

Reinforcement Learning at the Hyperscale

Jakob N. Foerster

Associate Professor



Foerster
Lab for AI
Research



St John's
College
University of Oxford

UNIVERSITY OF
OXFORD

Outline

- The Hyperscale Revolution of Deep RL
- Unlock #1: Faster, diverse Eval
- Unlock #2: Theory Inspired, simpler RL
- Unlock #3: Theory Inspired, scalable Meta-RL
- Conclusions and Open Problems



A tale of two revolutions: #1 The Deep Learning Revolution

AlexNet

文 A 10 languages ▾

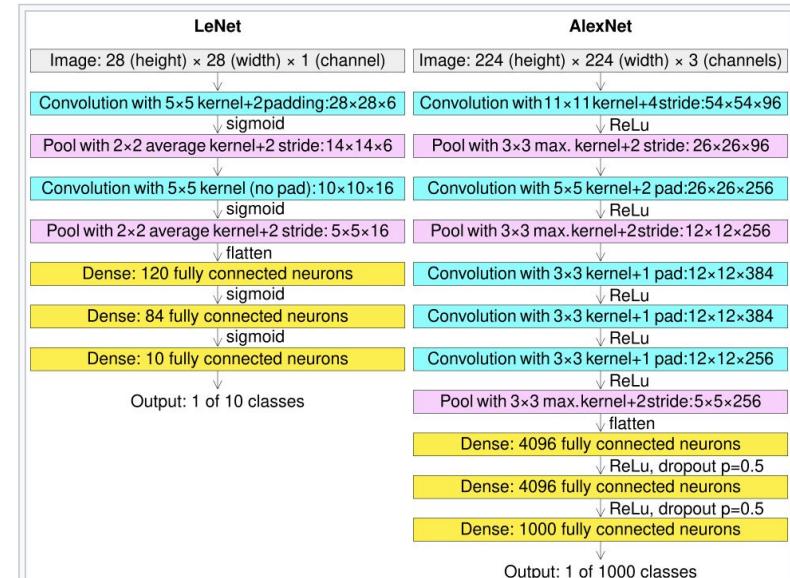
Article Talk

Read Edit View history Tools ▾

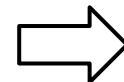
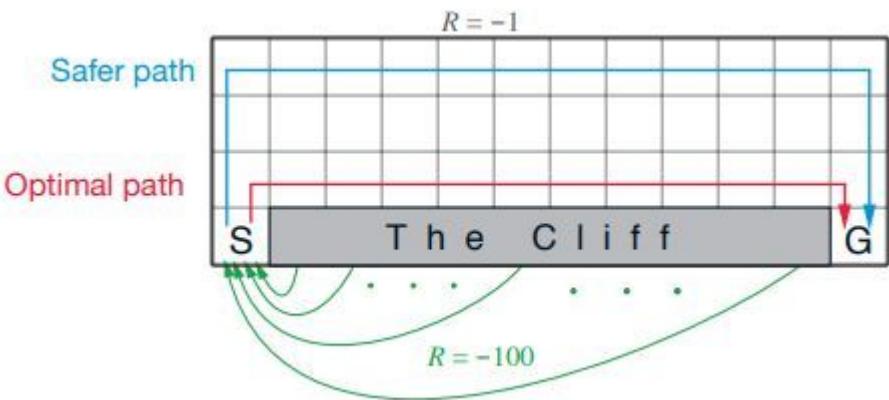
From Wikipedia, the free encyclopedia

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by [Alex Krizhevsky](#) in collaboration with [Ilya Sutskever](#) and [Geoffrey Hinton](#), who was Krizhevsky's Ph.D. advisor at the University of Toronto.^{[1][2]}

AlexNet competed in the [ImageNet Large Scale Visual Recognition Challenge](#) on September 30, 2012.^[3] The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of [graphics processing units \(GPUs\)](#) during training.^[2]



2: The Deep RL Revolution..



“Reinforcement Learning – An Introduction”,
Sutton and Barto

“Playing Atari with Deep Reinforcement Learning”,
Mnih et al, NIPS Deep Learning Workshop 2013



Volodymyr Mnih

FOLLOW

DeepMind

Verified email at cs.toronto.edu - [Homepage](#)

Machine Learning

Looking back 12+
years later...

TITLE

CITED BY

YEAR

[Human-level control through deep reinforcement learning](#)

30723 2015

V Mnih, K Kavukcuoglu, D Silver, AA Rusu, J Veness, MG Bellemare, ...

Nature 518 (7540), 529-533

[Playing Atari with Deep Reinforcement Learning](#)

15140 2013

V Mnih, K Kavukcuoglu, D Silver, A Graves, I Antonoglou, D Wierstra, ...

arXiv preprint arXiv:1312.5602



Alex Krizhevsky

FOLLOW

[University of Toronto](#)

Verified email at cs.toronto.edu - [Homepage](#)

Machine Learning

TITLE

CITED BY YEAR

[Imagenet classification with deep convolutional neural networks](#)

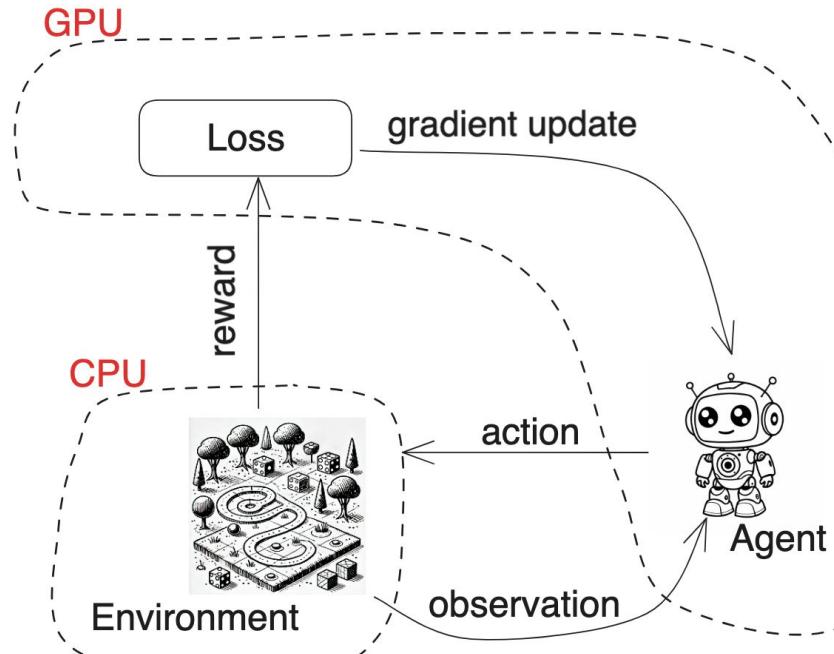
158869* 2012

A Krizhevsky, I Sutskever, GE Hinton

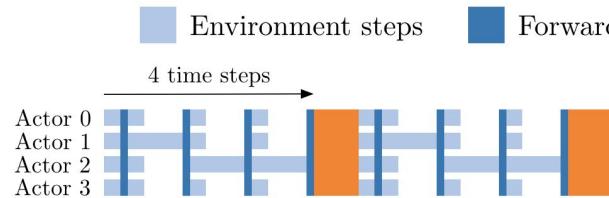
Advances in neural information processing systems 25

RL's Computational Inefficiency Problems

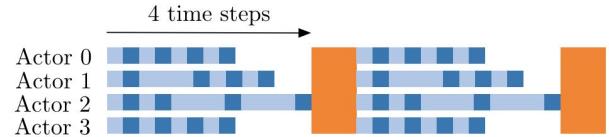
RL typically requires policy rollouts to happen in environments written for the CPU



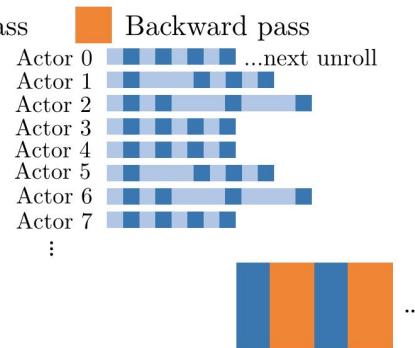
From Computational Inefficiency to Algorithmic Complexity:



(a) Batched A2C (sync step.)



(b) Batched A2C (sync traj.)



(c) IMPALA

- Difficult to keep the GPU “busy”
- Data gets stale
- Algorithmic and Engineering Complexity

“IMPALA: Importance Weighted Actor-Learner Architectures”, L Espeholt et al

What else does this mean? Example: Hanabi!



What else does this mean? Example: Hanabi!

Slow: “..Training a Rainbow agent for the *sample limited regime* of 100 million steps took approximately **seven days** using an NVIDIA V100 GPU.” [1]

- 165 steps / second on an NVIDIA V100

Expensive: “.. agents for 20 billion steps. We estimate the computation took 100 CPU years for a population size of 30..” [1]

- 190 steps / CPU second

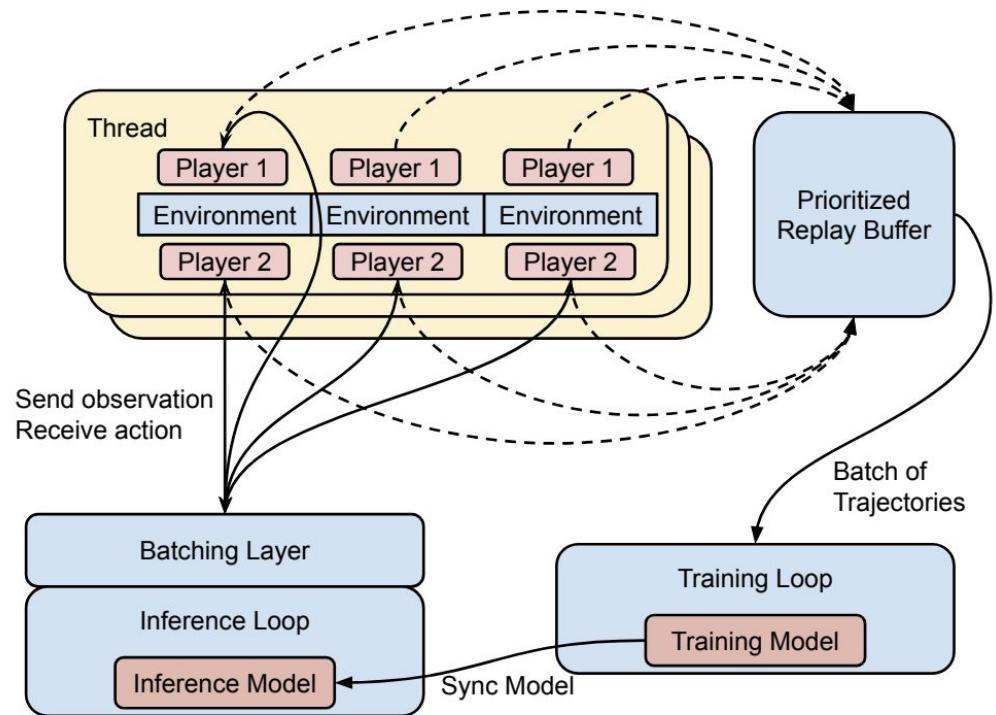
1: “The Hanabi Challenge: A New Frontier for AI Research”, Bardt et al, <https://arxiv.org/pdf/1902.00506.pdf>



What else does this mean? Example: Hanabi!

Hyper-engineered: requiring C++ for env and training loop with complicated thread handling

However, this achieves roughly 12.000 SPS on 3 GPUs.



Take-Away:

Deep RL had lost the “Hardware Lottery”

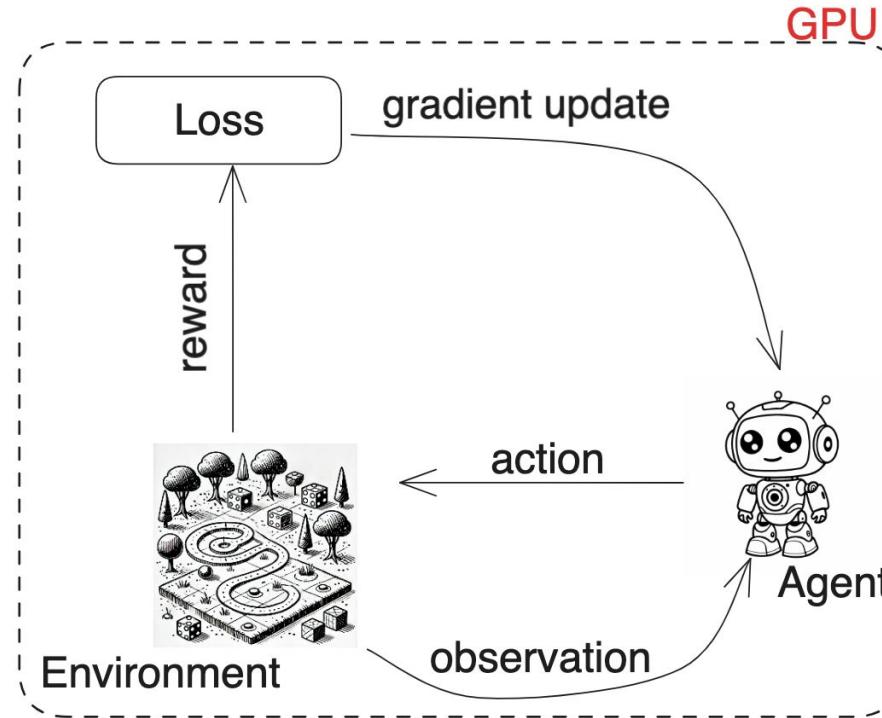


A Path Forward: The Second Coming of the GPU (to deep RL)

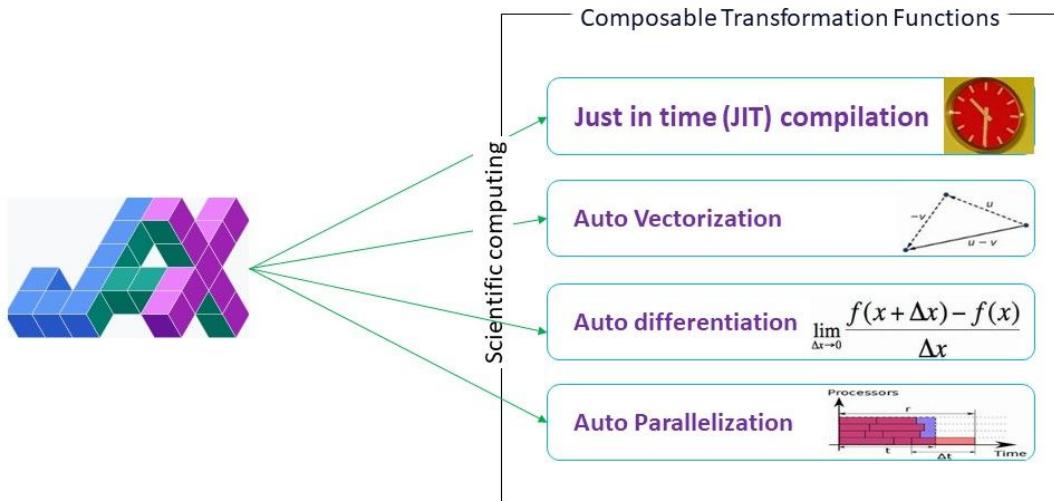


The solution:

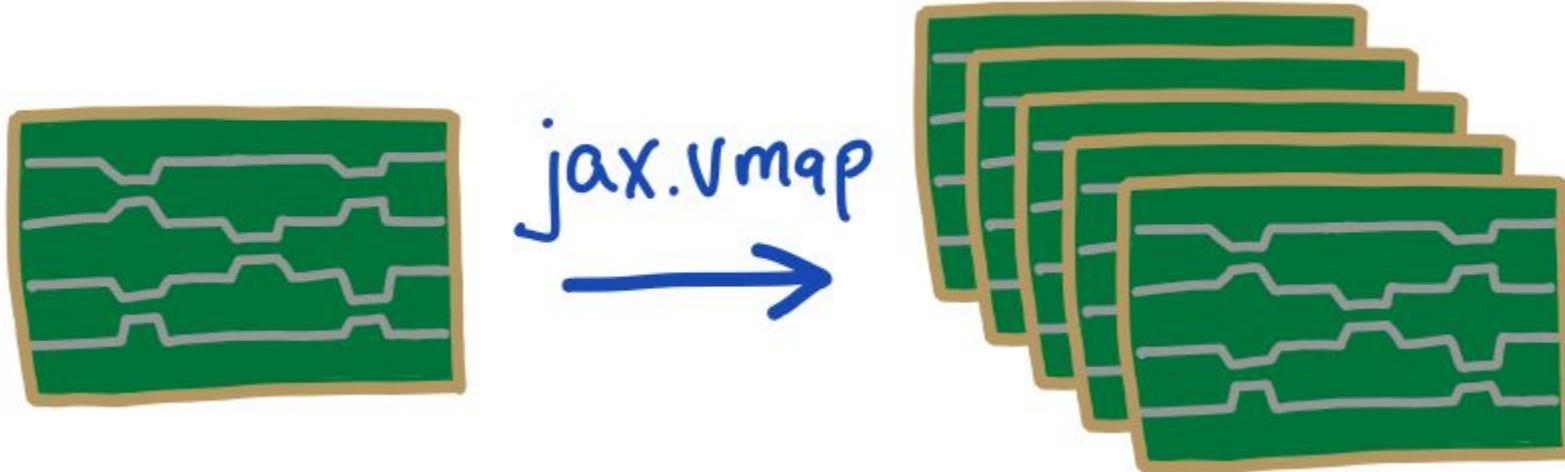
What if we run *everything* on the GPU instead? Wouldn't this be difficult?



Solution: Jax!



Solution: Jax vmap!



Scaling Up: PureJaxRL

**NORMAL
RL TRAINING**



**MULTIPLE
ENVIRONMENT
THREADS ON CPUS**

**VECTORIZED
ENVIRONMENTS
ON GPUs**

**END-TO-END
VECTORIZED RL
TRAINING ON GPUs**

Achieving 4000x Speedups and Meta-Evolving Discoveries with PureJaxRL

AUTHORS

Chris Lu

AFFILIATIONS

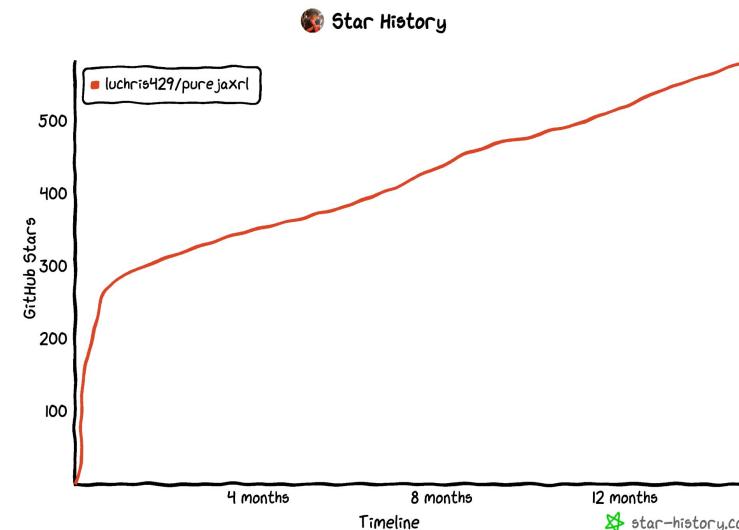
University of Oxford,
Foerster Lab for AI
Research

PUBLISHED

April 3, 2023

FULL PAPER

arXiv



PureJaxRL Speedups

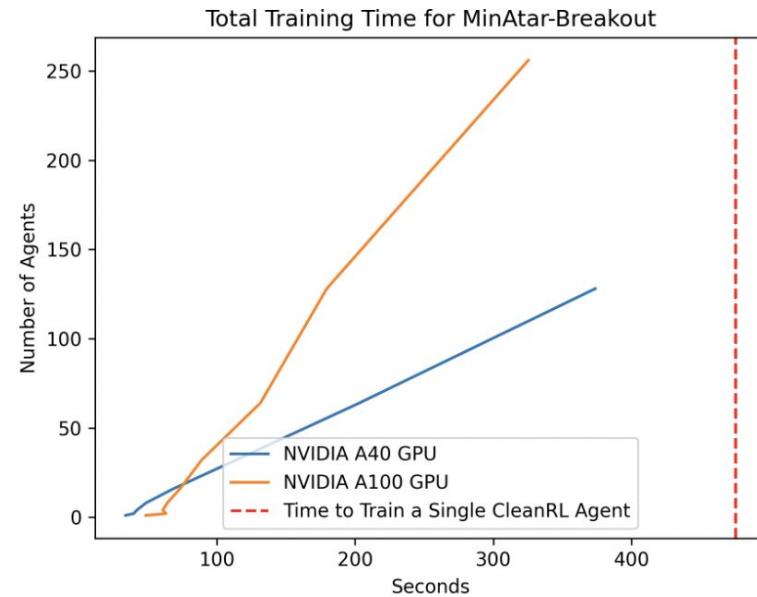
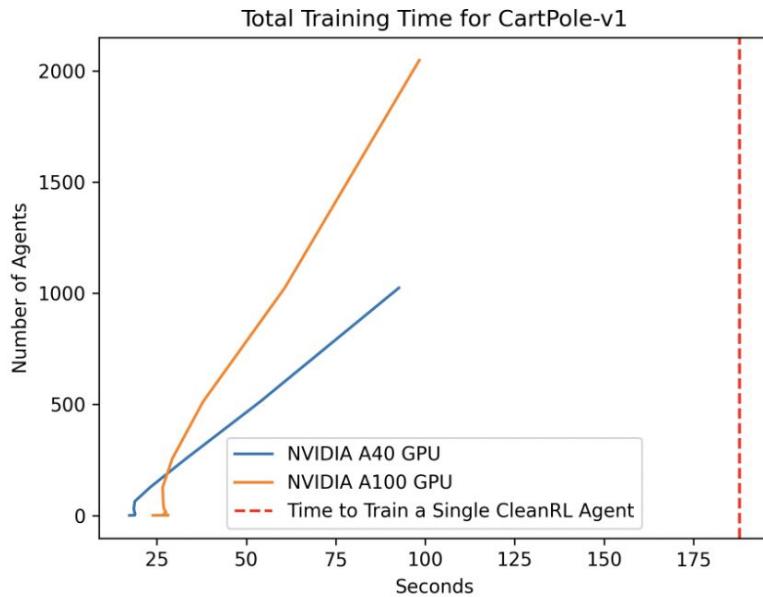


Figure 3: CleanRL vs. Our Jax PPO on CartPole-v1 and MinAtar-Breakout. We can parallelise the agent training itself!

On CartPole-v1 we can train 2048 agents in about half the time it takes to train a single CleanRL agent!



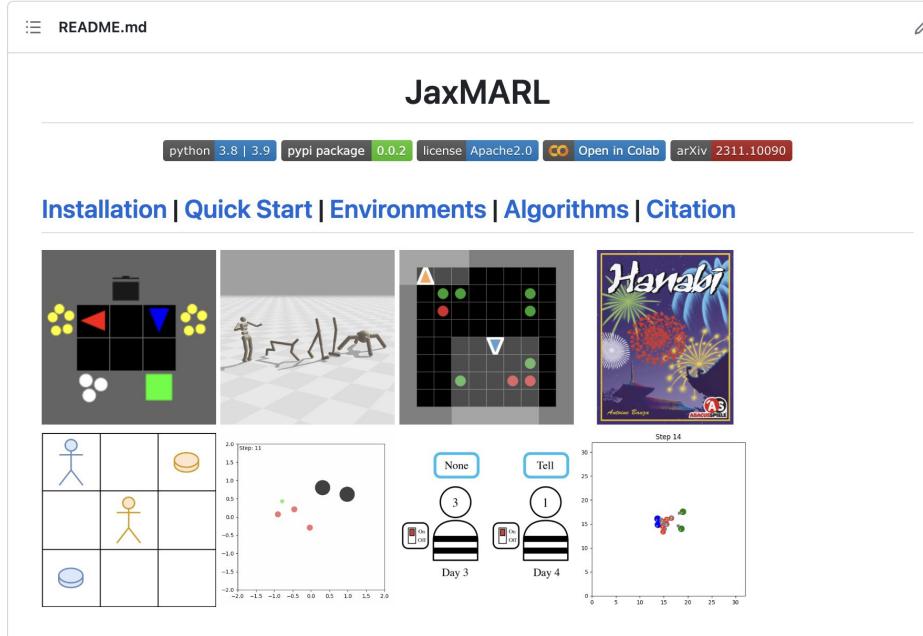
#1:

Faster, diverse Eval

JaxMARL: Multi-Agent RL Environments in JAX

Alexander Rutherford^{1*}†, Benjamin Ellis^{1*}†, Matteo Gallici^{2*}†, Jonathan Cook^{1*}, Andrei Lupu^{1*}, Garðar Ingvarsson^{3*}, Timon Willi^{1*}, Akbir Khan³, Christian Schroeder de Witt¹, Alexandra Souly³, Saptarashmi Bandyopadhyay⁴, Mikayel Samvelyan³, Minqi Jiang³, Robert Tjarko Lange⁵, Shimon Whiteson¹, Bruno Lacerda¹, Nick Hawes¹, Tim Rocktäschel³, Chris Lu^{1*}†, Jakob Nicolaus Foerster¹

¹University of Oxford ²Universitat Politècnica de Catalunya ³UCL ⁴University of Maryland ⁵Technical University Berlin



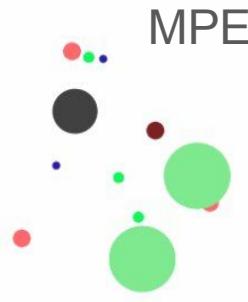
(Slides from
Chris Lu)



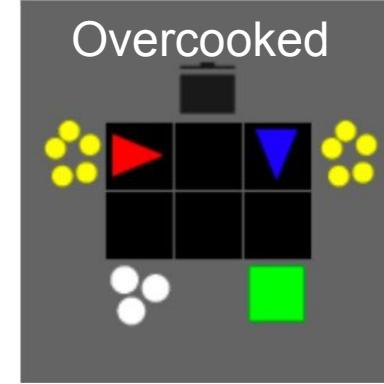
JaxMARL - So far Eight Environment Suites



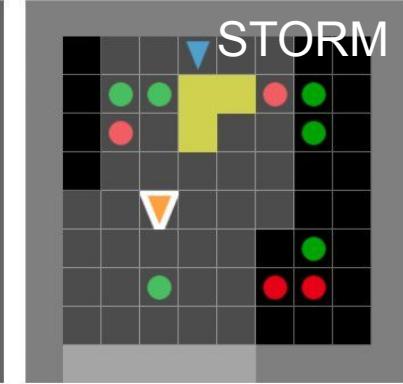
MABrax



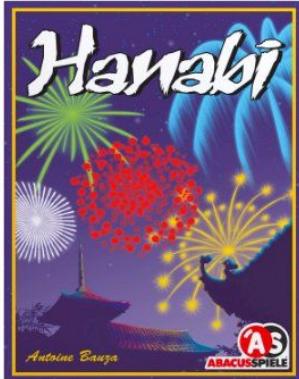
MPE



Overcooked

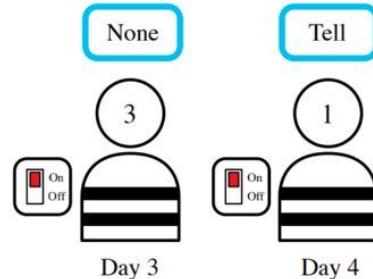


STORM

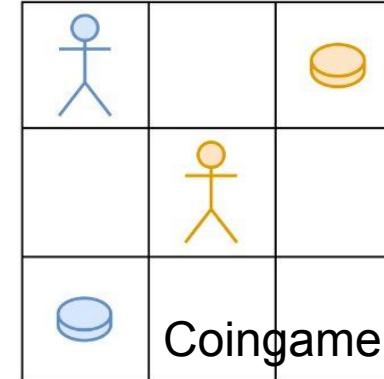


Antoine Bauza

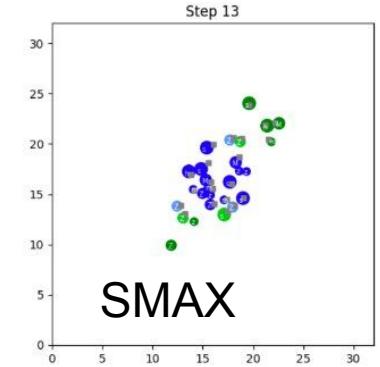
ABACUSPIELE



Prisoner's
Riddle



Coingame



SMAX

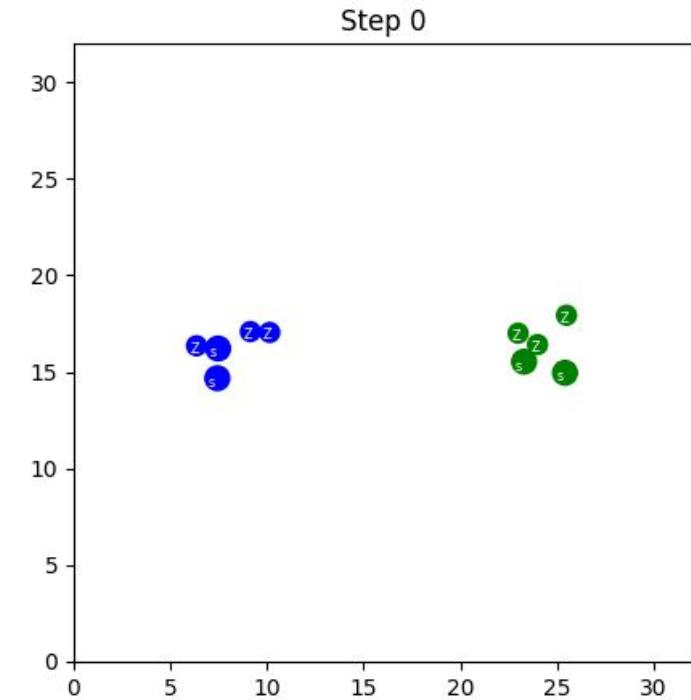
Environment Speedups

Table 2: Benchmark results for JAX-based MARL environments (steps-per-second) when taking random actions. All environments are significantly faster than existing CPU implementations.

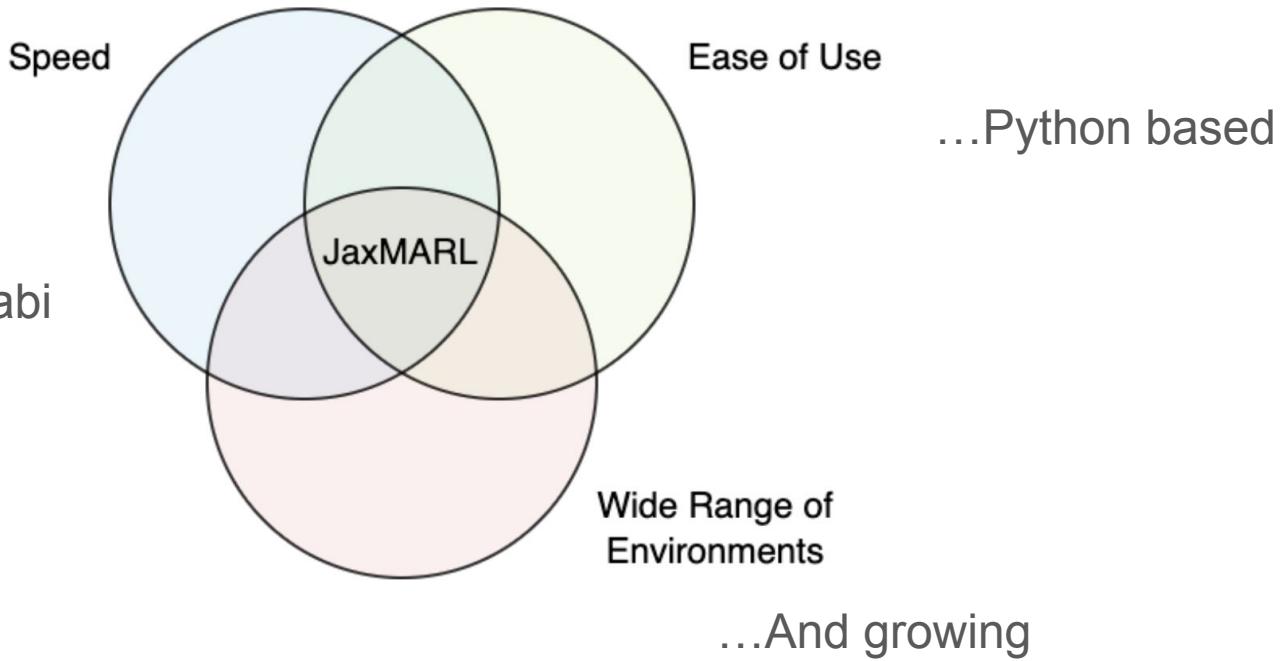
Environment	Original, 1 Env	Jax, 1 Env	Jax, 100 Envs	Jax, 10k Envs	Maximum Speedup
MPE Simple Spread	8.34×10^4	5.48×10^3	5.24×10^5	3.99×10^7	4.78×10^2
MPE Simple Reference	1.46×10^5	5.24×10^3	4.85×10^5	3.35×10^7	2.29×10^2
Switch Riddle	2.69×10^4	6.24×10^3	7.92×10^5	6.68×10^7	2.48×10^3
Hanabi	2.10×10^3	1.36×10^3	1.05×10^5	5.02×10^6	2.39×10^3
Overcooked	1.91×10^3	3.59×10^3	3.04×10^5	1.69×10^7	8.85×10^3
MABrax Ant 4x2	1.77×10^3	2.70×10^2	1.81×10^4	7.62×10^5	4.31×10^2
Starcraft 2s3z	8.31×10^1	5.37×10^2	4.53×10^4	2.71×10^6	3.26×10^4
Starcraft 27m vs 30m	2.73×10^1	1.45×10^2	1.12×10^4	1.90×10^5	6.96×10^3
STORM	—	2.48×10^3	1.75×10^5	1.46×10^7	—
Coin Game	1.97×10^4	4.67×10^3	4.06×10^5	4.03×10^7	2.05×10^3



SMAX



...e.g.
20k SPS on Hanabi



See our repo and blog post for more information

JaxMARL: Multi-Agent RL Environments and Algorithms in JAX

AUTHORS	AFFILIATIONS	PUBLISHED	FULL PAPER
Alex Rutherford, Ben Ellis, Chris Lu	University of Oxford	Nov. 17, 2023	arXiv

Overview

We present JaxMARL, a library of multi-agent reinforcement learning (MARL) environments and algorithms based on end-to-end GPU acceleration that achieves up to 12500x speedups. The environments in JaxMARL span cooperative, competitive, and mixed games; discrete and continuous state and action spaces; and zero-shot and CTDE settings. We specifically include implementations of the Hanabi Learning Environment, Overcooked, Multi-Agent Brax, MPE, Switch Riddle, Coin Game, and Spatial-Temporal Representations of Matrix Games (STORM). **Because of JAX's hardware acceleration, our per-run training pipeline is 12500x faster than existing approaches.** We also introduce **SMAX**, a vectorised version of the popular StarCraft Multi-Agent Challenge, which removes the need to run the StarCraft II game engine. By significantly speeding up training, JaxMARL enables new research into areas such as multi-agent meta-learning, as well as significantly easing and improving evaluation in MARL. Try it out here: <https://github.com/flairox/jaxmarl>!

The screenshot shows the GitHub repository page for JaxMARL. At the top, there are buttons for 'Edit Pins', 'Unwatch' (11), 'Fork' (17), and 'Starred' (192). Below the header, it says 'JaxMARL Public'. It shows 30 branches and 2 tags. A dropdown menu indicates the current branch is 'main'. The commit history table has columns for author, commit message, time, and commits. The commits listed are:

Author	Commit Message	Time	Commits
amacrutherford	overcooked multiple seeds	8665ca2 2 hours ago	448
	.github/workflows Create python-publish.yml	2 weeks ago	
	baselines overcooked multiple seeds	2 hours ago	
	docs/imgs add new doc images	3 days ago	
	jaxmarl Merge branch 'main' into qlearning	last week	
	requirements updates for flashbox	last week	
	tests name change to jaxmarl	3 weeks ago	

On the right side, there are sections for 'About' (Multi-Agent Reinforcement Learning with JAX), 'Readme', 'Apache-2.0 license', 'Activity', '192 stars', '11 watching', '17 forks', and a 'Report repository' button. There is also a small FLAIR logo in the bottom right corner.

JAX-LOB: A GPU-Accelerated Limit Order Book Simulator to Unlock Large Scale Reinforcement Learning for Trading

International Conference on AI in Finance: ICAIF'23
Best Academic Paper

Sascha Frey, Kang Li, Peer Nagy, Silvia Sapora, Chris Lu,
Stefan Zohren, Jakob Foerster, Anisoara Calinescu



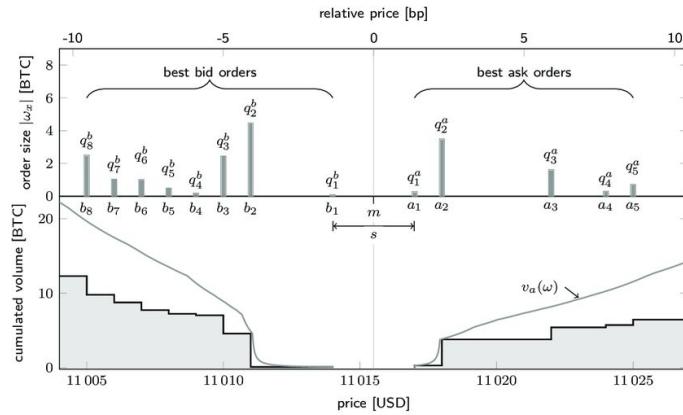
Foerster
Lab for AI
Research



(Slides from
Sasha Frey)

The Financial Markets are a complex multi-agent System

Limit Order Book

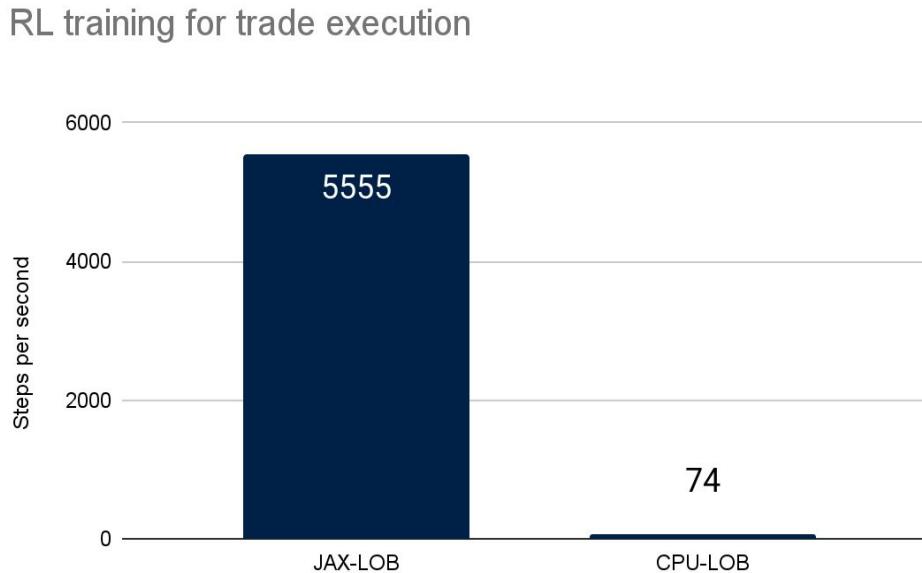


Interesting Emergent Dynamics



JAX-LOB Benefits: 75x Speed-Up Over CPU Equivalent

Automatic parallelisation: major benefit for **RL** since the entire training loop is GPU native. Similar speed-up expected for **Monte Carlo** methods



Craftax: A Lightning-Fast Benchmark for Open-Ended Reinforcement Learning

Michael Matthews¹ Michael Beukman¹ Benjamin Ellis¹ Mikayel Samvelyan² Matthew Jackson¹
Samuel Coward¹ Jakob Foerster¹



Michael Matthews¹



Michael Beukman¹



Benjamin Ellis^{1,2}



Mikayel Samvelyan³



Matthew Jackson^{1,2}



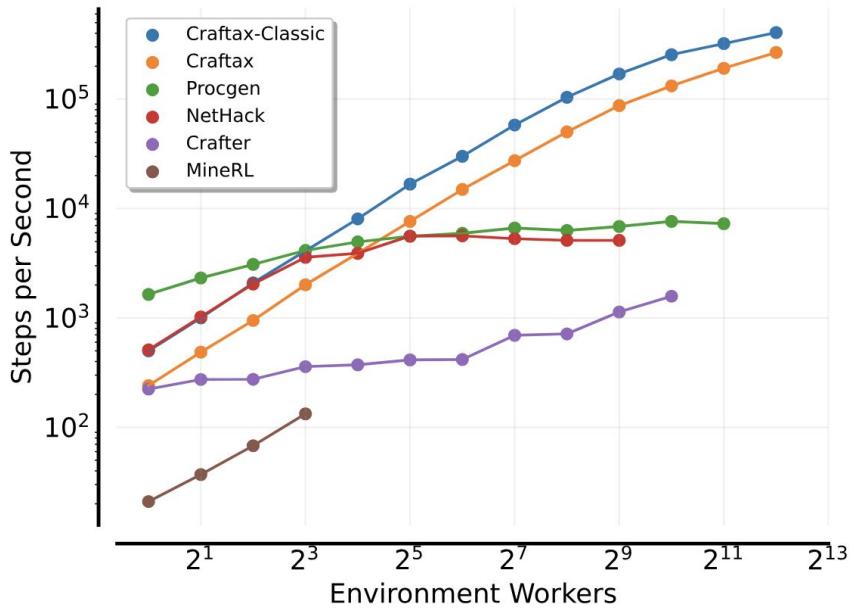
Samuel Coward¹



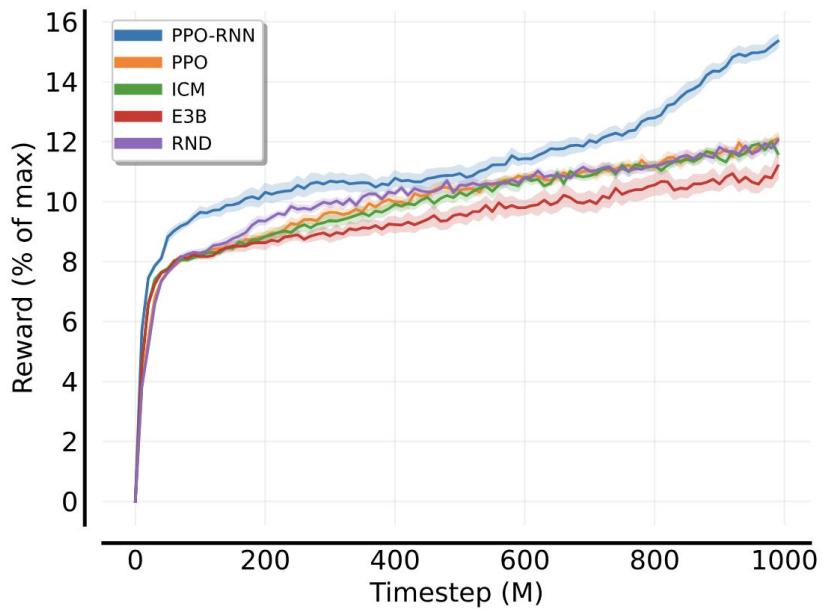
Jakob Foerster¹

¹ [FLAIR](#), University of Oxford ² [WhiRL](#), University of Oxford ³ [DARK](#), University College London

Blazing Fast



Challenging...





#2:

GPU Acceleration + (some)

Theory =

Simplified Reinforcement

Learning

Simplifying Deep Temporal Difference Learning

Matteo Gallici*¹ **Mattie Fellows***²

Benjamin Ellis² **Bartomeu Pou**^{1,3} **Ivan Masmitja**⁴

Jakob Nicolaus Foerster² **Mario Martin**¹

¹Universitat Politècnica de Catalunya ²University of Oxford

³Barcelona Supercomputing Center ⁴ Institut de Ciències del Mar

{gallici,mmartin}@cs.upc.edu

{matthew.fellows,benjamin.ellis,jakob.foerster}@eng.ox.ac.uk

bartomeu.poumulet@bsc.es masmitja@icm.csic.es



Part 1: Analysing and Stabilizing TD....

Why is TD so unstable?

It's not a gradient of anything

How can we analyse TD?

Using the Jacobian

How can we stabilise TD?

+ By introducing and LayerNorm
+ L2 regularisation

Theoretical insights

- 1) Batchnorm layers make the policy myopic:

Theorem 1. *Under Assumption 1, using the BatchNorm Q-function defined in Eq. (3)*
 $\lim_{N \rightarrow \infty} \mathbb{E}_{x_t \sim d^\mu} [\mathcal{B}[Q_\phi](x_t)] = \mathbb{E}_{r \sim P_R(x_t), x_t \sim d^\mu} [r]$ *almost surely.*

- 2) LayerNorm + L2 regularisation instead mitigates Off-Policy Instability:

Theorem: *Under mild regularity assumptions, using the k width LayerNorm Q-function:*

$$\frac{\left| (r(s, a) + \gamma Q_\omega(s', a') - Q_\omega(s, a)) x^\top \nabla_\omega^2 Q_\omega(s, a) x \right|}{\|x\|^2} = \mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$$

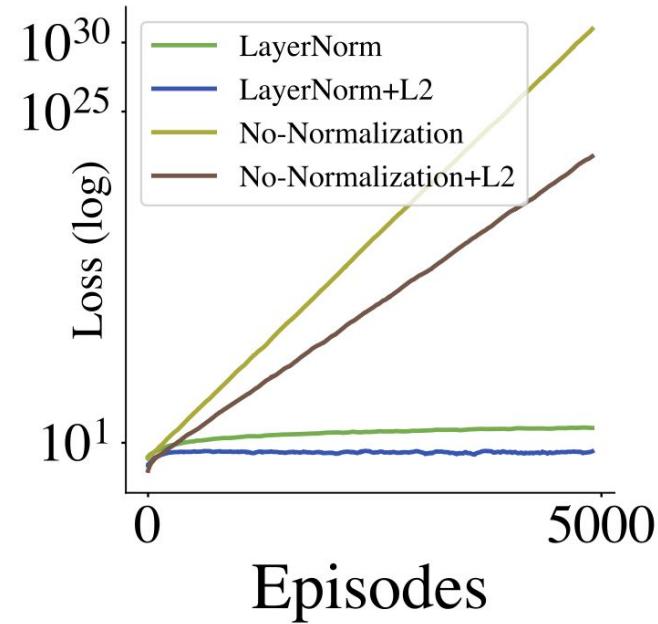
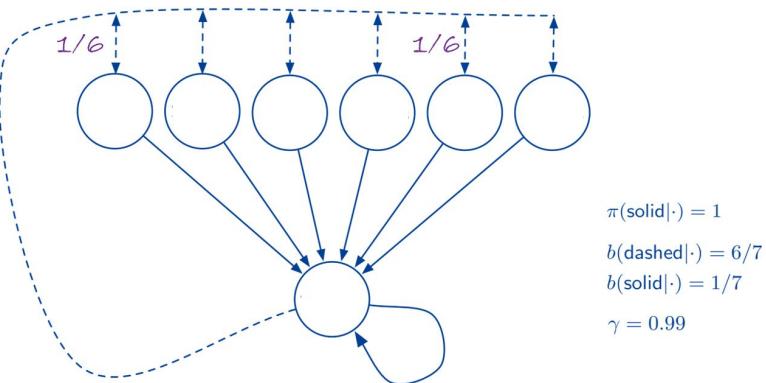
almost surely for a test vector x and any s, a, s', a'

Shrinks with increasing k

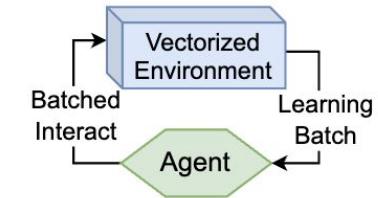
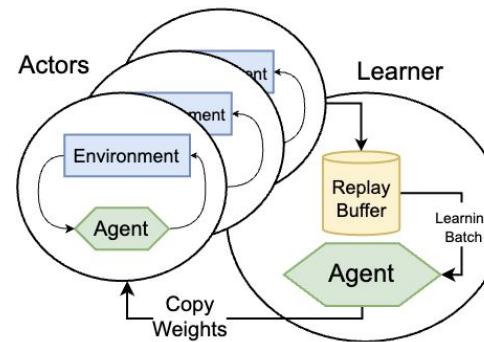
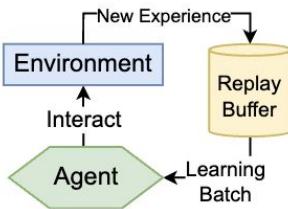
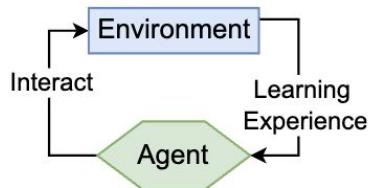


Tabular Validation

Baird's counterexample:
Simple MDP with linear function
approximator and off-policy
sampling



Part 2: Simplifying Temporal Difference Learning



(a) Online Q -Learning

(b) DQN

(c) Distributed DQN

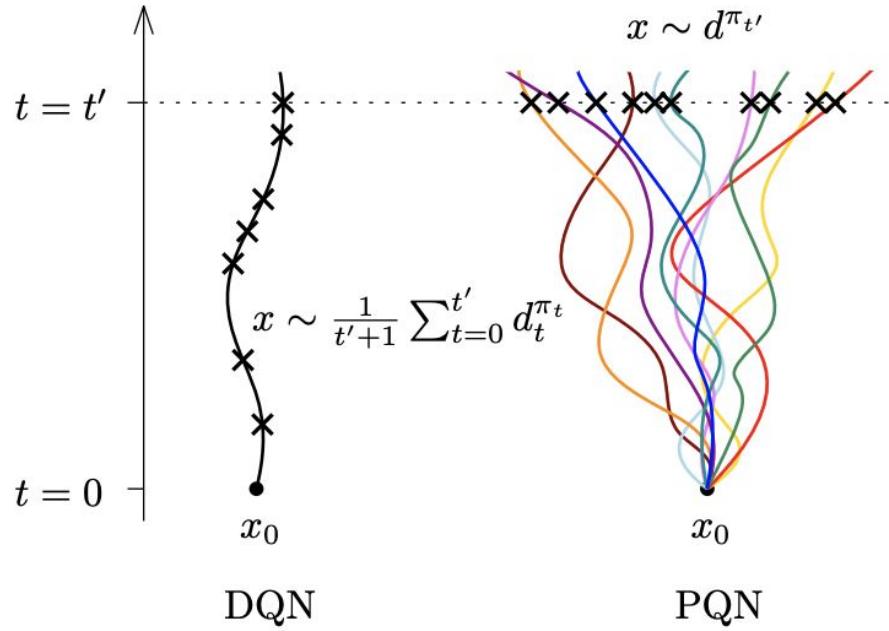
(d) PQN

No Target network, no replay – just TD Learning

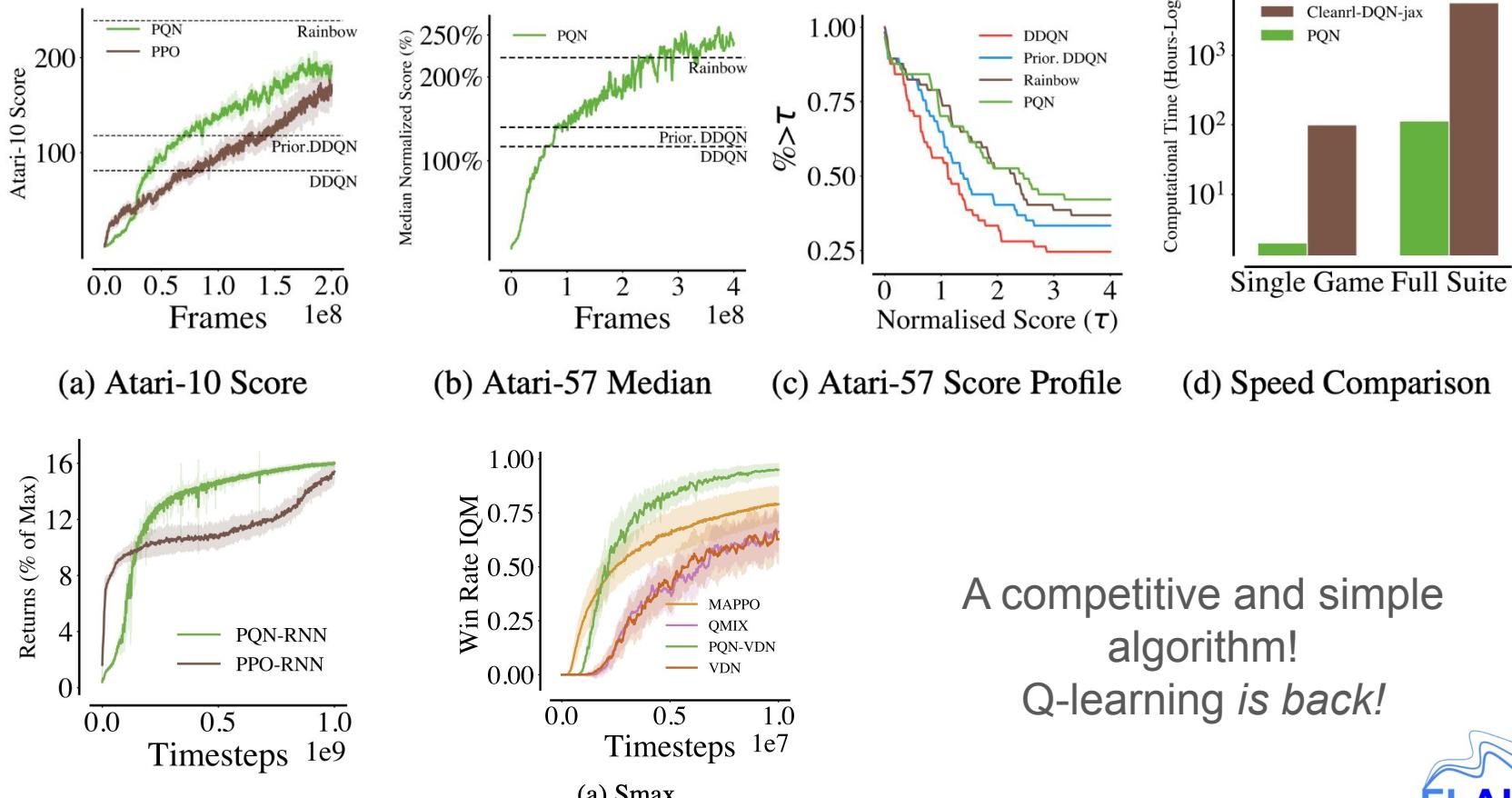
Algorithm 1 PQN ($s, a, s', r, \text{target}$ are B -dim vectors)

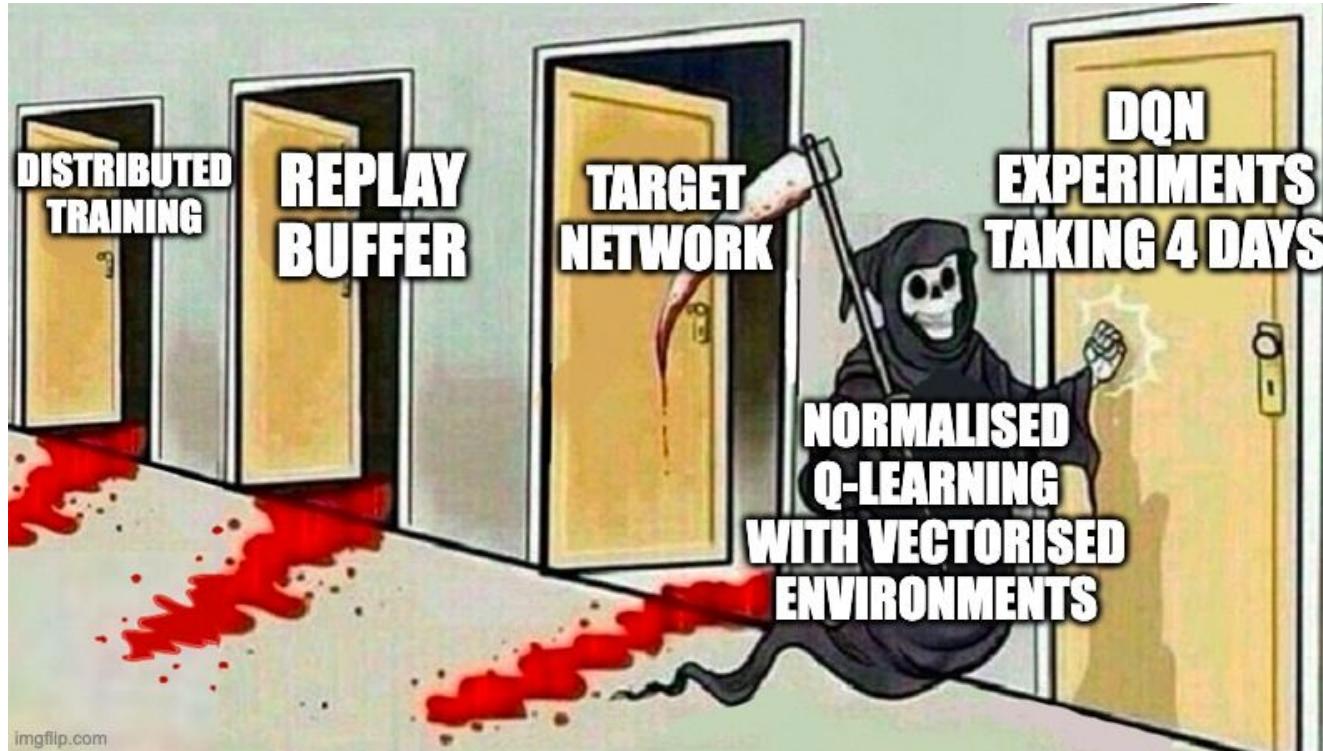
- 1: $\phi \leftarrow$ initialise regularised Q -network parameters
 - 2: $s \leftarrow$ initial state $s_0 \sim P_0$,
 - 3: **for each episode do**
 - 4: **for each $i \in B$ (in parallel) do**
 - 5: $a_i \leftarrow \begin{cases} a_i \sim \text{Unif}(\mathcal{A}), & \text{with prob. } \epsilon, \\ \arg \max_{a'} Q_\phi(s_i, a'), & \text{otherwise,} \end{cases}$
 - 6: $s'_i, r_i \leftarrow s'_i \sim P_S(s_i, a_i), r_i \sim P_R(s_i, a_i)$,
 - 7: $\text{target}_i \leftarrow r_i + \gamma \mathbb{1}(\text{not terminal}) \max_{a'} Q_\phi(s'_i, a')$
 - 8: **end for**
 - 9: $\phi \leftarrow \phi - \alpha \nabla_\phi \| \text{StopGrad[target]} - Q_\phi(s, a) \|^2$
 - 10: **end for**
-

The sampling regime of PQN



Results







#3: GPU Acceleration +
(some) Theory =

Scalable Meta - RL

Mirror Learning Framework (ICML 2022)

Mirror Learning: A Unifying Framework of Policy Optimisation

Jakub Grudzien Kuba¹ Christian Schroeder de Witt¹ Jakob Foerster¹

Mirror Learning: The Update Formula (Simplified)

At every iteration, maximize

$$\mathbb{E}_{a \sim \pi_{\text{new}}} [Q_{\pi_{\text{old}}}(s, a)] - \mathfrak{D}_{\pi_{\text{old}}}(\pi_{\text{new}} | s)$$

Drift Function

Mirror Learning: Properties

Theorem 1 (The Fundamental Theorem of Mirror Learning). *Let \mathfrak{D}^ν be a drift, \mathcal{N} be a neighbourhood operator, and the sampling distribution β_π depend continuously on π . Let $\pi_0 \in \Pi$, and the sequence of policies $(\pi_n)_{n=0}^\infty$ be obtained by mirror learning induced by \mathfrak{D}^ν , \mathcal{N} , and β_π . Then, the learned policies*

1. *Attain the strict monotonic improvement property,*

$$\eta(\pi_{n+1}) \geq \eta(\pi_n) + \mathbb{E}_{s \sim d} \left[\frac{\nu_{\pi_n}^{\pi_{n+1}}(s)}{\beta_{\pi_n}(s)} \mathfrak{D}_{\pi_n}(\pi_{n+1}|s) \right],$$

2. *Their value functions converge to the optimal one,*

$$\lim_{n \rightarrow \infty} V_{\pi_n} = V^*,$$

3. *Their expected returns converge to the optimal return,*

$$\lim_{n \rightarrow \infty} \eta(\pi_n) = \eta^*,$$

4. *Their ω -limit set consists of the optimal policies.*

Space of RL Algorithms

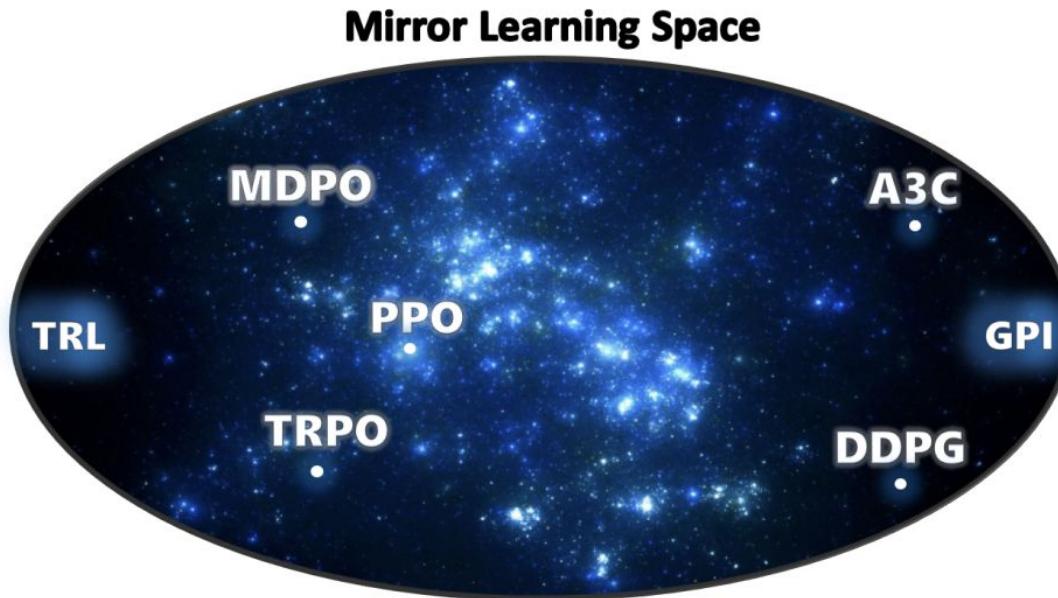


Figure 1. Known RL frameworks and algorithms as points in the infinite space of *theoretically sound* mirror learning algorithms.

Space of RL Algorithms

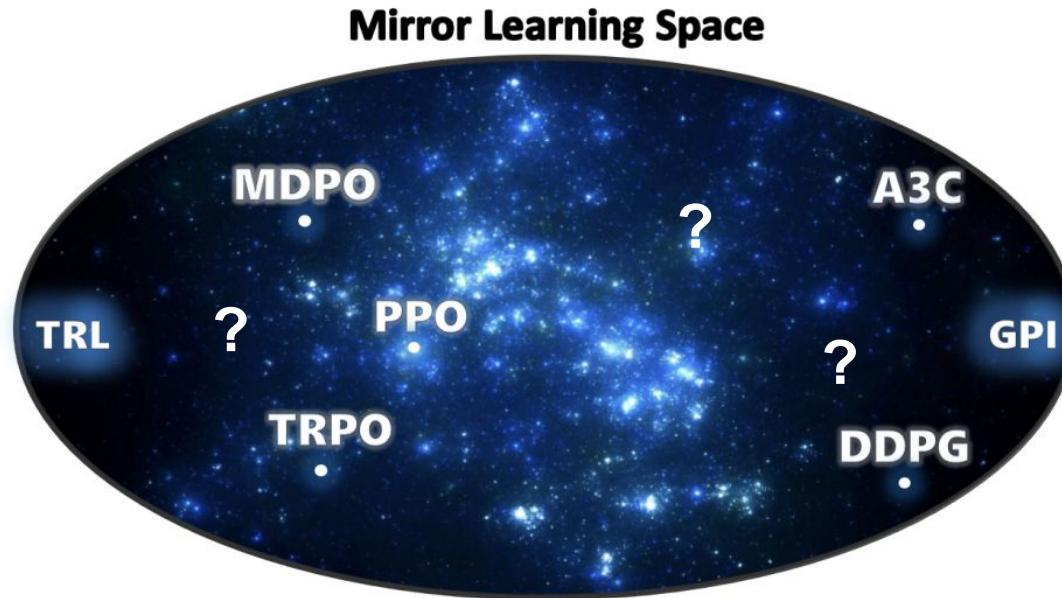


Figure 1. Known RL frameworks and algorithms as points in the infinite space of *theoretically sound* mirror learning algorithms.

Discovered Policy Optimisation

Chris Lu*

FLAIR, University of Oxford
christopher.lu@exeter.ox.ac.uk

Jakub Grudzien Kuba* †

BAIR, UC Berkeley
kuba@berkeley.edu

Alistair Letcher

aletcher.github.io
ahp.letcher@gmail.com

Luke Metz

Google Brain
Luke.s.metz@gmail.com

Christian Schroeder de Witt

FLAIR, University of Oxford
cs@robots.ox.ac.uk

Jakob Foerster

FLAIR, University of Oxford
jakob.foerster@eng.ox.ac.uk

NeurIPS 2022



A Meta Learning Approach

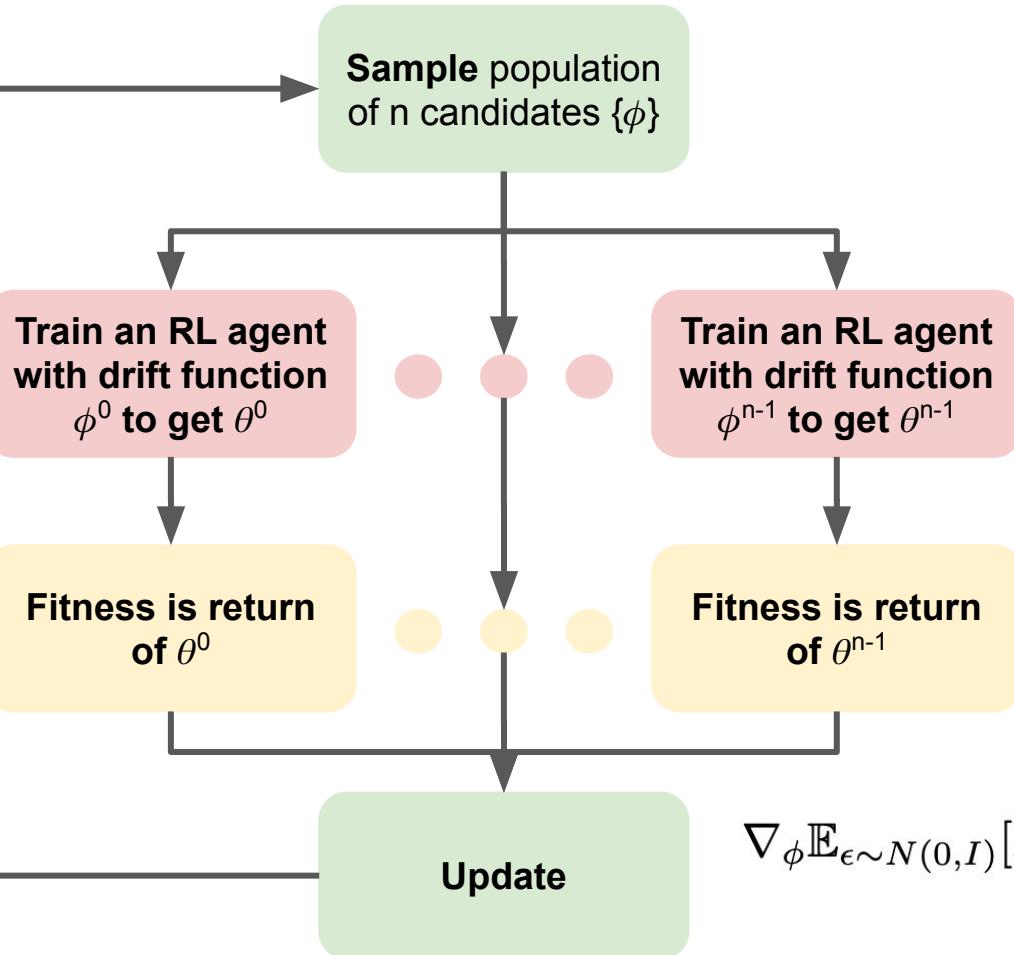
At every iteration, maximize

$$\mathbb{E}_{a \sim \pi_{\text{new}}} [Q_{\pi_{\text{old}}}(s, a)] - \mathfrak{D}_{\pi_{\text{old}}}(\pi_{\text{new}} | s)$$

Drift Function

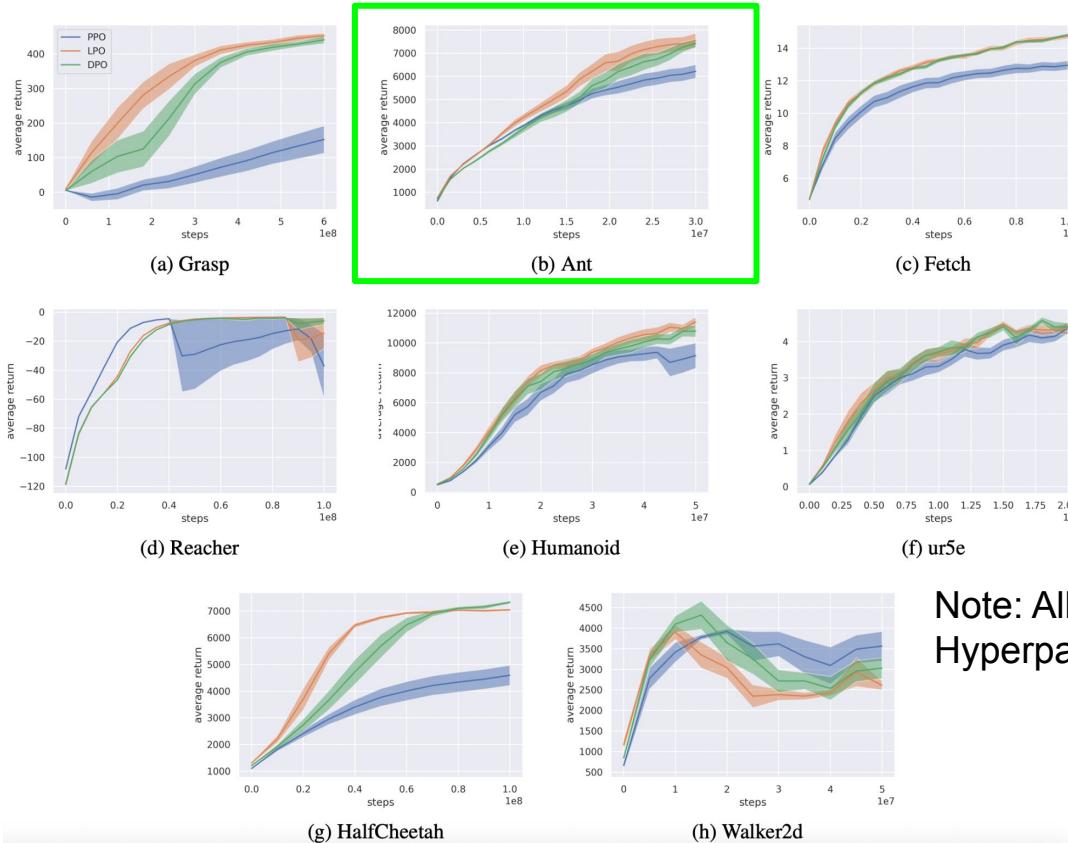
Parameterise as a neural network, ϕ , compliant with theory

How to estimate Meta-Gradients?



$$\nabla_{\phi} \mathbb{E}_{\epsilon \sim N(0, I)} [F(\phi + \sigma \epsilon)] = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0, I)} [F(\phi + \sigma \epsilon) \epsilon],$$

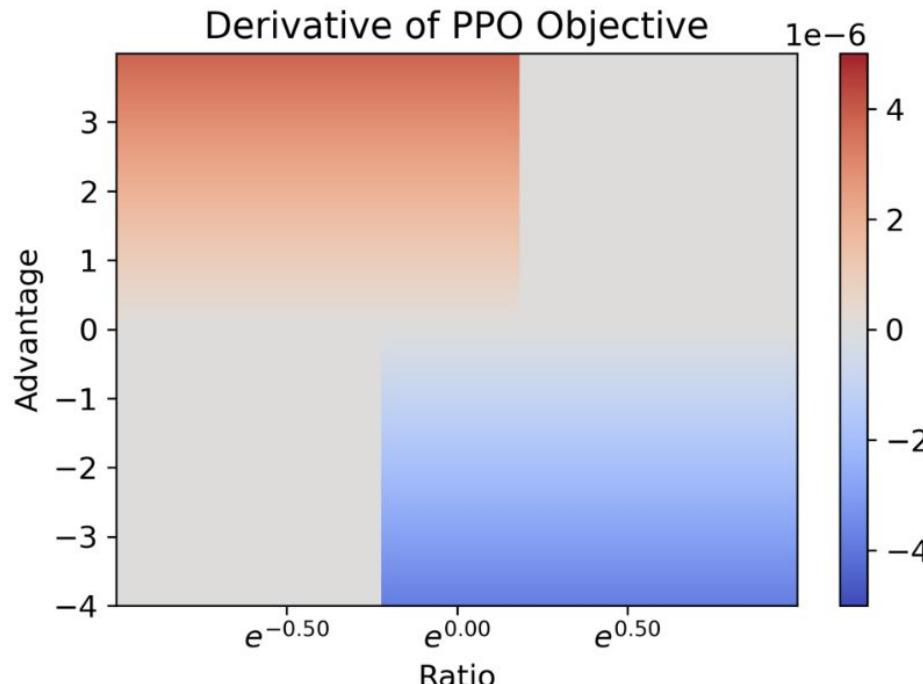
Learned Policy Optimisation (LPO) Performance



Note: All runs use BRAX default Hyperparameters for PPO

Visualizing Objective Functions: PPO

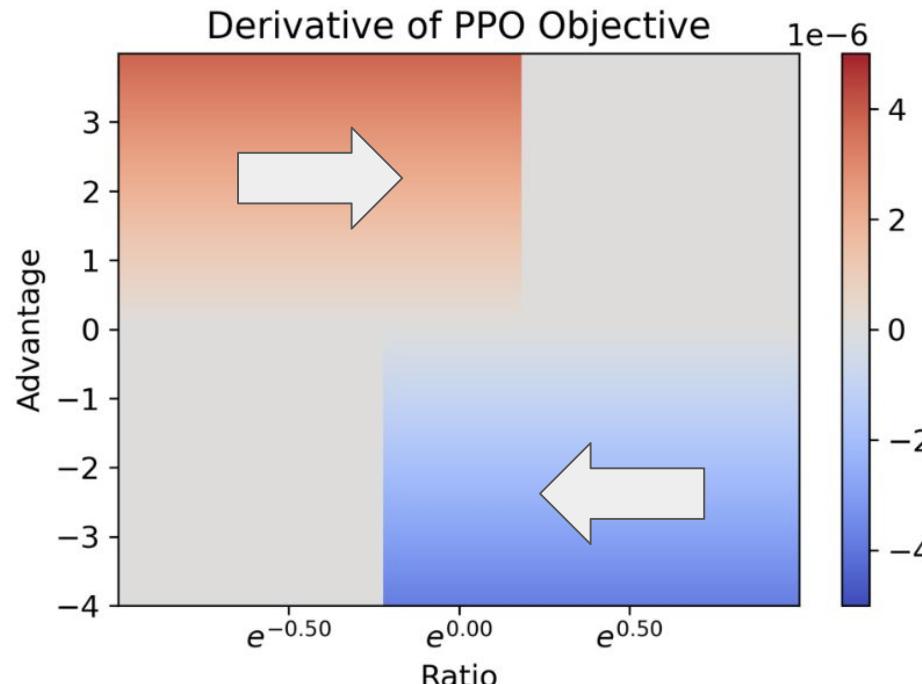
$$A_{\pi}(s, a)$$



$$\mathbf{r}(\bar{\pi}) \triangleq \bar{\pi}(a|s)/\pi(a|s)$$

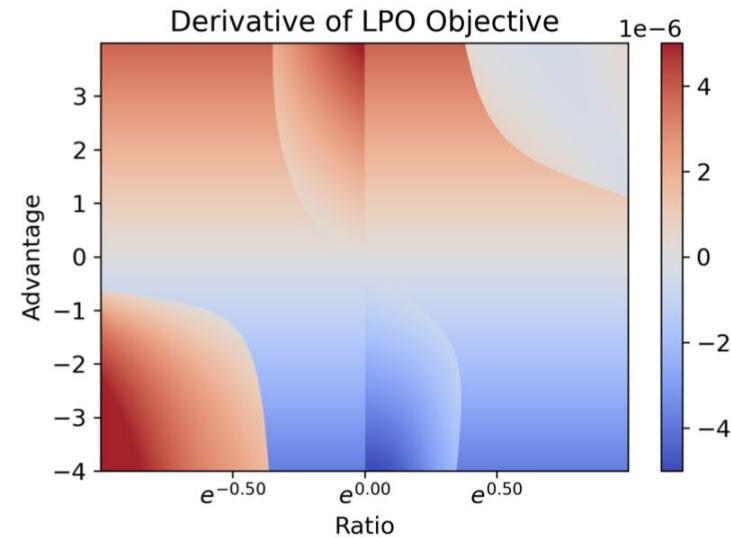
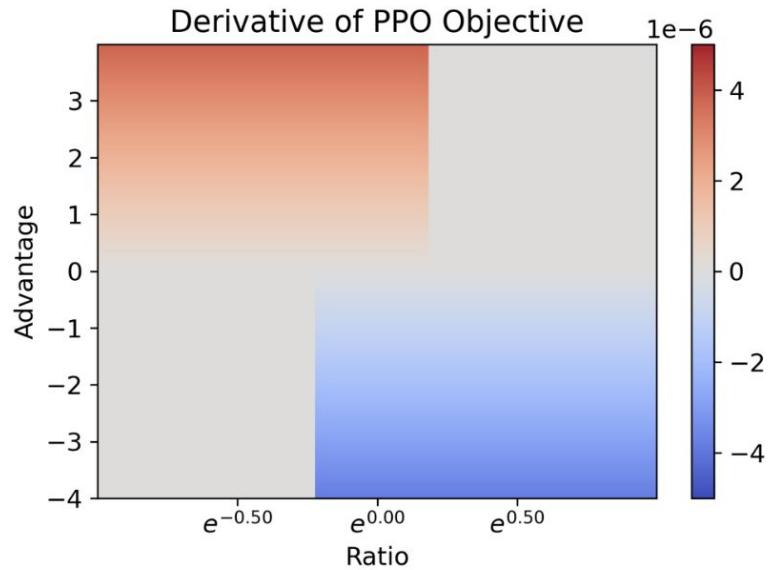
Visualizing Objective Functions: PPO

$$A_{\pi}(s, a)$$

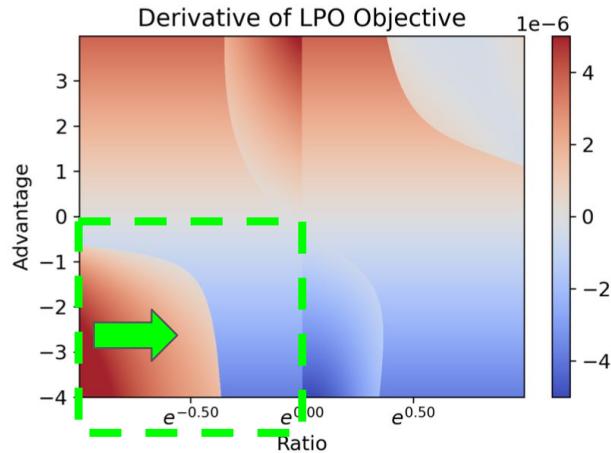


$$\mathbf{r}(\bar{\pi}) \triangleq \bar{\pi}(a|s)/\pi(a|s)$$

Visualizing Objective Functions: LPO

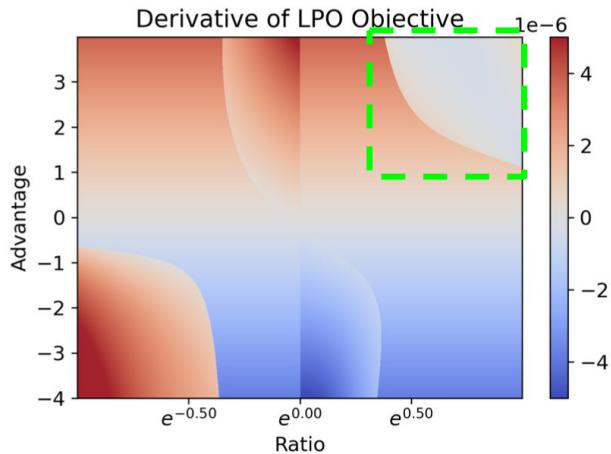


Interpreting LPO



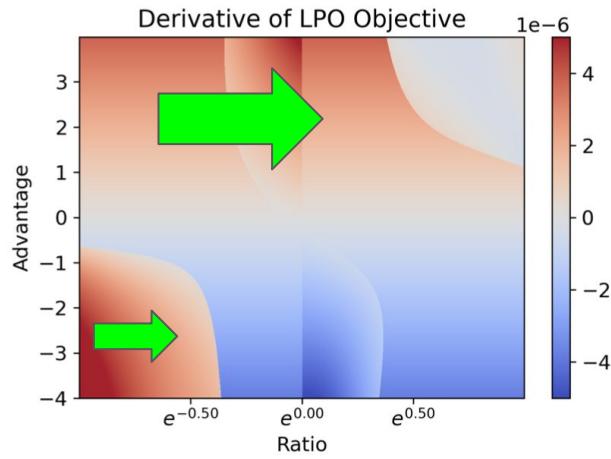
1. “Rollback” for Negative Advantage
2. Cautious Optimism for Positive Advantage
3. Implicit Entropy Maximisation
4. Secondary Features

Interpreting LPO



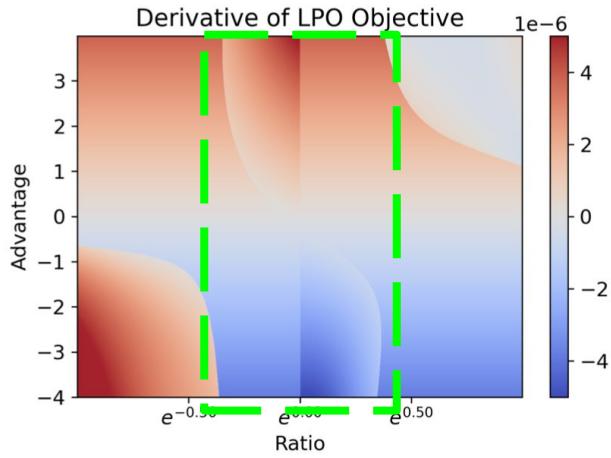
1. Rollback for Negative Advantage
2. Cautious Optimism for Positive Advantage
3. Implicit Entropy Maximisation
4. Secondary Features

Interpreting LPO



1. Rollback for Negative Advantage
2. Cautious Optimism for Positive Advantage
3. Implicit Entropy Maximisation
4. Secondary Features

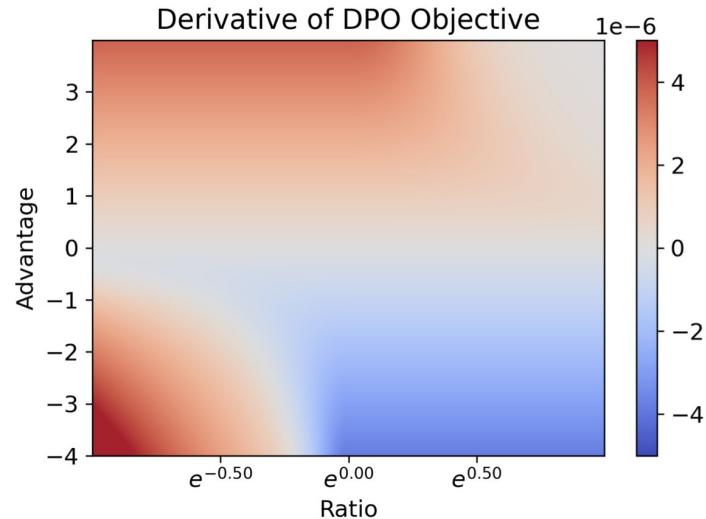
Interpreting LPO



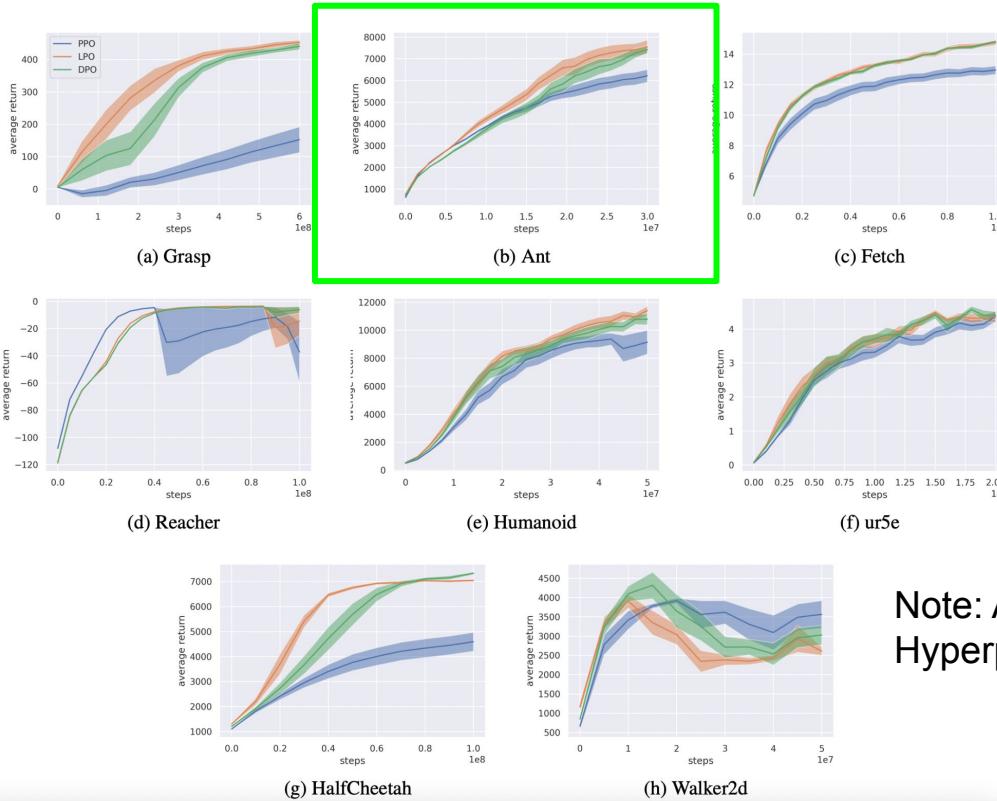
1. Rollback for Negative Advantage
2. Cautious Optimism for Positive Advantage
3. Implicit Entropy Maximisation
4. Secondary Features

Discovered Policy Optimisation (DPO)

$$f(r, A) = \begin{cases} \text{ReLU}\left((r - 1)A - \alpha \tanh\left((r - 1)A/\alpha\right)\right) & A \geq 0 \\ \text{ReLU}\left(\log(r)A - \beta \tanh\left(\log(r)A/\beta\right)\right) & A < 0 \end{cases}$$

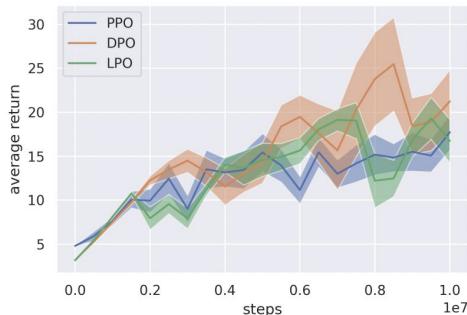


Discovered Policy Optimisation Performance

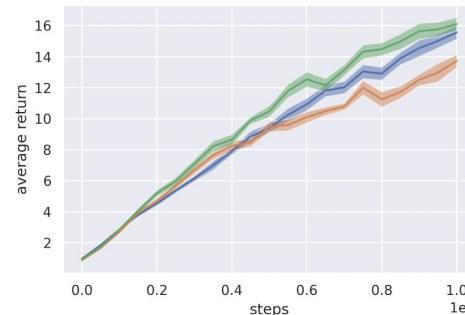


Note: All runs use BRAX default Hyperparameters for PPO

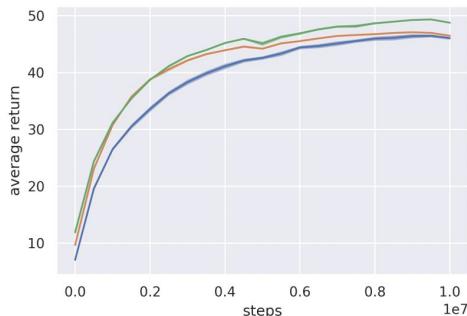
Discovered Policy Optimisation: Far OOD Performance



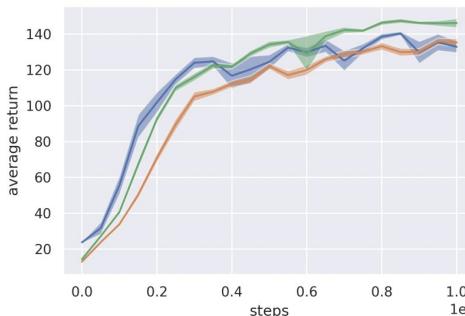
(a) Breakout-Minatar



(b) Asterix-Minatar



(c) Freeway-Minatar



(d) SpaceInvaders-Minatar

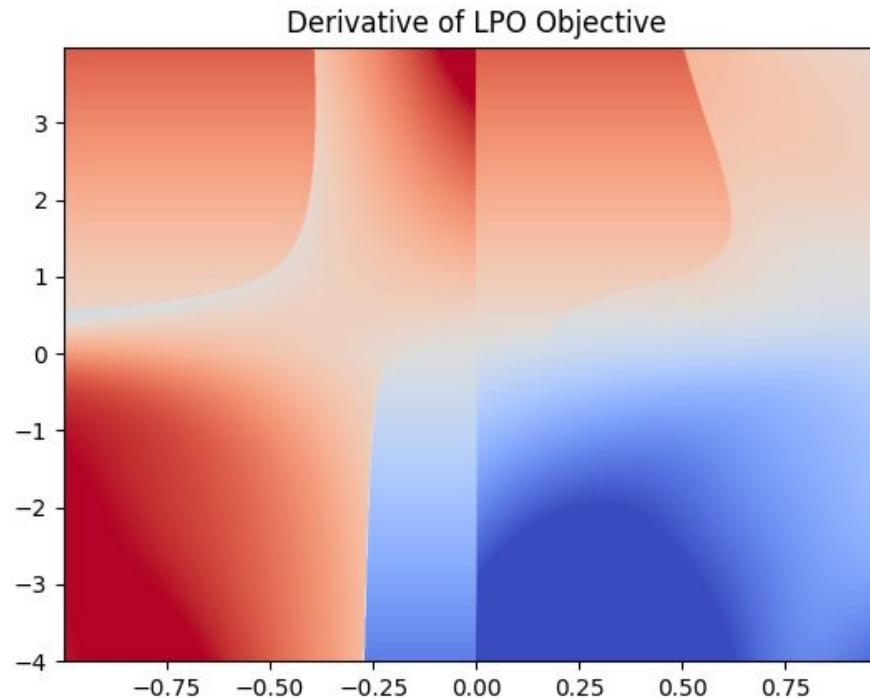
Discovery: DPO

1. Picked ϕ that corresponds to a core part of hand-crafted RL objective functions. Parameterised based on theory

2. Meta-Optimized ϕ using evolution and PureJaxRL

3. Analyzed ϕ to gain insight into policy optimization and create Discovered Policy Optimisation

Temporally-Adaptive LPO (ICLR 2024)



We get much better performance!

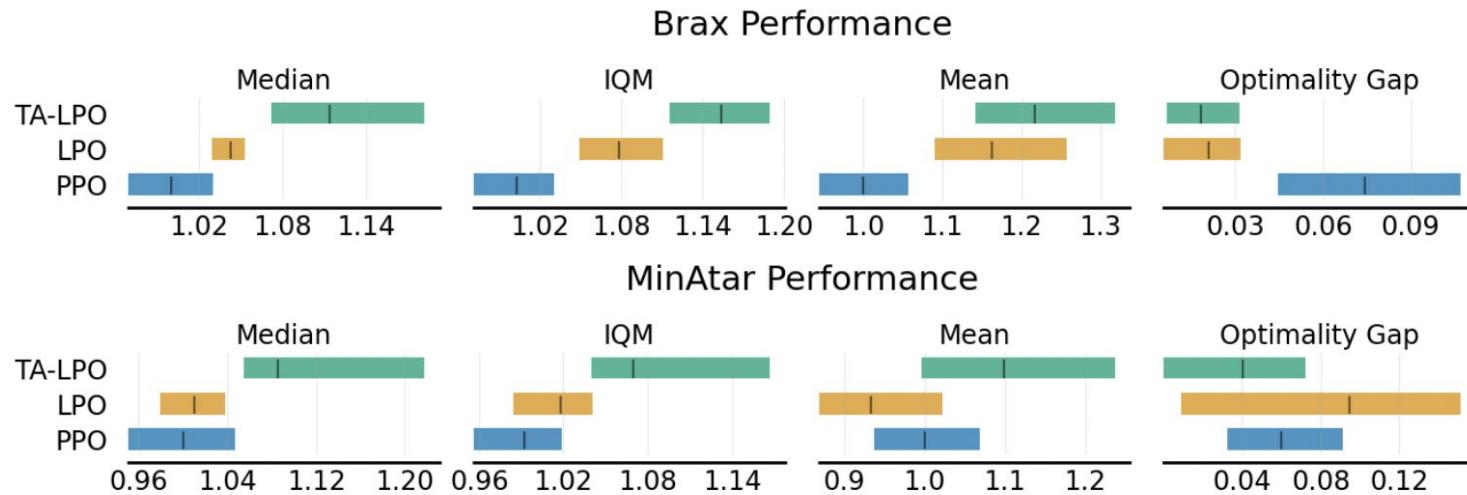


Figure 2: **TA-LPO leverages lifetime information and generalizes to a wide range of environments.** Results of TA-LPO, LPO and PPO on the Brax and MinAtar suites across three seeds. TA-LPO was only meta-trained on SpaceInvaders-MinAtar. We provide complete training curves in Appendix B.

Further example work from FLAIR at the *Hyperscale*..

1. Reward Functions

- a. Sapora, Silvia, et al. "EvIL: Evolution Strategies for Generalisable Imitation Learning." *ICML 2024*

2. Expert Datasets

- a. Lupu, Andrei, et al. "Behaviour Distillation." *ICLR 2024*

3. High-Dimensional Opponent Shaping

- a. Khan, Akbir, et al. "Scaling Opponent Shaping to High Dimensional Games." *AAMAS 2023*

4. Better Environments

- a. Matthews, Michael et al. "Craftax" *ICML 2024*
- b. Frey, Sasha et al, "JAX-LOB" *ICAIF 2023*

5. Synthetic Environments

- a. Lu, Chris et al. "Adversarial Cheap Talk." *ICML 2023*
- b. Liesen, Jarek et al. "Discovering Minimal Reinforcement Learning Environments"

6. Learned RL Optimizers

- a. Goldie, Alexander et al. "Can Learned Optimization Make Reinforcement Learning Less Difficult?" *ICML 2024 AutoRL Workshop (Spotlight)*

All of these experiments would have taken *years* to run in the old paradigm...



Conclusion

- We are in the middle of a revolution for deep RL: *end-to-end GPU acceleration*
- Speed-ups of many orders of magnitude are possible
- We have implemented a number of environments in JAX
- This unlocks theory-inspired, simpler, highly performant algorithms
- It also pushes the frontier for meta-RL
- All code and many tools are open source on our github
- Thanks for listening!

Open questions:

- RL at the hyperscale in the Age of (Agentic) LLMs?
- Breaking out of the (JAX) Box?

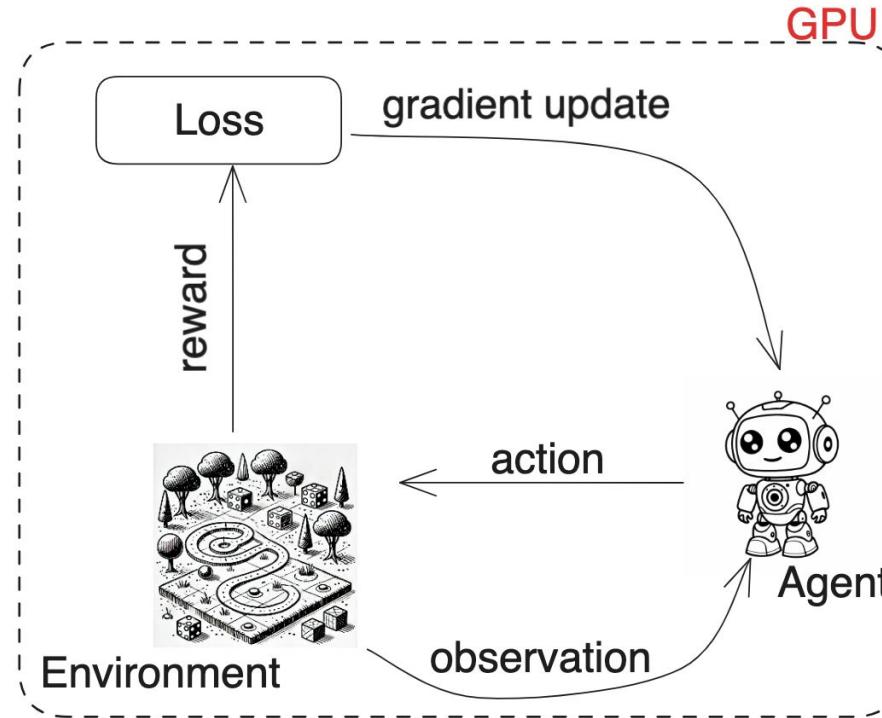
Thanks for listening!

More info and code [@j_foerst](https://github.com/jfoerster) and www.foersterlab.com



The solution:

What if we run *everything* on the GPU instead? Wouldn't this be difficult?



RL at the Hyperscale

