nature

https://doi.org/10.1038/s41586-025-09761-x

Accelerated Article Preview

Discovering state-of-the-art reinforcement learning algorithms

Received: 11 December 2024

Accepted: 15 October 2025

Accelerated Article Preview

Cite this article as: Oh, J. et al. Discovering state-of-the-art reinforcement learning algorithms. *Nature* https://doi.org/10.1038/s41586-025-09761-x (2025)

Junhyuk Oh, Greg Farquhar, Iurii Kemaev, Dan A. Calian, Matteo Hessel, Luisa Zintgraf, Satinder Singh, Hado van Hasselt & David Silver

Discovering state-of-the-art reinforcement learning algorithms

Junhyuk Oh*, **Greg Farquhar***, **Iurii Kemaev***, **Dan A. Calian***, **Matteo Hessel**, **Luisa Zintgraf**, **Satinder Singh**, **Hado van Hasselt**, **David Silver**

All authors are affiliated with Google DeepMind, London, UK.

**These authors contributed equally to this work.*

Humans and other animals use powerful reinforcement learning (RL) mechanisms that have been discovered by evolution over many generations of trial and error. By contrast, artificial agents typically learn using hand-crafted learning rules. Despite decades of interest, the goal of autonomously discovering powerful RL algorithms has proven elusive[7-12]. In this work, we show that it is possible for machines to discover a state-of-the-art RL rule that outperforms manually-designed rules. This was achieved by meta-learning from the cumulative experiences of a population of agents across a large number of complex environments. Specifically, our method discovers the RL rule by which the agent's policy and predictions are updated. In our large-scale experiments, the discovered rule surpassed all existing rules on the well-established Atari benchmark and outperformed a number of state-of-the-art RL algorithms on challenging benchmarks that it had not seen during discovery. Our findings suggest that the RL algorithms required for advanced artificial intelligence may soon be automatically discovered from the experiences of agents, rather than manually designed.

The primary goal of artificial intelligence is to design agents that, like humans, can predict and act in complex environments so as to achieve goals. Many of the most successful agents are based on reinforcement learning (RL), in which agents learn by interacting with environments. Decades of research have produced ever more efficient RL algorithms, resulting in numerous landmarks in artificial intelligence including the mastery of complex competitive games such as Go[1], chess[2], StarCraft[3], Minecraft[4], the invention of new mathematical tools[5], or the control of complex physical systems[6].

Unlike humans, whose learning mechanism has been naturally discovered by biological evolution, RL algorithms are typically manually designed. This is usually slow and laborious, and limited by reliance on human knowledge and intuition. Although a number of attempts have been made to automatically discover learning algorithms[7-12], none have proven to be sufficiently efficient and general to replace hand-designed RL systems.

In this work, we introduce an autonomous method for discovering RL rules solely through the experience of many generations of agents interacting with various environments (Fig. 1a). The discovered RL rule achieves state-of-the-art performance on a variety of challenging RL benchmarks. The success of our method contrasts to prior work in two dimensions. Firstly, while previous methods searched over narrow spaces of RL rules (e.g. hyperparameters[13,14] or policy loss[7,12]), our method allows the agent to explore a

far more expressive space of potential RL rules. Secondly, while previous work focused on meta-learning in simple environments (e.g. grid-worlds[9,15]), our method meta-learns in complex and diverse environments at a much larger scale.

To choose a general space of discovery, we observe that the essential component of standard RL algorithms is a rule that updates one or more predictions, as well as the policy itself, towards *targets* that are functions of quantities such as future rewards and future predictions. Examples of RL rules based on different targets include temporal-difference learning[16], Q-learning[17], PPO[18], auxiliary tasks[19], successor features[20], and distributional RL[21]. In each case, the choice of target determines the nature of the predictions, e.g. whether they become value functions, models, or successor features.

In our framework, an RL rule is represented by a *meta-network* that determines the targets towards which the agent should move its predictions and policy (Fig. 1c). This allows the system to discover useful predictions without predefined semantics, as well as how they are used. The system may in principle rediscover past RL rules, but the flexible functional form also allows the agent to invent new RL rules that may be specifically adapted to environments of interest.

During the discovery process we instantiate a population of agents, each of which interacts with its own instance of an environment taken from a diverse set of challenging tasks. Each agent's parameters are updated according to the current RL rule. We then use the *meta-gradient* method[13] to incrementally improve the RL rule such that it could lead to better performing agents.

Our large-scale empirical results show that our discovered RL rule, which we call DiscoRL, surpasses all existing RL rules on the environments in which it was meta-learned. Notably, this includes Atari games[22], arguably the most established and informative of RL benchmarks. Furthermore, DiscoRL achieved state-of-the-art performance on a number of other challenging benchmarks, such as ProcGen[23], that it had never been exposed to during discovery. We also show that the performance and generality of DiscoRL improves further as more diverse and complex environments are used in discovery. Finally, our analysis shows that DiscoRL has discovered unique prediction semantics that are distinct from existing RL concepts such as value functions. To the best of our knowledge, this is the empirical evidence that surpassing manually-designed RL algorithms in terms of both generality and efficiency is finally within reach.

Discovery Method

Our discovery approach involves two types of optimisations: agent optimisation and meta-optimisation. Agent parameters are optimised by updating their policies and predictions towards the targets produced by the RL rule. Meanwhile, the meta-parameters of the RL rule are optimised by updating its targets to maximise the cumulative rewards of the agents.

Agent network

Much RL research considers what predictions an agent should make (e.g., values), and what loss function should be used to learn those predictions (e.g. TD-learning) and improve the policy (e.g. policy gradient).

Instead of hand-crafting them, we define an expressive space of predictions without predefined semantics and meta-learn what the agent needs to optimise by representing it using a meta-network. It is desirable to maintain the ability to represent key ideas in existing RL algorithms, while supporting a large space of novel algorithmic possibilities.

To this end we let the agent, parameterised by θ, output two types of predictions in addition to a policy (π): an observation-conditioned vector prediction $y(s) \in \mathbb{R}^n$ and an action-conditioned vector prediction $z(s,a) \in \mathbb{R}^m$ where s and a are an observation and an action (Fig. 1b). The form of these predictions stems from the fundamental distinction between prediction and control[16]. For example, value functions are commonly divided into state value functions $v(s)$ (for prediction) and action value functions $q(s,a)$ (for control), and many other concepts in RL, such as rewards and successor features also have an observation-conditioned version $s \mapsto \mathbb{R}^m$ and an action-conditioned version $s,a \mapsto \mathbb{R}^m$. Therefore, the functional form of the predictions (y,z) is general enough to represent, but is not restricted to, many existing fundamental concepts in RL.

In addition to the predictions to be discovered, in most of our experiments the agent makes predictions with pre-defined semantics. Specifically, the agent produces an action-value function $q(s,a)$ and an action-conditional auxiliary policy prediction $p(s,a)$.[2] This encourages the discovery process to focus on discovering new concepts through y and z.

Meta-network

A large proportion of modern RL rules use the *forward view* of RL[16]. In this view, the RL rule receives a trajectory from timestep t to t+n, and uses this information to update the agent's predictions or policy. They typically update the predictions or policy towards *bootstrapped* targets, i.e., towards future predictions.

Correspondingly, our RL rule uses a meta-network (Fig. 1c) as a function that determines targets towards which the agent should move its predictions and policy. To produce targets at timestep t, the meta-network receives as input a trajectory of the agent's predictions and policy as well as rewards and episode termination from timestep t to t+n. It uses a standard LSTM[24] to process these inputs, although other architectures may be used (Extended Data Fig. 3).

The choice of inputs and outputs to the meta-network maintains certain desirable properties of handcrafted RL rules. First, the meta-network can deal with any observation, and with discrete action spaces of any size. This is possible because the meta-network does not receive the observation directly as input, but only indirectly via predictions. In addition, it processes action-specific inputs and outputs by sharing weights across action dimensions. As a result it can generalise to radically different environments. Second, the meta-network is agnostic to the design of the agent network, as it only sees the output of the agent network. As long as the agent network produces the required form of outputs (π,y,z), the discovered RL rule can generalise to arbitrary agent architectures or sizes. Third, the search space defined by the meta-network includes the important algorithmic idea of bootstrapping. Fourth, since the meta-network processes both policy and predictions together, it can not only meta-learn auxiliary tasks [25] but also

directly use predictions to update the policy (e.g., to provide a baseline for variance reduction). Finally, outputting targets is strictly more expressive than outputting a scalar loss function, as it includes semi-gradient methods like Q-learning in the search space. While building on these properties of standard RL algorithms, the rich parametric neural network allows the discovered rule to implement algorithms with potentially much greater efficiency and contextual nuance.

Agent optimisation

The agent's parameters (θ) are updated to minimise the distance from its predictions and policy to the targets from the meta-network. The agent's loss function can be expressed as:

$$L(\theta) = \mathbb{E}_{s,a,s' \sim \pi_\theta}[D(\hat{\pi}, \pi_\theta(s)) + D(\hat{y}, y_\theta(s)) + D(\hat{z}, z_\theta(s,a)) + L_{aux}]$$

where D(p,q) is a distance function between p and q. We chose the Kullback–Leibler (KL) divergence as the distance function, as it is sufficiently general and has previously been found to make meta-optimisation easier[9]. Here $\pi_\theta, y_\theta, z_\theta$ and $\hat{\pi}, \hat{y}, \hat{z}$ are the outputs of the agent network and the meta-network, respectively, with a softmax function applied to normalise each vector.

The auxiliary loss L_{aux} is used for predictions with pre-defined semantics: action-values (q) and auxiliary policy predictions (p) as follows: $L_{aux}=D(\hat{q},q_\theta(s,a))+D(\hat{p},p_\theta(s,a))$, where $\hat{q}$ is an action-value target from Retrace[26] projected to a two-hot vector[2], and $\hat{p}=\pi_\theta(s')$ is the policy at the one-step future state. To be consistent with the rest of losses, we use the KL divergence as the distance function D.

Meta-optimisation

Our goal is to discover an RL rule, represented by the meta-network with meta-parameters η, that allows agents to maximise rewards in a variety of training environments. This discovery objective $J(\eta)$ and its *meta-gradient* $\nabla_\eta J(\eta)$ can be expressed as:

$$J(\eta) = \mathbb{E}_{\mathcal{E}}\mathbb{E}_\theta[J(\theta)], \qquad \nabla_\eta J(\eta) \approx \mathbb{E}_{\mathcal{E}}\mathbb{E}_\theta[\nabla_\eta \theta \nabla_\theta J(\theta)],$$

where $\mathcal{E}$ indicates an environment sampled from a distribution, and θ denotes agent parameters induced by an initial parameter distribution and their evolution over the course of learning with the RL rule. $J(\theta)=\mathbb{E}[\sum_t \gamma^t r_t]$ is the expected discounted sum of rewards, which is the typical RL objective. The meta-parameters are optimised using gradient ascent following the above equations.

To estimate the meta-gradient, we instantiate a population of agents which learn according to the meta-network in a set of sampled environments. To ensure this approximation is close to the true distribution of interest, we use a large number of complex environments taken from challenging benchmarks, in contrast to prior work that focused on a small number of simple environments. As a result the discovery process surfaces diverse RL challenges, such as the sparsity of rewards, the task horizon, and the partial observability or stochasticity of environments.

Each agent's parameters are periodically reset to encourage the update rule to make fast learning progress within a limited agent lifetime. As in prior work on meta-gradient RL[13], the meta-gradient term $\nabla_\eta J(\eta)$ can be divided into two gradient terms by the chain rule: $\nabla_\eta \theta$ and $\nabla_\theta J(\theta)$. The first term can be understood as a gradient over the agent update procedure[27], while the second term is the gradient of the standard RL objective. To estimate the first term, we iteratively update the agent multiple times and backpropagate through the entire update procedure as illustrated in Fig. 1d. To make it tractable, we backpropagate over 20 agent updates using a sliding window. Finally, to estimate the second term, we use the advantage actor-critic (A2C) method[28]. To estimate the advantage, we train a *meta-value* function, which is a value function used only for discovery.

Empirical Result

We implemented our discovery method with a large population of agents in a set of complex environments. We call the discovered RL rule **DiscoRL**. In evaluation, the aggregated performance was measured by the *interquartile mean* (IQM) of normalised scores for benchmarks that consist of multiple tasks, which has proven to be a statistically reliable metric[29].

Atari

The Atari benchmark[22], one of the most studied benchmarks in the history of RL, consists of 57 Atari 2600 games. They require complex strategies, planning, and long-term credit assignment, making it non-trivial for AI agents to master. Hundreds of RL algorithms have been evaluated on this benchmark over the last decade, which include MuZero[2] and Dreamer[4].

To see how strong the rule can be when discovered directly from this benchmark, we meta-trained an RL rule, **Disco57**, and evaluated it on the same 57 games (Fig. 2a). In this evaluation we used a network architecture that has a number of parameters comparable to the number used by MuZero. This is a larger network than the one used during discovery; the discovered RL rule must therefore generalise to this setting. Disco57 achieved an IQM of 13.86, outperforming all existing RL rules[2,4,14,30] on the Atari benchmark, with a substantially higher wall-clock efficiency compared to the state-of-the-art MuZero (Extended Data Fig. 4) This shows that our method can automatically discover a strong RL rule from such challenging environments.

Generalisation

We further investigated the generality of Disco57 by evaluating it on a variety of held-out benchmarks that it was never exposed to during discovery. These benchmarks include unseen observation and action spaces, diverse environment dynamics, various reward structures, and unseen agent network architectures. Meta-training hyperparameters were only tuned on training environments (i.e., Atari) to prevent the rule from being implicitly optimised for held-out benchmarks.

The result on the ProcGen[23] benchmark (Fig. 2b, Extended Data Table 2), which consists of 16 procedurally generated 2D games, shows that Disco57 outperformed all existing published methods,

including MuZero[2] and PPO[18], even though it had never interacted with ProcGen environments during discovery. In addition, Disco57 achieved a competitive performance on Crafter[31] (Fig. 2d, Extended Data Table 5), where the agent needs to learn a wide spectrum of abilities to survive. Disco57 reached the 3rd place on the leaderboard of NetHack NeurIPS 2021 Challenge[32] (Fig. 2e, Extended Data Table 4), where more than 40 teams participated. Unlike the top submitted agents in the competition[33], Disco57 did not use any domain-specific knowledge for defining subtasks or reward shaping. For a fair comparison, we trained an agent with the IMPALA algorithm[34] using the same settings as Disco57. IMPALA's performance was much weaker, suggesting that Disco57 has discovered a more efficient RL rule than standard approaches. In addition to environments, Disco57 turned out to be robust to a range of agent-specific settings such as network size, replay ratio, and hyperparameters in evaluation (Extended Data Fig. 1).

Complex and diverse environments

To understand the importance of complex and diverse environments for discovery, we further scaled up meta-learning with additional environments. Specifically, we discovered another rule, **Disco103**, using a more diverse set of 103 environments consisting of the Atari, ProcGen, and DMLab-30[35] benchmarks. This rule performs similarly on the Atari benchmark while improving scores on every other seen and unseen benchmark in Fig. 2. In particular, Disco103 reached human-level performance on Crafter and neared MuZero's state-of-the-art performance on Sokoban[36]. These results show that the more complex and diverse the set of environments used for discovery, the stronger and more general the discovered rule becomes, even on held-out environments that were not seen during discovery. Discovering Disco103 required no changes to the discovery method compared to Disco57 other than the set of environments. This shows that the discovery process itself is robust, scalable, and general.

To further investigate the importance of using complex environments, we ran our discovery process on 57 grid-world tasks that are extended from prior work[9], using the same meta-learning settings as for Disco57. The new rule had a significantly worse performance (Fig. 3c) on the Atari benchmark. This verifies our hypothesis about the importance of meta-learning directly from complex and challenging environments. While using such environments was crucial, there was no need for a careful curation of the correct set of environments; we simply used popular benchmarks from the literature.

Efficiency and scalability

To further understand the scalability and efficiency of our approach, we evaluated multiple Disco57s over the course of discovery (Fig. 3a). The best rule was discovered within approximately 600 million steps per Atari game, which amounts to just 3 experiments across 57 Atari games. This is arguably more efficient than the manual discovery of RL rules, which typically requires many more experiments to be executed, in addition to the time of the human researchers.

Furthermore, DiscoRL performed better on the unseen ProcGen benchmark as more Atari games were used for discovery (Fig. 3b), showing that the resulting RL rule scales well with the number and diversity of environments used for discovery. In other words, the performance of the discovered rule is a function of data (i.e., environments) and compute.

Effect of discovering new predictions

To study the effect of the discovered semantics of predictions (y,z in Fig. 1b), we compared different rules by varying the outputs of the agent, with and without certain types of predictions. The result in Fig. 3c shows that the use of a value function dramatically improves the discovery process, which highlights the importance of this fundamental concept of RL. On the other hand, the result in Fig. 3c also shows the importance of discovering new prediction semantics (y and z) beyond pre-defined predictions. Overall, increasing the scope of discovery compared to prior work[7-12] was essential. In the following section, we provide further analysis to uncover what semantics have been discovered.

Analysis

Qualitative analysis

We analysed the nature of the discovered rule, using Disco57 as a case study (Fig. 4). Qualitatively, the discovered predictions spike in advance of salient events such as receiving rewards or changes in the entropy of the policy (Fig. 4a). We also investigated which features of the observation cause the meta-learned predictions to respond strongly, by measuring the gradient norm associated with each part of the observation. The result in Fig. 4b shows that meta-learned predictions tend to pay attention to objects that may be relevant in the future, which is distinct from where the policy and the value function pay attention to. These results indicate that DiscoRL has learned to identify and predict salient events over a modest horizon, and thus complements existing concepts such as the policy and value function.

Information analysis

To confirm the qualitative findings, we further investigated what information is contained in the predictions. We first collected data from the DiscoRL agent on 10 Atari games and trained a neural network to predict quantities of interest from either the discovered predictions, the policy, or the value function. The results in Fig. 4c shows that the discovered predictions contain greater information about upcoming large rewards and the future policy entropy, compared to the policy and value. This suggests that the discovered predictions may capture unique task-relevant information that is not well-captured by the policy and value.

Emergence of bootstrapping

We also found evidence that DiscoRL uses a bootstrapping mechanism. When the meta-network's prediction input at future timesteps (z_{t+k}) is perturbed, it strongly affects the target $\hat{z}_t$ (Fig. 4d). This means that the future predictions are used to construct targets for the current predictions. This bootstrapping mechanism and the discovered predictions turned out to be critical for performance (Fig. 4e). If the y and z inputs to the meta-network are set to zero when computing their targets $\hat{y}$ and $\hat{z}$ (thus preventing bootstrapping), performance degrades substantially. If the y and z inputs are set to zero for computing all targets including the policy target, the performance drops even further. This shows the discovered predictions are heavily used to inform the policy update, rather than just serving as auxiliary tasks.

Previous Work

The idea of meta-learning, or learning-to-learn, in artificial agents dates back to the 1980s[37], with proposals to train meta-learning systems with backpropagation of gradients[38]. The core idea of using a slower meta-learning process to meta-optimise a fast learning or adaptation process[39,40] has been studied for numerous applications in various contexts, including transfer learning[41], continual learning[42], multi-task learning[43], hyperparameter optimisation[44], and automated machine learning[45].

Early efforts to use meta-learning for RL agents comprised attempts to meta-learn information-seeking behaviours[46]. Many later works have focused on meta-learning a small number of hyperparameters of an existing RL algorithm[13,14]. Such approaches have produced promising results but cannot drastically depart from the underlying hand-crafted algorithms. Another line of work has attempted to eschew inductive biases by meta-learning entirely black-box algorithms implemented, for example, as recurrent neural networks[47] or, as a synaptic learning rule[48]. While conceptually appealing, these methods are prone to overfit to tasks seen in meta-training[49].

The idea of representing knowledge using a wider class of predictions was first introduced in temporal-difference networks[50] but without any meta-learning mechanism. A similar idea has been explored for meta-learning auxiliary tasks[25]. Our work extends this idea to effectively discover an entire loss function that the agent optimises, covering a much broader range of possible RL rules. Furthermore, unlike prior work the discovered knowledge can generalise to unseen environments.

Recently, there have been growing interests in discovering general-purpose RL rules[7,9-12,15]. However, most of them were limited to small agents and simple tasks, or the scope of discovery was limited to a partial RL rule. Therefore, their rules were not extensively compared to state-of-the-art rules on challenging benchmarks. In contrast, we search over a larger space of rules, including entirely new predictions, and scale up to a large number of complex environments for discovery. As a result, we demonstrate that it is possible to discover a general-purpose RL rule that outperforms a number of state-of-the-art rules on challenging benchmarks.

Conclusion

Enabling machines to discover learning algorithms for themselves is one of the most promising ideas in artificial intelligence due to its potential for open-ended self-improvement. This work has taken a step towards machine-designed reinforcement learning algorithms that can compete with and even outperform some of the best manually-designed algorithms in challenging environments. We also showed that the discovered rule becomes stronger and more general as it gets exposed to more diverse environments. This suggests that the design of RL algorithms for advanced artificial intelligence may in the future be led by machines that can scale effectively with data and compute.

References

1. Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484 (2016).
2. Schrittwieser, J. et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 604-609 (2020).
3. Vinyals, O. et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575, 350-354 (2019).
4. Hafner, D., Pasukonis, J., Ba, J. & Lillicrap, T. Mastering diverse control tasks through world models. *Nature* (2025).
5. Fawzi, A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* 610, 47-53 (2022).
6. Degrave, J. et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* 602, 414-419 (2022).
7. Kirsch, L., van Steenkiste, S. & Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. *International conference on learning representations* (2020).
8. Kirsch, L. et al. Introducing symmetries to black box meta reinforcement learning. *AAAI conference on artificial intelligence* (2022).
9. Oh, J. et al. Discovering reinforcement learning algorithms. *Advances in neural information processing systems* 33 (2020).
10. Xu, Z. et al. Meta-gradient reinforcement learning with an objective discovered online. *Advances in neural information processing systems* 33 (2020).
11. Houthooft, R. et al. Evolved policy gradients. *Advances in neural information processing systems* 31 (2018).
12. Lu, C. et al. Discovered policy optimisation. *Advances in neural information processing systems* (2022).
13. Xu, Z., van Hasselt, H. P. & Silver, D. Meta-gradient reinforcement learning. *Advances in neural information processing systems* 31 (2018).
14. Zahavy, T. et al. A self-tuning actor-critic algorithm. *Advances in neural information processing systems* 33 (2020).
15. Jackson, M. T. et al. Discovering general reinforcement learning algorithms with adversarial environment design. *Advances in neural information processing systems* (2023).
16. Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* (MIT press, 2018).
17. Watkins, C. J. & Dayan, P. Q-learning. *Machine learning* 8, 279-292 (1992).
18. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
19. Jaderberg, M. et al. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397* (2016).
20. Barreto, A. et al. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems* 30 (2017).
21. Bellemare, M. G., Dabney, W. & Munos, R. A distributional perspective on reinforcement learning. *International conference on machine learning* (2017).

22. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research* 47, 253-279 (2013).
23. Cobbe, K., Hesse, C., Hilton, J. & Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *International conference on machine learning* (2020).
24. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Computation* 9, 1735-1780 (1997).
25. Veeriah, V. et al. Discovery of useful questions as auxiliary tasks. *Advances in neural information processing systems* (2019).
26. Munos, R., Stepleton, T., Harutyunyan, A. & Bellemare, M. Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems* 29 (2016).
27. Finn, C., Abbeel, P. & Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *International conference on machine learning* (2017).
28. Mnih, V. et al. Asynchronous methods for deep reinforcement learning. *International conference on machine learning* (2016).
29. Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C. & Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems* 34 (2021).
30. Kapturowski, S. et al. Human-level atari 200x faster. *International conference on learning representations* (2023).
31. Hafner, D. Benchmarking the spectrum of agent capabilities. *International conference on learning representations* (2022).
32. Küttler, H. et al. The nethack learning environment. *Advances in neural information processing systems* 33 (2020).
33. Hambro, E. et al. Insights from the NeurIPS 2021 NetHack challenge. *NeurIPS 2021 Competitions and Demonstrations Track* 41-52 (2022).
34. Espeholt, L. et al. IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. *International conference on machine learning* (2018).
35. Beattie, C. et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801* (2016).
36. Racanière, S. et al. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems* 30 (2017).
37. Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Ph.D. thesis, Technische Universität München (1987).
38. Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. *International conference on simulation of adaptive behavior: From animals to animats* (1991).
39. Schmidhuber, J., Zhao, J. & Wiering, M. Simple principles of metalearning. *Technical report IDSIA* 69, 1-23 (1996).
40. Thrun, S. & Pratt, L. Learning to learn: Introduction and overview, 3-17 (Springer, 1998).
41. Pan, S. J. & Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 1345-1359 (2009).
42. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C. & Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks* 113, 54-71 (2019).

43. Caruana, R. Multitask learning. *Machine learning* 28, 41-75 (1997).
44. Feurer, M. & Hutter, F. Hyperparameter optimization, 3-33 (Springer, Cham, 2019).
45. Yao, Q. et al. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).
46. Storck, J., Hochreiter, S., Schmidhuber, J. et al. Reinforcement driven information acquisition in non-deterministic environments. *International conference on artificial neural networks* (1995).
47. Duan, Y. et al. Rl 2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779* (2016).
48. Niv, Y., Joel, D., Meilijson, I. & Ruppin, E. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors (2002).
49. Xiong, Z., Zintgraf, L., Beck, J., Vuorio, R. & Whiteson, S. On the practical consistency of meta-reinforcement learning algorithms. *Workshop on meta-learning, NeurIPS* (2021).
50. Sutton, R. S. & Tanner, B. Temporal-difference networks. *Advances in neural information processing systems* (2004).

Figure Legends

Figure 1: **Discovering a reinforcement learning rule from a population of agents. (a) Discovery.** Multiple agents, interacting with various environments, are trained in parallel according to the learning rule, defined by the meta-network. In the meantime, the meta-network is optimised to improve the agents' collective performances. **(b) Agent architecture.** An agent produces the following outputs: 1) a policy (π), 2) an observation-conditioned prediction vector (y), 3) action-conditioned prediction vectors (z), 4) action-values (q), and 5) an auxiliary policy prediction (p). The semantics of y and z are determined by the meta-network. **(c) Meta-network architecture.** A trajectory of the agent's outputs is given as input to the meta-network, together with rewards and episode termination indicators from the environment (omitted for simplicity in the figure). Using this information, the meta-network produces targets for all of the agent's predictions from the current and future timesteps. The agent is updated to minimise the prediction errors with respect to their targets. **(d) Meta-optimisation.** The meta-parameters of the meta-network are updated by taking a meta-gradient step calculated from backpropagation through the agent's update process ($\theta_0 \rightarrow \theta_N$), where the meta-objective to maximise the collective returns of the agents in their environments.

Figure 2: **Evaluation of DiscoRL.** The x-axis represents the number of environment steps in millions. The y-axis represents the human-normalised interquartile mean (IQM) score for benchmarks consisting of multiple tasks (Atari, ProcGen, DMLab-30) and average return for the rest. **Disco57** (blue) is discovered from the Atari benchmark, and **Disco103** (orange) is discovered from Atari, ProcGen, and DMLab-30 benchmarks. The shaded areas show 95% confidence intervals. The dashed lines represent manually-designed RL rules.

Figure 3: **Properties of discovery process.** (a) **Discovery efficiency:** The best DiscoRL was discovered within 3 simulations of the agent's lifetimes (200 million steps) per game. (b) **Scalability:** DiscoRL becomes stronger on the ProcGen benchmark as the training set of environments grows. (c) **Ablation:** The plot shows the performances of variations of DiscoRL on Atari. 'w/o Aux-Pred' is meta-learned without the auxiliary prediction (p). 'Small Agents' uses a smaller agent network during discovery. 'w/o Prediction' is meta-learned without learned predictions (y,z). 'w/o Value' is meta-learned without the value function (q). 'Toy Envs' is meta-learned from 57 grid world tasks instead of Atari games.

Figure 4: **Analysis of DiscoRL.** (a) **Behaviour of discovered predictions:** The plot shows how the agent's discovered prediction (y) changes along with other quantities in *Ms Pacman* (left) and *Breakout* (right). 'Confidence' is calculated as negative entropy. Spikes in prediction confidence are correlated with upcoming salient events. For example, they often precede large rewards in *Ms Pacman* and strong action preferences in *Breakout*. (b) **Gradient analysis:** Each contour shows where each prediction focuses on in the observation through a gradient analysis in *Beam Rider*. The predictions tend to focus more on enemies at a distance, whereas the policy and the value tend to focus on nearby enemies and the scoreboard, respectively. (c) **Prediction analysis:** Future entropy and large-reward events can be better predicted from discovered predictions. (d) **Bootstrapping horizon:** The plot shows how much the prediction target produced by DiscoRL changes when the prediction at each timestep is perturbed. The individual curves correspond to 16 randomly sampled trajectories, while the bold curve corresponds to the average over

them. (e) **Reliance on predictions:** The plot shows the performance of the controlled DiscoRL on *Ms Pacman* without bootstrapping when updating predictions and without using predictions at all.

Methods

Meta-network

The meta-network maps a trajectory of agent outputs along with relevant quantities from the environment to targets: $m_\eta: f_\theta(s_t), f_{\theta^-}(s_t), a_t, r_t, b_t, \ldots, f_\theta(s_{t+n}), f_{\theta^-}(s_{t+n}), a_{t+n}, r_{t+n}, b_{t+n} \mapsto \hat{\pi}, \hat{y}, \hat{z}$ where η represents meta-parameters, and $f_\theta=[\pi_\theta(s), y_\theta(s), z_\theta(s), q_\theta(s)]$ is the agent output with parameters θ. a,r,b are an action taken by the agent, a reward, and an episode termination indicator, respectively. θ^- is an exponential moving average of parameters θ. This functional form allows the meta-network to search over a strictly larger space of rules compared to meta-learning a scalar loss function. This is further discussed in Supplementary Information.

The meta-network processes the inputs by unrolling an LSTM backward in time as illustrated in Fig. 1c. This allows it to take into account n-step future information to produce targets, as in multi-step RL methods such as TD(λ)[51]. We found that this architecture is computationally more efficient than alternatives such as transformers, while achieving a similar performance as shown in Extended Data Fig. 3b.

The action-specific inputs and outputs are processed in the meta-network using shared weights over the action dimension, and an intermediate embedding is computed by averaging across it. This allows the meta-network to process any number of actions. More details about it can be found in Supplementary Information.

In order to allow the meta-network to discover a wider class of algorithms, such as reward normalisation, that require maintaining statistics over an agent's lifetime, we add an additional recurrent neural network. This 'meta-RNN' is unrolled forward across agent updates (from θ_i to θ_{i+1}), rather than across timesteps in an episode. The core of the meta-RNN is another LSTM module. For each of the agent updates, the whole batch of trajectories is embedded into a single vector which is passed to this LSTM. The meta-RNN can potentially capture the learning dynamics throughout the agent's lifetime, producing targets that are adaptive to the specific agent and the environment. The meta-RNN slightly improved the overall performance as shown in Extended Data Fig. 3a. Further details are described in Supplementary Information.

Meta-optimisation stabilisation

A number of challenges arise when we discover at a large scale, mainly because of unbalanced gradient signals coming from agents in different environments and myopic gradients caused by long lifetimes of agents. We introduce a few methods to alleviate these problems.

First, when estimating the advantage term in A2C to estimate $\nabla_\theta J(\theta)$ in the meta-gradient, we normalise the advantage term as follows: $\bar{A}=(A-\mu)/\sigma$, where $\bar{A}$ is a normalised advantage and μ and σ are the exponentially moving average and standard deviation of advantages accumulated over the agent's lifetime. We found that this makes the scale of the advantage term balanced across different environments. In addition, when aggregating the meta-gradient from the population of agents, we take the average of the

meta-gradients over all agents after applying a separate Adam optimiser to the meta-gradient calculated from each agent: $\eta \leftarrow \eta + \frac{1}{n}\sum_{i=1}^{n} ADAM(g_i)$ where g_i is the meta-gradient estimation from the i-th agent in the population. We found that this helps to normalise the magnitude of the meta-gradients from each agent.

We add two meta-regularisation losses (L_{ent} and L_{kl}) to the meta-objective $J(\eta)$ as follows: $\square_{\mathscr{E}}\square_{\theta}[J(\theta)-L_{ent}(\theta)-L_{kl}(\theta)]$. $L_{ent}(\theta)=-\square_{s,a}[H(y_\theta(s))+H(z_\theta(s,a))]$ is an entropy regularisation of predictions y and z, where $H(\cdot)$ is the entropy of the given categorical distribution. We found that this helps prevent the predictions from converging prematurely. $L_{kl}(\theta)=D_{KL}(\pi_{\theta^-}||\hat{\pi})$ is the KL divergence between the policy of a target network with an exponential moving average of the agent parameters (θ^-) and the meta-network's policy target ($\hat{\pi}$). This prevents the meta-network from proposing excessively aggressive updates which could lead to collapse.

Note that these methods are used only to stabilise meta-optimisation, and they do not determine how the agents are updated. The meta-learned rule still solely determines how the agents are updated.

Implementation details

We developed a JAX-based framework[52,53] that distributes computation across TPUs[54] inspired by the Podracer architectures[55]. In this framework, each agent is simulated independently, with the meta-gradients of all agents being calculated in parallel. The meta-parameters are updated synchronously by aggregating meta-gradients across all agents. We used MixFlow-MG[56] to minimise the computational cost of the runs.

For Disco57, we instantiate 128 agents by cycling through the 57 Atari environments in lexicographic order. For Disco103, we instantiate 206 agents, with two copies of each environment from Atari, ProcGen and DMLab-30. Disco57 was discovered using 1024 TPUv3 cores for 64 hours, and Disco103 was discovered using 2048 TPUv3 cores for 60 hours.

The meta-value function used to calculate the meta-gradient is updated using V-Trace[34], with a discount factor of 0.997 and a TD(λ) coefficient of 0.95. The meta-value function and agent networks are optimised using an Adam optimiser with a learning rate of 0.0003. For meta-parameter updates, we use the Adam optimiser with a learning rate of 0.001 and gradient clipping of 1.0. Each agent is updated based on a batch of 96 trajectories with 29 timesteps each. In each batch, on-policy trajectories and trajectories sampled from the replay buffer are mixed, with replay trajectories accounting for 90% of each batch. At each meta-step, 48 trajectories are generated to calculate the meta-gradient and update the meta-value function.

Each agent's parameters are reset after it has consumed its allocated experience budget. When resetting, a new experience budget is sampled from the categories (200M, 100M, 50M, 20M) with a weight inversely proportional to the budget, such that the same amount of total experience is sampled in each category. This was based on our observation that much of learning happens early in the lifetime and demonstrated a marginal improvement in our preliminary small-scale investigation.

Hyperparameters and evaluation

For evaluation on held-out benchmarks, we only tuned the learning rate from {0.0001, 0.0003, 0.0005}. The rest of the hyperparameters were selected based on baseline algorithms from the literature.

The evaluation on Atari games (shown in Fig. 2a and Extended Data Table 1) used a version of the IMPALA[34] network with an increased parameter count which matches the agent network size used by MuZero[2]. Specifically, we used a network with four convolutional residual blocks with 256, 384, 384, 256 filters, a shared fully-connected final layer of 768 dimensions, and an LSTM-based action-conditional predictions component that is composed of an LSTM with a 1024 hidden state dimension and a 1024 dimensional fully-connected layer. DMLab-30 evaluations (Fig. 2c and Extended Data Table 3) use the same action space discretisation and agent network architecture as used in IMPALA. See Extended Data Table 6 for the list of hyperparameters. To verify the statistical significance of our evaluations we used two random seeds for initialisation on each environment from Atari, ProcGen, and DMLab; three seeds on Crafter and Nethack, and five on Sokoban.

Analysis details

For the prediction analysis in Fig. 4c, we train multiple 3-layer MLPs with 128, 64, and 32 hidden units for each layer respectively. The MLPs are trained to predict quantities such as future entropy and rewards from the outputs of an agent that has been trained on different Atari games using Disco57. We use 10 Atari games (Alien, Amidar, Battle Zone, Frostbite, Gravitar, Qbert, Riverraid, Road Runner, Robotank, Zaxxon). The values shown in Fig. 4c are R2 scores for future entropy and test accuracy for large-reward events using 5-fold cross-validation. Extended Data Fig. 2 provides an additional prediction analysis for more quantities. For high-dimensional outputs (y, z, z_a) we used a larger 3-layer MLP with 256 hidden units each.

Methods references

51. Sutton, R. S. Learning to predict by the methods of temporal differences. Machine Learning 3, 9-44 (1988). URL https://api.semanticscholar.org/CorpusID:3349598.
52. Bradbury, J. et al. Jax: composable transformations of python+ numpy programs (2018).
53. DeepMind et al. The DeepMind JAX Ecosystem (2020). URL http://github.com/google-deepmind.
54. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture, 1-12 (2017).
55. Hessel, M. et al. Podracer architectures for scalable reinforcement learning. arXiv preprint arXiv:2104.06272 (2021).
56. Kemaev, I., Calian, D. A., Zintgraf, L. M., Farquhar, G. & van Hasselt, H. Scalable meta-learning via mixed-mode differentiation. *International conference on machine learning* (2025).

Data availability

No external data were used for the results presented in the article.

Code availability

We provide the meta-training and evaluation code, with the meta-parameters of Disco103, under an open source licence at https://github.com/google-deepmind/disco_rl. All of the benchmarks presented in the article are publicly available.

End Notes

Acknowledgements

We thank Sebastian Flennerhag, Zita Marinho, Angelos Filos, Surya Bhupatiraju, Andras György, and Andrei A. Rusu for their feedback and discussions about related ideas. We also thank Blanca Huergo Muñoz, Manuel Kroiss, and Dan Horgan for their help with the engineering infrastructure. Finally, we thank Raia Hadsell, Koray Kavukcuoglu, Nando de Freitas, and Oriol Vinyals for their high-level feedback on the project, and Simon Osindero and Doina Precup for their feedback on the early version of this work.

Author contributions

J.O., I.K., G.F. and D.C. contributed equally; J.O., G.F., I.K., D.C. and L.Z. developed and analysed the method with advice from H.H., S.B. and D.S.; J.O., G.F., M.H., D.C. and I.K. evaluated the method; I.K. led engineering with contributions from J.O., D.C. and G.F; J.O. and G.F. wrote the paper with contributions from L.Z. and D.C.; H.H., S.B. and D.S. advised the team. J.O. and D.S. led the project.

Author information

All authors are affiliated with Google DeepMind, London, UK.

Correspondence and requests for materials should be addressed to Junhyuk Oh (junhyuk@google.com) or David Silver (davidsilver@google.com).

Competing interests

A patent application(s) directed to aspects of the work described has been filed and is pending as of the date of manuscript submission. Google LLC has ownership and potential commercial interests in the work described.

Additional information

Supplementary information

Supplementary Information is available for this paper.

Extended Data

Extended Data Figure 1: **Robustness of DiscoRL.** The plots show the performance of Disco57 and Muesli on *Ms Pacman* by varying agent settings. 'Discovery' and 'Evaluation' represent the setting used for discovery and evaluation, respectively. (a) Each rule was evaluated on various agent network sizes. (b) Each rule was evaluated on various replay ratios, which define the proportion of replay data in a batch compared to on-policy data. (c) A sweep over optimisers (Adam or RMSProp), learning rates, weight decays, and gradient clipping thresholds was evaluated (36 combinations in total) and ranked according to the final score.

Extended Data Figure 2: **Detailed results for the regression and classification analysis.** Each cell represents the test score of one MLP model that has been trained to predict some quantity (columns) given the agent's outputs (rows).

Extended Data Figure 3: **Effect of meta-network architecture.** (a) The x-axis represents the number of environment steps in evaluation and the y-axis the IQM on the Atari benchmark. All methods are discovered from 16 randomly selected Atari games. The meta-RNN component slightly improves performance. (b) The x-axis represents the number of environment steps in evaluation and the y-axis the IQM on the Atari benchmark. All methods are discovered from 16 randomly selected Atari games. Each curve corresponds to a different meta-network architecture, with varying number of LSTM hidden units, or its LSTM component is replaced by a transformer. The choice of the meta-net architecture minimally affects performance.

Extended Data Figure 4: **Computational cost comparison.** The x-axis represents the amount of TPU hours spent for evaluation. The y-axis represents the performance on the Atari benchmark. Each algorithm was evaluated on 57 Atari games for 200M environment steps. DiscoRL reached MuZero's final performance with approximately 40% less computation.

Extended Data Table 1: **Atari57 result.** Each algorithm was evaluated at 200M environment steps. The best result for each game is shown in **bold**.

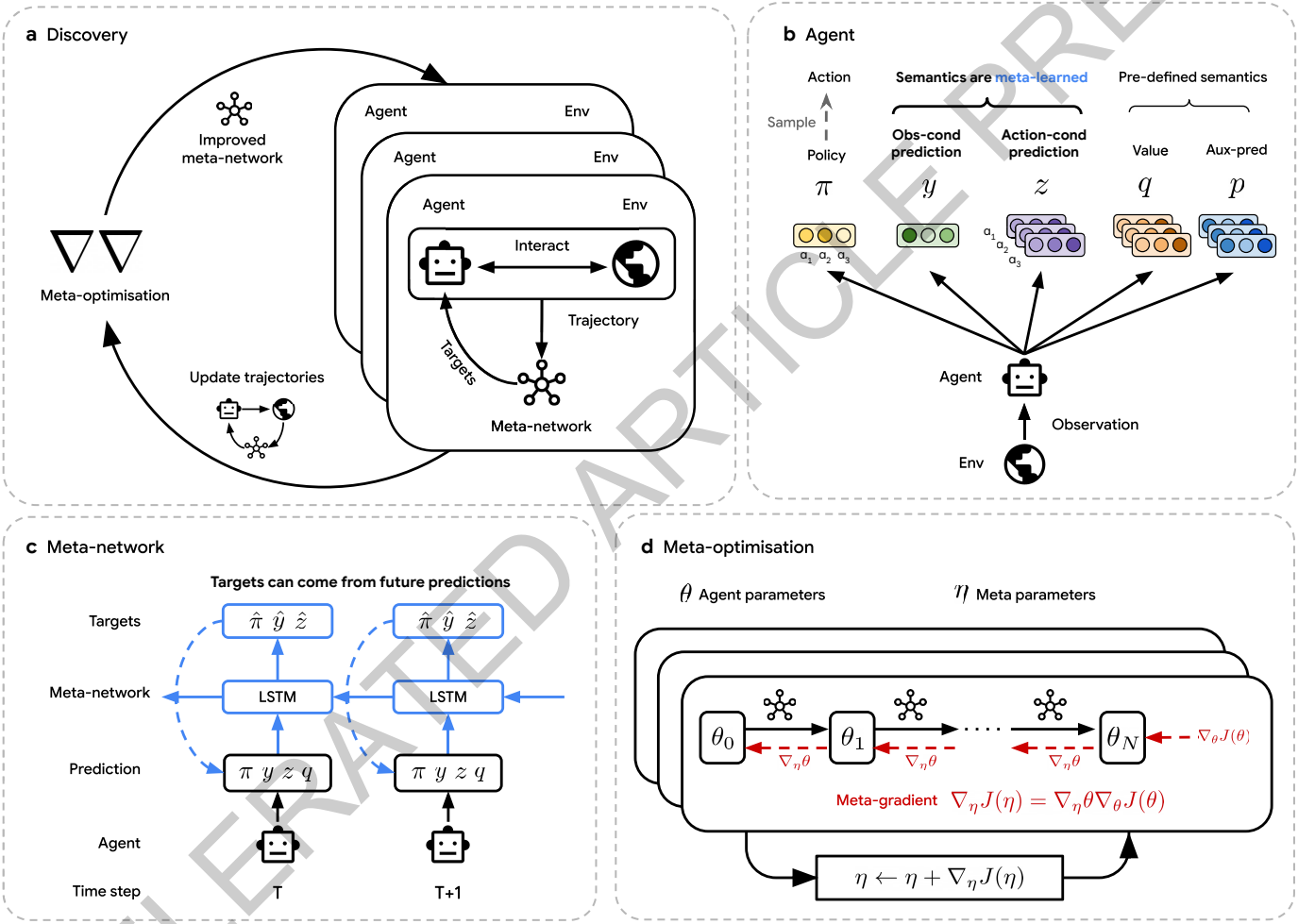
Extended Data Table 2: **ProcGen result.** Best result for each game shown in **bold**. The agents were trained and evaluated on 500 training levels.

Extended Data Table 3: **DMLab-30 result.** Best result for each task shown in **bold**.

Extended Data Table 4: **NetHack result.** Comparison to top 4 submissions from the "Final (Neural)" leaderboard of the NetHack NeurIPS 2021 Challenge[33]. Note that median scores were used for ranking submissions on the leaderboard, whereas we report mean scores to be consistent with other benchmarks.

Extended Data Table 5: **Crafter result.** Dreamer, Rainbow, PPO, Disco57 @1M and Disco103 @1M all use the same data budget of 1M environment steps, while Disco57 @20M and Disco103 @20M use 20M steps.

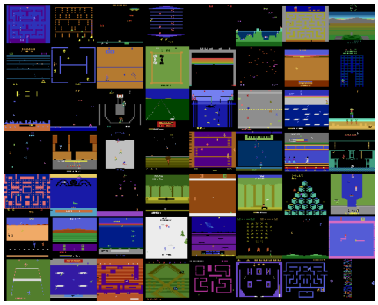
Extended Data Table 6: **Hyperparameters of agents in evaluation**.

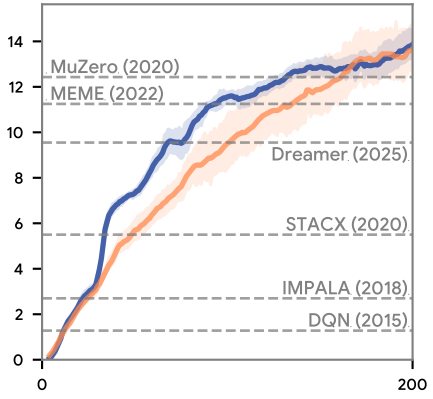


Used to discover Disco57

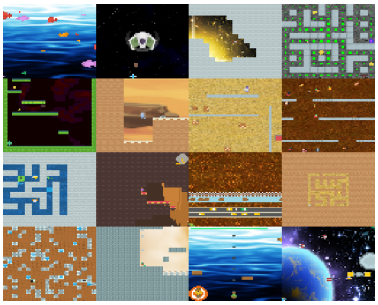
Used to discover Disco103

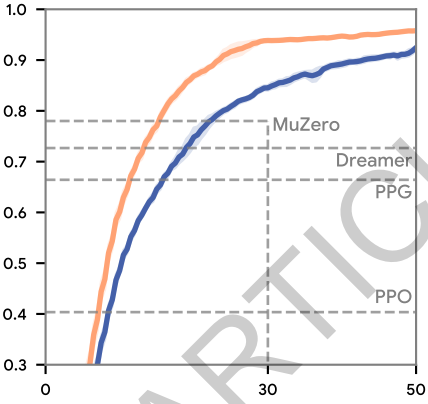
a Atari



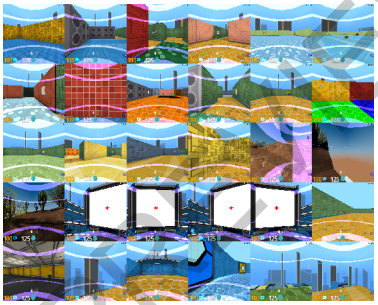


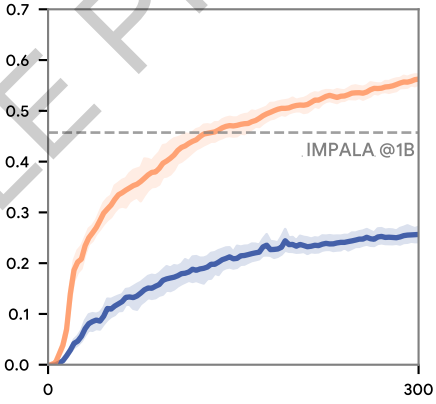
b ProcGen



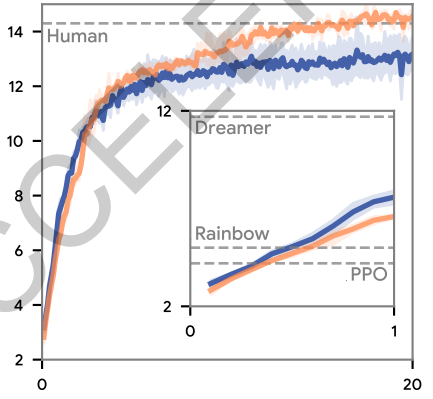


c DMLab-30

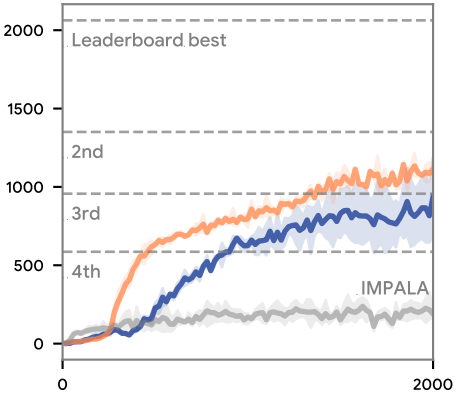




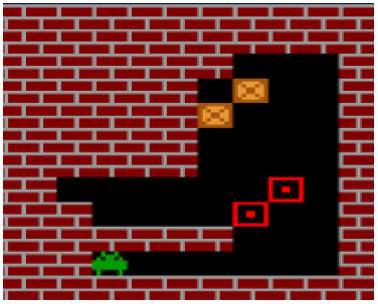
d Crafter

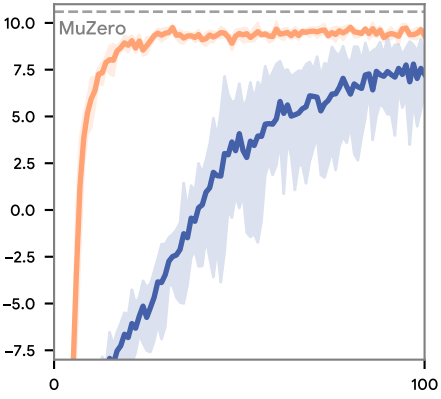


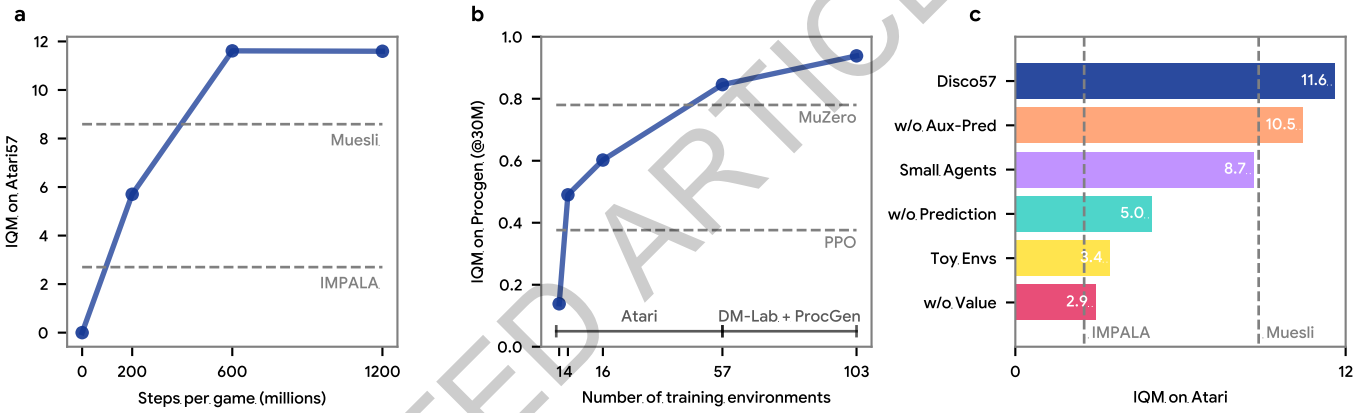
e Nethack Challenge

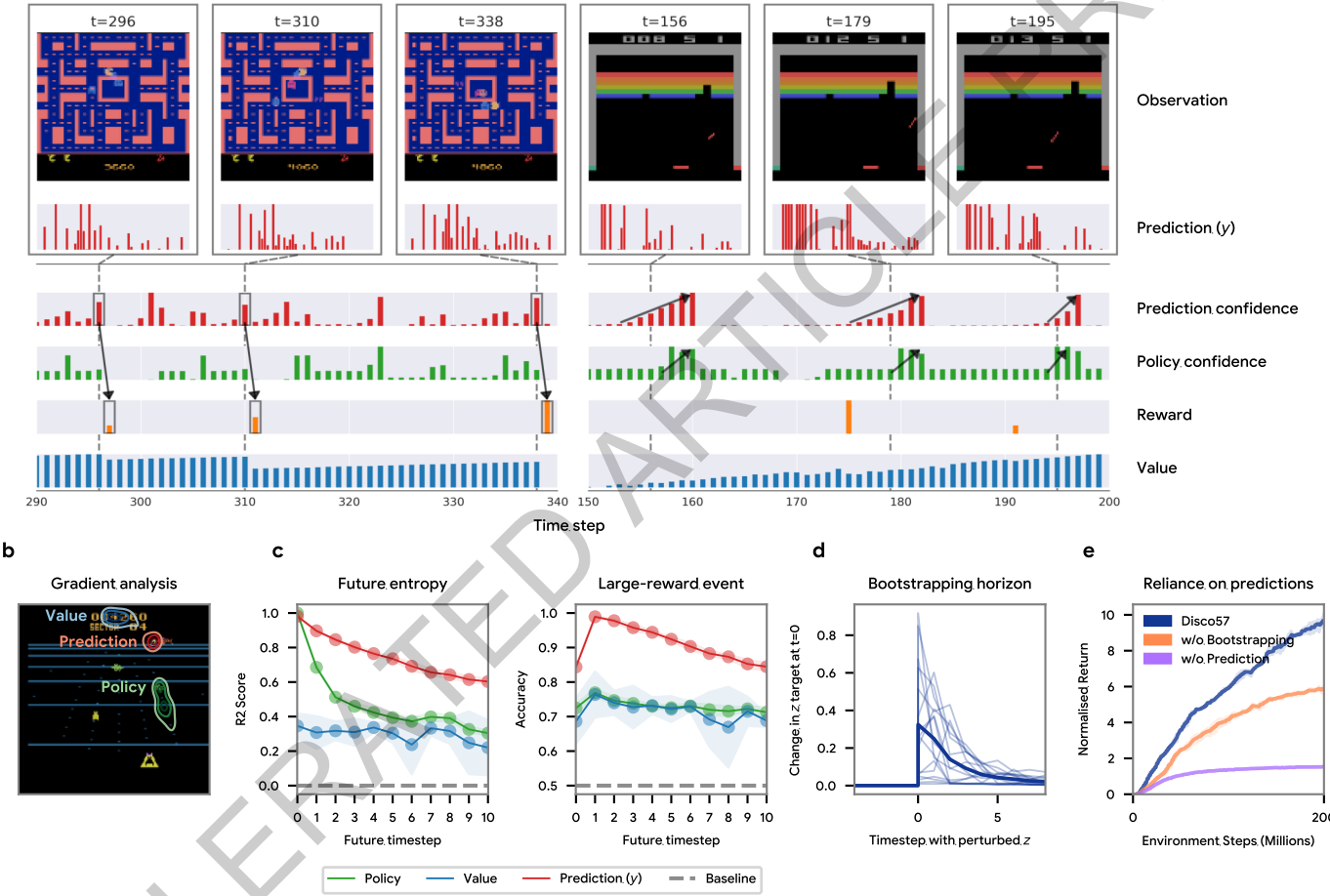


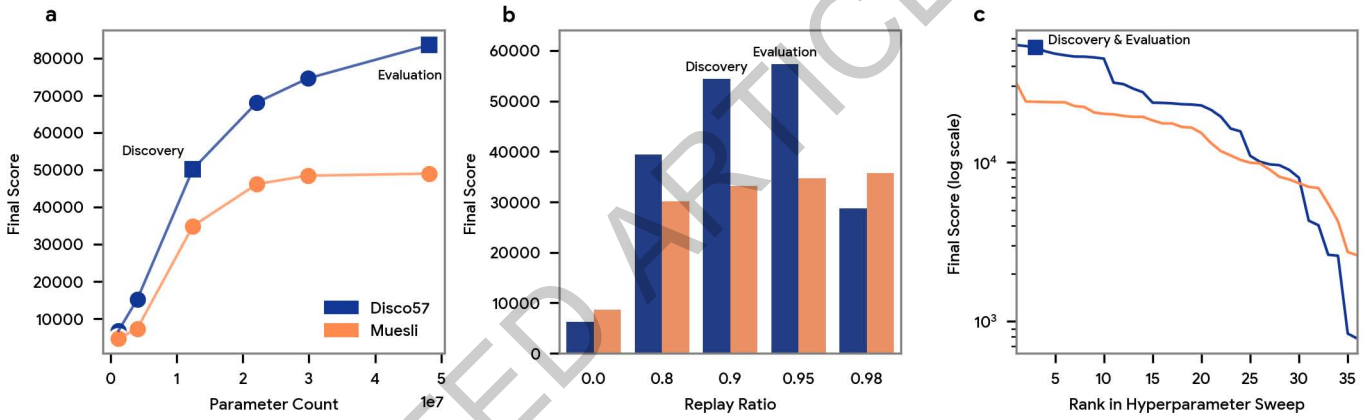
f Sokoban



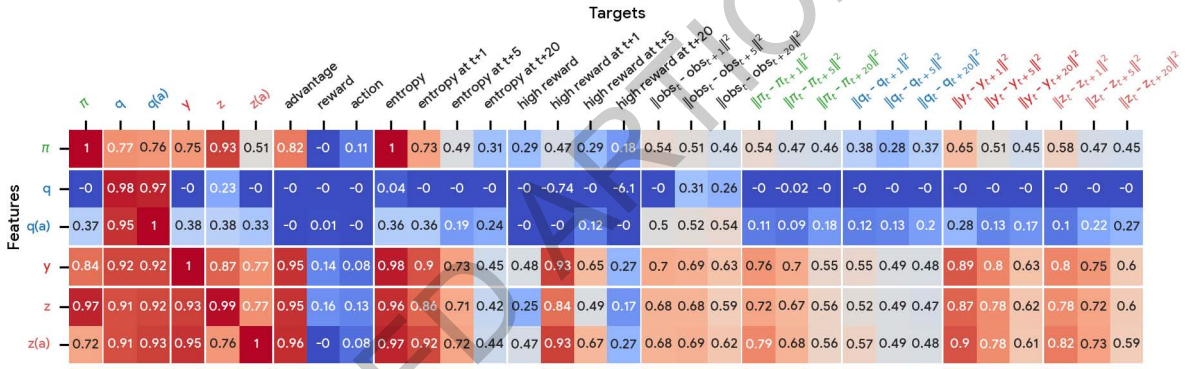



a
IQM on Atari57
Muesli
IMPALA
Steps per game (millions)
b
IQM on Procgen (@30M)
MuZero
PPO
Atari
DM-Lab + ProcGen
Number of training environments
c
Disco57
w/o Aux-Pred
Small Agents
w/o Prediction
Toy Envs
w/o Value
11.6
10.5
8.7
5.0
3.4
2.9
IMPALA
Muesli
IQM on Atari

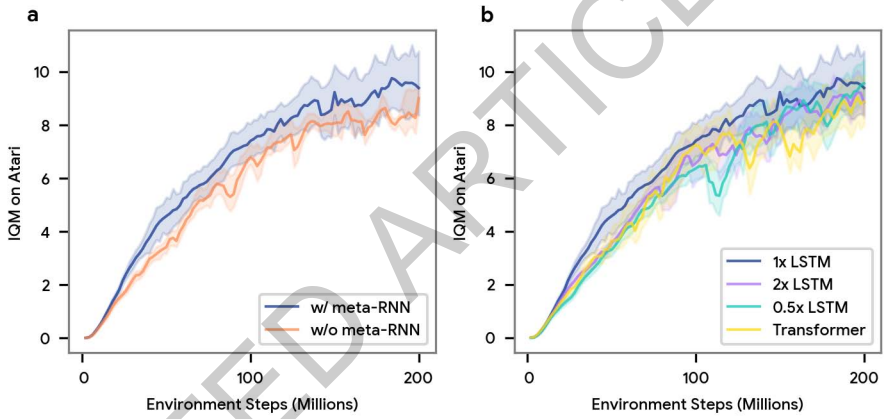




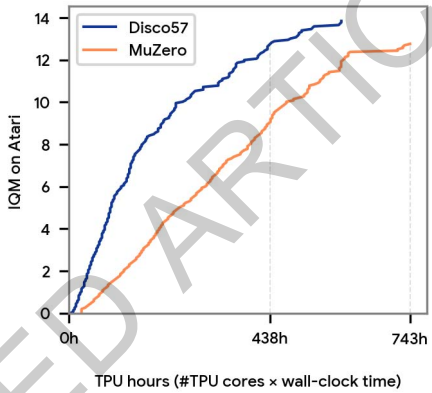
Extended Data Fig. 1



Extended Data Fig. 2



Extended Data Fig. 3



Extended Data Fig. 4

	Random	Human	IMPALA	STACX	Muesli	MuZero	Dreamer	Disco57	Disco103
Alien	228	7128	15962	31809	139409	135541	10977	322137	**333105**
Amidar	6	1720	1554	3719	**21653**	1061	3612	1384	14593
Assault	222	742	19148	19648	**36963**	29697	26010	34676	33546
Asterix	210	8503	300732	245617	316210	**918628**	441763	814458	773870
Asteroids	719	47389	108590	156096	484609	**509953**	348684	499430	490463
Atlantis	12850	29028	849967	848007	1363427	1136009	**1553222**	1112262	1083115
Bank Heist	14	753	1223	1329	1213	**14176**	1083	1219	1368
Battle Zone	2360	37188	20885	78359	414107	320641	**419653**	98764	94730
Beam Rider	364	16927	32463	62892	288870	319684	37073	215580	**328418**
Berzerk	124	2630	1852	1523	44478	19523	10557	**74633**	72922
Bowling	23	161	59	28	191	156	**250**	197	219
Boxing	0	12	99	**100**	99	**100**	**100**	99	99
Breakout	2	30	787	717	791	778	384	**798**	719
Centipede	2091	12017	11049	478347	869751	862737	554553	893158	**899572**
Chopper Command	811	7388	28255	846788	101289	494578	802698	**967905**	337167
Crazy Climber	10780	35829	136950	182617	175322	176172	**193204**	176195	173933
Defender	2874	18689	185203	344453	629482	544320	579875	640052	**650392**
Demon Attack	152	1971	132826	130741	129544	**143846**	142109	143656	143710
Double Dunk	-19	-16	0	**24**	-3	**24**	**24**	23	23
Enduro	0	861	0	259	2362	2363	2166	2364	**2373**
Fishing Derby	-92	-39	44	62	51	69	**82**	57	76
Freeway	0	30	0	18	33	**34**	**34**	33	16
Frostbite	65	4335	317	2522	301694	**410173**	41888	1652	1214
Gopher	258	2412	66782	87094	104441	121342	87600	**125875**	120536
Gravitar	173	3351	359	2746	11660	10926	12570	**13722**	10384
Hero	1027	30826	33730	35559	37161	37249	**40677**	31605	29031
Ice Hockey	-11	1	3	19	25	40	**57**	31	40
Jamesbond	29	303	1632	26123	19319	32107	24010	**35441**	31636
Kangaroo	52	3035	1632	3182	**14096**	13928	12229	13603	13196
Krull	1598	2666	8147	10480	34221	50137	**69858**	62579	60703
Kung Fu Master	258	22736	43375	67823	134689	148533	**154893**	102679	89081
Montezuma Revenge	0	4753	0	0	2359	1450	1852	**2559**	400
Ms Pacman	307	6952	7342	12647	65278	79319	24079	**101876**	101290
Name This Game	2292	8049	21537	24616	105043	**108133**	77809	98867	103914
Phoenix	761	7243	210996	370270	805305	748424	316606	**867704**	841925
Pitfall	-229	6464	-1	**0**	**0**	**0**	**0**	-1	**0**
Pong	-21	15	**21**	**21**	20	**21**	20	20	20
Private Eye	25	69571	98	100	10323	7600	26432	**27833**	5598
Qbert	164	13455	**351200**	27264	157353	85926	201084	127630	133099
Riverraid	1338	17118	29608	47671	47323	**172266**	48080	67916	44260
Road Runner	12	7845	57121	62191	327025	**554956**	150402	311239	356701
Robotank	2	12	12	61	59	85	**132**	108	108
Seaquest	68	42055	1753	1744	815970	501236	356584	**999994**	999991
Skiing	-17098	-4337	**-10180**	-10504	-18407	-30000	-29965	-29949	-30072
Solaris	1236	12327	2365	2326	3031	4401	**5851**	2314	2396
Space Invaders	148	1669	43595	34875	59602	31265	15005	**64092**	44213
Star Gunner	664	10250	200625	298448	214383	158608	408961	**568369**	564206
Surround	-10	7	7	3	9	**10**	9	9	8
Tennis	-24	-8	0	19	12	0	-3	**23**	**23**
Time Pilot	3568	5229	48481	49932	359105	**413988**	314947	411305	388002
Tutankham	11	168	292	101	252	318	**395**	230	347
Up N Down	533	11693	332546	315588	549190	606602	614065	**640537**	629513
Venture	0	1188	0	0	**2104**	866	0	1960	0
Video Pinball	0	17668	572898	441220	685436	921563	**940631**	877143	762964
Wizard Of Wor	564	4757	9157	47854	93291	**103463**	99136	100942	102435
Yars Revenge	3093	54577	84231	113651	557818	187731	**675774**	144298	201832
Zaxxon	32	9173	32935	56952	65325	**106935**	78443	36011	37081

Extended Data Table 1

	Random	Human	PPO	PPG	Dreamer	MuZero	Disco57	Disco103
Steps	-	-	50M	50M	50M	30M	50M	50M
Bigfish	0.00	40.00	10.92	31.26	7.67	30.80	34.61	**35.80**
Bossfight	0.50	13.00	10.47	11.46	11.16	11.88	12.53	**12.65**
Caveflyer	2.00	13.40	6.03	10.02	10.78	**12.37**	9.41	9.93
Chaser	0.50	14.20	4.48	8.57	5.21	4.34	12.83	**13.26**
Climber	1.00	12.60	7.59	10.24	11.77	9.35	11.74	**12.37**
Coinrun	5.00	10.00	7.93	8.98	9.87	9.90	9.93	**9.95**
Dodgeball	1.50	19.00	4.80	10.31	9.36	15.15	18.61	**18.72**
Fruitbot	-0.50	27.20	20.28	24.32	7.64	8.64	26.41	**27.84**
Heist	2.00	10.00	2.25	3.77	**6.86**	3.04	3.53	5.56
Jumper	1.00	10.00	5.09	5.84	**9.18**	7.75	8.74	9.05
Leaper	1.50	10.00	5.90	8.76	8.34	9.83	**9.91**	9.89
Maze	4.00	10.00	4.97	7.06	8.93	7.48	9.16	**9.52**
Miner	1.50	20.00	7.56	9.08	6.07	16.67	18.52	**21.48**
Ninja	2.00	10.00	6.16	9.38	9.89	9.84	9.88	**9.94**
Plunder	3.00	30.00	11.16	13.44	**24.26**	13.26	7.04	7.44
Starpilot	1.50	35.00	17.00	21.57	25.64	41.36	41.42	**44.07**

Extended Data Table 2

	Random	Human	IMPALA	Disco57	Disco103
Steps	-	-	1000M	300M	300M
Explore Goal Locations Large	3.10	194.50	83.10	36.86	**96.62**
Explore Goal Locations Small	7.70	267.50	209.40	208.78	**224.11**
Explore Object Locations Large	4.70	65.70	37.00	32.84	**41.78**
Explore Object Locations Small	3.60	74.50	57.80	56.08	**64.89**
Explore Object Rewards Few	2.10	77.70	**39.80**	30.11	37.99
Explore Object Rewards Many	2.40	106.70	**58.70**	48.50	55.28
Explore Obstructed Goals Large	2.60	119.50	**39.50**	5.84	39.46
Explore Obstructed Goals Small	6.80	206.00	135.20	78.18	**162.72**
Language Answer Quantitative Question	-0.30	184.50	219.40	2.42	**318.62**
Language Execute Random Task	-5.90	254.10	-49.90	-1.65	**310.43**
Language Select Described Object	-0.10	389.50	324.60	600.19	**605.19**
Language Select Located Object	1.90	280.70	189.00	1.27	**655.57**
Lasertag One Opponent Large	-0.20	12.70	-0.20	**-0.01**	**-0.01**
Lasertag One Opponent Small	-0.10	18.60	-0.10	**-0.00**	-0.05
Lasertag Three Opponents Large	-0.20	18.60	-0.10	0.45	**7.27**
Lasertag Three Opponents Small	-0.10	31.50	19.10	27.42	**34.95**
Natlab Fixed Large Map	2.20	36.90	34.70	41.17	**55.05**
Natlab Varying Map Randomized	7.30	42.40	36.10	37.02	**38.97**
Natlab Varying Map Regrowth	3.00	24.40	20.70	20.44	**25.66**
Psychlab Arbitrary Visuomotor Mapping	0.20	58.80	**16.40**	13.41	16.26
Psychlab Continuous Recognition	0.20	58.30	29.90	**30.49**	30.35
Psychlab Sequential Comparison	**0.10**	39.50	0.00	0.00	0.00
Psychlab Visual Search	**0.10**	78.50	0.00	0.00	0.00
Rooms Collect Good Objects Test	0.10	10.00	**9.00**	8.61	8.99
Rooms Exploit Deferred Effects Test	8.50	85.70	15.60	**39.63**	39.45
Rooms Keys Doors Puzzle	4.10	53.80	28.00	26.27	**39.45**
Rooms Select Nonmatching Object	0.30	65.90	**7.30**	5.68	6.13
Rooms Watermaze	4.10	54.00	26.90	19.89	**31.46**
Skymaze Irreversible Path Hard V2	0.10	100.00	**13.60**	0.00	0.05
Skymaze Irreversible Path Varied V2	14.40	100.00	45.10	43.58	**68.17**

Extended Data Table 3

	Mean Score
Leaderboard 4th (“Team JustPaulsAI”)	586.06
Leaderboard 3rd (“Team Chaotic-Dwarven”)	956.73
Leaderboard 2nd (“Team KakaoBrain”)	**2062.79**
Leaderbord Best (“Team RAPH”)	1350.81
IMPALA	188.95
Disco57	938.10
Disco103	1114.24

Extended Data Table 4

Domain	Return
Random	2.10
Human	14.30
PPO	4.20
Rainbow	5.00
Dreamer	11.70
Disco57 @1M	7.58
Disco57 @20M	13.18
Disco103 @1M	6.59
Disco103 @20M	**14.56**

Extended Data Table 5

	Atari	Crafter	DMLab-30	NetHack	Procgen	Sokoban
Initial learning rate	0.0003	0.0003	0.0005	0.0001	0.0003	0.0003
End learning rate	0.0	0.0003	0	0.0	0.0	0.0
Learning rate schedule	Cosine	Constant	Cosine	Cosine	Cosine	Cosine
Learning rate warmup steps	5×10^6	0	10^6	10^7	10^6	0
Weight decay	0.1	0.3	0.01	0.03	0.1	0.1
Max abs update	1.0	1.0	1.0	0.3	1.0	1.0
Discount factor	0.997	0.997	0.99	0.997	0.997	0.997
Frame stacking	4	32	-	4	4	4
Batch size	96	24	96	96	192	96
TPU cores	16	8	8	16	8	8
Environment steps	2×10^8	5×10^7	10^9	2×10^8	2×10^8	2×10^8
Replay buffer size	1760000	400000	220000	1760000	880000	880000
Replay ratio	0.95	0.99	0.9	0.95	0.9	0.9
Unroll length	29	29	100	29	29	29

Extended Data Table 6