breakout, though we hypothesise it is due to suboptimal hyperparameters

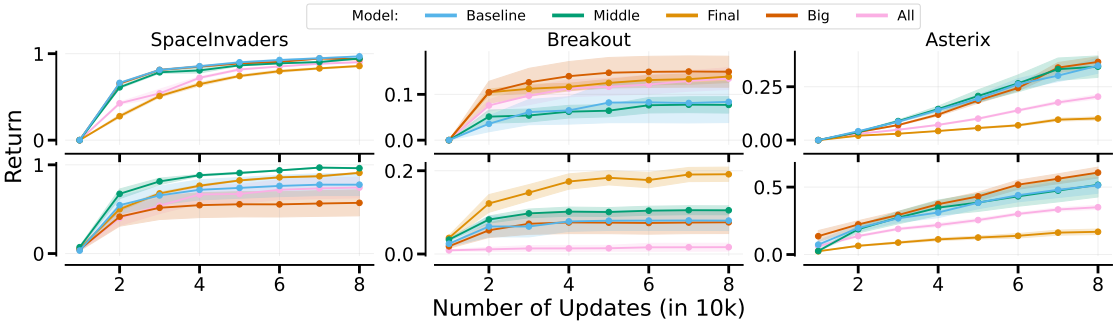


Figure 27: **CRL: All TopK variants** perform worse than the baseline except for Middle.

Game	Baseline	All	Big	Final	Middle
SI	0.97 ± 0.01	0.86 ± 0.01	0.90 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
BO	0.08 ± 0.05	0.14 ± 0.01	0.13 ± 0.02	0.15 ± 0.03	0.08 ± 0.02
Ast	0.35 ± 0.04	0.10 ± 0.01	0.20 ± 0.01	0.37 ± 0.02	0.34 ± 0.05
SI-2	0.78 ± 0.12	0.91 ± 0.01	0.74 ± 0.12	0.57 ± 0.15	0.96 ± 0.01
BO-2	0.08 ± 0.02	0.19 ± 0.02	0.02 ± 0.01	0.08 ± 0.03	0.10 ± 0.01
Ast-2	0.51 ± 0.06	0.17 ± 0.02	0.35 ± 0.02	0.61 ± 0.04	0.52 ± 0.09
Total	$\mathbf{0.46 \pm 0.03}$	0.39 ± 0.01	0.39 ± 0.02	$\mathbf{0.45 \pm 0.02}$	$\mathbf{0.49 \pm 0.02}$

Table 19: **CRL: All TopK variants** perform worse than the baseline except for Middle.

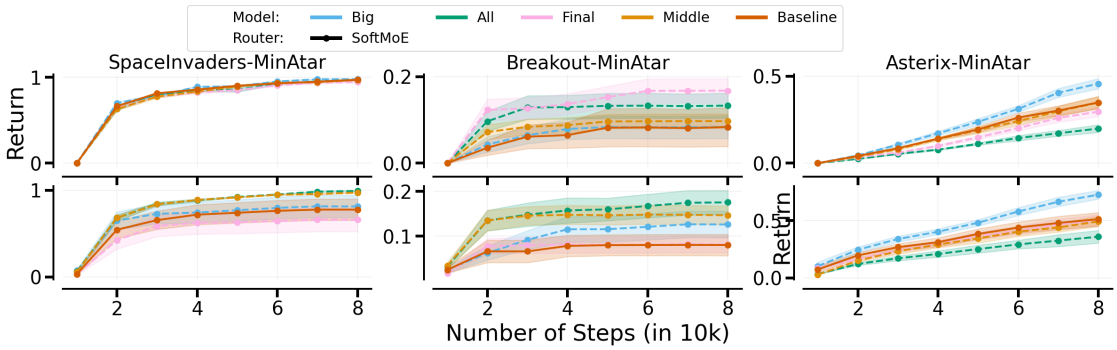


Figure 28: **CRL: All SoftMoE Architectures**: Big-SoftMoE is the only variant that performs better than the baseline.

Game	Baseline	All	Big	Final	Middle
SI	0.97 ± 0.01	0.97 ± 0.01	0.98 ± 0.01	0.94 ± 0.01	0.97 ± 0.01
BO	0.08 ± 0.05	0.13 ± 0.03	0.08 ± 0.03	0.17 ± 0.03	0.10 ± 0.02
Ast	0.35 ± 0.04	0.20 ± 0.02	0.46 ± 0.03	0.30 ± 0.03	0.35 ± 0.03
SI-2	0.78 ± 0.12	0.99 ± 0.01	0.81 ± 0.13	0.66 ± 0.14	0.97 ± 0.01
BO-2	0.08 ± 0.02	0.18 ± 0.03	0.13 ± 0.03	0.08 ± 0.02	0.15 ± 0.02
Ast-2	0.51 ± 0.06	0.36 ± 0.06	0.73 ± 0.04	0.50 ± 0.05	0.49 ± 0.05
Total	$\mathbf{0.46 \pm 0.03}$	0.47 ± 0.01	$\mathbf{0.53 \pm 0.02}$	0.44 ± 0.03	$\mathbf{0.50 \pm 0.01}$

Table 20: **CRL: All SoftMoE Architectures:** Big-SoftMoE is the only variant that performs better than the baseline.

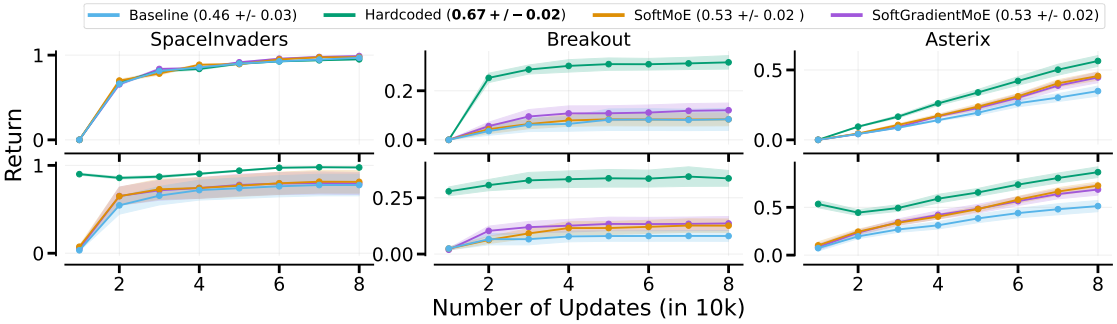


Figure 29: **CRL: Adding gradient information to the input** does not improve performance significantly.

G Single Env

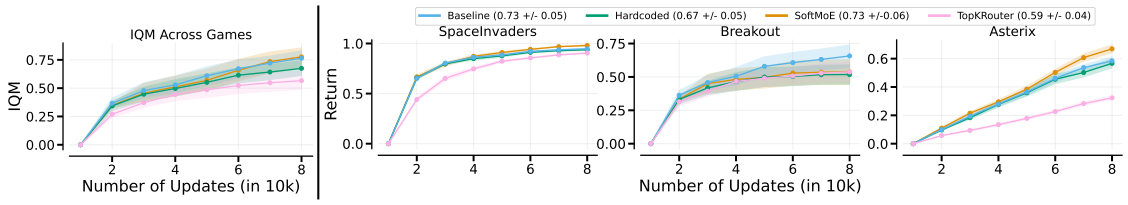


Figure 30: **Left:** Aggregate Interquantile Mean (Agarwal et al., 2021) of scores. **Right** Comparison of different models and routers on the single environment setup. We report the mean of the normalised scores for 3 Atari games. All games run with 10 independent seeds, shaded areas representing the standard error. We normalise performance according to the single environment results reported in Jesson et al. (2023). BigMoE improves performance over the baseline, especially due to performance improvements in Asterix.

Game	Big-SoftMoE	Baseline	Big-Hardcoded	Big-TopK
SI	0.98 ± 0.01	0.95 ± 0.01	0.94 ± 0.01	0.90 ± 0.01
BO	0.54 ± 0.09	0.66 ± 0.08	0.52 ± 0.07	0.54 ± 0.07
Ast	0.67 ± 0.03	0.58 ± 0.03	0.57 ± 0.03	0.32 ± 0.01
Total	$\mathbf{0.73 \pm 0.06}$	$\mathbf{0.73 \pm 0.05}$	$\mathbf{0.67 \pm 0.05}$	0.59 ± 0.04

Table 21: **Single environment: Normalised Performance of algorithms across games with average total performance**. We normalise performance according to the single environment results reported in Jesson et al. (2023). We do not achieve the same performance as we use smaller networks due to computational limits.

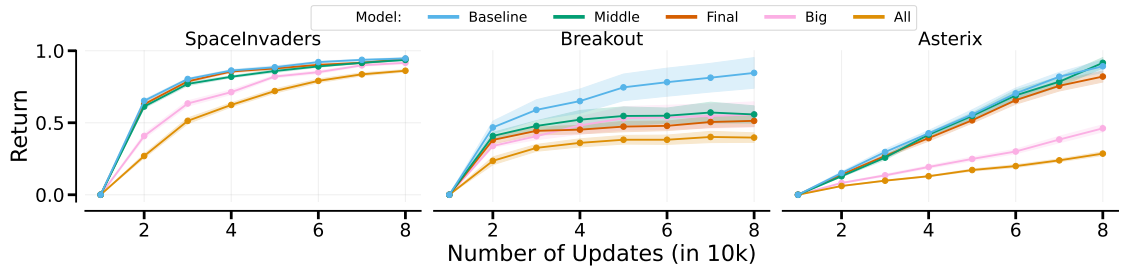


Figure 31: **Single environment: All TopK Architectures**, all variants perform worse than the baseline.

Game	All	Final	Middle	Baseline	Big
SI	0.86 ± 0.01	0.94 ± 0.01	0.94 ± 0.00	0.95 ± 0.01	0.92 ± 0.01
BO	0.31 ± 0.03	0.40 ± 0.03	0.43 ± 0.05	0.66 ± 0.08	0.43 ± 0.07
Ast	0.19 ± 0.01	0.54 ± 0.02	0.60 ± 0.02	0.58 ± 0.03	0.30 ± 0.02
Total	0.45 ± 0.02	0.63 ± 0.02	$\mathbf{0.66 \pm 0.03}$	$\mathbf{0.73 \pm 0.05}$	0.55 ± 0.04

Table 22: **Single environment: All TopK Architectures**, all variants perform worse than the baseline.

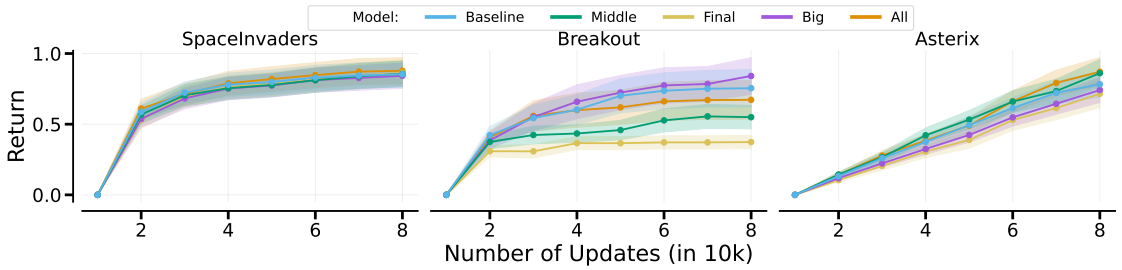


Figure 32: **Single environment: All SoftMoE Architectures:** Big, Final, and Middle all perform as well as the baseline.

Game	Big	All	Final	Baseline	Middle
SI	0.98 ± 0.01	0.96 ± 0.00	0.94 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
BO	0.54 ± 0.09	0.31 ± 0.02	0.67 ± 0.08	0.66 ± 0.08	0.44 ± 0.05
Ast	0.67 ± 0.03	0.55 ± 0.04	0.56 ± 0.03	0.58 ± 0.03	0.65 ± 0.02
Total	$\mathbf{0.73 \pm 0.06}$	0.61 ± 0.03	$\mathbf{0.72 \pm 0.05}$	$\mathbf{0.73 \pm 0.05}$	$\mathbf{0.68 \pm 0.03}$

Table 23: **Single environment: All SoftMoE Architectures:** Big, Final, and Middle all perform as well as the baseline.

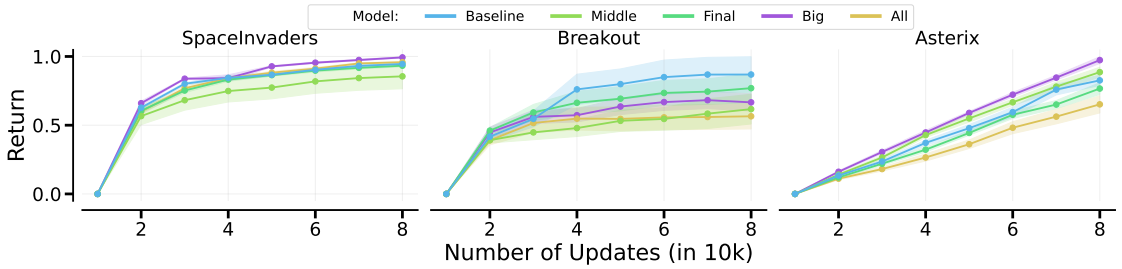


Figure 33: **Single environment: All SoftGradientMoE Architectures.**, adding gradient information does not change the conclusions of the softmoe architectures above.

Game	Baseline	Middle	Big	Final	All
SI	0.95 ± 0.01	0.86 ± 0.09	0.99 ± 0.00	0.93 ± 0.01	0.96 ± 0.01
BO	0.60 ± 0.09	0.42 ± 0.07	0.46 ± 0.04	0.53 ± 0.07	0.39 ± 0.06
Ast	0.59 ± 0.03	0.64 ± 0.03	0.70 ± 0.02	0.55 ± 0.03	0.47 ± 0.04
Total	$\mathbf{0.71 \pm 0.05}$	$\mathbf{0.64 \pm 0.07}$	$\mathbf{0.72 \pm 0.03}$	$\mathbf{0.67 \pm 0.05}$	0.60 ± 0.04

Table 24: **Single environment: All SoftGradientMoE Architectures**, adding gradient information does not change the conclusions of the softmoe architectures above.

Game	w/ Gradient Info	w/o Gradient Info
SI	0.95 ± 0.01	0.95 ± 0.01
BO	0.66 ± 0.08	0.60 ± 0.09
Ast	0.58 ± 0.03	0.59 ± 0.03
Total	$\mathbf{0.73 \pm 0.05}$	$\mathbf{0.71 \pm 0.05}$

Table 25: **Baseline vs. Baseline with Gradient Information:** Adding gradient information to the input of the baseline does not affect its performance.

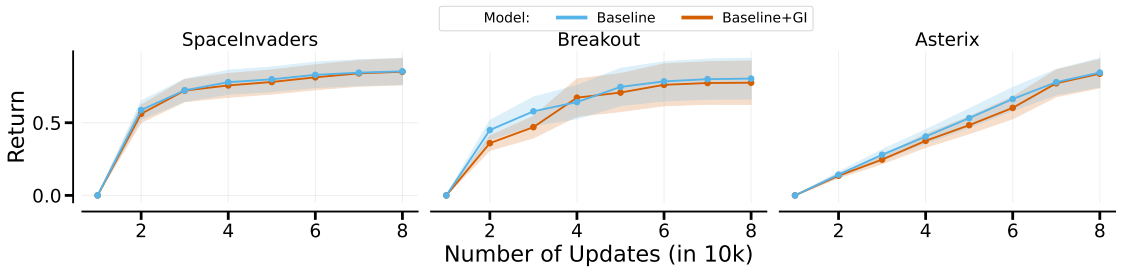


Figure 34: **Single environment: Baseline vs. Baseline with Gradient Information**, adding gradient information to the input of the baseline does not affect its performance.

H Hyperparameters

Hyperparameter	Value
Number of Environments	128
Learning Rate	9e-4
Steps	64
Total Timesteps	1e7
Updates	total_timesteps // num_steps // num_envs
Update Epochs	10
Minibatches	8
Minibatch Size	num_envs * num_steps // num_minibatches
GAE-γ	0.99
GAE-λ	0.7
Clip ϵ	0.2
Entropy Coefficient	0.01
Value Function Coeffficient	0.5
Max Gradient Norm	1.9
Activation	relu
Environment	{SpaceInvaders-MinAtar, Breakout-Minatar, Asterix-MinAtar}
Anneal learning rate	True
# Experts	3
Layer Size	64
Expert Hidden Size	64
Model	{BigMoE, FinalLayer, AllLayers, MiddleLayer}
MoE	{SoftMoE, MoE, SoftGradientMoE}
Expert	{BigExpert, Expert}
Router	{TopKRouter, HardcodedRouter}
Number of Selected Experts	1
Task ID	{True, False}
Actor MoE	{True, False}
Critic MoE	{True, False}
Gradient Buckets	5
Router Entropy	{True, False}

Table 26: **Potential Hyperparameters configurations**. We did not run a grid search over all potential combinations but report meaningful selections.

“The Road goes ever on and on
Down from the door where it began,
Now far ahead the Road has gone,
And I must follow, if I can.”
(Tolkien, 1954, p. 44)