Improving Deep Reinforcement Learning Through Better Benchmarking and Tools



Benjamin Ellis

Keble College

University of Oxford

Doctor of Philosophy

January 18, 2026

Acknowledgements

The work in this thesis was made possible by a large number of people. First, my advisors Jakob and Shimon, who gave me ideas when I didn't have them, helped me to refine and communicate the ones that I did have, and supported my research until I could conduct it on my own. They taught me to disagree with them and for that I am grateful. Thank you to my examiners for taking the time to read my thesis and provide valuable feedback.

Secondly, my collaborators: Jonny Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Alex Rutherford, Matteo Gallici, Chris Lu, Matt Jackson, Andrei Lupu, Alex Goldie and Mattie Fellows. The work and support of my collaborators made all of these projects possible— finishing research projects is a difficult and, without help, very lonely process. I certainly could not have done it alone.

I would like to particularly thank all the experienced researchers who took the time to give me advice, support or ideas throughout the years. In particular, a very special thank you to Mikayel Samvelyan, Mingfei Sun, Minqi Jiang, Vitaly Kurin, Luisa Zintgraf, Mattie Fellows and Tim Rocktaschel. I am very grateful for your counsel.

I would like to thank all the members of my two research groups, FLAIR and WhiRL, for making the office such an enjoyable place to be. There are too many people to name, but thank you in particular to Timon Willi, Silvia Sapora, Sebastian Towers, Irene Zhang, Ola Kalisz, Jake Beck, Risto Vuorio and Alex Goldie.

There are a few people who deserve special mention. Thank you to Andrei, who was my office companion in a difficult summer when nothing worked. Thank you to Matt, who collaborated with me at a stressful time and helped me finish a project when I could not sleep or concentrate. Thank you to Mingfei for supervising my first miniproject and introducing me to RL research. Thank you to Risto, with whom I greatly enjoyed running and who I am lucky enough to work with now. Thank you to Chris, whose advice helped me finish Adam-Rel, reassured me and guided my exploration in other projects. Thank you to Timon, who suffered through hunting for a job with me.

I frequently also received support from friends, too many to name, both within Oxford and elsewhere. Again, without their help I am sure this thesis would not be here at all. In the latter years of the PhD, I was fortunate to have the support of

my girlfriend Maddy. I found the times in the PhD with her around much easier than those without her. Maybe that was because I had already worked things out. I think I was probably just happier.

Finally, I would like to thank my parents, Susan and Richard, for their support of my education throughout my life. My parents gave me the opportunity to pursue anything I put my mind to. I did, and now I am privileged to do interesting, fulfilling and challenging work in a field with significant societal impact. I am extremely grateful.

Originality Statement

The research described in this thesis was carried out by the author between January 2021 and March 2025 and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. The use of the first person plural is strictly a matter of style in keeping with standard scientific conventions.

Abstract

Reinforcement Learning (RL) has been a powerful approach for training machines to perform tasks such as game playing and, more recently, solving mathematics and coding puzzles. However, despite these successes, it has suffered from numerous issues. First, algorithms in reinforcement learning commonly rely on a number of implementation details which are crucial for performance, often used to adapt tools originally designed for supervised learning to the very different world of reinforcement learning. This harms reproducibility and makes the causes of performance improvements in reinforcement learning unclear. Secondly, RL has suffered from poor evaluation practices, with many papers only evaluating on a single benchmark, benchmarks being poorly understood or designed, and inconsistent statistical practices. Finally, RL has a significant computational issue. Whereas supervised learning achieves high GPU usages, RL requires gathering long trajectories from CPU-based environment simulators, harming computational efficiency.

This thesis presents work aimed at solving these problems in two parts. In the first part, we consider evaluation in Multi-Agent Reinforcement Learning (MARL). The first content chapter considers the popular SMAC benchmark and demonstrates that it fundamentally lacks partial observability and stochasticity. We then present SMACv2, a new MARL benchmark that addresses the stochasticity issues in SMAC by adding randomised unit types and start positions. We then present the Extended Partial Observability Challenge (EPO), which addresses SMAC's lack of partial observability. The second content chapter of this part introduces JaxMARL, a multi-agent RL research library combining hardware acceleration with a wide range of MARL environments and baselines. In most previous MARL libraries, environments were run on the CPU, whereas the core models were trained on the GPU. Compared to this setup, JaxMARL has numerous advantages. JaxMARL allows easy parallelisation of environment rollouts using the `vmap` operation, which also has the benefit of removing complicated multiprocessing code from CPU-based rollout parallelisation, allows end-to-end JITting of the environment and model code, and avoids the slow transfers from the CPU to the GPU required when environments are run on the CPU. This allows JaxMARL to improve performance by 30x when compared with libraries using CPU-based environments.

The second part of this thesis aims to re-examine common deep learning tools used in RL with the aim of adapting them more effectively for use in a non-stationary RL setting. To that end we consider the Adam optimiser in RL. We first demonstrate theoretically that after a change in gradient scale, such as after a change in target network, the Adam optimiser produces consistently larger updates. Then, by observing that many RL algorithms are sequences of stationary optimisation problems and examining the Adam timestep, we instead use the *local* timestep, i.e. restarting Adam's timestep from 0 on each new optimisation problem, rather than the *global* timestep as is usual. We show that the update size is robust to a change of gradient scale and that if updates are smaller this reduces to learning rate annealing.

Overall this thesis contributes to making multi-agent RL better evaluated and the usage of Adam in non-stationary settings more effective and better understood. Applying the lessons learned from this research to create cooperative foundation models is an open question, and an exciting possibility for future work.

Contents

1 Introduction **1**
1.1 Modern Reinforcement Learning 1
1.2 Contributions and Thesis Structure 4
1.2.1 Part I: Improving Reinforcement Learning Evaluation 4
1.2.2 Part II: Usage of Deep Learning Tools in Reinforcement Learning 7

2 Background **9**
2.1 Reinforcement Learning 9
2.1.1 Markov Decision Processes 10
2.1.2 Value Functions 10
2.1.3 Value-based Methods 12
2.1.4 Policy Gradient and Policy Optimisation 13
2.1.5 Nonstationarity in Reinforcement Learning 15
2.1.6 Reinforcement Learning as Sequential Stationary Optimisation 15
2.2 Infrastructure and Tooling for Reinforcement Learning 16
2.2.1 Training and Distributed Training 16
2.2.2 Reinforcement Learning Training Frameworks and Infrastructure 18
2.2.3 Reinforcement Learning Environments 19
2.3 Multi-Agent Reinforcement Learning 21
2.3.1 Decentralised Partially Observable Markov Decision Processes 21
2.3.2 Key Properties of Dec-POMDPs 22
2.3.3 Centralised Training with Decentralised Execution 22
2.3.4 Independent Learning 23
2.3.5 Value-based Multi-agent Reinforcement Learning 23
2.3.6 Multi-Agent Reinforcement Learning Policy Gradient Methods 24
2.3.7 The StarCraft Multi-Agent Challenge 24

I Improving Reinforcement Learning Evaluation **27**

3 SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning **29**
3.1 Limitations and Future Work 46

4 JaxMARL: Multi-Agent RL Environments and Algorithms in JAX 49
4.1 Limitations and Future Work 66

II Usage of Deep Learning Tools in Reinforcement Learning 69

5 Adam on Local Time: Addressing Nonstationarity in RL with Relative Adam Timesteps 71
5.1 Limitations and Future Work 86

6 Conclusion 89
6.1 The Future of Multi-Agent Research 90
6.1.1 Summary 92

Bibliography 93

Appendices

A SMACv2 Appendices 103

B JaxMARL Appendices 117

C Adam-Rel Appendices 131

1 Introduction

One of the goals of modern machine learning is to enable machines to perform useful work in the real world. Such tasks often require long-term reasoning and planning. Reinforcement Learning (RL) (Sutton et al., 2018), which trains an agent to maximise reward in an environment, represents a promising framework for pursuing this goal.

1.1 Modern Reinforcement Learning

Researchers have pursued advances in Reinforcement Learning with significant success, including DQN (Mnih et al., 2013), which attained human-level performance across a wide range of games from the Atari 2600 console; AlphaStar (Vinyals et al., 2019), which achieved the grandmaster rank in StarCraft II; and more recently, reasoning language models such as OpenAI's o1 (Jaech et al., 2024), DeepSeek's R1 (Guo et al., 2025), and others (Team et al., 2025), which have surpassed many humans in domains such as coding and mathematics.

In the real world, many tasks are made significantly easier, or even possible, by cooperation. Therefore, alongside these single agent successes, researchers have focused intensively on training agents to coordinate effectively with others, whether human or machine. Studies have investigated training teams of agents to excel in

decentralized StarCraft micromanagement (Samvelyan et al., 2019; Rashid et al., 2018), coordinate with humans in Overcooked (Carroll et al., 2019; Strouse et al., 2021), and collaborate in Hanabi (Bard et al., 2020; Hu et al., 2021a). However, these results have been confined to training from scratch on narrow domains such as games, in contrast to the use of RL in LLM reasoning recently, which has produced more broadly capable systems. Multi-agent RL has yet to produce agents capable of cooperating with humans and machines on a wide range of tasks.

Despite this vision and these successes, however, RL also has significant challenges.

RL Toolchain Many RL advances have been built upon tools first designed for supervised learning, such as convolutional neural networks (CNNs; Mnih et al., 2013; LeCun et al., 1989; Lecun et al., 1998) and Adam (Kingma and Ba, 2014), enabling RL to fulfil its potential. However, unlike supervised learning, RL is inherently *nonstationary* because the learner must learn to maximise a return distribution that shifts each time the learner updates. Additionally, as the learner updates, the distribution of observations changes. For example, if learning to solve a maze, as the learner reaches further, it will see new parts of the maze that pose new challenges. Adapting these tools originally designed for stationary optimisation to work in a nonstationary setting has led to a wide array of poorly documented implementation tricks crucial for performance (Engstrom et al., 2020; Andrychowicz et al., 2020). RL is not only a very different optimisation problem, but also very different computationally from supervised learning approaches, often requiring evaluating the agent using long sequential episodes rather than the parallelisable batched inference of supervised learning. These considerations suggest that it is important to consider the usage of deep learning tools in reinforcement learning and revisit how they could be best applied.

Evaluation A problematic combination of poor evaluation practices and these implementation details has led to poor reproducibility in the field. Agarwal et al. (2021) demonstrated that the statistical metrics calculated in many reinforcement

learning papers for aggregate performance were misleading and Machado et al. (2018) found that a lack of stochasticity in the popular Atari Learning Environment benchmark (Bellemare et al., 2013) meant it could be solved without conditioning on the observation at all. In multi-agent RL (MARL), Gorsane et al. (2022) collated many papers and found numerous issues with evaluation on the popular benchmark SMAC, including inconsistent choice of environments, ineffective statistical practices, and many papers evaluating on only a single benchmark. The authors also found some crucial details of evaluations were missing, making reproducing these works difficult. These examples highlight the need for better benchmarks, frameworks and evaluation practices in the field of MARL. Establishing clear evaluation standards is vital to understanding whether a new algorithm represents empirical and technological progress or not.

Computational Inefficiency Reinforcement learning also suffers from fundamental computational issues that limit its progress. Unlike in other fields such as natural language processing or computer vision, most reinforcement learning requires interaction with an environment, rather than simply loading a dataset from disk. Most RL training is through simulation, where a hand-coded simulator and the policy rollouts are run on the CPU while model training is performed on a hardware accelerator such as a GPU. This setup constrains RL training performance for several reasons. First, unlike in supervised learning, the latency of transferring data between the CPU and GPU cannot be hidden because the most up-to-date policy is required for each new rollout. Secondly, running the environment on the CPU limits the effectiveness of just-in-time compilation, which can drastically improve performance. Thirdly, this training setup limits the parallelisation of environment rollouts available with typical academic compute, where large CPU clusters for distributed training are not available, because complicated and heavyweight multiprocessing tools must be used with only a relatively small number of CPU cores. These problems are particularly an issue in multi-agent RL where observation sizes are larger, and batch sizes scale with the number of agents.

However, more modern tools such as JAX (Bradbury et al., 2018) and PyTorch's `functorch` module have the power to remove these computational bottlenecks by running environment rollouts alongside model training on the GPU. This enables large-scale parallelisation via the `vmap` operation, just-in-time compilation, and removes the need to transfer data between the CPU and GPU. These tools enabled the development of gymnax (Lange, 2022) and PureJaxRL (Lu et al., 2022), improving the training speed of RL algorithms by 10x, and over 4000x when training many agents in parallel.

This thesis will present new benchmarks and frameworks aimed at improving evaluation in RL, as well as an analysis of the use of the Adam optimiser in nonstationary settings. Together these advancements present steps forward in addressing the problems outlined above.

1.2 Contributions and Thesis Structure

We begin the thesis with a literature review which will cover the necessary background in multi-agent reinforcement learning, hardware acceleration and optimisation. The remainder of the thesis is divided into two parts. The first part will contain papers aimed at improving evaluation in reinforcement learning, and the second part a paper that aims to improve RL's underlying toolchain. This is an integrated thesis, with each chapter corresponding to a published conference paper. The beginning of each chapter introduces the paper presented, explains its link to the overall thesis, and highlights key aspects. The appendices of each paper are listed at the end of the thesis.

1.2.1 Part I: Improving Reinforcement Learning Evaluation

This part introduces work aimed at improving evaluation in multi-agent reinforcement learning, chiefly through the introduction of new evaluation frameworks and benchmarks.

- **Chapter 3** presents an investigation of the widely popular StarCraft Multi-Agent Challenge (SMAC) benchmark in cooperative multi-agent reinforcement learning and shows that, in addition to its well-documented ceiling effect limiting the effectiveness of results, that it fundamentally lacks the stochasticity and partial observability vital to multi-agent learning. In particular, we demonstrate that an open-loop algorithm – one that only conditions on the timestep and agent ID – can achieve high win rates in many scenarios. We then introduce SMACv2, a new benchmark designed to rectify these issues. We demonstrate that state-of-the-art algorithms cannot achieve high win rates on the new SMACv2 scenarios. We also introduce the extended partial observability challenge (EPO), which addresses the lack of meaningful partial observability in SMAC, and show that the introduced partial observability leads to a performance gap between training with perfect communication and no communication. This chapter is based on the publication: ***Benjamin Ellis****, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, Shimon Whiteson. SMACv2: An Improved Benchmark for Multi-Agent Reinforcement Learning.* ***In NeurIPS Datasets and Benchmarks Track 2023****.*

  Contributions. Benjamin Ellis led the project, implemented SMACv2, collected the open-loop and baseline experiments, and collected many of the final results for the paper. Jonathan Cook implemented and ran EPO and all related experiments. Skander Moalla implemented the QMIX feature experiments and collected results. Mikayel Samvelyan contributed ideas and implemented an early version of SMACv2. Mingfei Sun ran initial ablation experiments. Anuj Mahajan contributed design. Jakob Foerster and Shimon Whiteson supervised the project. Benjamin Ellis led the writing of the paper, to which Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Jakob Foerster, and Shimon Whiteson also contributed.

- **Chapter 4** presents JaxMARL. This is a multi-agent RL library that for the first time combines a large number of popular environments, many important baseline algorithm implementations, and end-to-end hardware-accelerated training. This is around 14x faster than existing implementations and up to 12500x faster when the training of many agents is vectorised. By enabling efficient evaluations across a wide range of previously separated benchmarks, this work alleviates many evaluation and reproducibility issues in multi-agent RL. This chapter also introduces SMAX, a stripped-back implementation of the StarCraft game engine in JAX. This retains the elements that make SMAC an interesting challenge, while allowing hardware acceleration and fixing some of the issues present in the SMAC and SMACv2 environments. This chapter is based on the publication: *Alexander Rutherford*, **Benjamin Ellis***, Matteo Gallici*, Jonathan Cook, Andrei Lupu, Gardar 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 Rocktaschel, Chris Lu*, Jakob Foerster. JaxMARL: Multi-Agent RL Environments and Algorithms in JAX. **In NeurIPS Datasets and Benchmarks Track 2024**.* This work also appeared as an extended abstract at ***AAMAS 2024***.

  Contributions. Alexander Rutherford led the design of the JaxMARL API and interface as well as the implementation of IPPO and MPE environments. Benjamin Ellis led the design and implementation of the SMAX environments and IPPO evaluations. Alexander Rutherford and Benjamin Ellis also led the paper writing. Matteo Gallici led the implementation of the off-policy MARL algorithms and led their evaluations, as well as debugging the hanabi environment. Alexander Rutherford and Benjamin Ellis led the preparations for submission to the NeurIPS datasets and benchmarks track, which included new benchmarking and significant writing changes. Chris Lu and Matteo Gallici helped with this effort.

Jonathan Cook led the implementation of the Hanabi environment and heavily assisted with benchmarking and verifying its performance. Ravi Hammond debugged and verified the correctness of the hanabi environment. Andrei Lupu led the implementation of the Overcooked environments. Gardar Ingvarsson led the implementation of the Multi-Agent Brax environments. Timon Willi led the implementation of the STORM environments. Akbir Khan and Alexandra Souly worked on the STORM environments. Christian Schroeder de Witt led the implementation of the Predator-Prey environment.

Christian Schroeder de Witt, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, and Robert Lange provided invaluable discussions for project planning and implementations across the project. Saptarashmi Bandyopadhyay helped initiate the project plan. Mikayel Samvelyan worked on the Multi-Agent Brax environments. Minqi Jiang worked on the Overcooked and Hanabi environments. Robert Lange assisted with the design of the API and testing infrastructure. Shimon Whiteson, Bruno Lacerda, Nick Hawes, and Tim Rocktaschel provided invaluable feedback on the project, manuscript and results. Chris Lu initiated the project and led the organizational efforts and speed-based benchmarking. Jakob Foerster is the primary advisor for the project.

1.2.2 Part II: Usage of Deep Learning Tools in Reinforcement Learning

The second part of this thesis introduces work aimed at re-examining the toolchain inherited from supervised learning used in reinforcement learning and understanding how it could be used better in a non-stationary, RL setting.

- **Chapter 5** investigates the usage of the Adam optimiser in RL. Through a very simple model, we examine the impact of a change in gradient scale, such as that from a change in target network, on the Adam optimiser and show that such a change can lead to large updates and sub-optimal performance. To address this, we propose using the *local* timestep within an epoch, essentially

resetting the Adam timestep to 0 when the target changes. We find that this approach removes the possibility of large updates and reduces to learning rate annealing in the case of there being no spike in gradient. We demonstrate that this new approach, which we call *Adam with Relative Timesteps* (Adam-Rel), improves performance for both PPO and DQN across the Atari and Craftax domains. Finally we show that increases in gradient norm *do* occur in practice, and examine differences between our theoretical model and the observed data. This chapter is based on the publication: ***Benjamin Ellis****, *Matthew T. Jackson**, *Andrei Lupu, Alexander Goldie, Mattie Fellows, Shimon Whiteson, Jakob Foerster. Adam on Local Time: Addressing Nonstationarity in RL with Relative Adam Timesteps.* ***In NeurIPS 2024***.

Contributions. Benjamin Ellis and Matthew T. Jackson jointly conceived the project and many of the ideas. Matthew T. Jackson implemented the first version of the algorithm and collected the Craftax results. Benjamin Ellis collected the Atari results, performed the analysis, and wrote the first version of the proofs. Andrei Lupu also collected Atari results. Alexander Goldie contributed ideas. Mattie Fellows checked, formalised and wrote the final version of the proofs. Benjamin Ellis and Matthew T. Jackson wrote the paper. Shimon Whiteson and Jakob Foerster supervised the project.

The thesis will end by examining the limitations of the work in the thesis and suggesting how Reinforcement Learning could improve on the problems highlighted in this thesis in the future.

2
Background

In this chapter, I present background on topics necessary to understand the thesis. I start by introducing basic topics in RL necessary for understanding the thesis, before discussing nonstationarity, and finally discussing hardware acceleration in Reinforcement Learning. I then introduce topics in cooperative multi-agent reinforcement learning, which are necessary for understanding the first two content chapters of the thesis. Any remaining background will be introduced in the content chapters as needed.

2.1 Reinforcement Learning

Reinforcement learning considers an *agent* interacting with an *environment* with the goal of trying to maximise its long term *reward*. At each step, the agent observes the environment's state and takes an action. This is shown in Figure 2.1. This reward is usually assumed to be externally given and part of the problem setting. This interaction is most commonly modelled through a Markov Decision Process (Bellman, 1957).

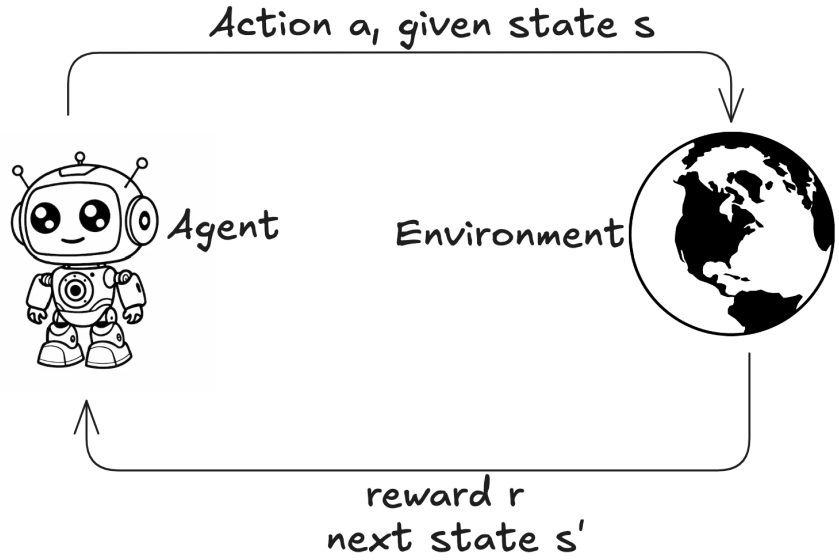


Figure 2.1: An environment represented by an MDP. An agent receives state s, takes an action a and receives next state s' and reward r.

2.1.1 Markov Decision Processes

A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, P, P_0, R, \gamma \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, P is the transition kernel $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ where $\mathcal{P}(X)$ is the set of all probability distributions on the set X, $P_0 \in \mathcal{P}(\mathcal{S})$ is the initial state distribution, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. After starting in an initial state $P_0(s)$, at each timestep t, the agent observes the state s_t, takes an action a_t drawn from its policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and receives reward $R(s_t, a_t)$. The environment then transitions to the next state according to the distribution $P(s_{t+1} \,|\, s_t, a_t)$. $\gamma \in [0, 1)$ is the discount factor.

2.1.2 Value Functions

The goal of the agent is to learn a policy that maximises its *discounted return*, which is defined by $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} R(s_{t'}, a_{t'})$. More specifically, the agent's goal is to maximise this return in expectation over the distribution over trajectories induced by the policy π. A trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots)$ is a sequence of states, actions and rewards. We can then state the joint probability distribution

over trajectories induced by the policy π as

$$P(\tau|\pi) = P(s_0)\prod_{t=0}^{\infty}\pi(a_t|s_t)P(s_{t+1}, r_{t+1}|s_t, a_t)$$

where $P(s_0)$ is the initial state distribution. For ease of notation, we have defined a joint distribution over the next states and rewards. This then allows us to define the RL objective as

$$J(\pi) = \mathbb{E}_{\tau\sim P(\tau|\pi)}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\right].$$

We can also define a trajectory distribution starting from a particular state s as

$$P(s_t = s, \tau|\pi) = P(s_t = s|\pi)\prod_{t'=t}^{\infty}\pi(a_{t'}|s_{t'})P(s_{t'+1}, r_{t'+1}|s_{t'}, a_{t'})$$

where $P(s_t = s|\pi)$ is the probability of the agent being in state s at time t under policy π.

A useful quantity is the *Value Function*, which is defined as the expected discounted return for a specific policy from a specific state by

$$V^{\pi}(s_t) = \mathbb{E}_{\tau\sim P(s_t=s,\tau|\pi)}\left[\sum_{t'=t}^{\infty}\gamma^{t'-t}R(s_{t'}, a_{t'})\right].$$

It's also useful to define the action-value, or Q, function as

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\tau\sim P(s_t=s,\tau|\pi)}\left[\sum_{t'=t}^{\infty}\gamma^{t'-t}R(s_{t'}, a_{t'})\right]$$

Intuitively, the value function and action-value function tell us how well a policy is expected to perform from a given state or when taking a given action in a particular state. For example, if the task is to navigate in a maze, and we are considering a uniformly random policy, we would expect the value function to have relatively high reward one step away from the centre of the maze compared to at the start, where the random policy is very unlikely to combine the correct actions to solve the maze. Similarly if we get r reward for solving the maze, the Q value of taking the correct action in the penultimate state would be r, whereas for a random policy its value would be $\frac{r}{N_a}$ where N_a is the number of actions, because there is only one action that leads to success and the rest to 0 reward.

We refer to the optimal policy as π^*. It is defined by $\pi^* \in \arg\max_{\pi} J(\pi)$. All optimal policies share the same value function, which we refer to as V^*. Similarly its action-value function is Q^*.

2.1.3 Value-based Methods

There are several approaches to learning an optimal policy π that rely on learning a value or action-value function. These are called *value-based* methods. These methods rely on the Bellman optimality equations (Bellman, 1966), which state:

$$V^*(s) = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[R(s,a) + \gamma V^*(s') \right].$$

More generally, we can define the Bellman operator (Bellman, 1952), $\mathcal{T}^\pi$:

$$\mathcal{T}^\pi[Q^\pi](s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(\cdot|s')} \left[R(s,a) + \gamma Q^\pi(s',a') \right].$$

The Bellman operator $\mathcal{T}^\pi$ has a fixed point, which corresponds to the true action value function Q^π.

We can use the Bellman operator for the optimal Q function, $\mathcal{T}^*$, to learn the optimal policy. In practice, we parameterise the Q function with parameters θ and update our network according to the loss function

$$\mathcal{L}(Q) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{B}} \left[\left(r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a) \right)^2 \right]$$

where s, a is the current state and action and s' is the next state. Training this way is referred to as *Q-learning* (Watkins and Dayan, 1992; Mnih et al., 2013), and is an instance of a broader family of methods referred to as temporal-difference learning (TD learning) (Sutton, 1988).

Training using modern neural networks in this way is unstable and does not learn. To solve this problem, DQN (Mnih et al., 2013) introduced a number of alterations to stabilise training. First, they use a *target network*, Q_{θ^-}. This is a lagged and periodically updated version of Q_θ which replaces Q_θ in the maximum taken in the loss function. This ensures that, rather than the targets changing at the same rate as the Q function updates, the targets are fixed for a time before being updated. DQN stores experience using a replay buffer, $\mathcal{B}$. This is a chunk of memory with a maximum size which stores tuples of (s, a, r, s'). Samples to

update on are then drawn uniformly from that buffer, and the network updated using stochastic gradient descent.

Unlike other methods in reinforcement learning (notably policy gradients), Q-learning has no requirement that the samples it trains on come from its current policy. This is known as being an *off-policy* method. DQN uses an ϵ-greedy policy as its exploration policy. This samples an action at random with probability ϵ, and samples the current best action, $\arg\max_a Q_\theta(s, a)$, with probability $1 - \epsilon$.

A related technique is used to learn value functions. This is another instance of TD learning. This updates the parameters of a learned value network V_θ as

$$\theta := \theta + \alpha(r + \gamma V_\theta(s') - V_\theta(s))\nabla V_\theta(s).$$

Note that, similarly to the use of target networks in DQN, this does not account for the gradient through $V_\theta(s')$ and hence is not a stochastic gradient of the mean-squared Bellman error objective.

2.1.4 Policy Gradient and Policy Optimisation

Another mechanism for learning optimal policies comes more directly from the RL objective. The policy gradient theorem (Sutton et al., 1999) states that for a learned policy π_θ

$$\nabla J(\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi(\cdot|s)}\left[Q^\pi(s, a)\nabla \log \pi(a|s)\right]$$

where d^π is the discounted steady-state distribution for the policy π, $d^\pi(s) = (1 - \gamma)\sum_{t=0}^{\infty} \gamma^t p_t^\pi(s)$, where $p_t^\pi(s)$ is the marginal distribution over states after t timesteps under policy π. This gives us a simple way of learning a policy via gradient ascent, provided that we can estimate $Q^\pi(s, a)$. The simplest way to do this is to use a Monte-Carlo estimate of $Q^\pi(s, a)$, the empirical return of the episode used for training, G_t. This gives rise to the REINFORCE algorithm (Williams, 1992). This works by iterating through the following steps:

1. Roll out the policy π_θ, collecting a batch $\mathcal{B}$ of tuples s_t, a_t, G_t

2. Update π_θ with

$$\theta := \theta + \alpha \mathbb{E}_{s,a,G\sim\mathcal{B}} \left[G \nabla \log \pi(a|s) \right]$$

A significant problem with this approach is the high variance of the return estimates G_t, which leads to very noisy gradients and unstable learning. To counteract this, two techniques are often employed. The first is to learn a representation of the value function, V_ϕ. Then, instead of using G_t as the Q^π estimator, it is instead possible to use $r_t + \gamma V^\pi(s_{t+1})$, where r_t is the reward received and s_{t+1} is the next state. Additionally, we can use any state-dependent function $b(s)$ with 0 expectation as a baseline. It can be shown that using $V^\pi(s)$ minimises the variance of the resulting estimator (Sutton, 1984). This gives the form

$$\nabla J(\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi(\cdot|s)} \left[(Q^\pi(s,a) - V^\pi(s)) \nabla \log \pi(a|s) \right].$$

The quantity $Q^\pi(s,a) - V^\pi(s)$ is called the *advantage function* and written $A^\pi(s,a)$. The policy we learn using this method is called the *actor*. It is possible to estimate the advantage using $r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$. Learning a value function using TD learning (the *critic*) and then using it to estimate the advantage, along with learning the policy using the policy gradient above, is referred to as an *actor-critic* algorithm.

Actor-critic algorithms are significantly more stable than REINFORCE, but still suffer from instability. This instability can arise because actor-critic algorithms aim to produce, via gradient descent, a capped-size update in *parameter* space, but do not place any restriction on the effect such an update might have in *policy* space. *Policy optimisation* algorithms are a family of methods aiming to do this. The most commonly used algorithm in this family is proximal policy optimisation (Schulman et al., 2017), which is a heuristic version of trust-region policy optimisation (Schulman et al., 2015). It learns a new policy $\tilde{\pi}$ given a current policy π by optimising the following objective

$$\mathbb{E}_{s\sim d^\pi, a\sim\pi} \left[\min \left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} A^\pi(s,a), \operatorname{clip} \left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}, 1-\epsilon, 1+\epsilon \right) A^\pi(s,a) \right) \right],$$

where clip is defined by

$$\operatorname{clip}(x,y,z) = \begin{cases} x & y < x < z \\ y & x \leq y \\ z & x \geq z \end{cases}.$$

2.1.5 Nonstationarity in Reinforcement Learning

Unlike supervised learning, RL is nonstationary. This means that the objective being optimised changes throughout optimisation. To see this, consider TD learning. As previously seen, this involves minimising the loss

$$\mathbb{E}_{(s,a,s',r)\sim\mathcal{B}}\left[(r+\gamma\,\mathrm{sg}\{V_\theta(s')\}-V_\theta(s))^2\right]$$

where sg is the stop gradient (i.e. the identity function with a zero gradient everywhere). Here we are attempting to minimise the distance between $V_\theta(s)$ and $r+\gamma V_\theta(s')$, but V_θ changes every iteration and with it the targets of the optimisation. This does not arise only in TD learning. Any time we update our policy, the return distribution, on which our objective function must necessarily depend to improve, changes.

2.1.6 Reinforcement Learning as Sequential Stationary Optimisation

To help to stabilise training in the face of this nonstationarity, different RL methods use different techniques. DQN stabilises training via the use of a target network. Target networks typically are fixed for a given number of steps before being updated to the current version of the network. In this way, DQN transforms a nonstationary optimisation problem into a sequence of stationary ones.

Similarly, PPO can also be phrased as a sequence of stationary optimisation problems. Like REINFORCE, PPO repeats the following steps:

1. Rollout the policy π in parallel to collect (s,a,r) tuples

2. Use this data and the learned value function to compute the advantages A_t and values V_t

3. Update the policy and value function by optimising for a fixed number of steps.

Crucially, because the values are pre-computed after the rollouts, during the optimisation all targets of both policy learning and value function learning are fixed. This means that PPO similarly exists as a sequence of stationary optimisations.

2.2 Infrastructure and Tooling for Reinforcement Learning

As machine learning methods have grown in compute scale and sophistication, so too has the tooling and infrastructure required to support them. This section will provide some background on the development of that tooling and infrastructure for reinforcement learning.

2.2.1 Training and Distributed Training

The most obvious piece of infrastructure that has developed is the ML training framework. Early systems such as Theano (Al-Rfou et al., 2016) were designed to support training of neural networks on CPUs. Later, TensorFlow (Abadi et al., 2016) was designed to support large-scale training of neural networks on GPUs. Early TensorFlow versions required defining the computational graph ahead of time and then compiling and executing it. This hampered debuggability. Other define-by-run frameworks such as Torch7 (Paszke et al., 2019), Chainer (Tokui et al., 2015) and DyNet (Neubig et al., 2017) allowed graphs to be defined and executed more flexibly. These frameworks inspired the design of PyTorch (Paszke et al., 2019), which, along with JAX (Bradbury et al., 2018), have become the dominant choices for training neural networks.

In addition to training frameworks becoming easier to use through better debuggability, distributed training has allowed massive experimental scale. Pre-training experiments involving thousands of GPUs, millions of dollars and many billions of parameters are now commonplace. NVIDIA's NCCL collective communication primitives (NVIDIA, 2024b) allowed the building of sophisticated distributed training frameworks with many-dimensional parallelism meshes.

Initially, distributed training was simple distributed data parallel. Frameworks such as Horovod (Sergeev and Del Balso, 2018) allowed large batches to be split across multiple GPUs, with gradient synchronisation happening through all-reduce operations. Later, distributed optimizer implementations such as ZeRO (Rajbhandari et al., 2020) and NVIDIA's Apex (NVIDIA, 2024a) allowed the memory-intensive optimiser states to be partitioned across the data parallel workers. Goyal et al. (2017) used this distributed training to demonstrate the possibility of training ImageNet in one hour.

However, data parallel can only allow scaling via bigger batches. It does not allow training models which cannot fit on a single GPU. Pipeline parallelism (Huang et al., 2019) allowed models to be split layerwise across multiple GPUs. When done poorly, this can cause poor GPU efficiency via 'pipeline bubbles', where one GPU is sitting idle waiting for another to complete its work. Huang et al. (2019) mitigated this via micro-batching. This splits each batch into smaller micro-batches, significantly improving GPU utilisation. PipeDream (Narayanan et al., 2019) allowed GPUs to alternate between forward and backward passes, reducing the number of idle GPUs and improving GPU efficiency.

In addition to pipeline parallelism, modern distributed training frameworks use tensor model parallelism to split model parameters across multiple GPUs. Instead of splitting networks layerwise, tensor parallelism splits individual tensors across multiple GPUs. Each rank of the tensor parallel group then either processes a single row of the input tensor or a single column of the output tensor. When the full output is required, an all-gather or all-reduce operation is used to combine results. To minimise communication overhead, tensor parallelism is typically used within a node, with pipeline parallelism used between nodes. Megatron-LM (Shoeybi et al., 2019) made tensor parallelism practical for training multi-billion parameter models, while Pytorch's FSDP implementation allows for practical training up to a few hundred GPUs.

These parallelism techniques can be combined orthogonally to form multi-dimensional parallelism meshes. This allows for data and model parallelism to be

combined. This combination underlies the efficiency of modern distributed training frameworks, which can comfortably scale model size to trillions of parameters and data scale to tens of trillions of tokens.

2.2.2 Reinforcement Learning Training Frameworks and Infrastructure

In addition to the infrastructure for training neural networks using supervised learning, reinforcement learning has its own infrastructure challenges. A common approach is Distributed Reinforcement Learning (Espeholt et al., 2018; Nair et al., 2015). In this approach, *actor* workers roll out the policy in parallel and collect trajectories, before writing them to a *buffer*. The *learner* then pulls samples from this buffer and uses them to update its policy, before synchronising the policy with the actors. Several early approaches to this used off-policy Q-learning based methods (Nair et al., 2015; Horgan et al., 2018; Kapturowski et al., 2018). This has the natural advantage of the rollout policy and learned policy not needing to be as frequently synchronised as on-policy methods. Espeholt et al. (2018) instead used an on-policy actor-critic method and define a V-trace estimator using truncated importance sampling to correct for differences between the rollout policy and the learned policy. This actor-critic method of distributed training has found application in the post-training of LLMs (Team et al., 2025; Guo et al., 2025), where highly optimised inference servers (Kwon et al., 2023) are used to generate rollouts on a wide range of tasks. The model is then trained using a variant of PPO (Schulman et al., 2017) with grouped rollout advantage estimation (Guo et al., 2025). This avoids learning an expensive critic network. To improve the training-inference mismatch, weights are beamed every learner step between the inference server and the trainer using optimised point-to-point GPU communication primitives from Nvidia's NIXL library (ai-dynamo contributors, 2025). This PipelineRL (Piché et al., 2025) approach allows for much greater throughput by allowing the inference servers to hot-reload their weights. This is in contrast to a traditional synchronous

approach, where it would be necessary to wait for the slowest sequences to complete before training can continue.

In addition to attempts at scaling reinforcement learning training across many machines, there has recently been a significant body of work focussed on accelerating RL workloads with smaller compute budgets. Traditionally, in academic settings, policy rollouts are most often performed with the environment code running on the CPU, while the policy updates are performed on the GPU.

However, more recent tooling such as JAX (Bradbury et al., 2018), has made it significantly easier to write environment code for the GPU. This has led to recent work (Lange, 2022; Lu et al., 2022; Dalton et al., 2020) running both the environment rollouts and learning on the GPU. This has a number of significant advantages. First, running all code on the GPU means that it can benefit from just-in-time (JIT) compilation, which significantly improves performance. Secondly, data need no longer be transferred from CPU to GPU memory when training. Thirdly, the environments can then be parallelised much more easily using JAX's `vmap` operation, which, in addition to its increased ease of use, allows much wider parallelisation than typically available when running environments on CPU. These developments can lead to significant speedups, with PureJAXRL achieving a 4000x speedup when parallelising over agent training (Lu et al., 2022). JaxMARL (Rutherford et al., 2023), which is included in this thesis, introduces this JAX-based training approach to multi-agent reinforcement learning. This leads to significant speed-ups compared to more traditional training approaches. Another approach is that of PufferLib (Suarez, 2024), which uses asynchronous learning to improve the speed of training RL on a single GPU. This can lead to similar speedups to those achieved by native JAX environments.

2.2.3 Reinforcement Learning Environments

Another extremely important area of reinforcement learning tooling is the development of environments. The breadth and complexity of available popular environments has a significant impact on the progress and direction of research.

Many environments in reinforcement learning were based on games, which provide a natural testbed for reinforcement learning agents. Games are customisable, and can easily be simulated on a GPU or CPU. For example, the Atari learning environment (Bellemare et al., 2013) made a collection of over 50 Atari 2600 games available as playable RL environments. Deepmind Lab (Beattie et al., 2016) provided a suite of first-person 3D environments for training agents. Procgen (Cobbe et al., 2019a) provided a suite of procedurally generated environments with special held-out levels for testing.

Another popular category of reinforcement learning environments has been continuous control. The MuJoCo (Todorov et al., 2012) physics engine was used to develop a suite of continuous control environments with different complexities (Duan et al., 2016). These environments require the control of robots with complex topologies using a multi-dimensional continuous action space.

However, the design of environments is very difficult, and there has been a steady stream of papers pointing out the limitations of even extremely popular environments. For example, Mania et al. (2018) demonstrated that simple random search of static linear policies is competitive with common reinforcement learning methods. Machado et al. (2018) showed that the Atari learning environment was not stochastic, and therefore could be solved with a simple open-loop policy. They then proposed using 'sticky actions' to mitigate this issue and add stochasticity to the benchmark. Within this thesis, SMACv2 (Ellis et al., 2023) makes a similar critique of the popular SMAC benchmark, demonstrating that it is not sufficiently stochastic to require complex closed-loop policies and proposes using procedural generation to make it more stochastic and partially observable.

Another, related critique of reinforcement learning environments is the use of the same distributions for training and testing. Cobbe et al. (2019b) investigated this and found that models can overfit to a wide training distribution, but that when held-out levels are used for testing, regularisation techniques from supervised learning, such as weight decay, again become necessary.

2.3 Multi-Agent Reinforcement Learning

Although RL is suitable for many environments, it does not account for many settings we encounter in the real world. Whether cooking in a kitchen, driving a car, or at work, we must collaborate with people every day. Multi-agent reinforcement learning (Albrecht et al., 2024) aims to bridge this gap.

Broadly, multi-agent RL can be divided into cooperative, competitive and mixed settings. In cooperative MARL, all agents' goals are perfectly aligned through a shared reward. For example, we might reward agents each time they successfully collaborate to save victims from a burning building. Competitive MARL is the exact opposite – agents have directly opposite goals. Many competitive settings are games such as chess, go, and poker. In mixed settings, each agent has its own reward function that it must optimise. Agents will therefore collaborate or compete as they see fit. A very famous example of a mixed setting is the prisoner's dilemma, where each prisoner can choose to either defect or cooperate as they see fit.

The work in this thesis is mostly in the cooperative setting, and my contributions to those papers were entirely in the cooperative setting. I will therefore provide an introduction to only the cooperative setting below.

2.3.1 Decentralised Partially Observable Markov Decision Processes

The theoretical framework used for modelling cooperative multi-agent RL settings is the decentralised partially observable Markov process (Oliehoek and Amato, 2016). A Dec-POMDP is a tuple $\langle N, \mathcal{S}, \mathcal{A}, \mathcal{O}, P, P_0, O, r, \gamma \rangle$, where N is the number of agents, $\mathcal{S}$ is a collection of states, $\mathcal{A}$ is the set of joint actions, $\mathcal{O}$ is the set of joint observations, P is the transition kernel, P_0 is the initial state distribution, $O : \mathcal{S} \to \mathcal{P}(\mathcal{O})$ is the observation kernel, r is the reward function and γ is the discount factor. $\mathcal{S}$, P, r, γ and P_0 behave exactly as in the MDP setting. Throughout this background section we will use the subscript i to refer to an individual agent. So a_i is the ith agent's action, o_i its observation and so on. $\mathcal{A}$ rather than being

an individual action is now a set of *joint* actions $\mathbf{a} = (a_1, a_2, \ldots, a_N)$. The largest change is the presence of *observations*.

At each timestep, rather than observing the underlying Markov state, the agents receive a joint observation $\mathbf{o} = (o_1, o_2, \ldots, o_N)$. The agents must hence each learn a policy $\pi_i : \mathcal{O}_i^* \to \mathcal{P}(\mathcal{A}_i)$ that conditions on an observation history $h_i \in \mathcal{O}_i^*$, where X^* is the set of all sequences from the set X and $\mathcal{O}_i$ is the observation space for agent i. Because each agent receives exactly the same reward, such environments are inherently cooperative – all agents must collaborate to improve reward. Conditioning on the observation history is necessary because each agent's observation does not reveal the underlying Markov state.

2.3.2 Key Properties of Dec-POMDPs

Dec-POMDPs, like MDPs, are stochastic. This is essential to their complexity. If P, P_0 and O were deterministic, it would be possible to choose an optimal sequence of joint actions $\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^t, \ldots$ rather than having to condition on the observation received and learn a joint policy $\boldsymbol{\pi}$.

Secondly, Dec-POMDPs are partially observable. This is essential to their representing a multi-agent setting. Without partial observability, all agents know the same information, and therefore the setting reduces to an MDP with one centralised agent that selects joint actions. This centralised setting is known as a multiagent MDP (Boutilier, 1996).

2.3.3 Centralised Training with Decentralised Execution

A common setting in MARL is *centralised training with decentralised execution* (Lowe et al., 2017; Foerster et al., 2018). This setting replicates situations in which it might be possible during training in a lab setting to have access to more accurate sensors, or a more accurate model of the world. In this setting agents have access to some centralised information (often the state s) at training time, but must use this to learn *decentralised* policies $\pi_i : \mathcal{O}_i^* \to \mathcal{P}(\mathcal{A}_i)$ at test time.

2.3.4 Independent Learning

The simplest way to adapt a single-agent RL algorithm for the multi-agent setting is to simply run that algorithm per-agent, treating all the other agents as part of the environment. This approach is called *independent learning*. Independent PPO (de Witt et al., 2020; Yu et al., 2022; Foerster et al., 2018) is perhaps the most common algorithm using this approach and performs very well across a number of domains.

Despite its strong performance, the constantly updating ally policies make the environment *non-stationary*, which can make learning more difficult and present convergence problems. This is particularly true for independent Q-learning (Tan, 1993), where changes in Q-values can lead to large changes in the implied policy.

2.3.5 Value-based Multi-agent Reinforcement Learning

Like in single-agent reinforcement learning, MARL methods are split into value-based and policy-gradient methods. In a multi-agent setting, these methods learn a *joint* Q-function $\boldsymbol{Q}(\boldsymbol{o}, \boldsymbol{a})$. This can be done as in the single-agent case using a Q-learning loss:

$$\mathcal{L}(\boldsymbol{Q}) = \mathbb{E}_{(\boldsymbol{h},\boldsymbol{a},r,\boldsymbol{h'})\sim\mathcal{B}}\left[\left(r + \gamma \max_{\boldsymbol{a'}} \boldsymbol{Q}_{\theta^-}(\boldsymbol{h'}, \boldsymbol{a'}) - \boldsymbol{Q}_{\theta}(\boldsymbol{h}, \boldsymbol{a})\right)^2\right]$$

This can be stabilised using replay buffers and target networks as in single-agent learning. However, once this has trained there is potentially a problem. It is not clear that this *centralised* Q function can easily be distilled into *decentralised* policies.

A common way of resolving this problem is to learn the joint Q-function by combining Q-functions learnt per agent and then forcing the joint Q-function to abide by the individual global-max (Son et al., 2019) constraint:

$$\arg\max_{\boldsymbol{a}} \boldsymbol{Q}(\boldsymbol{h}, \boldsymbol{a}) = \begin{pmatrix} \arg\max_{a_0} Q_0(h_0, a_0) \\ \arg\max_{a_1} Q_1(h_1, a_1) \\ \vdots \\ \arg\max_{a_{N-1}} Q_{N-1}(h_{N-1}, a_{N-1}) \end{pmatrix}.$$

If this condition holds, then we can always recover the joint Q-function's action simply by choosing for each agent the action that maximises their Q-function.

The simplest way to achieve this is to combine the individual Q-functions with a simple addition

$$\boldsymbol{Q}(\boldsymbol{h}, \boldsymbol{a}) = Q_0(h_0, a_0) + Q_1(h_1, a_1) + \cdots + Q_{N-1}(h_{N-1}, a_{N-1}).$$

This method is known as VDN (Sunehag et al., 2017) and performs very well in practice (Gallici et al., 2024; Rutherford et al., 2023), although due to its simple form it can only learn a subset of the functions representable via the IGM constraint.

A more sophisticated version of this is to learn a monotonic mixing network for the individual Q-functions. This is known as QMIX (Rashid et al., 2018). QMIX learns a hypernetwork which outputs strictly positive weights for the mixing network, thereby enforcing the monotonicity constraint.

2.3.6 Multi-Agent Reinforcement Learning Policy Gradient Methods

The application of typical policy gradient and policy optimisation algorithms to multi-agent settings is generally more straightforward. Broadly these algorithms can be divided into those which learn individual critics (and are therefore independent learning algorithms such as IPPO) and those which use a single global critic (e.g. Multi-Agent PPO (Yu et al., 2022)).

Although these two approaches seem fairly different, it can be shown that this design choice amounts to a bias-variance trade-off (Lyu et al., 2021). While there are other notable approaches, such as COMA (Foerster et al., 2018), which uses a counterfactual baseline to improve credit assignment, these algorithms are not used in practice due to either poor performance or unnecessary complexity.

2.3.7 The StarCraft Multi-Agent Challenge

A common MARL benchmark is the StarCraft Multi-Agent Challenge (Samvelyan et al., 2019). This benchmark focuses on the *micromanagement* aspect of StarCraft II, rather than, as in AlphaStar (Vinyals et al., 2019), tackling the full game, which also requires complex resource and economy management. In SMAC, two teams

of units are pitched against each other, with one side (referred to as the enemies) being controlled by a heuristic policy, and another (the allies) by the policy learned during training. Each allied unit is controlled by a separate decentralised policy.

Because this is a CTDE benchmark, during training algorithms have access to a centralised state, but must condition only on observation histories at test time. SMAC features 14 micromanagement scenarios that can broadly be divided into three categories: *symmetric*, *asymmetric* and *micro-trick*. The symmetric scenarios have equal numbers of units on each side. The asymmetric scenarios increase the difficulty by adding more units to the enemy side. Micro-trick scenarios feature a specific situation in which some micromanagement technique must be learned. For example, `3s_vs_5z` requires the stalkers, which have a large attack range, to 'kite' the zealots by running away and attacking from outside the zealots' attack range.

SMAC has been enormously popular as a benchmark (Gorsane et al., 2022), dominating the evaluations of submitted multi-agent RL papers to the top conferences. Gorsane et al. (2022) examined the usage of SMAC in MARL papers accepted at top conferences and found a number of issues including inconsistent usage of StarCraft versions, only evaluating on a subset of maps, inconsistencies in reported benchmark performance, and saturating benchmark performance.

Part I

Improving Reinforcement Learning Evaluation

3

SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning

I will start the content in this thesis by examining benchmarks in MARL. Benchmarks are vital to RL research. Better performance on benchmarks is the typical yardstick through which researchers measure algorithmic progress. Therefore any issues in benchmarks imply issues in the evaluation of papers that use them.

In this chapter, we examine the popular CTDE environment the StarCraft Multi-Agent Challenge (SMAC). While this has been central to the evaluation of many MARL algorithms, it also has documented issues. First, it is not difficult enough, with MAPPO, IPPO and QMIX all achieving nearly perfect scores on nearly all scenarios (Yu et al., 2022; Hu et al., 2021b). Secondly, Gorsane et al. (2022) found issues with its usage, with lack of standardisation of scenario choices, poor documentation of evaluation choices, and StarCraft version inconsistencies all being common flaws. In addition to these issues, we uncover two, more fundamental, issues in SMAC's design.

We begin with a demonstration that SMAC lacks *stochasticity*. This is a key problem because it implies that policies learned in SMAC need not pay attention to their observation, and hence that SMAC does not present a significant in-distribution

generalisation challenge. To show this, we train *open-loop* policies, namely those that condition only on the timestep and agent ID. Each agent trained this way only knows which agent it is, and the current episode timestep. All information that you might think is relevant to succeed in SMAC, such as the location of allies and enemies, is not available. Despite this, we find policies that perform as well as their closed-loop counterparts in a significant proportion of SMAC scenarios. To reinforce this, we examine the Q-values of learned policies. We show that by removing all of the information in the observation except the timestep, it is possible to regress to the Q-value with a high accuracy. This is further evidence of the lack of stochasticity present in the environment.

We also point out that SMAC suffers from a lack of *meaningful partial observability*. This concept is a refinement of the concept of a *fully observable* Dec-POMDP (Oliehoek and Amato, 2016). Meaningful partial observability is information relevant to one agent's policy that is unknown and uninferrable to it, but known (or inferrable) to another agent. Without meaningful partial observability, a multi-agent setting is not truly decentralised. SMAC is not meaningfully partially observable because its large sight ranges render almost all information common among agents.

To address the stochasticity issue, we introduce SMACv2. This new benchmark randomises the start position and unit type of allies and enemies while retaining winnable scenarios. Therefore each agent at the start of each episode does not know where its allies and enemies are, and what unit types are on it or the enemy team. We show that this addresses the stochasticity problems in SMAC by both training open-loop policies and demonstrating they do not succeed, and by analysing the Q-values of trained policies.

We also introduce the Extended Partial Observability challenge (EPO). This addresses SMAC's lack of meaningful partial observability by only making enemies generally visible with a fixed probability, while guaranteeing that any enemy can be seen by at least one ally. We demonstrate that this leads to a communication gap, where perfect observability, which corresponds to perfect communication,

recovers high win rates, while low probability of observing an enemy's properties leads to much lower win rates.

arXiv:2212.07489v2 [cs.LG] 17 Oct 2023

SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning

Benjamin Ellis[1*] **Jonathan Cook**[1] **Skander Moalla**[1,5] **Mikayel Samvelyan**[2,3]
Mingfei Sun[4] **Anuj Mahajan**[1] **Jakob N. Foerster**[1] **Shimon Whiteson**[1]

[1]University of Oxford [2]University College London [3]Meta AI [4]University of Manchester
[5]EPFL

Abstract

The availability of challenging benchmarks has played a key role in the recent progress of machine learning. In cooperative multi-agent reinforcement learning, the StarCraft Multi-Agent Challenge (SMAC) has become a popular testbed for the centralised training with decentralised execution paradigm. However, after years of sustained improvement on SMAC, algorithms now achieve near-perfect performance. In this work, we conduct new analysis demonstrating that SMAC lacks the stochasticity and partial observability to require complex *closed-loop* policies (i.e., those that condition on the observation). In particular, we show that an *open-loop* policy conditioned only on the timestep can achieve non-trivial win rates for many SMAC scenarios. To address this limitation, we introduce SMACv2, a new benchmark where scenarios are procedurally generated and require agents to generalise to previously unseen settings during evaluation.[2] We show that these changes ensure the benchmark requires the use of *closed-loop* policies. We also introduce the extended partial observability challenge (EPO), which augments SMACv2 to ensure meaningful partial observability. We evaluate state-of-the-art algorithms on SMACv2 and show that it presents significant challenges not present in the original benchmark. Our analysis illustrates that SMACv2 addresses the discovered deficiencies of SMAC and can help benchmark the next generation of MARL methods. Videos of training are available on our website.

1 Introduction

In many real-world cases, control policies for cooperative multi-agent reinforcement learning (MARL) can be learned in a setting where the algorithm has access to all agents' observations, e.g., in a simulator or laboratory, but must be deployed where such centralisation is not possible. Centralised training with decentralised execution (CTDE) [10, 25] is a paradigm for tackling these settings and has been the focus of much recent research [32, 44, 12, 26, 27]. In CTDE, the learning algorithm sees all observations during training, as well as possibly the Markov state, but must learn policies that can be executed without such privileged information.

As in other areas of machine learning, benchmarks have played an important role in driving progress in CTDE. Examples of such benchmarks include the Hanabi Learning Environment [2], Google football [20], PettingZoo [46], and Multi-Agent Mujoco [31]. Perhaps the most popular one is the StarCraft Multi-Agent Challenge (SMAC) [37], which focuses on decentralised micromanagement challenges in the game of StarCraft II. Rather than tackling the full game with centralised control [50], SMAC tasks a group of learning agents, each controlling a single army unit, to defeat the units of the

*Correspondence to benellis@robots.ox.ac.uk.

[2]Code is available at `https://github.com/oxwhirl/smacv2`

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks.



(a) Terran (b) Zerg (c) Protoss

Figure 1: Screenshots from SMACv2 showing agents battling the built-in AI.

enemy army controlled by the built-in heuristic AI. SMAC requires learning several complex joint action sequences such as focus fire,[3] and kiting enemy units,[4]. For these reasons, many prominent MARL papers [51, 26, 52, 33, 9, 55, 53] rely on SMAC to benchmark their performance.[5]

However, there is significant evidence that SMAC is outliving its usefulness [55, 13, 11]. Recent work reports near-perfect win rates on most scenarios [55, 13] and strong performance without using the centralised state via independent learning [55, 8]. This suggests that, due to ceiling effects, SMAC may no longer reward further algorithmic improvements and that new benchmarks are needed.

In this paper, we present new, more fundamental problems with SMAC and hence reason for new benchmarks. In particular, we show that an open-loop policy, which conditions only on the timestep and ignores all other observations, performs well on many SMAC scenarios. We also extend this analysis by examining the estimates of the joint Q-function learnt by QMIX. We demonstrate that, even with all features masked, regression to the joint Q-function is possible with less than 10% error in all but two scenarios from just timestep information.

Together these results suggest that SMAC is not stochastic enough to necessitate complex closed-loop (i.e. conditioned on the observation) control policies on many scenarios. Therefore, although SMAC scenarios may require difficult to discover action sequences such as focus fire, agents do not need to adapt to a diverse range of situations, but can largely repeat a fixed action sequence with little deviation. Additionally, *meaningful partial observability* (see Section 4) is minimal in SMAC due to large fields-of-view. For partial observability to be meaningful, one agent must observe information that is relevant to the current or future action selection of another agent, unknown to that other agent, and uninferrable from the other agent's observation. *Meaningful partial observability* is crucial to decentralisation and the NEXP-completeness of Dec-POMDPs [3].

To address these shortcomings, we propose *SMACv2*, a new benchmark that uses procedural content generation (PCG) [36] to address SMAC's lack of stochasticity. In SMACv2, for each episode we randomly generate team compositions and agent start positions. Consequently, it is no longer sufficient for agents to repeat a fixed action sequence, but they must learn to coordinate across a diverse range of scenarios. To remedy the lack of meaningful partial observability, we introduce the extended partial observability challenge (EPO), where we only allow the first agent to spot an enemy to see it for certain when it is in range. This requires agents to implicitly communicate enemy information to prioritise targets. Additionally, we update the sight and attack ranges to increase the diversity of the agents and complexity of scenarios.

Our implementation is extensible, allowing researchers to implement new distributions easily and hence further expand on the available SMACv2 scenarios. We also analyse SMACv2 and demonstrate that the added stochasticity prevents an open-loop policy from learning successfully, but instead requires policies to condition on ally and enemy features.

We evaluate several state-of-the-art algorithms on SMACv2 and show that they struggle with many scenarios, confirming that SMACv2 poses substantial new challenges. Finally, we perform an ablation on the new observation features to demonstrate how each contributes to the difficulty, highlighting the specific areas on which future MARL research should focus.

[3] In focused fire, allied units jointly attack and kill enemy units one after another.

[4] Kiting draws enemy units to give chase while maintaining distance so that little damage is incurred.

[5] SMAC has been cited over 600 times on Google Scholar.

2 Related Work

The MARL community has made significant use of games for benchmarking cooperative MARL algorithms. The Hanabi Learning Environment [2] tasks agents with learning to cooperate in the card-game Hanabi. Here, each agent observes the cards of its teammates but not its own, which must be implicitly communicated via gameplay. Hanabi is partially observable and stochastic, but only features teams of 2 to 5 players, which is fewer than all but the smallest SMACv2 scenarios. Kurach et al. [20] propose Google Football as a complex and stochastic benchmark, and also have a multi-agent setting. However, it assumes fully observable states, which simplifies coordination.

Peng et al. [31] propose a multi-agent adaptation of the MuJoCo environment featuring a complex continuous action space. Multi-Particle Environments (MPE) [25] feature simple communication-oriented challenges where particle agents can move and interact with each other using continuous actions. In contrast to both multi-agent MuJoCo and MPE, SMACv2 has a discrete action space, but challenges agents to handle a wide range of scenarios using procedural content generation. A plethora of work in MARL uses grid-worlds [25, 23, 54, 4, 48, 35, 24] where agents move and perform a small number of discrete actions on a 2D grid, but these tasks have much lower dimensional observations than SMACv2. OpenSpiel [22] and PettingZoo [46] provide collections of cooperative, competitive, and mixed sum games, such as grid-worlds and board games. However, the cooperative testbeds in either of these suites feature only simple environments with deterministic dynamics or a small number of agents. Neural MMO [41] provides a massively multi-agent game environment with open-ended tasks. However, it focuses on emergent behaviour within a large population of agents, rather than the fine-grained coordination of fully cooperative agents. Furthermore, none of these environments combines meaningful partial observability, complex dynamics, and high-dimensional observation spaces, whilst also featuring more than a few agents that need to coordinate to solve a common goal.

StarCraft has been frequently used as a testbed for RL algorithms. Most work focuses on the full game whereby a centralised controller serves as a puppeteer issuing commands for the two elements of the game: *macromanagement*, i.e., the high-level strategies for resource management and economy, and *micromanagement*, i.e., the fine-grained control of army units. TorchCraft [45] and TorchCraftAI [1] provide interfaces for training agents on *StarCraft: BroodWar*. The StarCraft II Learning Environment (SC2LE) [49] provides a Python interface for communicating with the game of *StarCraft II* and has been used to train AlphaStar [50], a grandmaster-level but fully centralised agent that is able to beat professional human players. SMAC and SMACv2 are built on top of SC2LE and concentrate only on decentralised unit micromanagement for the CTDE setting.

One limitation of SMAC is the constant starting positions and types of units, allowing methods to memorise action sequences for solving individual scenarios (as we show in Section 5.1), while also lacking the ability to generalise to new settings at test time, which is crucial for real-world applications of MARL [28]. To address these issues, SMACv2 relies on procedural content generation [PCG; 36, 18] whereby infinite game levels are generated algorithmically and differ across episodes. PCG environments have recently gained popularity in single-agent domains [7, 6, 17, 21, 38] for improving generalisation in RL [19] and we believe the next generation of MARL benchmarks should follow suit. Iqbal et al. [15] and Mahajan et al. [28] consider updated versions of SMAC by randomising the number and types of the units, respectively, to assess the generalisation in MARL. However, these works do not include the random start positions explored in SMACv2, analyse the properties of SMAC to motivate these changes, or address SMAC's lack of meaningful partial observability. They also do not change the agents' field-of-view and attack ranges or provide a convenient interface to generate new distributions over these features.

3 Background

3.1 Dec-POMDPs

A partially observable, cooperative multi-agent reinforcement learning task can be described by a *decentralised partially observable Markov decision process (Dec-POMDP)* [30]. This is a tuple $(n, S, A, T, \mathbb{O}, O, R, \gamma)$ where n is the number of agents, S is the set of states, A is the set of individual actions, $T : S \times A^n \to \Delta(S)$ is the transition probability function, $\mathbb{O}$ is the set of joint observations, $O : S \times A^n \to \Delta(\mathbb{O})$ is the observation function, $R : S \times A^n \to \Delta(\mathbb{R})$ is the reward function, and γ is the discount factor. We use $\Delta(X)$ to denote the set of probability distributions over a set X. At

each timestep t, each agent $i \in \{1, \dots, n\}$ chooses an action $a \in A$. The global state then transitions from state $s \in S$ to $s' \in S$ and yields a reward $r \in \mathbb{R}$ after the joint action $\mathbf{a}$, according to the distribution $T(s, \mathbf{a})$ with probability $\mathbb{P}(s'|s, \mathbf{a})$ and $R(s, \mathbf{a})$ with probability $\mathbb{P}(r'|s, \mathbf{a})$, respectively. Each agent then receives an observation according to the observation function O so that the joint observation $\mathbf{o} \in \mathbb{O}$ is sampled according to $O(s, \mathbf{a})$ with probability $\mathbb{P}(\mathbf{o}|s, \mathbf{a})$. This generates a trajectory $\tau_i \in (\mathbb{O} \times A^n)^*$. The goal is to learn n policies $\pi_i : (\mathbb{O}_i \times A)^* \to \Delta(A)$ that maximise the expected cumulative reward $\mathbb{E}[\sum_i \gamma^i r_{t+i}]$.

3.2 StarCraft Multi-Agent Challenge (SMAC)

Rather than tackling the full game, the StarCraft Multi-Agent Challenge [37, SMAC] benchmark focuses on micromanagement challenges where each military unit is controlled by a single learning agent. Units have a limited field-of-view and no explicit communication mechanism when at test time. By featuring a set of challenging and diverse scenarios, SMAC has been used extensively by the MARL community for benchmarking algorithms. It consists of 14 micromanagement scenarios, which can be broadly divided into three different categories: *symmetric*, *asymmetric*, and *micro-trick*. The symmetric scenarios feature the same number and type of allied units as enemies. The asymmetric scenarios have one or more extra units for the enemy and are often more difficult than the symmetric scenarios. The micro-trick scenarios feature setups that require specific strategies to counter. For example, `3s_vs_5z` requires the three allied stalkers to kite the five zealots, and `corridor` requires six zealots to block off a narrow corridor to defeat twenty-four zerglings without being swarmed from all sides. Two state-of-the-art algorithms on SMAC are MAPPO [55] and QMIX [32]. We provide background to these methods in Section 2 of the appendix.

4 Partial Observability and Stochasticity in CTDE

If the initial state and transition dynamics of a Dec-POMDP are deterministic, then open-loop policies suffice, i.e., there exists an optimal joint policy that ignores all information but the timestep and agent ID. Such a policy amounts to playing back a fixed sequence of actions. **One way of determining whether SMAC has sufficient stochasticity therefore is to measure how strongly the learned policies condition on their observations**. For example, in a mostly deterministic environment, a policy may only be required to condition on a small subset of the observation space for a few timesteps, whereas a more stochastic environment might require conditioning on a large number of observation features. We use this concept in section 5.2 to evaluate SMAC's stochasticity.

Since open-loop policies do not rely on observations, the observation function and any partial observability it induces only become relevant when the environment is stochastic enough to require closed-loop policies. However, even in stochastic environments not all partial observability is meaningful. If the hidden information is not relevant to the task, or can be inferred from the observation, it is resolvable by a single agent alone, and if it is unknown and undiscoverable by any of the agents, then it cannot contribute to solving the task. **Without meaningful partial observability, a task cannot be considered truly decentralised because a policy can either ignore or infer any unknown information *without the intervention of other agents***. Meaningful partial observability is essential to the difficulty (and NEXP-completeness) of Dec-POMDPs [3]. While SMAC is technically partially observable, there is limited hidden information because large fields-of-view render most relevant observations common among agents.

5 Limitations of SMAC

In this section, we analyse stochasticity in SMAC by examining the performance of open-loop policies and the predictability of Q-values given minimal observations. We show that SMAC is insufficiently stochastic to require complex closed-loop policies.

5.1 SMAC Stochasticity

We use the observation that if the initial state and transition functions are deterministic, there always exists an optimal deterministic policy that conditions solely on the timestep information to investigate stochasticity in SMAC. In particular, we use a range of SMAC scenarios to compare

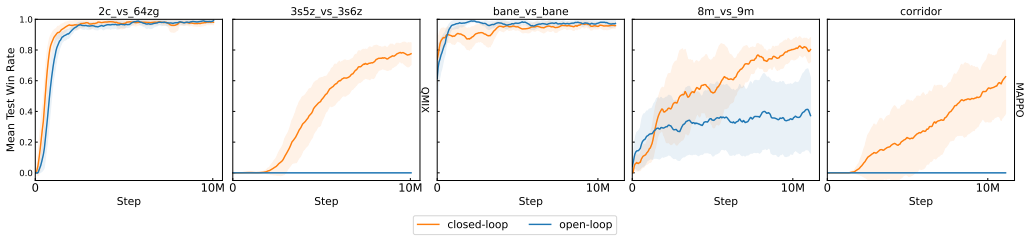


Figure 2: QMIX (left) and MAPPO (right) open-loop and closed-loop results.

policies conditioned only on the timestep (which we refer to as *open-loop*) to policies conditioned on the full observation history (i.e., all the information typically available to each agent in SMAC), which we refer to as *closed-loop*. The open-loop policies observe no ally or enemy information, and can only learn a distribution of actions at each timestep. If stochasticity in the underlying state is a serious challenge when training policies on SMAC scenarios, then policies without access to environment observations should fail. However, if the open-loop policies can succeed at the tasks, this demonstrates that SMAC is not representing the challenges of Dec-POMDPs well and so would benefit from added stochasticity.

We use MAPPO and QMIX as our base algorithms because of their widespread adoption and very strong performance and train open- and closed-loop versions of each. We train the *open-loop* policies on SMAC, but only allow the policies to observe the agent ID and timestep, whereas the closed-loop policies are given the usual SMAC observation as input with the timestep appended. The open-loop policies still have access to the central state as usual during training since this is only used in CTDE to aid in training and not during execution. The QMIX open-loop policy was trained only on the maps which the MAPPO open-loop policy was not able to completely solve. For QMIX, the hyperparameters used are the same as in [13] and for MAPPO the hyperparameters are listed in Appendix C.

Figure 2 shows the mean win rates and standard deviation of both policies on selected scenarios. The full results are available in Figure 8 in the appendix. Overall, the open-loop policies perform significantly worse than their closed-loop counterparts. However, this performance difference varies widely across scenarios and algorithms. For MAPPO, some scenarios, such as `bane_vs_bane`, `3s5z`, `1c3s5z` and `2s3z` achieve open-loop performance indistinguishable from that of closed-loop. In other scenarios, such as `8m_vs_9m` and `27m_vs_30m`, there are large performance differences, but the open-loop policy still achieves high win rates. There are also scenarios, such as `corridor` and `2c_vs_64zg`, where the MAPPO open-loop policy does not learn at all, whereas the closed-loop policies can learn successfully. QMIX also achieves good performance on `2c_vs_64zg`, `MMM2`, `27m_vs_30m` and `8m_vs_9m`, although for the latter maps the closed-loop policy strongly outperforms the open-loop QMIX. Altogether, there are only four SMAC maps where the open-loop approach cannot learn a good policy at all: `3s5z_vs_3s6z`, `corridor`, `6h_vs_8z` and `5m_vs_6m`. Overall, open-loop policies perform well on a range of scenarios in SMAC. The *easy* scenarios show no difference between the closed-loop and the open-loop policies, and some *hard* and *very hard* scenarios show either little difference or non-trivial win rates for the open-loop policy. These successes are striking given the restrictive nature of the open-loop policy. This suggests that stochasticity is not a significant challenge for a wide range of scenarios in SMAC.

These results highlight an important deficiency in SMAC. Stochasticity is either not evaluated or not part of the challenge for the vast majority of SMAC maps. Not testing stochasticity is a major flaw for a general MARL benchmark because without stochasticity the policy only needs to take optimal actions along a single trajectory. Additionally, without stochasticity, there can be no meaningful partial observability. Given widespread use of SMAC as a benchmark to evaluate algorithms for Dec-POMDPs, this suggests that a new benchmark is required to evaluate MARL algorithms.

5.2 SMAC Feature Inferrability & Relevance

In this section, we look at stochasticity from a different perspective. We mask (i.e. "zero-out") all features of the state and observations on which QMIX's joint Q-function conditions. If the Q-values can still be easily inferred via regression, then SMAC does not require learning complex closed-loop

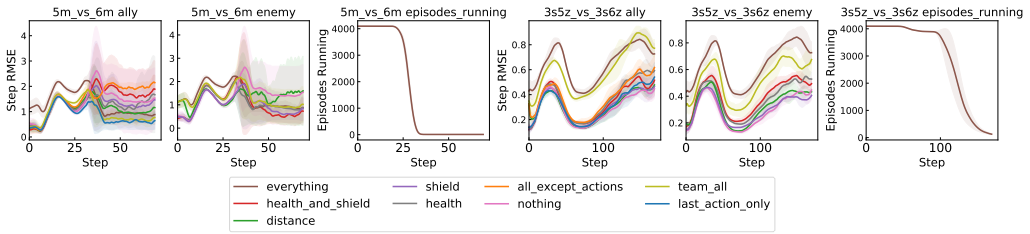


Figure 3: Comparison of the loss for different feature masks when regressing to a trained QMIX policy. The x-axis plots steps within an episode. 'Episodes Running' is the number of episodes that have not terminated by the given timestep. *everything* masks the entire observation and *nothing* masks no attributes. *team_all* masks all attributes of the team. *last_action_only* masks the last actions of all the allies in the state. The *all_except_actions* masks all the allied information *except* the last actions in the state. The mean is plotted for each mask and standard deviation across 3 seeds used as error bars. The low error rates for the everything mask imply that Q-values can be effectively inferred given only the timestep.

policies as Dec-POMDPs in general do, but is solvable by ignoring key aspects of StarCraft II gameplay. If the central Q-value could be inferred only from timestep information, this would also imply the task is not decentralised because all agents implicitly know the timestep and so could operate with the same central policy.

We also extend this experiment to masking subsets of the observation and state features. By only masking a subset of features, we measure the impact of different features on the learned policy. The more observation features the Q-function conditions on, the more confident we can be that SMAC requires understanding decentralised agent micromanagement rather than learning action sequences. We measure this conditioning by regressing to Q-values of a trained policy when masking different sets of features. The observation features hidden by each mask are in Table 1 in the appendix. Since we evaluate the regression performance of the joint Q-function, we always mask features in the state as well as the observation. More details of the regression procedure are in Appendix C.

Some interesting trends can be observed in Figure 3, which shows Q-value regression loss for each mask and scenario against steps in the RL episode. First, the root-mean-squared error of masking *everything* is low, reaching about 15% of the mean Q-value at peak, and 5-10% for most of the episode. Mean root-mean-squared error as a proportion of mean Q-value, as well as other metrics, is given in the appendix in Table 2. For all scenarios except `5m_vs_6m` this value is below 0.12. These are not much higher than the baseline *nothing* masks, suggesting the observation features do not really inform the Q-value. These results highlight the lack of stochasticity and meaningful partial observability in SMAC and the necessity of novel benchmarks to address these shortcomings.

6 SMACv2

As shown in the previous section, SMAC has some significant drawbacks. To address these, we propose three major changes: random team compositions, random start positions, and increasing diversity among unit types by using the true unit attack and sight ranges. These changes increase stochasticity and so address the deficiency discovered in the previous section.

Teams in SMACv2 are procedurally generated. Since StarCraft II does not allow units from different races to be on the same team, scenarios are divided depending on whether they feature Protoss, Terran or Zerg units. In scenarios with the same number of enemy and ally units, the unit types on each team are the same. Otherwise the teams are the same except for the extra enemy units, which are drawn identically to the other units. We use three unit types for each race, chosen to be the same unit types that were present in the original SMAC. We generate units in teams algorithmically, with each unit type having a fixed

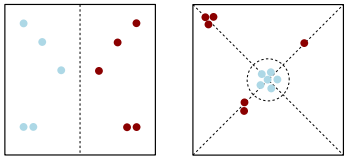


Figure 4: Examples of the two different types of start positions, *reflect* and *surround*. Allied units are shown in *blue* and enemy units in *dark red*.

probability. The probabilities of generating each unit type are given in Table 9 in the appendix. These probabilities are the same at test time and train time.

Random start positions in SMACv2 come in two different flavours. In *reflect* scenarios, the allied units are spawned uniformly at random on one side of the scenario. The enemy positions are the allied positions reflected in the vertical midpoint of the scenario. This is similar to how units spawned in the original SMAC benchmark but without clustering them together. In *surround* scenarios, allied units are spawned at the centre and surrounded by enemies stationed along the four diagonals. Figure 4 illustrates both flavours of start position.

The final change is to the sight range and attack range of the units. In the original SMAC, all unit types had a fixed sight and attack range, allowing short- and long-ranged units to choose to attack at the same distance. This did not affect the real attack range of units within the game – a unit ordered to attack an enemy while outside its striking range would approach the enemy before attacking. We change this to use the values from SC2, rather than the fixed values. However, we impose a minimum attack range of 2 because using the true attack ranges for melee units makes attacking too difficult.

Taken together, these changes mean that before an episode starts, agents no longer know what their unit type or initial position will be. Consequently, by diversifying the range of different scenarios the agents might encounter, they are required to understand the observation and state spaces more clearly, which should render learning a successful open-loop policy not possible. We also define a convenient interface for defining distributions over team unit types and start positions. By only implementing a small `Distribution` class one can easily change how these are generated.

Akin to the original version of SMAC [37], we split the scenarios into *symmetric* and *asymmetric* groups. In the symmetric scenarios, allies and enemies have the same number and type of units. In the asymmetric scenarios, the enemies have some extra units chosen from the same distribution as the allies. There are 5 unit, 10 unit and 20 unit symmetric scenarios, and 10 vs 11 and 20 vs 23 unit aymmetric scenarios for each of the three races.

6.1 Extended Partial Observability Challenge

We now introduce the Extended Partial Observability (EPO) challenge, where we make additional modifications to SMACv2 to ensure an extension of SMAC with meaningful partial observability. While SMAC is clearly partially observable in the sense that agents do not observe the global state, this partial observability is not particularly meaningful. SMAC certainly occludes information that would be relevant to each agent through the use of fixed sight ranges. However, the ranges of attack actions are always less than sight ranges and so ally teams can do without communication by moving within firing range and attacking any observed enemies. Finding enemies is not itself a significant challenge, as ally and enemy units spawn closely together. Therefore, much of the information that is relevant to the action selection of a given agent is within its observation.

To create a benchmark setting with meaningful partial observability, we introduce two further changes to SMACv2. First, enemy observations are stochastically masked for each agent. Once an enemy has been observed for the first time within an episode, the agent that was first to see it is guaranteed to observe it as normal. If two or more agents observe an enemy that had not yet been observed on the same timestep, there is a random tie break. As soon as this initial sighting is made, a random binary draw occurs for all other agents, which determines whether an agent will be able to observe that enemy for the remainder of the episode if it enters the agent's sight range. Agents for which the draw is unsuccessful have their observations masked such that they cannot observe any information about that enemy for the rest of the episode, even if it enters their sight range. The draw has a tunable probability p of success. We suggest $p = 0$ in the challenge. Figure 5 illustrates this setup.

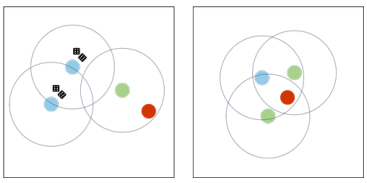


Figure 5: Example EPO enemy sighting. Allied units that do not observe the enemy are shown in *blue*, those that do are shown in *green* and the enemy unit in *dark red*. Initially, an ally spots an enemy. Later (right), when the enemy is within all allied sight ranges, only the first ally to observe the enemy and the ally for which the draw was successful can see it.

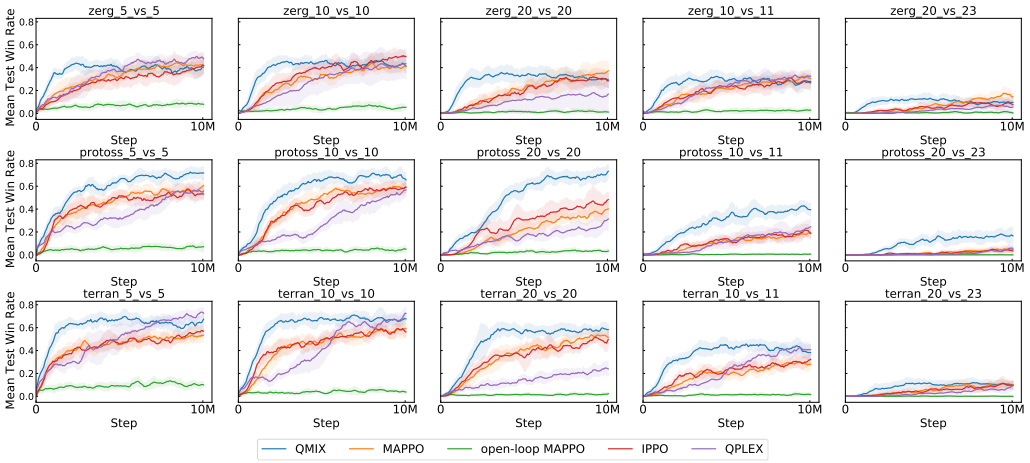


Figure 6: Comparison of the mean test win rate of QMIX, MAPPO, IPPO, QPLEX and an open-loop policy on SMACv2. Plots show the mean and standard deviation across 3 seeds.

In SMAC, the environment provides an available action mask that guarantees agents can only choose attack actions if the corresponding enemy is within shooting range. In this setting only, we remove this available actions mask. This makes deciding on a target more difficult because agents cannot rely on the available actions to infer which enemies they can attack. Hence they must either infer or communicate this information. If an invalid action is chosen, it becomes a no-op. We also found that without removing the available actions, there was little separation between the $p = 1$ (i.e. all allies can target all enemies) and $p = 0$ (i.e. only the first ally to see an enemy can target it) cases, as shown in Figure 13 in the appendix. We believe that this is because the available actions mask encourages heavy weight to be placed on attack actions, knowing that they will not be chosen unless available. In EPO, because of the increased difficulty, 6 allied agents battle against 5 enemy units. We found that this increased the gap between the $p = 1$ and $p = 0$ settings. We believe this is because in the more difficult `5_vs_5` setting, agents are too preoccupied with other more basic challenges for target prioritisation to be a large issue. Results for the `5_vs_5` setting are shown in Figure 14 in the appendix and for the `6_vs_5` setting in Figure 7. In EPO, there are 3 maps, one for each race, with 6 allies and 5 enemies per team. All other settings such as unit types and start positions are identical to SMACv2. We recommend a setting of $p = 0$.

7 SMACv2 Experiments

In this section, we present results from a number of experiments on SMACv2. We first train state-of-the-art SMAC baselines on SMACv2 to assess its difficulty. Currently, both value-based and policy optimisation approaches achieve strong performance on all SMAC scenarios [55, 13]. As baselines we use QMIX [32], a strong value-based method and MAPPO [55], a strong policy optimisation method. We also investigate the effect of the new SMACv2 environment features with ablations. These results are available in Figure 12 in the appendix.

7.1 SMACv2 Baseline Comparisons

First we compare the performance of MAPPO, QMIX, QPLEX[51] and IPPO[8] and an open-loop policy, based on MAPPO, on SMACv2. We run each algorithm for 10M training steps and 3 environment seeds. For MAPPO, we use the implementation from Sun et al. [43] and for QMIX the one from Hu et al. [13]. The results of the runs of the best hyperparameters are shown in Figure 6.

These results shown in Figure 6 reveal a few trends. First, QMIX generally performs better than MAPPO across most scenarios. QMIX strongly outperforms MAPPO in two Protoss scenarios (`protoss_20_vs_20` and `protoss_10_vs_11`), but otherwise final performance is similar, although QMIX is more sample efficient. However, QMIX is memory-intensive because of its large replay buffer and so requires more compute to run than MAPPO. Additionally, MAPPO appears to still be

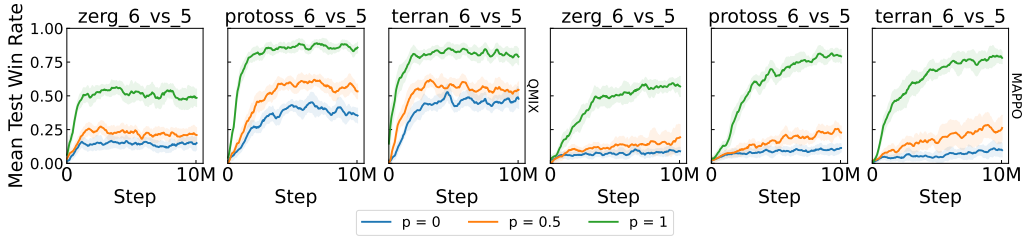


Figure 7: Comparison of mean test win rate when $p = 0, 0.5, 1$, with 6 agents against 5 enemy units, for QMIX (left) and MAPPO (right). Here, the available actions mask has been removed.

increasing its performance towards the end of training in several scenarios, such as the `10_vs_11` and `20_vs_20` maps. It is possible that MAPPO could attain better performance than QMIX if a larger compute budget were used. MAPPO and IPPO perform nearly identically across the maps, suggesting MAPPO gains little benefit from the centralised state. QPLEX performs worse than QMIX across a number of maps, particularly the `20_vs_20` maps.

Roughly even win rates are attained on the symmetric maps. By contrast, on the asymmetric maps, win rates are very low, particularly on the `20_vs_23` scenarios. Additionally, there is no real change in difficulty as the number of agents scales. However, the maps do not seem uniformly difficult, with all algorithms struggling with the Zerg scenarios more than other maps. Reassuringly the open-loop method cannot learn a good policy on any maps – even those where QMIX and MAPPO both achieve high win rates. This is strong evidence that SMAC's lack of stochasticity has been addressed.

7.2 EPO Baselines

We now compare QMIX and MAPPO on EPO. A comparison between the $p = 1$, $p = 0$ and $p = 0.5$ settings is shown in Figure 7. In this setting, $p = 0$ struggles to attain even modest performance, whereas $p = 1$ comes close to solving the task in a number of instances. This leaves a lot of room for the development of algorithms that learn to use implicit communication as a means of overcoming meaningfully restrictive partial observability and reduce the performance gap between the $p = 1$ and $p = 0$ settings. The performance in the $p = 0.5$ case is similar to the $p = 0$ case. This suggests that any communication protocol would have to be reasonably robust to have an impact on performance. This makes sense – an additional 50% chance to observe an enemy would still make it difficult to perform group target prioritisation. In general QMIX has a smaller gap between the $p = 0$ and $p = 1$ cases, suggesting it is perhaps better able to use the global state to resolve partial observability differences than MAPPO.

7.3 SMACv2 Feature Inferrability & Relevance

In this section we repeat the experiments from Section 5.2 on the 5 unit SMACv2 scenarios. We use three previously trained QMIX policies to generate 3 different datasets for training, and compute error bars in the regression across these three policies. The results are given in Table 3 and Figure 11 in the appendix.

When comparing results between SMAC and SMACv2, we focus on the difference between the *everything* (all features not visible) and *nothing* (no features hidden) masks. This is to account for some scenarios having much higher errors in the *nothing* mask than others. For example, `5m_vs_6m` has an error as a proportion of the mean Q-value in the *nothing* mask of 0.14, compared to 0.018 for `2c_vs_64zg`. The SMACv2 scenarios have higher errors than any of the SMAC scenarios when subtracting $\frac{\epsilon_{\text{rmse}}}{Q}$ for the *everything* and *nothing* masks. For example, this value is 0.12 for `5_gen_protoss`, which has the lowest value for any tested SMACv2 map. This is significantly higher than the 0.07 for `5m_vs_6m`, which has the highest value for any SMAC map. Tables 2 and 3 in the appendix list the full values for SMAC and SMACv2. These results suggest that SMACv2 is significantly more stochastic than SMAC. Second, examining stepwise results in Figure 11 in the appendix shows that SMACv2 maps have similar patterns of feature importance to SMAC maps. Enemy health is the most important individual feature and is important throughout an episode for all scenarios tested. Importantly, both the ally and enemy all masks reach significant thresholds of the

average Q-value, suggesting that it is important in SMACv2 for methods to attend to environment features.

8 Conclusion

Using a set of novel experimental evaluations, we showed that the popular SMAC benchmark for cooperative MARL suffers from a lack of stochasticity and meaningful partial observability. We then proposed SMACv2 and repeated these evaluations, demonstrating significant alleviation of this problem. We also demonstrated that baselines that achieve strong performance on SMAC struggle to achieve high win rates on the new scenarios in SMACv2. We performed ablations and discovered that the stochasticity of the unit types and random start positions combine to explain the tasks' difficulty. We also introduced and evaluated current methods on the extended partial observability (EPO) challenge and demonstrated that meaningful partial observability is a significant contributor to the challenge in this environment.

Gorsane et al. [11] found that it was most common for MARL papers accepted at top conferences to only evaluate on one benchmark, relying on the multiple scenarios within that benchmark to provide sufficient variety. However, in addition to the issues shown in that paper with cherry-picking of scenarios within a benchmark, this evaluation protocol is vulnerable to issues in environment dynamics, such as SMAC's lack of stochasticity, and community-wide overfitting. Given the difficulty in designing both benchmarks and sufficiently diverse scenarios within them, we recommend that evaluating on *multiple benchmarks* should become the norm where possible to help avoid issues.

We hope that SMACv2 can contribute to MARL research as a significantly challenging domain capturing practical challenges.

Acknowledgements

Benjamin Ellis and Jonathan Cook are supported by the EPSRC centre for Doctoral Training in Autonomous and Intelligent Machines and Systems EP/S024050/1. The experiments were made possible by a generous equipment grant from NVIDIA.

References

[1] Torchcraftai: A bot platform for machine learning research on starcraft: Brood war. `https://github.com/TorchCraft/TorchCraftAI`, 2018. GitHub repository.

[2] 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.

[3] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4): 819–840, 2002.

[4] 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. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf`.

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[6] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[7] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056, 2020.

[8] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[9] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alch'e Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/07a9d3fed4c5ea6b17e80258dee231fa-Paper.pdf`.

[10] 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/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf`.

[11] 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.

[12] Tarun Gupta, Anuj Mahajan, Bei Peng, Wendelin Böhmer, and Shimon Whiteson. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 3930–3941. PMLR, 2021.

[13] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and SW Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021.

[14] Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. Updet: Universal multi-agent rl via policy decoupling with transformers. In *International Conference on Learning Representations*, 2020.

[15] Shariq Iqbal, Christian A Schroeder De Witt, Bei Peng, Wendelin Boehmer, Shimon Whiteson, and Fei Sha. Randomized entity-wise factorization for multi-agent reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4596–4606. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/iqbal21a.html`.

[16] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[17] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning. In *IJCAI*, 2019.

[18] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729, 2018.

[19] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv e-prints*, pages arXiv–2111, 2021.

[20] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4501–4510, 2020.

[21] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[22] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019. URL `http://arxiv.org/abs/1908.09453`.

[23] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas, 2017. URL `https://arxiv.org/abs/1702.03037`.

[24] 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.

[25] Ryan Lowe, 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.

[26] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 32, 2019.

[27] Anuj Mahajan, Mikayel Samvelyan, Lei Mao, Viktor Makoviychuk, Animesh Garg, Jean Kossaifi, Shimon Whiteson, Yuke Zhu, and Animashree Anandkumar. Tesseract: Tensorised actors for multi-agent reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7301–7312. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/mahajan21a.html`.

[28] Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022.

[29] 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*, 2013.

[30] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276.

[31] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. FACMAC: Factored multi-agent centralised policy gradients. 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=WxH774NOmEu`.

[32] 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.

[33] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10199–10210. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf`.

[34] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[35] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

[36] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2, 08 2020. doi: 10.1038/s42256-020-0208-z.

[37] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

[38] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[39] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[40] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896. PMLR, 2019.

[41] Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

[42] Mingfei Sun, Vitaly Kurin, Guoqing Liu, Sam Devlin, Tao Qin, Katja Hofmann, and Shimon Whiteson. You may not need ratio clipping in ppo. *arXiv preprint arXiv:2202.00079*, 2022.

[43] 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*, AAMAS '23, page 5–13, Richland, SC, 2023. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321.

[44] 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*, pages 2085–2087, 2018.

[45] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games, 2016. URL `https://arxiv.org/abs/1611.00625`.

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

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[48] Eugene Vinitsky, Natasha Jaques, Joel Leibo, Antonio Castenada, and Edward Hughes. An open source implementation of sequential social dilemma games. `https://github.com/eugenevinitsky/sequential_social_dilemma_games/issues/182`, 2019. GitHub repository.

[49] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017. URL `https://arxiv.org/abs/1708.04782`.

[50] 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 L. 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, 2019. doi: 10.1038/s41586-019-1724-z.

[51] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2020.

[52] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020.

[53] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2020.

[54] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5571–5580. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/yang18d.html`.

[55] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

3.1 Limitations and Future Work

One of the main limitations of SMACv2 is its restriction to a fixed distribution of unit types and start positions. A more generalisable policy could perhaps be learned by instead using one of the many popular curriculum learning approaches (Rutherford et al., 2024; Dennis et al., 2020; Jiang et al., 2021). Secondly, the heuristic enemy policy in SMACv2 could be improved. Currently, it relies on knowing the positions of all the allies. This is so that enemies always attack the allies no matter where they are on the map. However, this relies on global information that would not be available in allied observations. This information asymmetry means that there is no guarantee that a 50% win rate is achievable and hence that the scenarios are soluble. Another, related issue is that we do not provide evidence in the paper that the communication gap presented by EPO is actually *bridgeable*. In other words we do not demonstrate a strategy to show how with $p = 0$, it is possible to recover some of the performance from when $p = 1$. Further work to investigate this property and show better performance via a simple communication mechanism would be useful in this regard.

Although this work finds an important issue in a popular benchmark, it does not address the widespread evaluation issues in MARL (Gorsane et al., 2022). Combining this work with a library aimed at standardising evaluation, similar to rliable in single-agent RL (Agarwal et al., 2021), would help to improve this work's impact on evaluation. Another limitation is that SMACv2 does not support self-play. Self-play has led to superhuman policies in games such as Go and StarCraft II (Silver et al., 2016; Vinyals et al., 2019), and therefore supporting it in SMACv2 would likely improve the quality of policies obtained.

SMACv2 also is only one benchmark. To address evaluation issues in MARL, and RL more generally, there needs to be a wide range of difficult tasks to test the generalisation of algorithms. SMACv2 therefore represents only one brick in that broader wall. Some of these issues, such as the lack of support for self-play and the issues with the heuristic policy are addressed by the introduction of SMAX in JaxMARL in the next chapter.

Additionally, although we demonstrate that the open-loop policies that we train cannot achieve good performance on SMACv2, this does not rule out that such a successful sequence of actions exists. It is highly likely that the stochasticity in the environment is enough to prevent such a strategy from succeeding, but using another method, such as evolutionary methods, would help to more conclusively rule this out.

4

JaxMARL: Multi-Agent RL Environments and Algorithms in JAX

One significant issue with evaluation in RL, and specifically in MARL, is the lack of usage of a wide range of benchmarks (Gorsane et al., 2022). Another is that long run times and poor computational efficiency make it difficult to run enough seeds to produce statistically meaningful results. A third is that many frameworks feature complicated and unnecessary abstractions that can make understanding algorithmic details difficult. The next chapter of this thesis aims to address these concerns. We introduce JaxMARL, a large collection of MARL environment and algorithm implementations across mixed, cooperative and competitive settings in JAX.

Recent work in RL has significantly improved computational efficiency by running environments on the GPU (Lu et al., 2022), allowing end-to-end JITting, as well as easy parallelisation using methods such as `vmap`. In this work, we bring these innovations to MARL, where they have not been so widely adopted, and as a result achieve significantly better computational efficiency and speed up training by around 31x when training a single agent. Additional speedups can be achieved by vectorising the training of many agents in parallel. This leads to speedups over training agents sequentially by a factor of 12500x. The speedups achieved by using JAX also depend significantly on the environment in question. For example, the

MPE environments run in pure NumPy, and so are already quite fast and simple. However, for more complex environments, such as SMAC, the speedups can be much larger since we do not need to run the StarCraft II game engine.

In addition to this improved efficiency, JaxMARL also has an impressive range of algorithms and benchmarks. It includes implementations for MAPPO, IPPO, QMIX, VDN and IQL across eight environments. This significantly improves the ease of evaluation for MARL researchers, allowing easy benchmarking on a range of domains. Improving the ease of evaluation has the potential to alleviate overfitting of evaluation and algorithms to certain benchmarks. This is particularly relevant in MARL given the findings of Gorsane et al. (2022), which shows that 30% of MARL papers accepted at top conferences only use one benchmark for evaluation. This is largely because of the disparate codebases for different environments, making integrating algorithmic changes into a separate environment challenging. Some codebases themselves are complex too, with hanabi for example requiring parallelisation in C++ across a wide range of CPU cores. Hanabi in JaxMARL runs more quickly and is written entirely in Python and therefore much simpler to understand.

JaxMARL also introduces two new benchmarks: STORM, a mixed benchmark for spatially and temporally extending matrix games, and SMAX, a version of SMAC written in JAX. SMAX addresses several of the issues with SMAC and SMACv2 highlighted in the previous chapter. It allows self-play and improves upon the heuristic AI with a hand-coded bot which relies solely on observation data. This means that a win rate of less than 50% in SMAX represents a concrete failure to learn because the learned algorithm could improve by copying the enemy's strategy. SMAX also does not rely on the StarCraft II game engine to function. This allows SMAX to run on the GPU, and hence enormously improve computational efficiency.

JaxMARL first provides recommendations for how to evaluate algorithms of certain types. The paper then shows aggregated results across a wide range of the environments, demonstrating that PPO performs better than the Q-learning

algorithms in general. This is likely because of the computational inefficiency of Q-learning algorithms and their large replay buffers when run end-to-end on the GPU. PQN (Gallici et al., 2024) for example performs on par with or better than PPO. We then present benchmarking results across the environments for different numbers of parallel environments and show that JaxMARL is much faster than CPU-based environments. We then present evidence that the environment implementations are correct, by, for example, comparing the observations and rewards for MPE from JaxMARL with the reference implementation.

arXiv:2311.10090v5 [cs.LG] 2 Nov 2024

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.

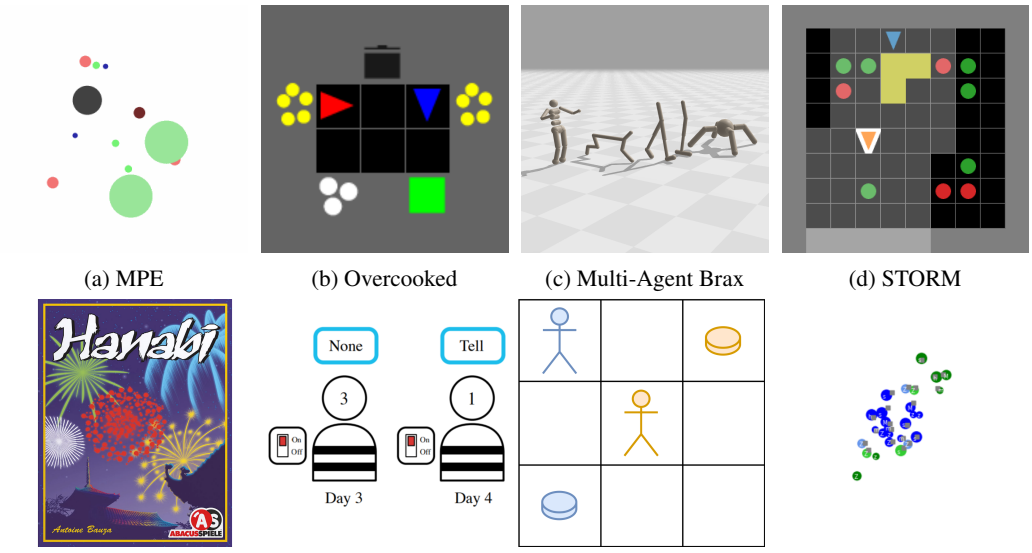
1 Introduction

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.

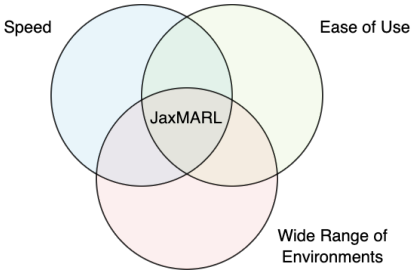


Figure 2: Our philosophy. JaxMARL combines a wide range of environments with ease of use and evaluation speed.

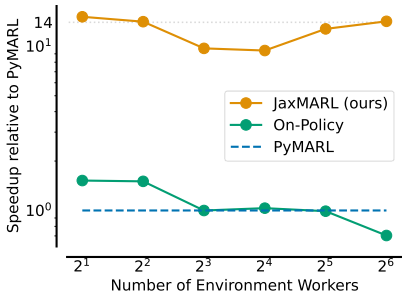


Figure 3: Speed of training an RNN agent using IPPO on a multi-particle environment in JaxMARL compared to two popular MARL libraries, see the Appendix for details.

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]

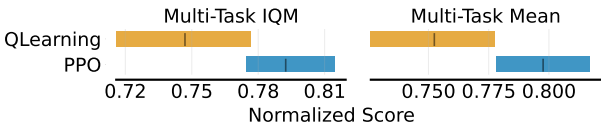


Figure 4: Normalised scores aggregated over SMAX, MPE and Overcooked. PPO shows a clear advantage.

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

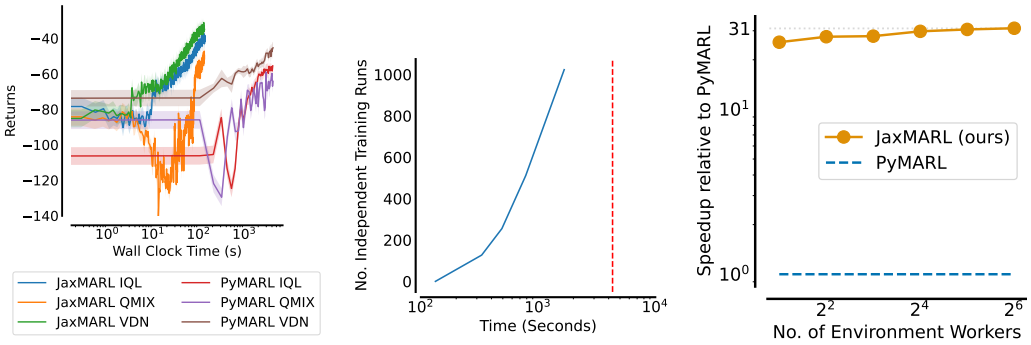
We compare the performance of our environments in steps per second when using random actions to the original environments in Table 3, with details of this test provided in the Appendix.

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.

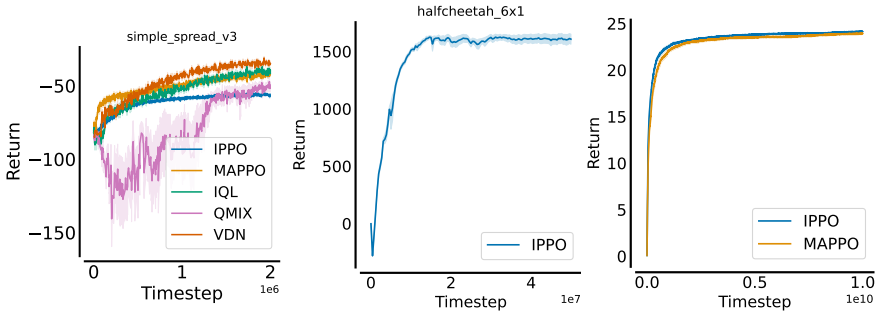
training process within a single hardware accelerator. For QMIX on MPE, this allows us to complete 1024 individual training runs in 198.4 seconds, compared to 1 hour and 10 minutes for a single training run with PyMARL, a speed up of 21,500x per agent. This analysis is repeated for IPPO in the Appendix 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.

4.1 Limitations and Future Work

While JaxMARL represents a significant step forward in the way that multi-agent RL algorithms are evaluated, it has a number of limitations and areas where this research could be further pursued. One important limitation is that a number of the environments in the benchmark, for various reasons, are not equivalent to the original environments they are based on. This makes comparing results difficult. Truly faithful versions of these environments where possible would help to compare results more effectively across papers.

Secondly, JaxMARL presents versions of common MARL benchmarks that run much faster. However, this computational efficiency represents an opportunity to design coordination environments that are much richer and more complicated. This has already happened in single-agent RL with the introduction of benchmarks such as Craftax (Matthews et al., 2024a) and Kinetix (Matthews et al., 2024b). These benchmarks use the drastically better sample efficiency of JAX to present challenges that would have previously been completely insurmountable. Similar should hold for MARL. JAX presents an opportunity to produce drastically more complicated MARL environments to inspire new algorithms.

Another opportunity presented by JAX, and not explored in this work, is to handle very large numbers of agents. The improved simulation speed of JAX should allow the creation of environments with vastly more agents than used previously. For example, the simulation of complex multi-agent environments, such as a shopping centre, is likely now on the verge of possible using very small 1B language models such as RWKV (Peng et al., 2023) and today's powerful GPUs. These larger numbers of agents would present an opportunity to study credit assignment. Independent learning is very successful in MARL, but should break down due to non-stationarity and difficulty with credit assignment as the number of agents becomes very large. However, testing this until now has been computationally prohibitive. JAX presents an opportunity to probe those limits.

JaxMARL also introduces self-play to SMAX. Exploiting the natural curriculum of self-play could produce agents significantly more capable than those that could

be trained using SMAC or SMACv2. These could then be evaluated in the regular SMAX scenarios to test their generalisation ability. Using such a setup would provide the opportunity to go beyond the traditional SMAC evaluation setup and train more capable agents.

Another prominent use of games recently has been as an out-of-distribution evaluation for frontier models. JaxMARL as a collection of multi-agent benchmarks presents an ideal starting point for this. Evaluating the ability of frontier models to cooperate will be increasingly important as their capabilities improve and they interact with humans and other unseen partners more often. Previous work has shown that LLMs struggle to do even relatively basic social reasoning (Alberts et al., 2024), and so complex coordination games are likely to remain challenging for VLMs for at least the near future. Hanabi, as a game that can easily be made text-based and relies on a number of complex communication conventions, is particularly ideal for this. A similar project in single-agent RL is BALROG (Paglieri et al., 2024), which provides evaluations for LLMs and VLMs on a range of games including NetHack.

Part II

Usage of Deep Learning Tools in Reinforcement Learning

5

Adam on Local Time: Addressing Nonstationarity in RL with Relative Adam Timesteps

Reinforcement learning has made significant progress since the advent of deep learning by adopting tools originally developed for supervised learning. For example, CNNs and the Adam optimiser were instrumental to impressive results on Atari (Schwarzer et al., 2023; Espeholt et al., 2018). However, RL has some significant differences to supervised learning. Most notably, because changing the policy implies a change in the return distribution, RL optimisation is *nonstationary*. This optimisation property requires that learning be stabilised by techniques such as target networks or clipping policy updates. Additionally, differences between supervised learning and reinforcement learning have led to the adoption of a number of important, but ill-documented and often opaque design choices in RL (Engstrom et al., 2020; Andrychowicz et al., 2020).

In this part of the thesis, we argue that this should be revisited. Instead of blindly building on top of deep learning tooling, we should instead understand how it can best be *adapted* for use in a nonstationary RL setting. In the first content chapter of this part, we examine the Adam optimizer.

We consider the Adam optimizer's timestep. This is used to ensure that Adam update sizes stay at around 1, despite the momentum coefficients β_1 and β_2 having different sizes. This works well, but was designed for stationary optimisation. However, RL is nonstationary. RL is also most often nonstationary in a particular way: it progresses as a sequence of stationary optimisation problems with nonstationarity between them. Viewing RL optimisation this way gives rise to the idea of using the Adam timestep within each stationary optimisation (the *local timestep*) rather than a single *global* timestep. We call this approach Adam-Rel.

By considering a simple theoretical model, we show that if the scale of gradient changes, as it might with a change in objective, Adam can cause large persistent updates and that Adam-Rel fixes this while reducing to learning rate annealing if update sizes stay consistent. We then examine the performance of Adam-Rel when applied to PPO and DQN. We show that for PPO, Adam-Rel outperforms Adam in Atari and Craftax. Because Adam-Rel changes the Adam update, we re-tuned some hyperparameters, in particular using a different value for GAE λ and learning rate. As part of this thesis, we ran an additional baseline that featured PPO with Adam using these changed hyperparameters. We found that this baseline performed poorly, achieving an IQM score of 0.35 (with a 95% confidence interval of $[0.33, 0.38]$), well below Adam-Rel and the baseline we previously used, which achieved IQM scores of close to 1. We can therefore conclude that these changed hyperparameters were not a key factor in the performance of Adam-Rel. We also evaluate DQN in Atari and show that Adam-Rel performs better whether used with Polyak averaging or not.

We then examine the real update sizes collected from training runs of PPO on Craftax and show that increases in gradient magnitude *do* occur, although of a lower magnitude than might be expected. We then discuss the differences between our theoretical model and the observed data.

Adam on Local Time: Addressing Nonstationarity in RL with Relative Adam Timesteps

Benjamin Ellis* University of Oxford
Matthew T. Jackson* University of Oxford
Andrei Lupu University of Oxford
Alexander D. Goldie University of Oxford
Mattie Fellows University of Oxford
Shimon Whiteson University of Oxford
Jakob N. Foerster University of Oxford

Abstract

In reinforcement learning (RL), it is common to apply techniques used broadly in machine learning such as neural network function approximators and momentum-based optimizers [1, 2]. However, such tools were largely developed for supervised learning rather than nonstationary RL, leading practitioners to adopt target networks [3], clipped policy updates [4], and other RL-specific implementation tricks [5, 6] to combat this mismatch, rather than directly adapting this toolchain for use in RL. In this paper, we take a different approach and instead address the effect of nonstationarity by adapting the widely used Adam optimiser [7]. We first analyse the impact of nonstationary gradient magnitude—such as that caused by a change in target network—on Adam's update size, demonstrating that such a change can lead to large updates and hence sub-optimal performance. To address this, we introduce *Adam with Relative Timesteps*, or Adam-Rel. Rather than using the global timestep in the Adam update, Adam-Rel uses the local timestep within an epoch, essentially resetting Adam's timestep to 0 after target changes. We demonstrate that this avoids large updates and reduces to learning rate annealing in the absence of such increases in gradient magnitude. Evaluating Adam-Rel in both on-policy and off-policy RL, we demonstrate improved performance in both Atari and Craftax. We then show that increases in gradient norm occur in RL in practice, and examine the differences between our theoretical model and the observed data.

1 Introduction

Reinforcement Learning (RL) aims to learn robust policies from an agent's experience. This has the potential for large scale real-world impact in areas such as autonomous driving or improving logistic chains. Over the last decade, a number of breakthroughs in supervised learning—such as convolutional neural networks and the Adam optimizer—have expanded the deep learning toolchain and been transferred to RL, enabling it to begin fulfilling this potential.

However, since RL agents are continuously learning from new data they collect under their changing policy, the optimisation objective is fundamentally *nonstationary*. Furthermore, temporal difference (TD) approaches bootstrap the agent's update from its own value predictions, exacerbating the nonstationarity in the objective function. This is in stark contrast to the *stationary* supervised learning setting for which the deep learning toolchain was originally developed. Therefore, to apply these tools successfully, researchers have developed a variety of implementation tricks *on top of* this base to stabilise training [8, 6, 5]. This has resulted in a proliferation of little-documented design choices that are vital for performance, contributing to the reproducibility crisis in RL [9].

*Equal Contribution

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

We believe that in the long term, a more robust approach is to *augment* this toolchain for RL, rather than building on top of it. To this end, in this paper we examine the interaction between nonstationarity and the Adam optimizer [7]. Adam's update rule, where equations are applied element-wise (i.e. per parameter), is defined by

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad \hat{m}_t = \frac{m_t}{(1-{\beta_1}^t)},$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2, \qquad \hat{v}_t = \frac{v_t}{(1-{\beta_2}^t)},$$
$$u_t = \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \qquad \theta_t = \theta_{t-1} - \alpha u_t.$$

Here, g_t is the gradient, θ_t a parameter to be optimized, and α the learning rate. The resulting update is the ratio of two different momentum terms: one for the first moment, m_t, and one for second moment, v_t, of the gradient. These terms use different exponential decay coefficients, β_1 and β_2. Under stationary gradients, the $(1-\beta_i)$ weighting ensures that, in the limit, the overall magnitude of the two momenta is independent of the value chosen for each of the coefficients. However, since both momentum estimates are initialised to 0, they must be renormalised for a given (finite) timestep t, to account for the "missing parts" of the geometric series [7], resulting in $\hat{v}_t$ and $\hat{m}_t$.

Crucially, t counts the update steps since the *beginning of training* and thus bakes in the assumption of stationarity that is common in supervised learning. In particular, this renormalisation breaks down if the loss is nonstationary. Consider a task change late in training, which results in gradients orders of magnitudes higher than those of the prior (near convergence) task. Clearly, this is analogous to the situation at the *beginning of training* where all momentum estimates are 0. However, the t parameter, and therefore the renormalisation, does not account for this.

In this paper, we demonstrate that changes in the gradient scale can lead to large updates that persist over a long horizon. Previous work [10, 11] has suggested that old momentum estimates can *contaminate* an agent's update and propose resetting the entire optimizer state when the target changes as a solution. However, by discarding previous momentum estimates, we hypothesise that this approach needlessly sacrifices valuable information for optimization. Instead, we propose retaining momentum estimates and only resetting t, which we refer to as **Adam-Rel**. In the limit of gradient sparseness, we show that the Adam-Rel update size remains bounded, converging to 1 in the limit of a large gradient, unlike Adam. Furthermore, if such gradient magnitude increases do not occur, Adam-Rel reduces to learning rate annealing, a common method for stabilising optimization.

When evaluated against the original Adam and Adam with total resets, we demonstrate that our method improves PPO's performance in Craftax-Classic [12] and the Atari-57 challenge from the Arcade Learning Environment [13]. Additionally, we demonstrate improved performance in the off-policy setting by evaluating DQN on the Atari-10 suite of tasks [14]. We then examine the gradients in practice and show that there are significant increases in gradient magnitude following changes in the objective. Finally, we examine the discrepancies between our theoretical model and observed gradients to better understand the effectiveness of Adam-Rel.

2 Background

2.1 Reinforcement Learning

Definition Reinforcement learning agents learn a policy π in a Markov Decision Process [15, MDP], a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and γ is the discount factor. At each timestep t, the agent observes a state $s_t \in \mathcal{S}$ and takes an action a_t drawn from $\pi(\cdot|s_t)$ before transitioning to a new state $s_{t+1} \in \mathcal{S}$ and receiving reward r_t drawn from $\mathcal{R}(s_t, a_t)$. The goal of the agent is to maximise the expected discounted return $\mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

Nonstationarity in RL In contrast with supervised learning, where a single stationary objective is typically optimised, reinforcement learning is inherently nonstationary. Updates to the policy induce changes not only in the distribution of observations seen at a given timestep, but also the return distribution, and hence value function being optimised. This arises regardless of how these updates

are performed. However, one particular reason for nonstationarity in RL is the use of bootstrapped value estimates via TD learning [15], which optimises the below objective

$$\mathcal{L}(\theta) = \left[\text{sg}\left\{r_t + \gamma V_\theta^\pi\left(s_{t+1}\right)\right\} - V_\theta^\pi\left(s_t\right)\right]^2,$$

where sg is the stop-gradient operator. In this update, the target $r_t + \gamma V_\theta^\pi(s_{t+1})$ depends on the parameters θ and therefore changes as these are updated.

These target changes can either be more gradual, as in the case of continuous updates to the value function in TD learning, or more abrupt, as in the case of the use of target networks in DQN.

Sequentially Optimized Stationary Objectives In this work, we focus on abrupt objective changes; changes of objectives that do not involve a smoothing method such as Polyak averaging [1], and the resulting sudden change of supervised learning problem. More explicitly, we consider optimising a stationary loss function $L(\theta, \phi)$, where θ are the parameters to be optimised and ϕ is the other parameters of the loss function (such as the parameters of a value network), which are not updated throughout optimisation, but does not include the training data.

We consider a setting where at a certain timestep t in our training, we transition from optimising $L(\theta_t, \phi_1)$ to optimising $L(\theta_{t+1}, \phi_2)$ for some ϕ_1, ϕ_2. Such individual objectives are still non-stationary. For example, significant changes in the policy would induce changes in the data distribution, which would then affect the underlying loss landscape, but we do not consider such non-stationarity in this work.

This setting is very common throughout RL. Bootstrapped value estimates are the most common cause of this, but it also occurs in PPO's actor update, where each new rollout induces a different supervised learning problem due to the actor and critic updates. This is optimised for a fixed number of updates before collecting new data.

We refer to these sequences of supervised learning problems as sequentially-optimised stationary objectives. In this work, we use this framing to propose an approach that is consistent throughout each stationary period of optimization and applies corrections to make optimization techniques valid when nonstationarity is introduced via objective changes. Bengio et al. [11] propose the gradient contamination hypothesis, which states that current optimizer momentum estimates can point in the opposite direction to the gradient following a change in objective, thereby hindering optimization. A previous approach to this problem is that of Asadi et al. [10], where they propose resetting Adam's momentum estimates and timestep to 0 throughout training. We refer to this method as **Adam-MR**. Finally, Dohare et al. [16] propose setting the Adam hyperparameters to equal values, such that $\beta_1 = \beta_2$, suggesting that this can help avoid performance collapse.

Proximal Policy Optimization Proximal Policy Optimization [4, PPO] is a policy optimisation based RL method. It uses a learned critic V_ϕ^π trained by a TD loss to estimate the value function, and a clipped actor update of the form

$$\min\left[\text{clip}\left(r_{(\theta,t)}, 1 \pm \epsilon\right) A^\pi(s_t, a_t), r_{(\theta,t)} A^\pi(s_t, a_t)\right], \tag{1}$$

where the policy ratio $r_{(\theta,t)} = \frac{\tilde{\pi}_\theta(a_t|s_t)}{\pi(a_t|s_t)}$ is the ratio of the stochastic policy to optimise $\tilde{\pi}_\theta$ and π, the previous policy. A^π is the advantage, which is typically estimated using generalised advantage estimation [17]. Clipping the policy ratio aims to avoid performance collapse by preventing policy updates larger than ϵ.

Optimisation of the PPO objective proceeds by first rolling out the policy to collect data, and then iterating over this data in a sequence of *epochs*. Each of these epochs splits the collected data into a sequence of *mini-batches*, over which the above update is calculated.

2.2 Momentum-Based Optimization

Momentum [1, 2] is a method for enhancing stochastic gradient descent by accumulating gradients in the direction of repeated improvement. The typical formulation of momentum for each element i is

$$m_t^i = \beta m_{t-1}^i + g_t^i,$$
$$\theta_t^i = \theta_{t-1}^i - \alpha m_t^i,$$

where β is the momentum coefficient, $g_t \in \mathbb{R}^n$ is the gradient at the current step, $m_t \in \mathbb{R}^n$ is the gradient incorporating momentum, α is the scalar learning rate and $\theta \in \mathbb{R}^n$ are the parameters to be optimised. With momentum, update directions with low curvature have their contribution to the gradient amplified, considerably reducing the number of steps required for convergence.

In the introduction, we described the update equations for Adam [7], the most popular optimizer that uses momentum. Adam's update is designed to keep its updates within a trust region, which depends on a learning rate α.

3 Nonstationary Optimization with Adam

We now investigate the effect of nonstationarity on Adam by analysing its update rule after a sudden change in gradient. As a simplified model of gradient instability, we assume optimization with Adam starts at timestep $t = -t'$ with a constant gradient $g^i_{-t'} = g$, $0 < g < \infty$ until timestep 0. Following $t = 0$, we model instability by increasing the gradient by a factor of k, as might occur in a nonstationary optimization setting. This gives

$$g^j_t = \begin{cases} g, & -t' \leq t < 0, \\ kg, & t \geq 0. \end{cases} \tag{2}$$

For larger values of t', the short term effects of Adam's initialisation on the momentum terms dissipate and $\hat{m}_t$ and $\hat{v}_t$ converge to stable values. By taking the limit of $t' \to \infty$, we investigate the effect of a sudden change in gradient g^i_t on the update size u^i_t after a long period of training. This allows for any effects from the initialisation of momentum terms $\hat{m}_{-t',t}$ and $\hat{v}_{-t',t}$ to dissipate:

Theorem 3.1. *Assume that $\epsilon = 0$. Let g^i_t be defined as in Equation (2) and $\hat{m}^i_{-t',t}$ and $\hat{v}^i_{-t',t}$ be the momentum terms at timestep t given Adam starts at timestep $-t'$. It follows that:*

$$\lim_{t' \to \infty} u^i_t = \lim_{t' \to \infty} \frac{\hat{m}^i_{-t',t}}{\sqrt{\hat{v}^i_{-t',t}}} = \frac{{\beta_1}^{t+1} + k(1 - {\beta_1}^{t+1})}{\sqrt{{\beta_2}^{t+1} + k^2(1 - {\beta_2}^{t+1})}}. \tag{3}$$

Proof. See Appendix A. □

For large k, Theorem 3.1 proves that the element-wise momentum term after the change in gradient at $t = 0$ is approximately $\frac{1-\beta_1}{\sqrt{1-\beta_2}}$. For the most commonly used values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, this is $\sqrt{10}$, which is much larger than the intended unit update which Adam is designed to maintain. The top plot in Figure 1, which shows the Adam update size against t for different values of k, demonstrates that the update peaks significantly higher than the desired 1 before slowly converging back to 1.

4 Adam with Relative Timesteps

To fix the problems analysed in the previous section, we introduce Adam-Rel. At the start of each new supervised learning problem, Adam-Rel resets Adam's t parameter to 0, rather than incrementing it from its previous value. This one-line change is illustrated for PPO in Algorithm 1.

At the start of training, both momentum terms in Adam are 0. Therefore, at the first timestep, when the first gradient is encountered, the magnitude of the gradient is infinite relative to the current momentum estimate. As explained in Section 3, this induces a large update. However, dividing the momentum estimates by $(1 - {\beta_1}^t)$ and $(1 - {\beta_2}^t)$ fixes this issue by correcting for this sparsity. Therefore, by resetting t to 0, Adam handles changes in gradient magnitude resulting from the change of supervised learning problem.

If we examine the same update as in the previous section adjusted by Adam-Rel, assuming that we reset Adam's t just before the gradient scales to kg, we find it comes to

$$\lim_{t' \to \infty} \frac{\hat{m}^i_{-t',t}}{\sqrt{\hat{v}^i_{-t',t}}} = \frac{\sqrt{1 - {\beta_2}^{t+1}}}{1 - {\beta_1}^{t+1}} \frac{{\beta_1}^{t+1} + k(1 - {\beta_1}^{t+1})}{\sqrt{{\beta_2}^{t+1} + k^2(1 - {\beta_2}^{t+1})}}. \tag{4}$$

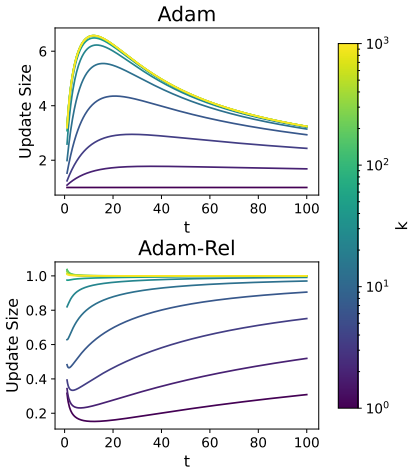


Figure 1: Update size of Adam and Adam-Rel versus k when considering nonstationary gradients. Assumes that optimization starts at time $-t'$, which is large, and that the gradients up until time 0 are g and then there is an increase in the gradient to kg.

Algorithm 1 Pseudocode for PPO with Adam, Adam-Rel, and Adam-MR.

$m_0 = 0$, $v_0 = 0$, $t = 0$ ▷ Initialise Adam state
$j = 0$ ▷ Initialise number of updates to 0
for $k = 1$ **to** K **do**
  Rollout policy π_{θ_k} to collect batch D
  $t = 0$
  $t = 0, m_j = 0, v_j = 0$
  for epoch $= 1$ **to** E **do**
    for mini-batch B in D **do**
      $g_j = \nabla_{\theta_{j-1}} \left[L^{\text{PPO}}(\theta_{j-1}) + L^{\text{TD}}(\theta_{j-1}) \right]$
      $j = j + 1$
      $t = t + 1$
      $m_j = \beta_1 m_{j-1} + (1 - \beta_1) g_j$
      $v_j = \beta_2 v_{j-1} + (1 - \beta_2) {g_j}^2$
      $\hat{m}_j = \frac{m_j}{(1-\beta_1{}^t)}$
      $\hat{v}_j = \frac{v_j}{(1-\beta_2{}^t)}$
      $\theta_j = \theta_{j-1} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j + \epsilon}}$
    end for
  end for
end for

As $k \to \infty$, this tends to 1. This means that Adam-Rel ensures approximately unit update size in the case of a large increase in magnitude in the gradient, at the expense of a potentially smaller update at the point t is reset. Figure 1 shows the update size of Adam-Rel as $t - t'$ increases. The update size is smaller at the start, but never reaches significantly above 1.

However, the above analysis does not show how Adam and Adam-Rel differ in practice, where large changes in gradient magnitude may not occur. Examining the bottom of Figure 1, we can see that for lower values of k, Adam-Rel rapidly decays the update size before increasing it. Functionally, this behaves like a learning rate schedule. Over a short horizon (e.g., 16 steps is common in PPO), this effect is similar to learning rate annealing, whilst over a longer horizon (e.g., approximately 1000 steps in DQN) it is akin to learning rate warmup, both of which are popular techniques in optimising stationary objectives. Therefore, the benefits of Adam-Rel are twofold: first, it guards against large increases in gradient magnitude by capping the size of potential updates, and secondly, if such large gradient increases do not occur, it reduces to a form of learning rate annealing, which is commonly employed in optimising stationary objectives.

5 Experiments

5.1 Experimental setup

To evaluate Adam-Rel, we explore its impact on DQN and PPO, two of the most popular algorithms in off-policy and on-policy RL respectively.

To do so, we first train DQN [18, 19] agents with Adam-Rel on the Atari-10 benchmark for 40M frames, evaluating performance against agents trained with Adam and Adam-MR. We then extensively evaluate our method's impact on PPO [4, 19, 20], training agents on Craftax-Classic-1B [12]—a JAX-based reimplementation of Crafter [21] where the agent is allocated 1 billion environment interactions—and the Atari-57[2] suite [13] for 40 million frames. In doing so, our benchmarks respectively evaluate the performance of Adam-Rel on exceedingly long training horizons and its

[2]We exclude 2 out of the 57 games, Montezuma's Revenge and Venture, after observing that all algorithms achieve a human-normalized score of 0.

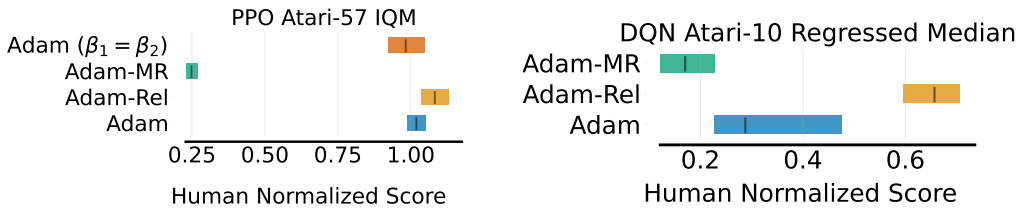


Figure 2: Performance of Adam-Rel, Adam, Adam-MR, and Adam ($\beta_1 = \beta_2$) for PPO and Adam, Adam-MR and Adam-Rel for DQN on Atari-57 and Atari-10 respectively. Atari-10 uses a subset of Atari tasks to estimate median performance across the whole suite. Details can be found in [14]. Error bars are 95% stratified bootstrapped confidence intervals. Results are across 10 seeds except for Adam ($\beta_1 = \beta_2$), which is 3 seeds.

robustness when applied to a diverse range of environments. We then analyse the differences between Adam-Rel and Adam's updates. We compare 8 seeds on the Craftax-Classic environment for this purpose, recording the update norm, maximum update, and gradient norm of every update.

5.2 Off-policy RL

Figure 2 shows the performance of DQN agents trained with Adam-Rel against those trained with Adam-MR and Adam on the Atari-10 benchmark [14]. We tune the learning rate of each method, keeping all other hyperparameters fixed at values tuned for Adam in CleanRL [19]. Adam-Rel outperforms Adam, achieving 65.7% vs. 28.8% human-normalized performance. Furthermore, the stark performance difference between Adam-Rel and Adam-MR (23.5%) demonstrates the advantage of retaining momentum information across target changes (so long as appropriate corrections are applied), thereby contradicting the gradient contamination hypothesis discussed in Bengio et al. [11] and Asadi et al. [10].

More surprisingly, Adam-MR performs substantially worse than Adam, contrasting with the findings of Asadi et al. [10]. We evaluate on a different set of Atari games and tune both Adam and Adam-MR separately, which may account for the differences. However, these results suggest that preventing any gradient information from crossing over target changes is an excessive correction and can even harm performance. We additionally evaluate on the set of games used by Asadi et al. [10], the results of which can be found in Appendix B. We find that Adam-Rel outperforms the Adam baseline in IQM. We also find that, although our implementation of Adam-MR again significantly under-performs relative to the Adam baseline, we approximately match the returns listed in their work.

We also evaluate Adam-Rel when soft target changes are used, by comparing Adam and Adam-Rel on Atari-10 when using DQN with Polyak averaging. We find that Adam-Rel also outperforms Adam in this setting. These results, along with a more detailed discussion, can be found in Appendix C.

5.3 On-policy RL

Craftax Figure 3 shows the performance of PPO agents trained on Craftax-1B over 8 seeds. Most strikingly, Adam-MR, which resets the optimizer completely when PPO samples a new batch, achieves dramatically poorer performance across all metrics. This deficit is unsurprising when compared to its performance on DQN, where the optimizer has many more updates between resets and so can achieve a superior momentum estimate, and demonstrates the impact of not retaining any momentum information after resets in on-policy RL. Similarly, Adam with $\beta_1 = \beta_2$ [16] achieves poorer performance than Adam-Rel on all metrics and has no significant different against Adam with default hyperparameters.

Furthermore, Adam-Rel outperforms Adam on all metrics. Whilst the performance on the number of achievements is similar, we follow the evaluation procedure recommended in Hafner [21] and report score, calculated as the geometric mean of success rates for all achievements. This metric applies logarithmic scaling to the success rate of each achievement, thereby giving additional weight to those that are hardest to accomplish. We see that Adam-Rel clearly outperforms Adam in score, as well as on the two hardest achievements (collecting diamonds and eating a plant). These behaviours require

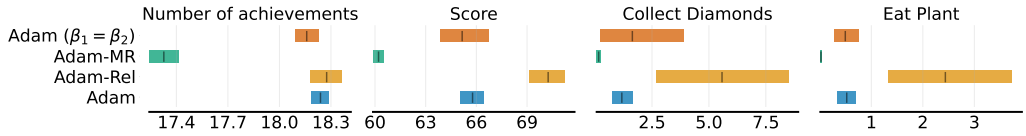


Figure 3: PPO on Craftax-1B — comparison of Adam-Rel against Adam, Adam-MR, and Adam with $\beta_1 = \beta_2$ [16]. Bars show the 95% stratified bootstrap confidence interval, with mean marked, over 8 seeds [22].

a strong policy to discover so are learned late in training, suggesting that Adam-Rel improves the plasticity of PPO.

Atari-57 Figure 2 shows the performance of PPO agents on Atari-57. As before, entirely resetting the optimizer significantly harms performance when compared to resetting only the count. Across all environments, Adam-Rel also improves over Adam, outperforming it in **33 out of the 55 games** tested and IQM across games. Adam with $\beta_1 = \beta_2$ also fails to improve over the baseline.

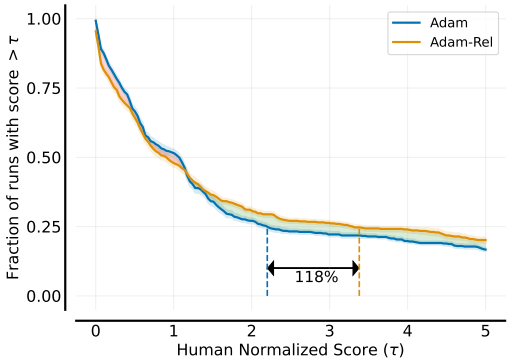


Figure 4: Performance Profile of Adam and Adam-Rel on Atari-57. Error bars represent the standard error across 10 seeds. Green-shaded areas represent Adam-Rel outperforming Adam and red-shaded areas the opposite.

To further analyse the impact of Adam-Rel over Atari-57, we plot the performance profile of human-normalized score (Figure 4). Whilst the performance of the two methods is similar over the bottom half of the profile, we see a major increase in performance in the top half. Namely, at the 75th percentile of scores Adam-Rel achieves a human-normalized performance of **338% vs. 220%** achieved by Adam. This demonstrates the ability of Adam-Rel to improve policy performance on tasks where Adam is successful but suboptimal, without sacrificing performance on harder tasks.

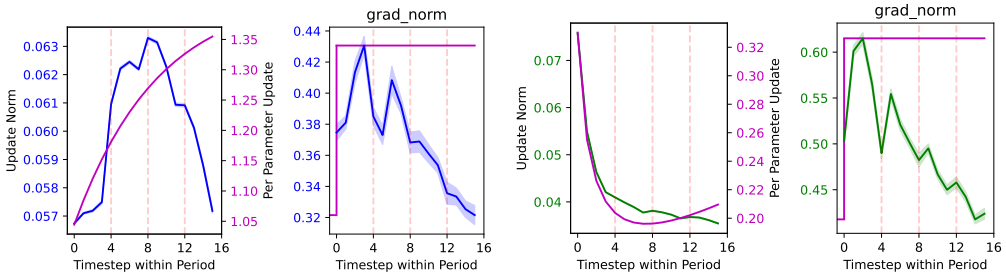
5.4 Method Analysis

In this section we connect our theoretical exposition in Section 3 to our experimental results. Specifically, we first examine whether gradients increase in magnitude due to nonstationarity, to what extent predictions from our model match the resulting updates, and how Adam's update differs from Adam-Rel's in practice.

To this end, we collect gradient (i.e., before passing through the optimizer) and update (i.e., the final change applied to the network) information from PPO on Craftax-Classic. We follow the experimental setup in Section 5 but truncate the Craftax-Classic runs to 250M steps to reduce the data processing required. The results are shown in Figure 5.

Comparing Theory and Practice In Figure 5, both Adam and Adam-Rel face a significant increase in gradient norm immediately after starting optimisation on a new objective resulting from a new batch of trajectories collected under an updated policy and value function. While this matches the assumptions we make in our work, the magnitude of the increase is much less than some of the values explored in Section 3.

For Adam, this is approximately 29% and for Adam-Rel it is around 45%. The grad norm profiles look similar in each case, with the norm peaking early before decreasing below its initial average value. This decrease and the initial ramp both deviate from the step function we assume in our model. It is obvious that our theoretical model of gradients, which requires an increase in the gradient magnitude on each abrupt change in the objective, cannot hold throughout training in its entirety because this would require the gradient norm to increase without bound.



(a) Adam (b) Adam-Rel

Figure 5: Adam and Adam-Rel compared to the theoretical model. To make this plot, we divided all the updates in the PPO run into chunks, each of which was optimising a stationary objective. We then averaged over all the chunks. The red dashed lines show the different epochs for each batch of data. The assumption about the gradient under the model is shown in the grad norm plot. Note that the update norm plot for Adam and Adam-Rel has separate y-axes. The shading represents standard error.

However, we find that despite this discrepancy, for Adam-Rel the update predicted by our model fairly closely matches the shape of the true update norm, i.e., a *fast drop* at the beginning followed by flattening (the scaling is not comparable between observed and predicted values).

For Adam, our model explains the initial *overshoot* of the update norm but then fails to predict the rapid decrease, which results from the fast drop in the true gradient norm. Given the simplicity of our modeling assumptions, we find these results overall encouraging.

On Spherical Cows Under the assumption of a *step increase* in gradients of an *infinite* relative magnitude Adam-Rel results in a flat update, while Adam would drastically overshoot. Clearly, this assumption does not hold in practice, as we have shown above. However, we believe that this mismatch between reality and assumption is encouraging, since our experimental results show that Adam-Rel is still effective in this regime. Our hypothesis is that there are two benefits to designing Adam-Rel under these assumptions. First of all, it avoids overshoots even under large gradient steps and secondly, when there are less drastic gradient steps it *undershoots*, which might have similar effects to a fast learning rate annealing. These kind of annealing schedules (over longer horizons) are popular when optimising stationary losses [23, 24].

6 Related Work

Optimization in Reinforcement Learning Plasticity loss [25–27] refers to the loss in ability of models to fit new objectives as they are trained. This is particularly relevant in nonstationary settings such as RL and continual learning, where the model is continuously fitting changing objectives. Many solutions have been proposed, including resetting network layers [28–32], policy distillation [25], LayerNorm [33, 34], regressing outputs to their initial values [26], resetting dead units [35] and adding output heads during training [36]. These solutions, in particular resetting layers during training [28, 32], have contributed towards state-of-the-art performance on Atari 100k [30]. However, of these works, only Lyle et al. [33] investigate the relationship between the optimizer and nonstationarity, demonstrating that by reducing the momentum coefficient of the second-moment gradient estimate in Adam, the fraction of dead units no longer increases. However, these works focus on plasticity loss, which is a symptom of nonstationarity, and only analyse off-policy RL. In contrast, we address nonstationarity directly and evaluate both on-policy and off-policy RL.

Meta-reinforcement learning [37–39] provides an alternative approach to designing optimizers for reinforcement learning. Rather than manually identifying problems and handcrafting solutions for RL optimization, this line of work seeks to automatically discover these solutions by meta-learning components of the optimization process. Often these methods parameterize the agent's loss function with a neural network, allowing it to be optimized through meta-gradients [40–42] or zeroth-order methods [43, 20, 44]. Recently, Lan et al. [45] proposed meta-learning a black-box optimizer directly,

demonstrating competitive performance with Adam on a range of RL tasks. However, these works are limited by the distribution of tasks they were trained on, and using handcrafted optimizers in RL is still far more popular.

Adam Extensions Cyclical update schedules [46] have previously been applied in supervised learning as a mechanism for simplifying hyperparameter tuning and improving performance, and Loshchilov and Hutter [47] propose the use of warm learning rate restarts with cosine decay for improving the training of convolutional nets. Liu et al. [48] examine the combination of Adam and learning rate warmup, proposing RAdam to stabilise training. However, all of these methods focus on supervised learning and therefore assume stationarity.

There has also been some investigation of the interaction between deep RL and momentum-based optimization. Henderson et al. [49] investigate the effects of different optimizer settings and recommend sensible parameters, but do not investigate resetting the optimizer. Bengio et al. [11] identify the problem of contamination of momentum estimates and propose a solution based on a Taylor expansion. Dohare et al. [16] investigate policy collapse in RL when training for longer than methods were tuned for and propose setting $\beta_1 = \beta_2$ to address this. By contrast, we investigate training for a standard number of steps and focus on improved overall empirical performance, rather than avoiding policy collapse. Asadi et al. [10], which is perhaps the most similar to our work, also aim to tackle contamination, but do so differently, by simply resetting the Adam momentum states to 0 whenever the target network changes in the value-based methods DQN and Rainbow. However, they do not consider resetting of Adam's timestep parameter, and explain their improved results by suggesting that old, bad, momentum estimates contaminate the gradients when training on a new objective. By contrast, we demonstrate that resetting only the timestep suffices for better performance on a range of tasks and therefore that the contamination hypothesis does not explain the better performance of resetting the optimizer. We also demonstrate that retaining momentum estimates can be essential for performance, particularly in on-policy RL.

Adam in RL To adapt Adam for use in RL, prior work has commonly applied a number of modifications compared to its use in supervised learning [8]. The first is to set the parameter ϵ to 10^{-5}, which is a higher value than the 10^{-8} typically used in supervised learning. Additionally many reinforcement learning algorithms use gradient clipping before passing the gradients to Adam. Typically gradient vectors are clipped by their L_2 norm.

A higher value of ϵ reduces the sensitivity of the optimizer to sudden large gradients. If an objective has been effectively optimized and hence the gradients are very small, then a sudden target change may lead to large gradients. $\hat{v}$ typically updates much more slowly than $\hat{m}$ and therefore this causes the update size to increase significantly, potentially causing performance collapse. However, this implementation detail is not mentioned in the PPO paper [4], and subsequent investigations omit it [6, 5]. Clipping the gradient by the norm also aims at preventing performance collapse. Andrychowicz et al. [6] find this to increase performance slightly when set to 0.5.

7 Limitations and Future Work

In this work we have mostly examined *abrupt* nonstationarity, where there are distinct changes of target, as it is in that setting where our method can be most cleanly applied. However, a range of RL methods face *continuous* nonstationarity, such as when applying Polyak averaging [1] to smoothly update target networks after every optimization step. We have demonstrated improved performance in this algorithm in Appendix C. However, further investigation into how to apply resetting in this setting would be beneficial.

There are also many promising avenues for future work. First, while we have focused on RL, it would be interesting to apply Adam-Rel to other domains that feature nonstationarity such as RLHF, training on synthetic data, or continual learning. Additionally, we also note that while our results are promising, it was not possible to investigate all RL settings and environments in this work, and we therefore encourage future work in settings such as continuous control. Secondly, Adam-Rel is designed with the principle that large updates can harm learning, but it is not clear in general what properties of update sizes are desirable in nonstationary settings. Understanding this more clearly may help produce meaningful improvements in optimisation. Relatedly, it would be beneficial to better understand the nature of gradients in RL tasks, in particular how they change throughout training for

different methods and what effect this has on performance. Finally, re-examining other aspects of the RL toolchain that are borrowed from supervised learning could produce further advancements by designing architectures, optimisers and methods specifically suited for problems in RL.

8 Conclusion

We presented a simple, theoretically-motivated method for handling nonstationarity via the Adam optimizer. By analysing the impact of large changes in gradient size, we demonstrated how directly applying Adam to nonstationary problems can lead to unstable update sizes, before demonstrating how timestep resetting corrects for this instability. Following this, we performed an extensive evaluation of Adam-Rel against Adam and Adam-MR in both on-policy and off-policy settings, demonstrating significant empirical gains. We then demonstrated that increases in gradient magnitude after abrupt objective changes occur in practice and compared the predictions of our simple theoretical model with the observed data in a complex environment. Adam-Rel can be implemented as a simple, single-line extension to any Adam-based algorithm with discrete nonstationarity (e.g. target network updates), leading to major improvements in performance across environments and algorithm classes. We hope that the ease of implementation and effectiveness of Adam-Rel will encourage researchers to use it as a de facto component of future RL algorithms, providing a step towards robust and performant RL.

Acknowledgements

BE and MJ are supported by the EPSRC centre for Doctoral Training in Autonomous and Intelligent Machines and Systems EP/S024050/1. MJ is also supported by Amazon Web Services and the Oxford-Singapore Human-Machine Collaboration Initiative. The experiments were made possible by a generous equipment grant from NVIDIA.

References

[1] Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[2] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

[5] 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.

[6] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/`. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[9] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *The Journal of Machine Learning Research*, 22(1):7459–7478, 2021.

[10] Kavosh Asadi, Rasool Fakoor, and Shoham Sabach. Resetting the optimizer in deep rl: An empirical study. *arXiv preprint arXiv:2306.17833*, 2023.

[11] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Correcting momentum in temporal difference learning. *arXiv preprint arXiv:2106.03955*, 2021.

[12] Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

[13] 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.

[14] Matthew Aitchison, Penny Sweetser, and Marcus Hutter. Atari-5: Distilling the arcade learning environment down to five games. In *International Conference on Machine Learning*, pages 421–438. PMLR, 2023.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[16] Shibhansh Dohare, Qingfeng Lan, and A Rupam Mahmood. Overcoming policy collapse in deep reinforcement learning. In *Sixteenth European Workshop on Reinforcement Learning*, 2023.

[17] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[18] 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*, 2013.

[19] 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`.

[20] 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.

[21] Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2021.

[22] 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.

[23] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[24] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[25] 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.

[26] Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2021.

[27] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[28] 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*, pages 16828–16847. PMLR, 2022.

[29] 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 *The Eleventh International Conference on Learning Representations*, 2022.

[30] 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 *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

[31] Hattie Zhou, Ankit Vani, Hugo Larochelle, and Aaron Courville. Fortuitous forgetting in connectionist networks. *arXiv preprint arXiv:2202.00155*, 2022.

[32] Charles Anderson. Q-learning with hidden-unit restarting. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL `https://proceedings.neurips.cc/paper_files/paper/1992/file/08c5433a60135c32e34f46a71175850c-Paper.pdf`.

[33] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. *arXiv preprint arXiv:2303.01486*, 2023.

[34] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[35] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. *arXiv preprint arXiv:2302.12902*, 2023.

[36] Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. *arXiv preprint arXiv:2305.15555*, 2023.

[37] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*, pages 87–94. Springer, 2001.

[38] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[39] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[40] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

[41] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[42] Matthew Thomas Jackson, Minqi Jiang, Jack Parker-Holder, Risto Vuorio, Chris Lu, Gregory Farquhar, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering general reinforcement learning algorithms with adversarial environment design. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[43] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[44] Matthew T. Jackson, Chris Lu, Louis Kirsch, Robert T. Lange, Shimon Whiteson, and Jakob N. Foerster. Discovering temporally-aware reinforcement learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=MJJcs3zbmi`.

[45] Qingfeng Lan, A Rupam Mahmood, Shuicheng Yan, and Zhongwen Xu. Learning to optimize for reinforcement learning. *arXiv preprint arXiv:2302.01470*, 2023.

[46] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[47] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2016.

[48] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2019.

[49] Peter Henderson, Joshua Romoff, and Joelle Pineau. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*, 2018.

[50] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

5.1 Limitations and Future Work

There are several limitations and interesting prospects for future work. First, it would be interesting to understand the role of non-stationarity better on *optimisation*. Is it possible to understand how much changing objectives actually cause policies to update in function space? Recent work (Milsom et al., 2025) demonstrated a computationally efficient method of measuring the update of a neural network in *function space*. Applying this to RL would undoubtedly have interesting results. For example, such techniques could be used to understand why, from a function space perspective, PPO is less likely to suffer policy collapse than actor-critic algorithms. This could also be used to examine TD learning. Recent work has shown that, in addition to target networks (Mnih et al., 2013; Fellows et al., 2023), TD learning can also be stabilised by LayerNorm (Gallici et al., 2024) and BatchNorm (Bhatt et al., 2024). Examining the effect of these stabilising effects in function space would help to understand the optimisation effect that this is having, and hence suggest new algorithmic innovation. This would help to understand how far these new stabilising techniques can be combined with off-policy exploration methods.

Secondly, we only evaluate Adam-Rel on a limited number of domains and algorithms in this paper. Understanding how this interacts with other algorithms such as distributional RL (Bellemare et al., 2017), or other Q-learning approaches such as PQN (Gallici et al., 2024) would also be an interesting avenue for further research. The theoretical model in this work is also extremely primitive. Expanding the analysis to include a linear decay from a new peak of gradient size, for example, could improve the match between theory and practice and provide new algorithmic insight. Additionally, we did not closely examine the *distribution* of gradients created by Adam-Rel. Further work to understand exactly how Adam-Rel affects this distribution, and what effect this has on performance and why would help to understand better how Adam-Rel improves performance.

However, one significant limitation of this work is how well we can reasonably expect to understand the training dynamics of such a high-dimensional nonstationary problem. Although Adam-Rel works in practice, true explanations of why will

always be difficult, or perhaps even not possible, to verify, and the same goes for many other aspects of RL training. A related and impactful approach to take to this research is to bring practice in RL in line with that from supervised learning. For example, when training PPO with CNNs in RL, learning rate warmup is hardly ever used, despite this being standard practice in supervised learning. The networks used in RL experiments often also do not use tried and tested deep learning techniques such as residual connections (He et al., 2016) and normalisation (Ba et al., 2016). While the transformer (Vaswani et al., 2017) has dominated computer vision and natural language processing research for years, it is not as widespread in reinforcement learning, perhaps because of its high inference costs. However, with increasingly efficient inference strategies being developed from LLMs, there is no longer a compelling reason to avoid transformer usage.

6
Conclusion

While RL has had a number of prominent successes, it has struggled with poor evaluation practices, poorly understood benchmarks, and a lack of understanding of the fundamental tools it uses. In this thesis, I demonstrated a number of frameworks, benchmarks, and approaches aimed at improving evaluation in RL and the understanding of its fundamentals. To improve upon these issues, this thesis is divided into two parts. The first part explored improving evaluation methods in MARL. Chapter 3 investigated the SMAC benchmark. We found that SMAC lacked stochasticity and *meaningful partial observability*, a property that we defined based on extending existing concepts inherent in the definition of a Dec-POMDP. We first demonstrated that SMAC is not stochastic by showing that an open-loop policy, that is one that does not observe environment state and conditions only on the timestep and agent ID, can attain impressive performance and examining the Q-values of a trained policy. We also observed that, because of large sight ranges and close proximity of allied units, SMAC is also not meaningfully partially observable. We then proposed SMACv2, a new benchmark which added random start positions and unit types to SMAC, and the extended partial observability challenge (EPO), which demonstrated how to introduce meaningful partial observability to SMAC. By demonstrating poor performance of the same open-loop policies on SMACv2, we show that SMACv2 addresses SMAC's lack of stochasticity. Similarly, by training

policies with perfect communication and comparing those to the partial observability setting, we demonstrate that EPO creates a meaningful *communication gap*.

Chapter 4 then introduced JaxMARL, a framework for multi-agent RL research in JAX. Writing the algorithms and environments in JAX enables experiments to be run end-to-end on the GPU, which we demonstrated could improve performance by 31× compared to CPU-based environments. Furthermore, JaxMARL incorporates a wide range of environments from cooperative and mixed settings, as well as high quality baselines. We also compared Q-learning algorithms to PPO and demonstrated their significantly worse computational performance in this new paradigm as well as their worse empirical performance. Subsequent work has expanded JaxMARL to include implementations of PQN (Gallici et al., 2024) as well as OvercookedV2 (Gessler et al., 2025), an update to the original overcooked environment (Carroll et al., 2019) which adds stochasticity and partial observability. JaxMARL combines environments from previously separated codebases and therefore stands set to improve evaluation practice in MARL significantly.

The second part of the thesis examined the RL toolchain, aiming to adapt it to better handle nonstationary optimisation. To this end, we explored the Adam optimiser and showed that a change in gradient scale leads to persistently large updates. We then demonstrated that using the local timestep (Adam-Rel) instead of the global timestep in Adam improves performance of both on-policy and off-policy algorithms on the Atari and Craftax domains. Analysing Adam-Rel, we show it limits the effect of changes in gradient scale on the update size, while reducing to learning rate decay in the case of no change in gradient scale. We then examined the correspondence between real gradient data and that predicted by our model, showing that gradient spikes do occur in practice and, in the case of Adam-Rel, the theoretical model matched our empirical results.

6.1 The Future of Multi-Agent Research

The work in this thesis has considered tabula rasa RL. While this was a popular setting for many years, the advent of ChatGPT and similar models has highlighted

the benefits of internet-scale data and self-supervised learning in achieving generalisation. Consequently, it became apparent that training neural networks from scratch in a large simulated environment was unlikely to lead to the generalisation and capabilities required for powerful AI. More broadly, if there are lessons to be drawn from the present line of MARL research, it is that the best solutions to MARL problems often just ignore their multi-agent aspect entirely. IPPO is state-of-the-art in many domains (Yu et al., 2022) and VDN performs on par with other, more complicated Q-learning approaches when tuned and trained properly (Gallici et al., 2024). Perhaps the mistake made by this line of research, then, was to treat multi-agent reinforcement learning as a problem setting, where finding settings that require uniquely multi-agent considerations is difficult. Instead, motivated by the broad need for multi-agent reasoning and skills to simply function in society, it is instead perhaps time to consider multi-agent learning a *solution* instead of a *problem*.

To make this more concrete, despite the impressive capabilities of frontier models, the work in this thesis touches on a number of their critical deficiencies. Notably, LLMs struggle with multi-agent reasoning and basic theory-of-mind (Alberts et al., 2024). These skills are fundamental to effectively collaborating with humans, and therefore important to develop if AI is to go beyond the current "assistant paradigm". Secondly, evaluation in LLMs is very difficult. Not only does training on internet-scale data open up the possibility of data leakage, but it is also difficult to construct a set of tasks or questions that provide a meaningful marker of progress. Often, it is difficult to predict which capabilities will require algorithm changes and which might be solvable by a brute-force computational solution, leading to a range of benchmarks quickly becoming saturated.

The role of multi-agent learning in this future then is currently unclear. The current paradigm seems to involve post-training LLMs on relevant tasks (such as coding and maths where results are easily verifiable) using RL, leading to significant improvements in those areas (Jaech et al., 2024). However, such an approach has a notable flaw: it is likely to lead to very large latency, particularly for difficult problems. Multi-agent learning is a natural way to address these latency problems,

and a major reason that organisations of humans use teams. By training LLMs in a multi-agent setting, tasks such as writing complicated code and proofs could be more effectively parallelised along different components or lines of enquiry. There is already some early work on training LLMs as part of MARL systems this way, but we envision a large increase in this type of work in the future. Agents in such a multi-agent system would be required to reason about the knowledge and tasks performed by their collaborators, and therefore such work would also likely have the benefit of producing systems that are capable of theory of mind, and other related collaborative reasoning, hence also likely improving their ability as assistants.

An important research direction which would contribute to this vision would be to improve the benchmarking of LLMs as assistants, particularly focusing on their ability to be coding assistants. For example, by setting a number of coding tasks and having expert software developers complete them with and without AI assistance (for example using an open-source system like Aider (AI, 2024)), it would be possible to better understand how effective current AI tools are at improving worker productivity. In general, introducing more evaluations that focus on simple AI assistant games would help to assess the abilities of LLMs, which, even at their most sophisticated, are trained on prompt-response pairs in longer, multi-step interactions.

6.1.1 Summary

The work in this thesis represents significant improvements in the evaluation of MARL algorithms in the cooperative setting, as well as improved understanding of the usage of Adam in nonstationary settings such as RL. The applications of these techniques, and in particular the most compelling way to create models capable of effectively collaborating to solve complex problems, remains an open question. However, the work in this thesis provides tools and methods for multi-agent learning to be better evaluated, better understood, and, eventually, more effective.

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.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29304–29320. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf`.

Aider AI. Aider: Ai pair programming in your terminal, 2024. URL `https://github.com/Aider-AI/aider`.

ai-dynamo contributors. NVIDIA Inference Xfer Library (NIXL). GitHub repository, November 2025. URL `https://github.com/ai-dynamo/nixl`. Version 0.8.0 (released 2025-11-20, commit 08d6094).

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv e-prints*, pages arXiv–1605, 2016.

Lize Alberts, Benjamin Ellis, Andrei Lupu, and Jakob Foerster. Curate: Benchmarking personalised alignment of conversational ai assistants. *arXiv preprint arXiv:2410.21159*, 2024.

Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

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.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK*, volume 96, pages 195–210. Citeseer, 1996.

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/jax-ml/jax`.

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. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf`.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019a.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019b.

Steven Dalton et al. Accelerating reinforcement learning through gpu atari emulation. *Advances in Neural Information Processing Systems*, 33:19773–19782, 2020.

Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Nicolaus Foerster, and Shimon Whiteson. SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL `https://openreview.net/forum?id=5OjLGiJW3u`.

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.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

Mattie Fellows, Matthew JA Smith, and Shimon Whiteson. Why target networks stabilise temporal difference methods. In *International Conference on Machine Learning*, pages 9886–9909. PMLR, 2023.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

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.

Tobias Gessler, Tin Dizdarevic, Ani Calinescu, Benjamin Ellis, Andrei Lupu, and Jakob Nicolaus Foerster. Overcookedv2: Rethinking overcooked for zero-shot coordination. In *The Thirteenth International Conference on Learning Representations*, 2025.

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.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Hengyuan Hu, Adam Lerer, Brandon Cui, Luis Pineda, Noam Brown, and Jakob Foerster. Off-belief learning. In *International Conference on Machine Learning*, pages 4369–4379. PMLR, 2021a.

Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021b.

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.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

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

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

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.

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.

Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024a.

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. *arXiv preprint arXiv:2410.23208*, 2024b.

Edward Milsom, Ben Anson, and Laurence Aitchison. Function-space learning rates. *arXiv preprint arXiv:2502.17405*, 2025.

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*, 2013.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

NVIDIA. Apex: A pytorch extension, 2024a. URL `https://github.com/NVIDIA/apex`. Accessed: 2024.

NVIDIA. Nvidia collective communications library (nccl), 2024b. URL `https://developer.nvidia.com/nccl`. Accessed: 2024.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276.

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.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation, September 2025. arXiv:2509.19128v2.

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`.

Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. *arXiv preprint arXiv:2408.15099*, 2024.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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 *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

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, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL `https://doi.org/10.1038/nature16961`.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896. PMLR, 2019.

DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34:14502–14515, 2021.

Joseph Suarez. Pufferlib: Making reinforcement learning libraries and environments play nice, 2024. URL `https://arxiv.org/abs/2406.12905`.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

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.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT Press, 2nd edition, 2018.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

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.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, pages 1–6, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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 L. 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, 2019. doi: 10.1038/s41586-019-1724-z.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8: 279–292, 1992.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

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. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Appendices

A

SMACv2 Appendices

A Links

- SMACv2: https://github.com/oxwhirl/smacv2. MIT License.
- MAPPO implementation: https://github.com/benellis3/mappo.
- QMIX implementation: https://github.com/benellis3/pymarl2. Apache-2 License.

B Further Background

B.1 StarCraft

StarCraft II is a real-time strategy game featuring 3 different races, Protoss, Terran and Zerg, with different properties and associated strategies. The objective is to build an army powerful enough to destroy the enemy's base. When battling two armies, players must ensure army units are acting optimally. This is called *micromanagement*. An important micromanagement strategy is *focus firing*, which is ordering all allied units to jointly target the enemies one by one to ensure that damage taken is minimised.

Another important strategy is *kiting*, where units flee from the enemy and then pick them off one by one as they chase.

B.2 QMIX

QMIX can be thought of as an extension of DQN[29] to the Dec-POMDP setting. The joint optimal action is found by forcing the joint Q to adhere to the individual global max (IGM) principle[40], which states that the joint action can be found by maximising individual agents' Q_i functions:

$$\arg\max_{\boldsymbol{a}} Q(s, \boldsymbol{\tau}, \boldsymbol{a}) = \begin{cases} \arg\max_a Q_1(\tau_1, a_1) \\ \arg\max_a Q_2(\tau_2, a_2) \\ \dots \\ \arg\max_a Q_n(\tau_n, a_n) \end{cases}$$

This central Q is trained to regress to a target $r + \gamma\hat{Q}(s, \boldsymbol{\tau}, \boldsymbol{a})$ where $\hat{Q}$ is a target network that is updated slowly. The central Q estimate is computed by a mixing network, whose weights are conditioned on the state, which takes as input the utility function Q_i of the agents. The weights of the mixing network are restricted to be positive, which enforces the IGM principle[40] by ensuring the central Q is monotonic in each Q_i.

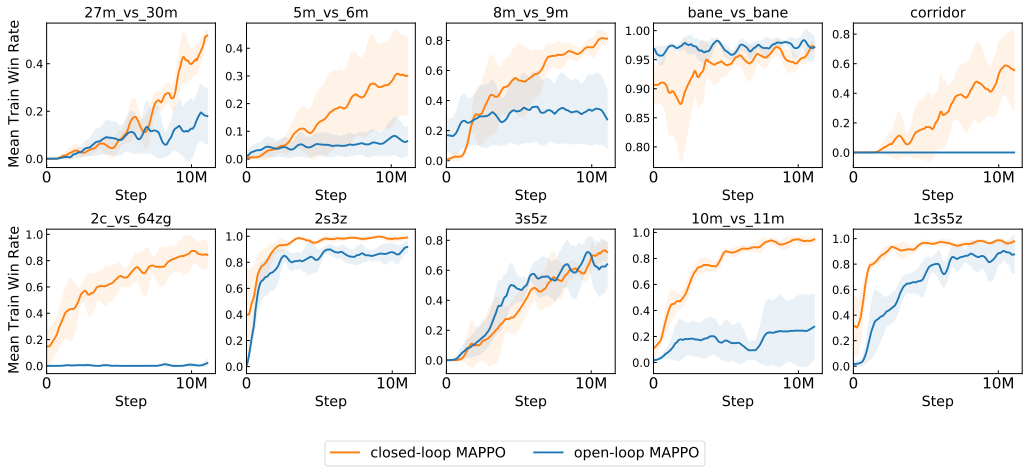
B.3 Independent and Multi-agent PPO

Proximal Policy Optimisation (PPO) is a method initially developed for single-agent reinforcement learning which aims to address performance collapse in policy gradient methods. It does this by heuristically bounding the ratio of action probabilities between the old and new policies. To this end, it optimises the below objective function.

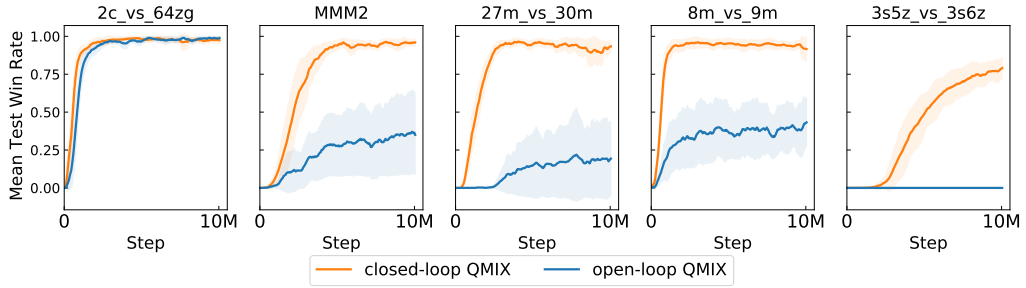
$$\mathbb{E}_{s\sim d^\pi, a\sim\pi}\left[\min\left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A^\pi(s,a), \text{clip}\left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}, 1-\epsilon, 1+\epsilon\right)A^\pi(s,a)\right)\right]$$

where $\text{clip}(t, a, b)$ is a function that outputs a if $t < a$, b if $t > b$ and t otherwise.

Extending this to the multi-agent setting can easily done in two ways. The first is to use independent learning, where each agent treats the others as part of the environment and learns a critic using its observation and action history. The second is to use a centralised critic conditioned on the central state. This is called multi-agent PPO. Both independent PPO (IPPO) and multi-agent PPO (MAPPO) have demonstrated strong performance on SMAC[55, 8]. Note that we do not apply the observation and state changes suggested by Yu et al. [55] anywhere in this paper. This is because these changes were not implemented as a wrapper on top of SMAC, but instead by modifying SMAC directly, leading to the environment code becoming unmanageably complicated.



(a) MAPPO open-loop and closed-loop results



(b) QMIX open-loop and closed-loop results

Figure 8: Plot of selected SMAC scenarios treated as an open-loop planning problem by limiting the observation to the current timestep and agent ID. Plots show the mean win rate and standard deviation across 3 training seeds for MAPPO and QMIX.

C Experimental Details

In this section we describe extra details of the experiments run as part of the paper.

C.1 Stochasticity

This section describes details of the experiments run in Section 5.1 in the main paper. Both the closed-loop and open-loop algorithms were based on the MAPPO implementation from Sun et al. [43]. The code used for these experiments can be found here. Both the open-loop and closed-loop algorithms use the same neural network architecture as in [43]. Both were also provided with access to the available actions mask of the environment and conditioned the critic on the state. We used the hyperparameters from Sun et al. [43], with the exception of the actor's learning rate, which we set to 0.0005 and decayed linearly throughout training. This is because learning rate decay has been found to be important for bounding PPO's policy updates [42]. Full hyperparameters can be found in Table 5. Full results are shown in Figure 8.

C.2 SMAC Feature Inferrability & Relevance

This section describes details of the experiments run in Section 5.2 and Section 7.3 in the main paper. The code used for these experiments can be found here.

For each scenario, we had 3 QMIX policies trained using the implementation and hyperparameters from Hu et al. [13], each with a different network and SMAC initialization. Training results for the policies for SMAC are shown in Figure 9. Each policy constitutes a seed and is used to collect a

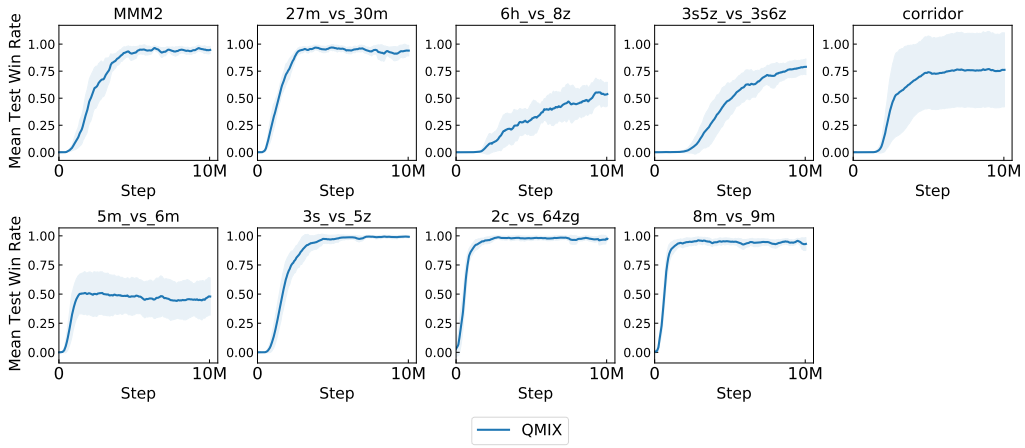


Figure 9: QMIX training results on SMAC

dataset of episodes for the feature-relevance experiment on the scenario. We call these policies *expert policies*. QMIX hyperparameters used are given in Table 4. A dataset consists of two folds: 8192 episodes used for training and 4096 episodes used for evaluation.

For a given mask, the experiment then consists of training a new QMIX network, termed *regression network*, whose input is trajectories with observations and states masked according to the mask, and whose task is to predict the Q-values output by the expert policy on the unmasked trajectories.

The *nothing* mask does not apply any effect on the observations and states in a trajectory. Otherwise, a mask, say 'health (ally)', masks the health feature of every ally in the observation of each agent (except its own health) and masks the health feature of all the units controlled by QMIX in the state. Table 1 shows the feature sets zeroed out for different masks.

The regression network has the same architecture as the expert network. We use the mean squared error (MSE) as a loss function and optimise the network via Adam with a batch size of 512 episodes and learning rate equal to 0.005. (The other Adam hyper-parameters are the default PyTorch values.) Training is performed with early stopping according to the validation fold with an evaluation every 5 epochs and a patience of 10 tries (i.e. training is stopped when the MSE on the validation fold does not decrease in 50 consecutive epochs and performance at the best epoch is retained). Models for all masks and scenarios hit early stopping and trained for about 200 epochs on average, except a few ones which trained to a limit of 500 epochs.

All hyper-parameters were tuned on a few different scenarios from both SMAC and SMACv2 to minimise validation MSE using datasets and expert policies not used for the reported results. The sizes of the training and validation folds are respectively 1.6 times and 0.8 times the size of the QMIX replay buffer (5000 episodes) and have been chosen to be large enough to minimize validation MSE while keeping experiments practical.

Figure 10 shows the full results of the feature quality experiments for SMAC and Figure 11 shows the full results for SMACv2. Table 3 shows summary results for the *everything* and *nothing* masks for SMACv2 and Table 2 shows the same data for SMAC.

Experiments for this section were conducted on 80-core CPU machines with NVIDIA GeForce RTX 2080 Ti or Tesla V100-SXM2-16GB GPUs. A single (scenario, seed, mask) combination took around 1 hour to train. In total, the experiments in this section took about 1500 GPU hours.

C.3 SMACv2 Runs

Here we describe the hyperparameter and training procedure used in section 7.1. As in previous experiments, we used the implementation by Hu et al. [13] for QMIX and by Sun et al. [43] for MAPPO. These implementations have both achieved very strong results on SMAC. We therefore tuned the hyperparameters by taking deviations from these for a few key parameters. For MAPPO, we tuned the actor's learning rate and the clipping range. Additionally, we added linear learning rate

Table 1: Masked features for each setting of the feature quality experiment. ✓means a feature is masked and ✗means a feature is not masked.

Mask	Ally						Enemy				
	Health	Shield	x	y	Distance	Actions	Health	Shield	x	y	Distance
everything	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
nothing	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
health (ally)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
shield (ally)	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
distance (ally)	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
health and shield (ally)	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
actions only	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
all except actions	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
ally all	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
health (enemy)	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
shield (enemy)	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
distance (enemy)	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
enemy all	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓

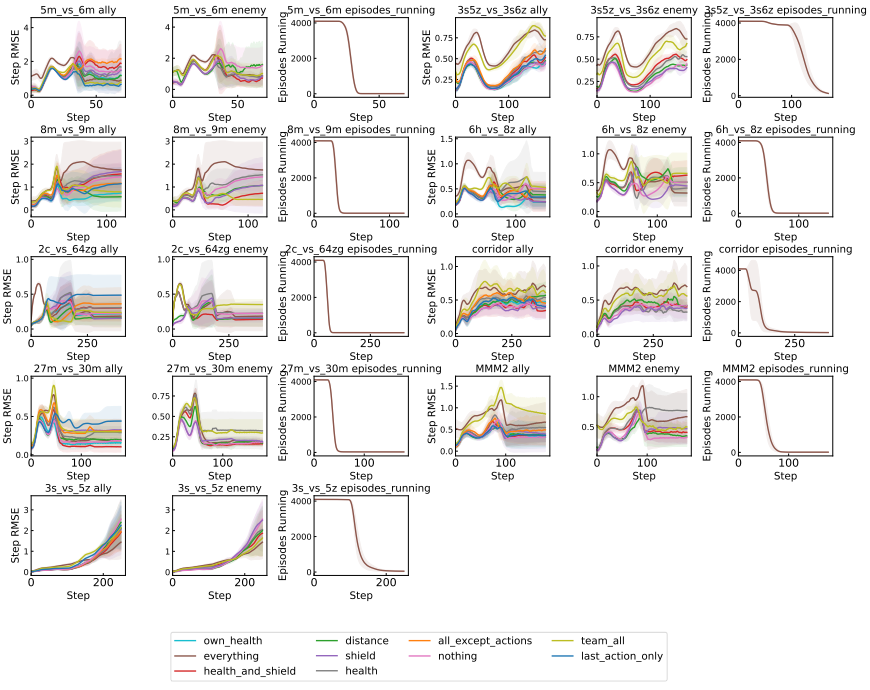


Figure 10: Results of the feature quality experiments for all tested SMAC scenarios not in the main paper.

Table 2: Mean Q values in the different feature quality experiments on SMAC scenarios

Map	Mask	$\bar{Q}$	ϵ_{rmse}	$\frac{\epsilon_{\text{rmse}}}{\bar{Q}}$	ϵ_{abs}	$\frac{\epsilon^{\text{mask}}_{\text{rmse}} - \epsilon^{\text{nothing}}_{\text{rmse}}}{\bar{Q}}$
corridor	everything	7.0 ± 0.08	0.55 ± 0.08	0.078 ± 0.002	0.39 ± 0.05	0.042 ± 0.001
	ally_all_except_actions		0.34 ± 0.09	0.047 ± 0.005	0.21 ± 0.04	0.01 ± 0.008
	ally_health_and_shield		0.29 ± 0.07	0.041 ± 0.003	0.19 ± 0.03	0.005 ± 0.006
	ally_health		0.29 ± 0.05	0.0413 ± 0.0007	0.19 ± 0.02	0.005 ± 0.004
	ally_distance		0.27 ± 0.03	0.038 ± 0.002	0.174 ± 0.009	0.001 ± 0.001
	ally_shield		0.26 ± 0.02	0.037 ± 0.003	0.166 ± 0.005	0.0004 ± 0.0003
	ally_all		0.4 ± 0.2	0.06 ± 0.01	0.28 ± 0.09	0.02 ± 0.02
	enemy_health_and_shield		0.29 ± 0.05	0.0414 ± 0.0005	0.2 ± 0.03	0.005 ± 0.004
	enemy_health		0.3 ± 0.04	0.043 ± 0.003	0.21 ± 0.02	0.006 ± 0.001
	enemy_distance		0.32 ± 0.04	0.046 ± 0.002	0.203 ± 0.009	0.009 ± 0.001
	enemy_shield		0.25 ± 0.03	0.036 ± 0.003	0.161 ± 0.005	-0.0001 ± 0.0008
	enemy_all		0.42 ± 0.09	0.059 ± 0.003	0.29 ± 0.06	0.022 ± 0.006
	nothing		0.25 ± 0.02	0.037 ± 0.003	0.164 ± 0.006	
3s5z_vs_3s6z	everything	6.97 ± 0.01	0.61 ± 0.01	0.087 ± 0.001	0.418 ± 0.006	0.046 ± 0.003
	ally_all_except_actions		0.34 ± 0.01	0.049 ± 0.002	0.193 ± 0.01	0.0083 ± 0.0009
	ally_health_and_shield		0.326 ± 0.01	0.047 ± 0.002	0.182 ± 0.009	0.006 ± 0.001
	ally_health		0.333 ± 0.006	0.0479 ± 0.0008	0.188 ± 0.001	0.007 ± 0.001
	ally_distance		0.29 ± 0.02	0.041 ± 0.003	0.16 ± 0.01	-0.0 ± 0.003
	ally_shield		0.299 ± 0.007	0.043 ± 0.001	0.167 ± 0.004	0.0018 ± 0.0004
	ally_all		0.53 ± 0.02	0.076 ± 0.003	0.35 ± 0.02	0.035 ± 0.004
	enemy_health_and_shield		0.38 ± 0.02	0.055 ± 0.003	0.25 ± 0.02	0.014 ± 0.002
	enemy_health		0.36 ± 0.01	0.051 ± 0.002	0.22 ± 0.01	0.01 ± 0.001
	enemy_distance		0.32 ± 0.01	0.046 ± 0.002	0.18 ± 0.008	0.005 ± 0.001
	enemy_shield		0.302 ± 0.004	0.0434 ± 0.0007	0.188 ± 0.006	0.0023 ± 0.0008
	enemy_all		0.5 ± 0.04	0.071 ± 0.007	0.33 ± 0.04	0.03 ± 0.006
	nothing		0.286 ± 0.008	0.041 ± 0.001	0.161 ± 0.007	
5m_vs_6m	everything	8.1 ± 0.08	1.71 ± 0.08	0.21 ± 0.02	1.24 ± 0.09	0.073 ± 0.007
	ally_all_except_actions		1.17 ± 0.04	0.15 ± 0.02	0.76 ± 0.06	0.007 ± 0.004
	ally_health_and_shield		1.13 ± 0.05	0.14 ± 0.02	0.71 ± 0.05	0.002 ± 0.002
	ally_health		1.13 ± 0.05	0.14 ± 0.02	0.72 ± 0.05	0.001 ± 0.006
	ally_distance		1.15 ± 0.05	0.15 ± 0.02	0.73 ± 0.05	0.004 ± 0.004
	ally_shield		1.12 ± 0.07	0.14 ± 0.02	0.71 ± 0.05	0.001 ± 0.002
	ally_all		1.24 ± 0.06	0.16 ± 0.02	0.83 ± 0.09	0.014 ± 0.007
	enemy_health_and_shield		1.28 ± 0.03	0.16 ± 0.02	0.81 ± 0.03	0.021 ± 0.001
	enemy_health		1.27 ± 0.05	0.16 ± 0.02	0.82 ± 0.06	0.019 ± 0.004
	enemy_distance		1.22 ± 0.02	0.15 ± 0.02	0.82 ± 0.02	0.013 ± 0.007
	enemy_shield		1.11 ± 0.06	0.14 ± 0.02	0.7 ± 0.06	-0.0 ± 0.003
	enemy_all		1.44 ± 0.06	0.18 ± 0.02	0.99 ± 0.07	0.04 ± 0.009
	nothing		1.11 ± 0.05	0.14 ± 0.02	0.71 ± 0.04	
8m_vs_9m	everything	12.3 ± 0.07	0.75 ± 0.07	0.061 ± 0.006	0.46 ± 0.06	0.025 ± 0.004
	ally_all_except_actions		0.6 ± 0.04	0.049 ± 0.004	0.32 ± 0.04	0.013 ± 0.003
	ally_health_and_shield		0.53 ± 0.05	0.043 ± 0.005	0.27 ± 0.04	0.007 ± 0.002
	ally_health		0.56 ± 0.03	0.045 ± 0.002	0.29 ± 0.02	0.01 ± 0.004
	ally_distance		0.49 ± 0.06	0.04 ± 0.005	0.27 ± 0.05	0.0039 ± 0.0006
	ally_shield		0.47 ± 0.02	0.038 ± 0.002	0.27 ± 0.02	0.002 ± 0.003
	ally_all		0.73 ± 0.07	0.06 ± 0.007	0.43 ± 0.06	0.024 ± 0.004
	enemy_health_and_shield		0.55 ± 0.04	0.045 ± 0.004	0.31 ± 0.03	0.009 ± 0.002
	enemy_health		0.55 ± 0.05	0.045 ± 0.004	0.32 ± 0.03	0.009 ± 0.001
	enemy_distance		0.46 ± 0.07	0.038 ± 0.006	0.27 ± 0.04	0.002 ± 0.002
	enemy_shield		0.42 ± 0.05	0.034 ± 0.005	0.23 ± 0.04	-0.001 ± 0.002
	enemy_all		0.64 ± 0.06	0.052 ± 0.006	0.38 ± 0.05	0.016 ± 0.003
	nothing		0.44 ± 0.07	0.036 ± 0.006	0.24 ± 0.04	
27m_vs_30m	everything	9.6 ± 0.04	0.49 ± 0.04	0.051 ± 0.006	0.33 ± 0.03	0.016 ± 0.002
	ally_all_except_actions		0.47 ± 0.05	0.049 ± 0.007	0.28 ± 0.04	0.014 ± 0.003
	ally_health_and_shield		0.44 ± 0.07	0.046 ± 0.009	0.26 ± 0.05	0.01 ± 0.003
	ally_health		0.45 ± 0.06	0.047 ± 0.007	0.26 ± 0.04	0.011 ± 0.002
	ally_distance		0.35 ± 0.06	0.037 ± 0.008	0.21 ± 0.04	0.001 ± 0.001
	ally_shield		0.35 ± 0.02	0.036 ± 0.004	0.2 ± 0.02	0.001 ± 0.003
	ally_all		0.5 ± 0.06	0.052 ± 0.008	0.32 ± 0.05	0.016 ± 0.003
	enemy_health_and_shield		0.46 ± 0.04	0.048 ± 0.006	0.31 ± 0.03	0.012 ± 0.004
	enemy_health		0.5 ± 0.05	0.052 ± 0.007	0.34 ± 0.04	0.0166 ± 0.0003
	enemy_distance		0.41 ± 0.03	0.042 ± 0.005	0.25 ± 0.03	0.007 ± 0.003
	enemy_shield		0.35 ± 0.06	0.036 ± 0.007	0.21 ± 0.03	0.0009 ± 0.0008
	enemy_all		0.53 ± 0.02	0.055 ± 0.004	0.37 ± 0.02	0.02 ± 0.002
	nothing		0.34 ± 0.05	0.036 ± 0.006	0.21 ± 0.04	
2c_vs_64zg	everything	7.72 ± 0.02	0.56 ± 0.02	0.072 ± 0.002	0.42 ± 0.01	0.054 ± 0.003
	ally_all_except_actions		0.15 ± 0.04	0.02 ± 0.005	0.08 ± 0.02	0.0018 ± 0.001
	ally_health_and_shield		0.15 ± 0.04	0.02 ± 0.005	0.08 ± 0.02	0.002 ± 0.002
	ally_health		0.15 ± 0.03	0.02 ± 0.004	0.08 ± 0.02	0.0018 ± 0.0005
	ally_distance		0.13 ± 0.03	0.017 ± 0.003	0.07 ± 0.01	-0.0007 ± 0.0004
	ally_shield		0.14 ± 0.02	0.018 ± 0.002	0.07 ± 0.01	0.0 ± 0.001
	ally_all		0.15 ± 0.04	0.019 ± 0.005	0.08 ± 0.02	0.002 ± 0.001
	enemy_health_and_shield		0.43 ± 0.006	0.056 ± 0.002	0.317 ± 0.004	0.038 ± 0.003
	enemy_health		0.426 ± 0.006	0.0553 ± 0.001	0.314 ± 0.006	0.037 ± 0.003
	enemy_distance		0.18 ± 0.02	0.024 ± 0.002	0.125 ± 0.009	0.006 ± 0.001
	enemy_shield		0.14 ± 0.03	0.018 ± 0.003	0.07 ± 0.01	0.0 ± 0.0008
	enemy_all		0.54 ± 0.02	0.07 ± 0.002	0.41 ± 0.01	0.052 ± 0.003
	nothing		0.14 ± 0.03	0.018 ± 0.004	0.07 ± 0.02	
6h_vs_8z	everything	7.5 ± 0.09	0.87 ± 0.09	0.12 ± 0.02	0.65 ± 0.09	0.07 ± 0.01
	ally_all_except_actions		0.43 ± 0.02	0.059 ± 0.008	0.31 ± 0.03	0.007 ± 0.002
	ally_health_and_shield		0.41 ± 0.03	0.056 ± 0.009	0.29 ± 0.03	0.0034 ± 0.0007
	ally_health		0.41 ± 0.03	0.055 ± 0.008	0.28 ± 0.03	0.003 ± 0.002
	ally_distance		0.41 ± 0.03	0.056 ± 0.009	0.29 ± 0.03	0.004 ± 0.001
	ally_shield		0.38 ± 0.03	0.051 ± 0.008	0.26 ± 0.03	-0.001 ± 0.001
	ally_all		0.49 ± 0.01	0.067 ± 0.007	0.36 ± 0.02	0.014 ± 0.002
	enemy_health_and_shield		0.45 ± 0.02	0.06 ± 0.008	0.32 ± 0.02	0.008 ± 0.001
	enemy_health		0.46 ± 0.03	0.063 ± 0.01	0.33 ± 0.03	0.011 ± 0.001
	enemy_distance		0.47 ± 0.03	0.064 ± 0.009	0.33 ± 0.03	0.012 ± 0.0004
	enemy_shield		0.39 ± 0.02	0.053 ± 0.008	0.27 ± 0.03	0.001 ± 0.002
	enemy_all		0.55 ± 0.02	0.075 ± 0.008	0.4 ± 0.02	0.0224 ± 0.001
	nothing		0.38 ± 0.03	0.052 ± 0.009	0.27 ± 0.03	
3s_vs_5z	everything	8.1 ± 0.04	0.3 ± 0.04	0.037 ± 0.007	0.2 ± 0.03	0.012 ± 0.003
	ally_all_except_actions		0.19 ± 0.02	0.024 ± 0.003	0.1 ± 0.02	-0.0014 ± 0.0008
	ally_health_and_shield		0.17 ± 0.02	0.021 ± 0.002	0.081 ± 0.01	-0.004 ± 0.001
	ally_health		0.17 ± 0.03	0.021 ± 0.004	0.07 ± 0.01	-0.004 ± 0.002
	ally_distance		0.2 ± 0.02	0.024 ± 0.002	0.09 ± 0.02	-0.0 ± 0.004
	ally_shield		0.18 ± 0.02	0.022 ± 0.002	0.083 ± 0.007	-0.003 ± 0.003
	ally_all		0.3 ± 0.08	0.04 ± 0.01	0.17 ± 0.05	0.012 ± 0.007
	enemy_health_and_shield		0.26 ± 0.03	0.032 ± 0.004	0.17 ± 0.03	0.007 ± 0.001
	enemy_health		0.24 ± 0.03	0.03 ± 0.005	0.15 ± 0.03	0.005 ± 0.002
	enemy_distance		0.19 ± 0.02	0.024 ± 0.003	0.09 ± 0.02	-0.0015 ± 0.0006
	enemy_shield		0.24 ± 0.02	0.03 ± 0.004	0.13 ± 0.01	0.0049 ± 0.0002
	enemy_all		0.26 ± 0.03	0.033 ± 0.004	0.17 ± 0.02	0.008 ± 0.001
	nothing		0.2 ± 0.02	0.025 ± 0.003	0.08 ± 0.01	
MMM2	everything	9.4 ± 0.1	0.7 ± 0.1	0.08 ± 0.02	0.47 ± 0.09	0.039 ± 0.008
	ally_all_except_actions		0.41 ± 0.07	0.04 ± 0.01	0.23 ± 0.05	0.007 ± 0.003
	ally_health_and_shield		0.38 ± 0.03	0.041 ± 0.005	0.22 ± 0.03	0.003 ± 0.002
	ally_health		0.39 ± 0.04	0.042 ± 0.006	0.21 ± 0.04	0.004 ± 0.002
	ally_distance		0.34 ± 0.05	0.036 ± 0.007	0.19 ± 0.04	-0.0015 ± 0.0007
	ally_shield		0.35 ± 0.06	0.038 ± 0.008	0.2 ± 0.04	-0.0001 ± 0.0009
	ally_all		0.6 ± 0.1	0.07 ± 0.02	0.36 ± 0.09	0.028 ± 0.008
	enemy_health_and_shield		0.43 ± 0.06	0.046 ± 0.008	0.29 ± 0.04	0.008 ± 0.001
	enemy_health		0.44 ± 0.05	0.047 ± 0.007	0.29 ± 0.04	0.0094 ± 0.0005
	enemy_distance		0.33 ± 0.05	0.036 ± 0.006	0.19 ± 0.04	-0.0022 ± 0.0009
	enemy_shield		0.33 ± 0.04	0.035 ± 0.006	0.18 ± 0.04	-0.002 ± 0.002
	enemy_all		0.59 ± 0.08	0.06 ± 0.01	0.39 ± 0.05	0.026 ± 0.004
	nothing		0.35 ± 0.05	0.038 ± 0.007	0.2 ± 0.04	

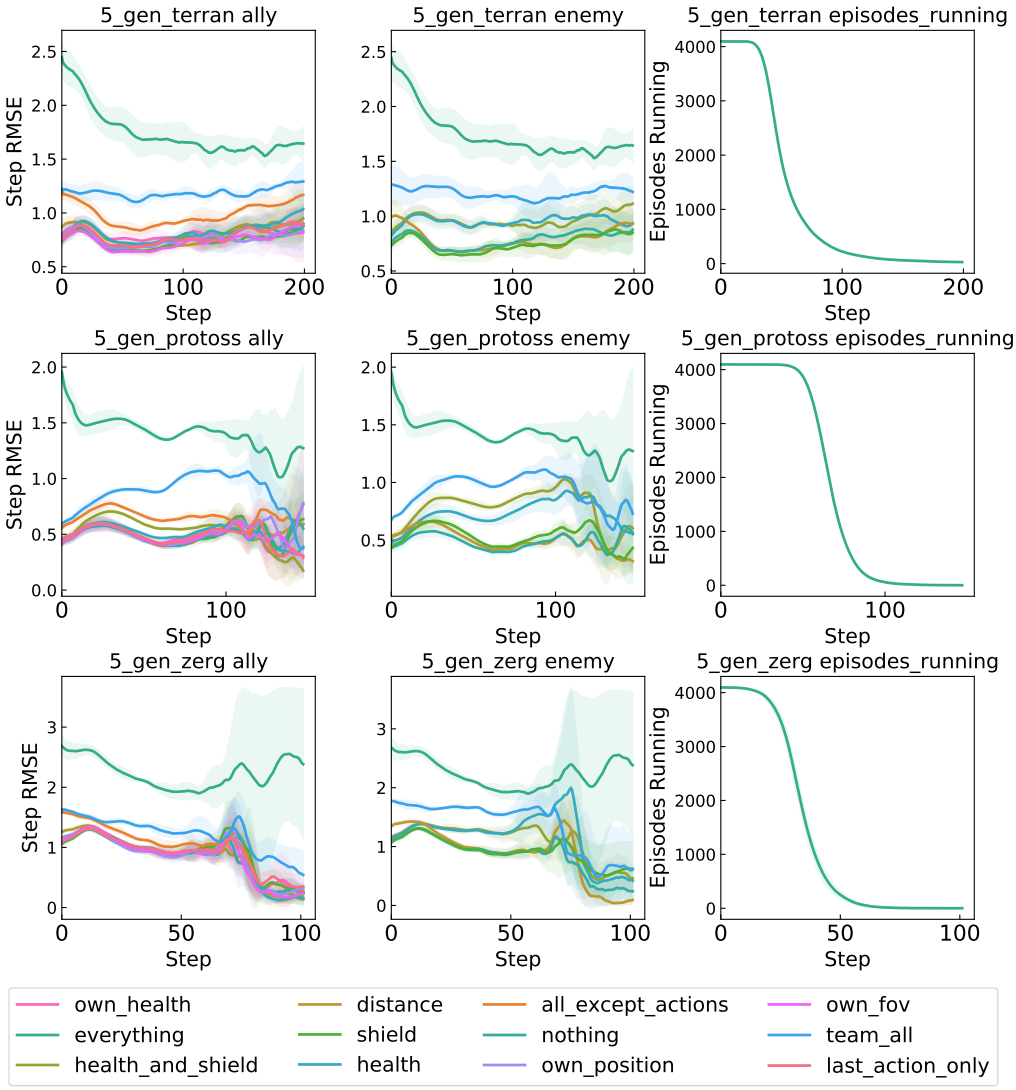


Figure 11: Feature Quality Experiments for the 5 and 10 unit SMACv2 scenarios.

decay to the MAPPO implementation because this has been shown to be important to bound the policy update[42]. For QMIX, we tuned the ϵ-annealing time and λ in the eligibility trace $Q(\lambda)$. All other hyperparameters, including neural network architecture, were unchanged from the implementations mentioned previously.

The MAPPO code for these experiments can be found here, and the QMIX code here. The QMIX code is distributed under the Apache license, and the MAPPO code under the MIT license. The only differences in these branches and the implementations used for previous experiments on SMAC are important to changing the environment from SMAC to SMACv2.

The open-loop policy is identical to MAPPO, except the policy receives as input only the timestamp and agent ID, as in Section 5.1.

We tuned hyperparameters by running each set of hyperparameters on all scenarios and then choosing the best results. The grids of hyperparameters for QMIX and MAPPO can be found in Table 7 and 8. For IPPO, we used mostly the same hyperparameters as MAPPO. For QPLEX we used the hyperparameters from [51]. All algorithms were trained for 10M environment steps with evaluations

Table 3: SMACv2 feature quality experiment results

Map	Mask	$\bar{Q}$	ϵ_{rmse}	$\frac{\epsilon_{\text{rmse}}}{\bar{Q}}$	ϵ_{abs}	$\frac{\epsilon^{\text{mask}}_{\text{rmse}} - \epsilon^{\text{nothing}}_{\text{rmse}}}{\bar{Q}}$
5_gen_protoss	everything	8.4 ± 0.05	1.5 ± 0.05	0.179 ± 0.006	1.18 ± 0.04	0.119 ± 0.004
	ally_all_except_actions		0.69 ± 0.02	0.082 ± 0.004	0.52 ± 0.01	0.022 ± 0.002
	ally_health_and_shield		0.62 ± 0.02	0.074 ± 0.004	0.45 ± 0.02	0.014 ± 0.002
	ally_health		0.54 ± 0.03	0.064 ± 0.005	0.39 ± 0.02	0.004 ± 0.001
	ally_distance		0.52 ± 0.04	0.062 ± 0.006	0.38 ± 0.03	0.0019 ± 0.0005
	ally_shield		0.52 ± 0.04	0.062 ± 0.006	0.38 ± 0.03	0.0018 ± 0.0008
	ally_all		0.85 ± 0.009	0.102 ± 0.004	0.656 ± 0.01	0.042 ± 0.002
	enemy_health_and_shield		0.79 ± 0.03	0.094 ± 0.004	0.6 ± 0.03	0.034 ± 0.003
	enemy_health		0.69 ± 0.05	0.082 ± 0.007	0.51 ± 0.04	0.022 ± 0.003
	enemy_distance		0.56 ± 0.03	0.067 ± 0.006	0.41 ± 0.03	0.0074 ± 0.0002
	enemy_shield		0.57 ± 0.03	0.068 ± 0.005	0.42 ± 0.02	0.008 ± 0.002
	enemy_all		0.96 ± 0.02	0.116 ± 0.004	0.74 ± 0.02	0.055 ± 0.002
	nothing		0.5 ± 0.03	0.06 ± 0.006	0.37 ± 0.03	
5_gen_terran	everything	8.2 ± 0.1	2.0 ± 0.1	0.243 ± 0.009	1.6 ± 0.1	0.148 ± 0.008
	ally_all_except_actions		1.0 ± 0.05	0.1221 ± 0.001	0.75 ± 0.04	0.028 ± 0.002
	ally_health_and_shield		0.82 ± 0.04	0.1 ± 0.001	0.6 ± 0.03	0.0052 ± 0.0009
	ally_health		0.83 ± 0.04	0.1007 ± 0.0003	0.6 ± 0.04	0.0061 ± 0.0007
	ally_distance		0.78 ± 0.04	$0.0945 \pm 9e-05$	0.56 ± 0.03	0.0002 ± 0.0007
	ally_shield		0.75 ± 0.02	0.092 ± 0.002	0.55 ± 0.02	-0.003 ± 0.001
	ally_all		1.18 ± 0.06	0.145 ± 0.002	0.92 ± 0.06	0.05 ± 0.003
	enemy_health_and_shield		0.98 ± 0.06	0.12 ± 0.003	0.72 ± 0.05	0.025 ± 0.003
	enemy_health		0.97 ± 0.07	0.118 ± 0.003	0.71 ± 0.05	0.023 ± 0.003
	enemy_distance		0.82 ± 0.04	0.1002 ± 0.0002	0.59 ± 0.03	0.0055 ± 0.0006
	enemy_shield		0.75 ± 0.05	0.091 ± 0.005	0.54 ± 0.04	-0.004 ± 0.004
	enemy_all		1.2 ± 0.1	0.151 ± 0.007	0.95 ± 0.09	0.057 ± 0.008
	nothing		0.78 ± 0.03	0.0945 ± 0.0007	0.56 ± 0.03	
5_gen_zerg	everything	7.2 ± 0.05	2.41 ± 0.05	0.33 ± 0.02	1.86 ± 0.04	0.176 ± 0.009
	ally_all_except_actions		1.36 ± 0.02	0.188 ± 0.008	0.99 ± 0.02	0.031 ± 0.002
	ally_health_and_shield		1.15 ± 0.03	0.16 ± 0.005	0.83 ± 0.03	0.002 ± 0.001
	ally_health		1.17 ± 0.03	0.161 ± 0.005	0.83 ± 0.02	0.0037 ± 0.0008
	ally_distance		1.18 ± 0.02	0.164 ± 0.007	0.85 ± 0.02	0.006 ± 0.001
	ally_shield		1.13 ± 0.02	0.157 ± 0.006	0.81 ± 0.02	-0.0005 ± 0.0003
	ally_all		1.46 ± 0.02	0.202 ± 0.01	1.09 ± 0.02	0.044 ± 0.002
	enemy_health_and_shield		1.31 ± 0.03	0.18 ± 0.008	0.95 ± 0.03	0.024 ± 0.002
	enemy_health		1.3 ± 0.02	0.179 ± 0.007	0.94 ± 0.02	0.022 ± 0.001
	enemy_distance		1.24 ± 0.01	0.173 ± 0.008	0.88 ± 0.02	0.014 ± 0.002
	enemy_shield		1.14 ± 0.02	0.159 ± 0.006	0.82 ± 0.02	0.0005 ± 0.0008
	enemy_all		1.67 ± 0.01	0.23 ± 0.01	1.24 ± 0.02	0.074 ± 0.008
	nothing		1.14 ± 0.02	0.157 ± 0.007	0.81 ± 0.02	

of 20 episodes every 2k steps. The hyperparameters used for QMIX are given in Table 4, for MAPPO and IPPO in Table 5, and for QPLEX in Table 6.

C.4 EPO Runs

This section provides the implementation details for the Extended Partial Observability challenge in SMACv2 and outlines the corresponding training procedure.

For the EPO baselines in Section 7.2, we used the same hyperparameters as Section 7.1. Implementations of MAPPO and QMIX were also consistent with those in Section 7.1. Each training run and corresponding evaluations were conducted across 3 random seeds. Experiments for this section were conducted on 80-core CPU machines with NVIDIA GeForce RTX690 2080 Ti or Tesla V100-SXM2-16GB GPUs. Both MAPPO and QMIX experiments took between 36-48 hours to complete, each running on 8 GPUs, for 3 p values, across 3 seeds and 3 races.

D Environment Additional Details

D.1 SMACv2 Additional Details

In this section we describe the generation of teams in SMACv2. For each race, 3 different types of units are used. Each race has a special unit that should not be generated too often. For Protoss this is the colossus. This is a very powerful unit. If a team has many colossi, the battle will devolve into a war about who can use their colossi most effectively. Terran has the medivac unit. This is a healing unit that cannot attack the enemy and so is only spawned sparingly. Zerg has the baneling unit. This is a suicidal unit which deals area-of-effect damage to enemy units by rolling into them and exploding. In scenarios with many banelings, the agents learn to spread out and hide in the corners

Table 4: QMIX hyperparameters used for experiments. Parameters with (SMAC) or (SMACv2) after them denote that parameter setting was only used for SMAC or SMACv2 experiments respectively. These are the values in the corresponding configuration file in PyMarl[37, 13]. Mac is the code responsible for marshalling inputs to the neural networks, learner is the code used for learning and runner determines whether experience is collected in serial or parallel.

Parameter Name	Value
Action Selector	epsilon greedy
ϵ Start	1.0
ϵ Finish	0.05
ϵ Anneal Time	100000
Runner	parallel
Batch Size Run	4
Buffer Size	5000
Batch Size	128
Optimizer	Adam
t_{max}	10050000
Target Update Interval	200
Mac	n_mac
Agent	n_rnn
Agent Output Type	q
Learner	nq_learner
Mixer	qmix
Mixing Embed Dimension	32
Hypernet Embed Dimension	64
Learning Rate	0.001
λ (SMAC)	0.6
λ (SMACv2)	0.4

with the hope that the enemy banelings explode and the allies win by default. All of these special units are spawned with a probability of 10%. The other units used spawn with a probability of 45%. This is summarised in Table 9.

There are two changes to the observation space from SMAC. First, each agent observes their own field-of-view direction. Secondly, each agent observes their own position in the map as x- and y-coordinates. This is normalised by dividing by the map width and height respectively. The only change to the state from SMAC was to add the field-of-view direction of each agent to the state.

Additionally, we made one small change to the reward function in SMACv2. This was to fix a bug where the enemies healing can give allied units reward. There are more details available about this problem in the associated Github issue. SMACv2 also has an identical API to the original SMAC, allowing for very simple transition between the two frameworks.

The code for SMACv2 can be found in the Github repo, where there is a README detailing how to run the benchmark with random agents.

D.2 EPO Additional Details

In EPO, the first ally agent to observe an enemy is guaranteed to observe the usual SMAC features associated with that enemy. Upon an enemy being observed for the first time, by any agent on the ally team, a random binary draw occurs for all other agents (i.e., those that had not yet observed this particular enemy). Random tie breaking ensures only one agent is guaranteed to see the enemy should two or more observe it for the first time on the same timestep. The draw is weighted by a tunable environmnet parameter p corresponding to the probability of success. If the draw is successful for a particular agent, any future instances of the agent observing the enemy will occur as normal, without masking. If the draw is unsuccessful, that agent will not be able to observe the enemy on future timesteps, irrespective of whether or not it is within sight range. If the first agent to have observed the enemy dies, the next time the enemy falls within an ally agent's sight range that agent is guaranteed to see the enemy and the random draw occurs again for all other agents.

Table 5: MAPPO and IPPO hyperparameters used for the experiments on SMAC and SMACv2. Parameters with (SMAC) or (SMACv2) after them denote that parameter setting was only used for SMAC or SMACv2 experiments respectively.

Hyperparameter	Value
Action Selector	multinomial
Mask Before Softmax	True
Runner	parallel
Buffer Size	8
Batch Size Run	8
Batch Size	8
Target Update Interval	200
Learning Rate Actor (SMAC)	0.001
Learning Rate Actor (SMACv2)	0.0005
Learning Rate Critic	0.001
τ	0.995
λ	0.99
Agent Output Type	pi_logits
Learner	trust_region_learner
Critic (MAPPO)	centralV_rnn_critic
Critic (IPPO)	decentral_rnn_critic
Detach Every	20
Replace Every	None
Mini Epochs Actor	10
Mini Epochs Critic	10
Entropy Loss Coeff	0.0
Advantage Calc Method	GAE
Bootstrap Timeouts	False
Surrogate Clipped	True
Clip Range	0.1
Is Advantage Normalized	True
Observation Normalized	True
Use Popart	False
Normalize Value	False
Obs Agent Id	True
Obs Last Action	False

E Analysis of Changes in SMACv2

To investigate the impact of the changes introduced in SMACv2, we perform three ablation studies. We focus only on the 5 unit scenarios, and only train MAPPO on the ablations because it is significantly faster to train than QMIX. We use the same hyperparameters and setup as in Section 7.1. We ablate different team compositions by using a single unit type for each of the non-special units in each race. We investigate random start positions by only using the *surround* or *reflect* scenarios and by removing the random start positions entirely. We also ablate the effect of using the true unit ranges. Additionally, we investigate partial observability by comparing feed-forward and recurrent networks.

The results of these experiments are shown in Figure 12. The position ablations show that stochastic starting positions and hence stochasticity itself contributes greatly to the challenge of SMACv2. Without random start positions, MAPPO achieves win rates of close to 100%. Both *surround* and *reflect* scenarios have a similar win rate, suggesting similar difficulty. There does not appear to be additional difficulty from combining the two. This is logical as the scenarios can be disambiguated using start positions at the beginning of the episode.

Our experiments of unit type ablations, shown in Figure 12b, indicate that unit variety significantly impacts the difficulty of the tasks. All three races show large differences between the easiest unit distribution and the baseline, suggesting that a diverse range of units contributes to SMACv2's difficulty. In both the Zerg and Protoss scenarios, the melee-only scenarios are easier than the ablation

Table 6: QPLEX hyperparameters used for experiments on SMACv2.

Hyperparameter	Value
Action Selector	epsilon_greedy
Epsilon Start	1.0
Epsilon Finish	0.05
Epsilon Anneal Time	50000
Runner	parallel
Batch Size Run	4
Buffer Size	5000
Batch Size	128
Optimizer	adam
Target Update Interval	200
Agent Output Type	q
Learner	dmaq_qatten_learner
Double Q	True
Mixer	dmaq_qatten
Mixing Embed Dim	32
Hypernet Embed	64
Adv Hypernet Layers	1
Adv Hypernet Embed	64
Td Lambda	0.8
Num Kernel	4
Is Minus One	True
Is Adv Attention	True
Is Stop Gradient	True
N Head	4
Attend Reg Coef	0.001
State Bias	True
Mask Dead	False
Weighted Head	False
Nonlinear	False
Burn In Period	100
Name	qplex_qatten_sc2

Table 7: QMIX hyperparameter tuning grid

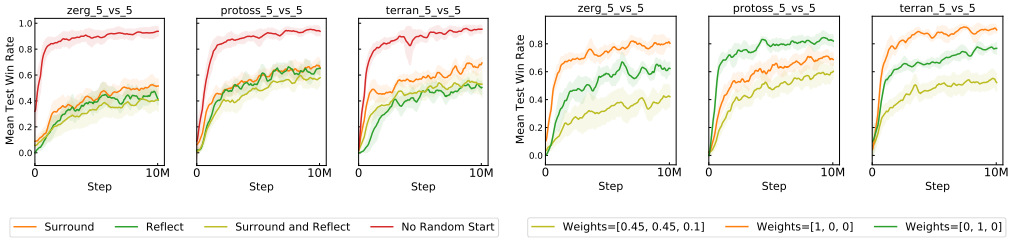
Hyperparameter	Values Tried
ϵ annealing time	$[100 \times 10^3, 500 \times 10^3]$
λ	[0.6, 0.8, 0.4]

Table 8: MAPPO hyperparameter tuning grid

Hyperparameter	Value
Actor Learning Rate	$[0.0007, 0.0004, 0.0001]$
Clip Range	$[0.15, 0.05, 0.1]$

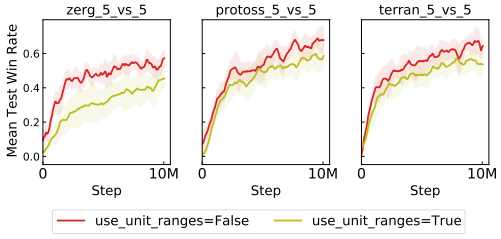
Table 9: Unit types used per-race in the SMACv2 scenarios.

Race	Unit Types	Probability of Generation
Terran	Marine	0.45
	Marauder	0.45
	Medivac	0.1
Zerg	Zergling	0.45
	Baneling	0.1
	Hydralisk	0.45
Protoss	Stalker	0.45
	Zealot	0.45
	Colossus	0.1

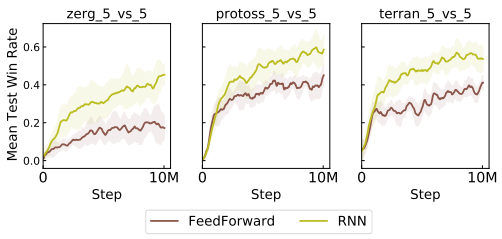


(a) Position ablations

(b) Unit type ablations



(c) Field-of-view ablations



(d) Feed-forward vs RNN ablation

Figure 12: Ablation of SMACv2 unit type, field-of-view and starting position features. Figure 12d shows the performance of a feed-forward policy compared to an RNN baseline. The unit weights are relative to a fixed order of units. For Zerg this is zergling, hydralisk, baneling, for Terran it is marine, marauder, medivac and for Protoss it is stalkers, zealots and colossi. Plots show mean and standard deviation across three seeds.

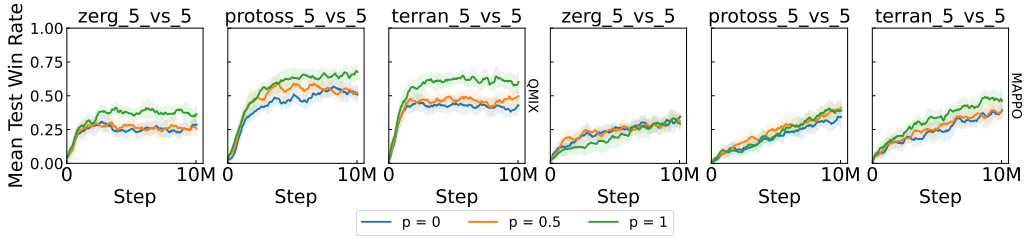


Figure 13: Comparison of the mean test win rate when $p = 0, 0.5, 1$ for QMIX (left) and MAPPO (right) on SMACv2. Here, the available action mask is still present.

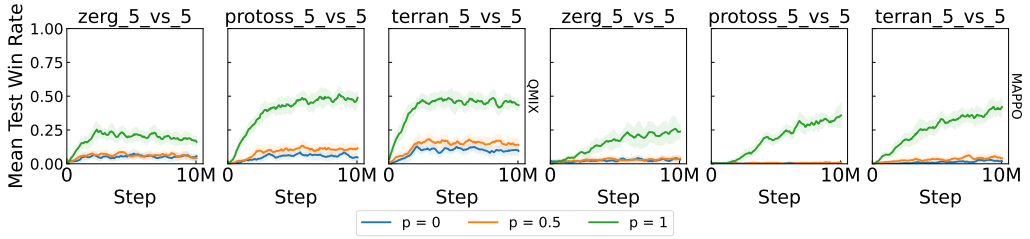


Figure 14: Comparison of mean test win rate when $p = 0, 0.5, 1$ with 5 agents against 5 enemy units, for QMIX (left) and MAPPO (right). Here, the available actions mask has been removed.

with ranged units. From observing episodes, the enemy AI tends to aggressively pursue allied units, which allows the allies to lure it out of position. This is easier to exploit with melee units than ranged ones. Terran scenarios have no melee units, but the marine scenarios are slightly easier. Overall these results show that generalising to different unit types is a significant part of the challenge in SMACv2.

Our field-of-view ablations, shown in Figure 12c, compare the fixed unit sight and attack ranges to the SMACv2 versions. There is an increase in difficulty from varying the unit ranges, but this effect is small. This suggests that the difficulty of SMACv2 is better explained by the diversity of start positions and unit types than the change to the true sight and attack ranges.

Finally, to evaluate the effect that the changes had on the partial observability of SMACv2, we compare a feedforward network with the performance of the RNN baseline, shown in Figure 12d. There is a significant decrease in performance from using the feedforward network. However, the feedforward network can still learn reasonable policies. This suggests that although being able to resolve partial observability is useful in SMACv2, the primary difficulty consists in generalising to a range of micro-management scenarios. This is in contrast to the extended partial observability challenge, where resolving partial observability through communication is the primary difficulty. The zerg scenario has a larger relative performance difference. Partial observability may be more of a problem here because the splash damage done to allies by banelings creates an incentive for allies to spread out more.

Overall, the results in Figures 12a and 12b show that current MARL methods struggle with the increased stochasticity due to map randomisation. To solve this problem, agents must be able to use their observations to make inferences about relevant state or joint observation features in the Dec-POMDP in more general settings. Current inference capabilities could be improved by research on more effective sequence models, such as specialised transformers [47, 16, 5]. The work in this area either applies transformer models to traditional RL methods [15], or focuses on paradigms similar to upside-down RL [39, 5]. While the latter work shows promising generalisation [34], it is mostly in the offline setting. Hu et al. [14] demonstrate a promising sequence model that can be applied to any MARL algorithm, but only demonstrate their work on marine units, and rely on the Dec-POMDP having a certain structure. Expanding such work to handle more diverse scenarios is an interesting avenue of future work.

The code for the ablations can be found here. This is distributed under an MIT license. The changes on this branch enable easy running of the ablations and add associated environment configurations for this purpose.

Table 10: SMACv2 Scenarios with their number of allies, number of enemies, and unit types.

Scenario Name	Number of Allies	Number of Enemies	Unit Types
protoss_5_vs_5	5	5	Stalker, Zealot, Colossus
protoss_10_vs_10	10	10	Stalker, Zealot, Colossus
protoss_20_vs_20	20	20	Stalker, Zealot, Colossus
protoss_10_vs_11	10	11	Stalker, Zealot, Colossus
protoss_20_vs_23	20	23	Stalker, Zealot, Colossus
terran_5_vs_5	5	5	Marine, Marauder, Medivac
terran_10_vs_10	10	10	Marine, Marauder, Medivac
terran_20_vs_20	20	20	Marine, Marauder, Medivac
terran_10_vs_11	10	11	Marine, Marauder, Medivac
terran_20_vs_23	20	23	Marine, Marauder, Medivac
zerg_5_vs_5	5	5	Zergling, Hydralisk, Baneling
zerg_10_vs_10	10	10	Zergling, Hydralisk, Baneling
zerg_20_vs_20	20	20	Zergling, Hydralisk, Baneling
zerg_10_vs_11	10	11	Zergling, Hydralisk, Baneling
zerg_20_vs_23	20	23	Zergling, Hydralisk, Baneling

F Limitations and Broader Impact

The main limitation of SMACv2 is that it is confined to scenarios within the game of StarCraft II. This is an environment which, while complex, cannot represent the dynamics of all multi-agent tasks. Evaluation of MARL algorithms therefore should not be limited to one benchmark, but should target variety with a range of tasks.

Whilst the scenarios in SMACv2 involve battles between two armies of units, only one side can be controlled by RL agents. It is technically possible to control two armies using two StarCraft II clients that communicate via LAN, which would allow to train two groups of decentralised agents against each other, e.g., via self-play. We leave the implementation of this functionality as future work.

SMACv2 aims to contribute to the development of MARL algorithms. As with any machine learning field, it is possible that improving the capabilities of these algorithms could lead to unethical uses. However, there are also many potential benefits to better cooperative AI, such as applications in automated driving among others. We believe that the potential benefits of developing more capable and cooperative AI outweigh the potential risks.

B

JaxMARL 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

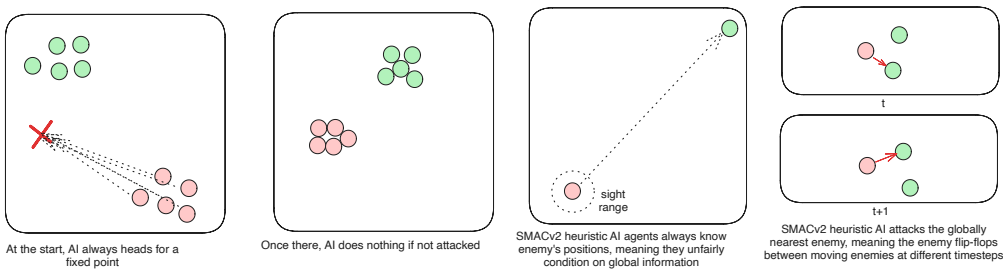
B.1 SMAX

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

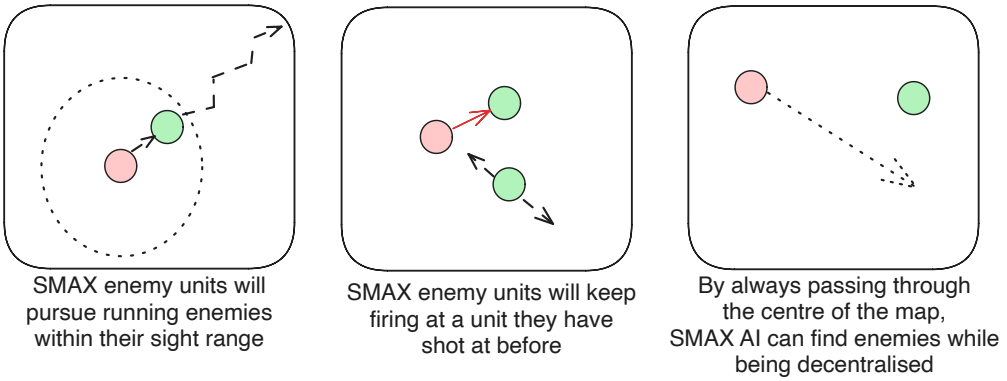


Figure 8: Explanation of the operation of the SMAX heuristic AI.

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].

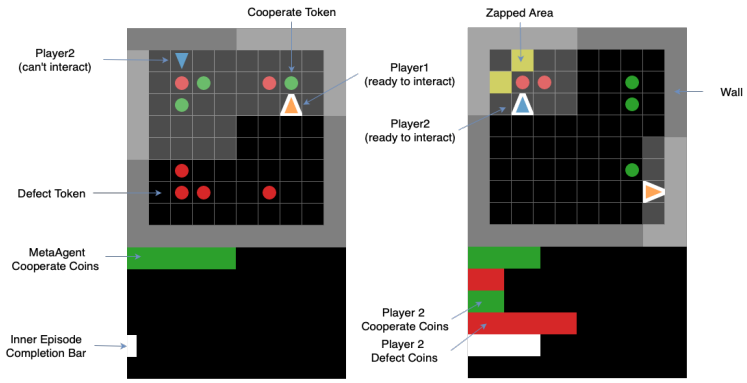


Figure 9: Annotated Image of IPDiTM renders, demonstrating the objects within the game

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.

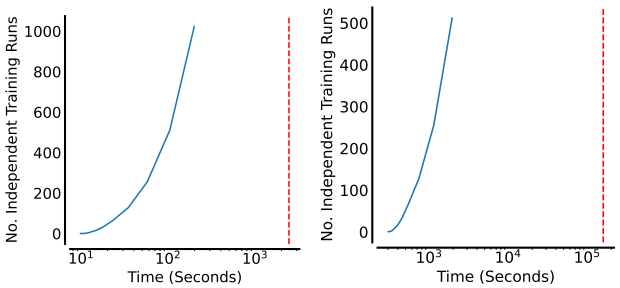
QMIX improves upon VDN by relaxing the full factorization requirement. It ensures that a global $argmax$ operation on the total Q-Value (Q_{tot}) is equivalent to individual $argmax$ operations on each agent's Q-Value. This is achieved using a feed-forward neural network as the mixing network, which 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`

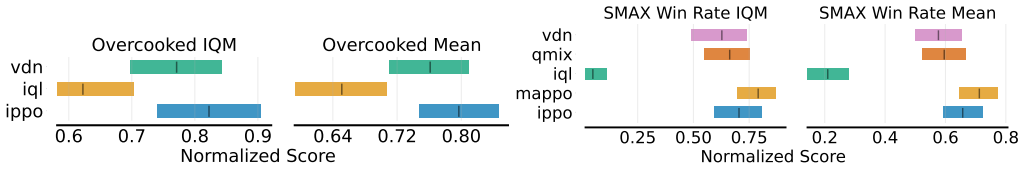
Figure 11: Time taken to train a varying number of seeds in parallel on the same device for JaxMARL IPPO (in blue) compared to the time taken to train one seed with MARLLIB (shown as the red dashed line)

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.

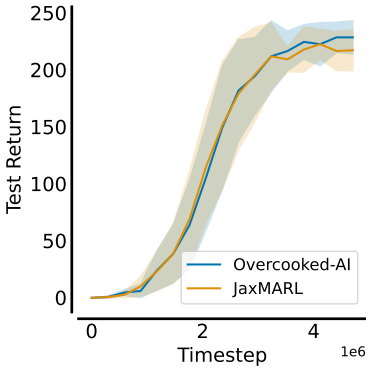


Figure 13: Evaluation performance throughout training of an IPPO policy trained with JaxMARL on our Overcooked Cramped Room scenario implementation and the original [10]. The similarity in performance demonstrates correspondence.

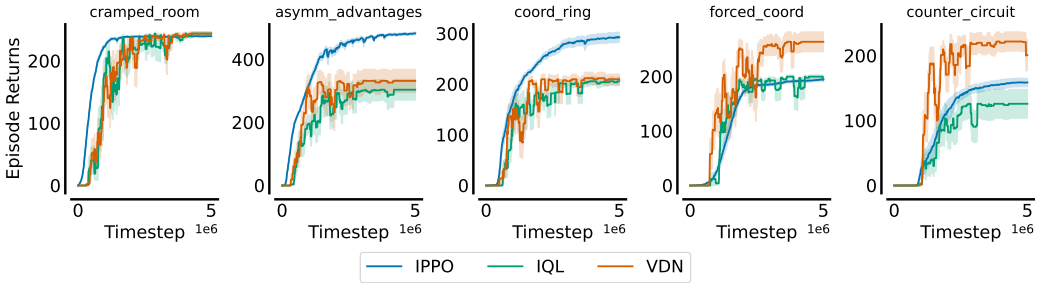


Figure 14: Evaluation of all algorithms in Overcooked scenarios. These scores are obtained after our own hypeparameter tuning, which held to better performances than the using original hyperparameters from Overcooked paper.

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.

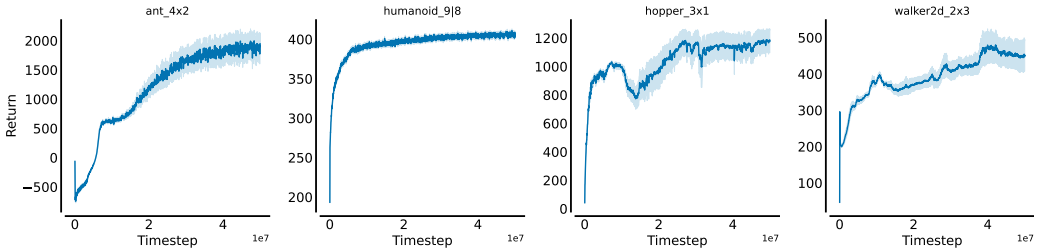


Figure 15: Performance of IPPO on MABrax Tasks

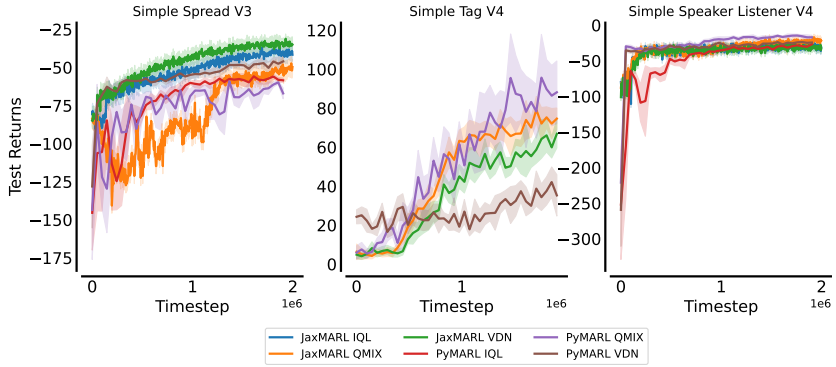


Figure 16: Comparison of the performances of Q-Learning baselines in PyMARL and JaxMARL in two cooperative scenarios of MPE (Spread and Speaker Listener) and one competitive scenario (Simple Tag). For Simple Tag, we pre-trained a prey in JaxMARL and then trained agents to compete with it in both PyMARL 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.

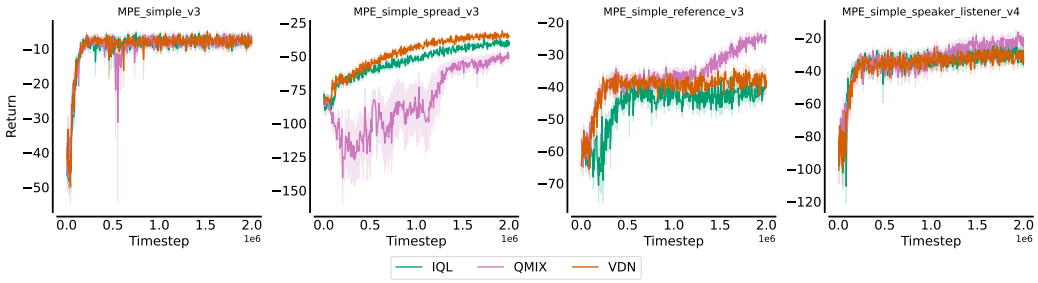


Figure 17: Evaluation of performances of QLearning in all the MPE cooperative scenarios.

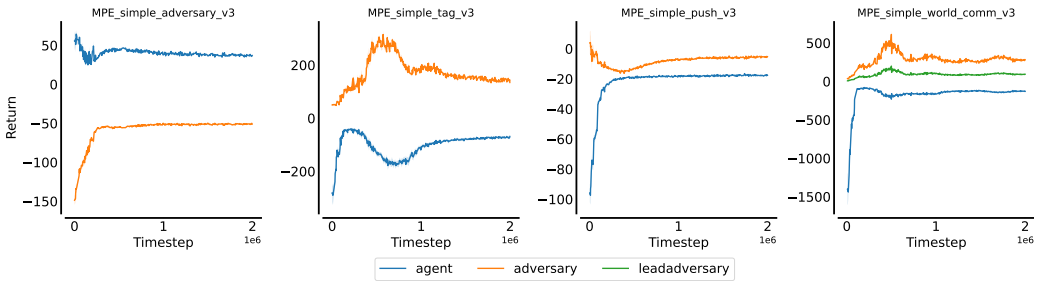


Figure 18: Evaluation of performances of IQL in all the MPE competetive scenarios. All the competetive agents are trained independently together. "Agent" and "Adversary" are teams, not single agents.

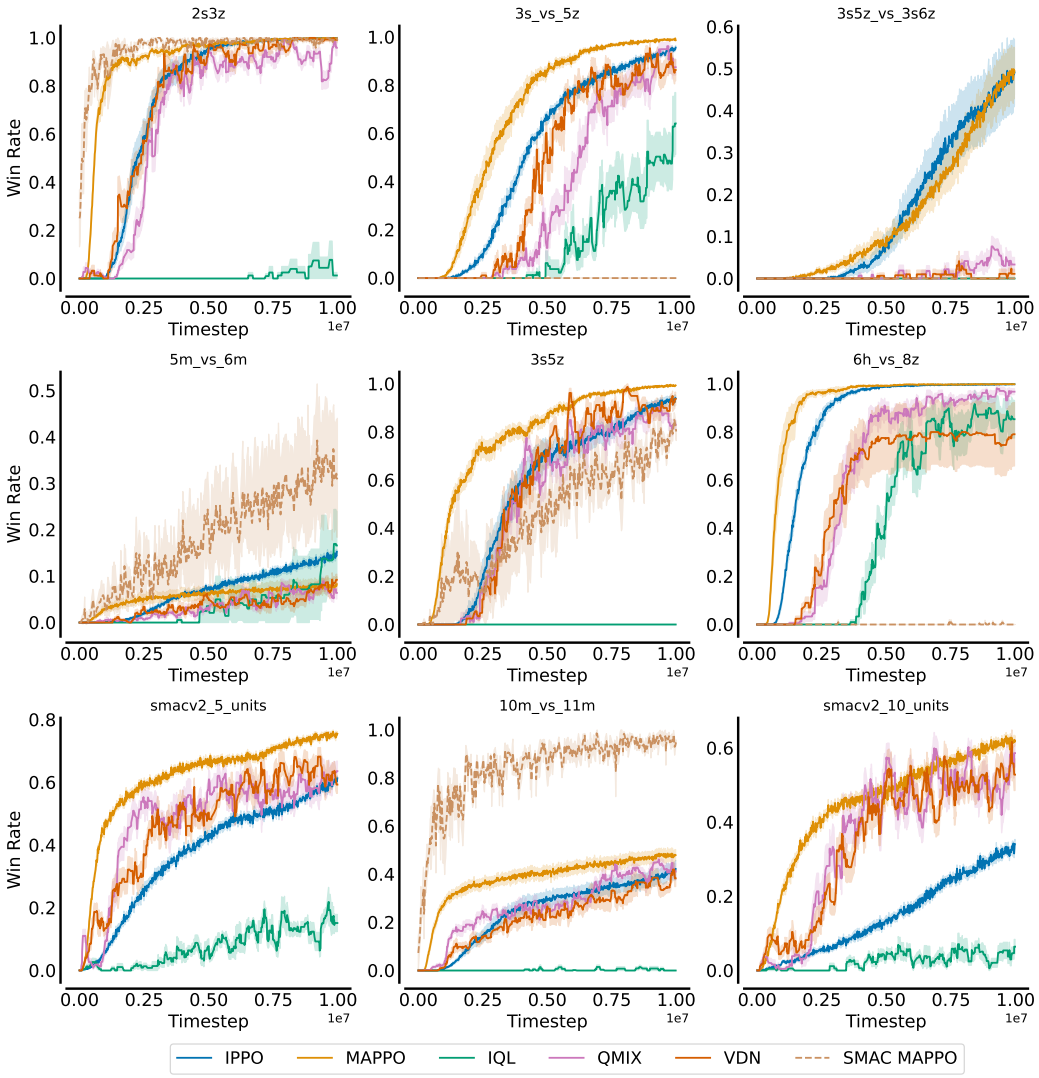


Figure 19: Comparison of IPPO, MAPPO, IQL, QMIX, VDN in SMAX with MAPPO in SMAC.

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}	-

C

Adam-Rel Appendices

A Proof of Theorem 1

Starting from the definition of the momentum term in Adam's update rule:

$$\begin{aligned}
m_t^i &= (1-\beta_1)\sum_{j=-t'}^{t} {\beta_1}^{t-j} g_j^i, \\
&= (1-\beta_1)\left[g\sum_{j=-t'}^{-1} {\beta_1}^{t-j} + kg\sum_{j=0}^{t} {\beta_1}^{t-j}\right], \\
&= (1-\beta_1){\beta_1}^{t} g\left[\sum_{j=-t'}^{-1} {\beta_1}^{-j} + k\sum_{j=0}^{t} {\beta_1}^{-j}\right], \\
&= (1-\beta_1){\beta_1}^{t} g\left[\beta_1\sum_{j=0}^{t'-1} {\beta_1}^{j} + k\sum_{j=0}^{t} \left(\beta_1^{-1}\right)^{j}\right].
\end{aligned}$$

From the solution to the sum of a geometric series:

$$\begin{aligned}
m_t^i &= (1-\beta_1){\beta_1}^{t} g\left[\beta_1\frac{1-{\beta_1}^{t'}}{1-\beta_1} + k\frac{1-{\beta_1}^{-(t+1)}}{1-{\beta_1}^{-1}}\right], \\
&= (1-\beta_1){\beta_1}^{t} g\left[\beta_1\frac{1-{\beta_1}^{t'}}{1-\beta_1} + k\frac{{\beta_1}^{-t}-\beta_1}{1-\beta_1}\right], \\
&= g\left[{\beta_1}^{t+1}(1-{\beta_1}^{t'}) + k(1-{\beta_1}^{t+1})\right].
\end{aligned}$$

Similarly for v_t^i, it follows:

$$v_t = g^2\left[{\beta_2}^{t+1}(1-{\beta_2}^{t'}) + k^2(1-{\beta_2}^{t+1})\right].$$

Substituting v_t^i and m_t^i into the Adam momentum updates with $\epsilon = 0$ yields:

$$\begin{aligned}
\frac{\hat{m}^i_{-t',t}}{\sqrt{\hat{v}^i_{-t',t}}} =& \frac{\sqrt{1-{\beta_2}^{t'+t+1}}}{1-{\beta_1}^{t'+t+1}} \\
&\cdot \frac{g\left[{\beta_1}^{t+1}(1-{\beta_1}^{t'}) + k(1-{\beta_1}^{t+1})\right]}{\sqrt{g^2\left[{\beta_2}^{t+1}(1-{\beta_2}^{t'}) + k^2(1-{\beta_2}^{t+1})\right]}}.
\end{aligned}$$

Taking the limit $t' \to \infty$ with $\beta_1, \beta_2 \in [0, 1)$ yields our desired result:

$$\lim_{t'\to\infty} \frac{\hat{m}^i_{-t',t}}{\sqrt{\hat{v}^i_{-t',t}}} = \frac{{\beta_1}^{t+1} + k(1-{\beta_1}^{t+1})}{\sqrt{{\beta_2}^{t+1} + k^2(1-{\beta_2}^{t+1})}}.$$

B Results comparison with Asadi et al.

Asadi et al. [10] find in their paper that their method, when applied to DQN, gives roughly comparable performance to their Adam baseline. However, in our paper we find that Adam-MR performs significantly worse than the Adam baseline, even when compared on the same games as in Figure 6. There Adam-Rel performs better than Adam on the inter-quartile mean, but worse on the median. However, given this is a selection of just 12 games of very different difficulties, the median is often likely in this case to reduce to a single game for most algorithms.

To investigate this disparity, we compare our results for Adam-MR to theirs in Table 1. We estimated their scores in each game from the appropriate figures in their paper. Overall we see that our

Table 1: Comparison with the results from Asadi et al. [10]. The scores are estimated by taking the performance at 40M frames from the figures in their paper. We compare to both $K = 1000$, which is our default hyperparameter, and $K = 8000$, which is their default hyperparameter.

	Score			Normalized Score		
Environment	Adam-MR (K=1000) [10]	Adam-MR(K=8000) [10]	Adam-Rel (K=1000)	Adam-MR (K=1000) [10]	Adam-MR (K=8000) [10]	Adam-Rel (K=1000)
Amidar	350	300	270 ± 20	0.20	0.17	0.16 ± 0.01
Asterix	3500	4200	3600 ± 700	0.39	0.48	0.40 ± 0.09
BeamRider	3800	4300	4800 ± 500	0.21	0.24	0.27 ± 0.03
Breakout	160	200	300 ± 20	5.5	6.9	10.5 ± 0.7
CrazyClimber	0	85000	80000 ± 9000	-0.41	2.85	2.6 ± 0.3
DemonAttack	3300	3500	8400 ± 500	1.73	1.84	4.5 ± 0.3
Gopher	3500	4000	1500 ± 300	1.50	1.74	0.6 ± 0.1
Hero	1500	6000	1200 ± 600	0.015	0.17	0.005 ± 0.02
Kangaroo	10500	8250	6000 ± 900	3.5	2.75	2.0 ± 0.3
Phoenix	4250	4500	3800 ± 1000	0.54	0.58	0.5 ± 0.2
Seaquest	1300	6000	1800 ± 300	0.03	0.14	0.042 ± 0.006
Zaxxon	1000	6200	2200 ± 300	0.11	0.67	0.24 ± 0.04
Mean				1.11	1.54	1.81 ± 0.2
Inter-Quartile Mean[3]				0.49	0.92	0.66
Median				0.30	0.63	0.43

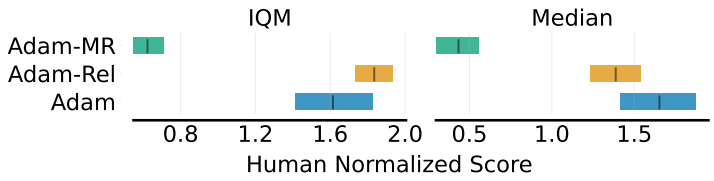


Figure 6: Comparison of the inter-quartile mean and median of Adam-MR, Adam-Rel and Adam on the Atari games evaluated on in Asadi et al. [10].

implementation, which uses $K = 1000$, performs significantly better than their implementation with $K = 1000$. It is also better in mean but worse in median and inter-quartile mean than their $K = 8000$ implementation. In short, our results broadly match theirs reported after a similar amount of training, but our Adam baseline performs significantly better than theirs. However, there are a number of differences in our evaluation. First we run for 10M steps (40M frames) whereas they run for 30M steps (120M frames). Secondly, they use the Dopamine [50] settings for Atari, whereas we use the more standard ones used by DQN [18]. We kept these settings throughout our paper to avoid significant hyperparameter tuning by evaluating in as standard settings as possible. We believe these results demonstrate the correctness of our implementation of their work and that our method still performs favourably.

C Results with Polyak Averaging

We also run experiments on DQN with Polyak averaging to examine the effect of Adam-Rel in cases where there are soft-target updates.

We set $\tau = 0.02$. This was chosen so that after 250 steps, the previous target update frequency, the original target parameters would contribute just 0.5% to the new target. We found the best learning rate to be lower, at 5×10^{-5}. The results are shown in Figure 7. As shown in that figure, Adam-Rel outperforms Adam in this setting as well, achieving a higher median value, although still retaining a long tail of negative results. We also note that although Adam achieves better performance than the baseline without Polyak averaging, Adam-Rel performs worse than when Polyak averaging is not used. This may be due to resetting t being less effective when soft-target changes are used, or that more extensive tuning may improve its performance.

D Code Repositories

For the Atari experiments (both DQN and PPO), we based our implementation on CleanRL [19]. This code is available here. For the Craftax experiments, we based our implementation on PureJaxRL [20]. This code is available here.

[3]This is the inter-quartile mean over *environments* as opposed to the more usual over *environments and seeds*. This is because Asadi et al. [10] do not provide individual seed data.

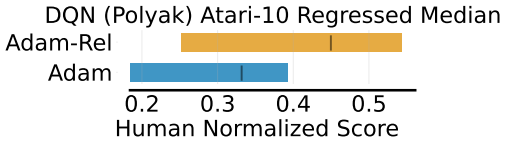


Figure 7: Comparison of the regressed median on the Atari-10 benchmark of Adam and Adam-Rel for DQN with Polyak Averaging.

E Compute and Additional Experiments

For our DQN experiments, we swept over learning rates for Adam-MR, Adam-Rel and Adam. For PPO experiments, we swept over learning rate, max gradient norm and GAE λ values, as we found these to differ from the PPO defaults. Experiments were performed on an internal cluster of NVIDIA V100 GPUs. Experiments were scheduled using slurm, with 10 CPU cores per GPU.

The Atari PPO experiments required around 10000 GPU hours to complete, including hyperparameter tuning. The DQN experiments, because of the computational inefficiency of DQN, take much longer to run (approximately 2 days per experiment), and hence used a total of 14000 GPU hours, despite there being many fewer parallel runs. The Craftax-Classic experiments took around 300 GPU hours to complete.

F Hyperparameters

Table 2: Atari Adam PPO hyperparameters

Hyperparameter	Value
Learning Rate	0.00025
Number of Epochs	4
Minibatches	4
γ	0.99
GAE λ	0.95
Normalise Advantages	`True`
ϵ	0.1
Value Function Clipping	`True`
Max Grad Norm	0.5
Number of Environments	8
Number of Rollout Steps	128

Table 3: Atari Adam-Rel and Adam-MR PPO hyperparameters

Hyperparameter	Value
Learning Rate	0.002
Number of Epochs	4
Minibatches	4
γ	0.99
GAE λ	0.9
Normalise Advantages	`True`
ϵ	0.1
Value Function Clipping	`True`
Max Grad Norm	5.0
Number of Environments	8
Number of Rollout Steps	128

Table 4: Atari-10 DQN hyperparameters

Hyperparameter	Value
Learning Rate	0.0001
Buffer Size	1×10^6
γ	0.99
GAE λ	0.9
Target Network Update Steps	1000
Batch Size	32
Start ϵ	1
End ϵ	0.01
Exploration Fraction	0.1
Number of Steps without Training	80000
Train Frequency	4

Table 5: Craftax Adam and Adam-MR PPO hyperparameters

Hyperparameter	Value
Learning Rate	0.0003
Number of Epochs	4
Minibatches	4
γ	0.99
GAE λ	0.9
Normalise Advantages	`True`
ϵ	0.2
Value Function Clipping	`True`
Max Grad Norm	1
Number of Environments	512
Number of Rollout Steps	64

Table 6: Craftax Adam-Rel hyperparameters

Hyperparameter	Value
Learning Rate	0.001
Number of Epochs	4
Minibatches	4
γ	0.99
GAE λ	0.7
Normalise Advantages	`True`
ϵ	0.2
Value Function Clipping	`True`
Max Grad Norm	5
Number of Environments	512
Number of Rollout Steps	64