OpenAI

Parameter Space Noise for Exploration

Matthias Plappert

APRIL 10, 2018

Agenda

- OpenAI
- Robotics at OpenAI
- A Brief Introduction to Reinforcement Learning
- Parameter Space Noise for Exploration

About Me

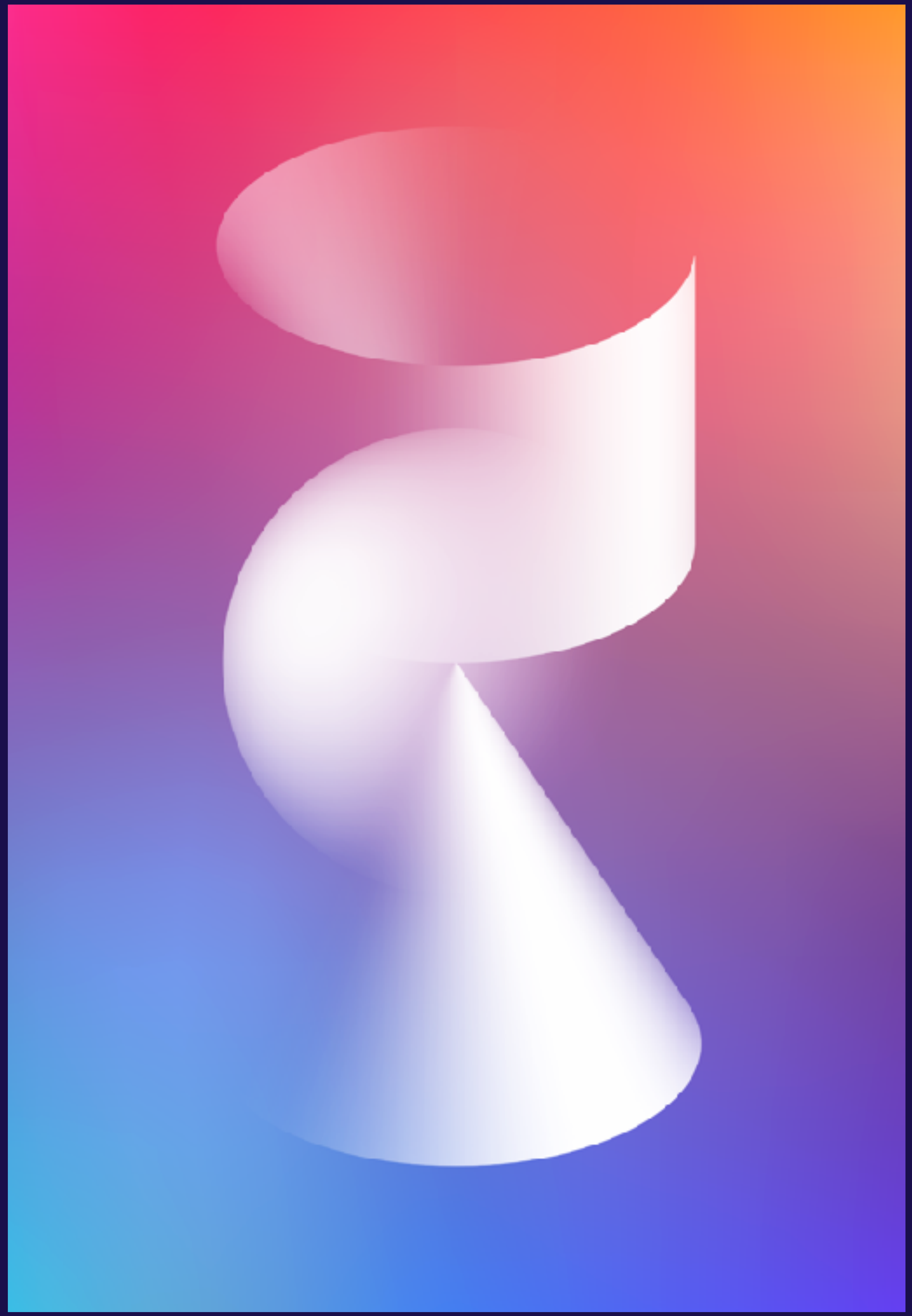
- Matthias Plappert
- since 2017: Research at OpenAI Robotics
- 2011 - 2017: Computer Science at Karlsruhe Institute of Technology (B.Sc. and M.Sc.)
- Before that and in between: iOS software development
- I like hiking, camping, running, and climbing

OpenAI's Mission

“OpenAI is a non-profit AI research company, discovering and enacting the path to safe artificial general intelligence.”

OpenAI Charter

- Broadly Distributed Benefits
- Long-Term Safety
- Technical Leadership
- Cooperative Orientation
- Full text available on our blog: https://blog.openai.com/openai-charter/



Releases

GYM RETRO

INGREDIENTS FOR ROBOTICS RESEARCH

Future of Humanity Institute
University of Oxford
Centre for the Study of Existential Risk
University of Cambridge
Center for a New American Security
Electronic Frontier Foundation
OpenAI

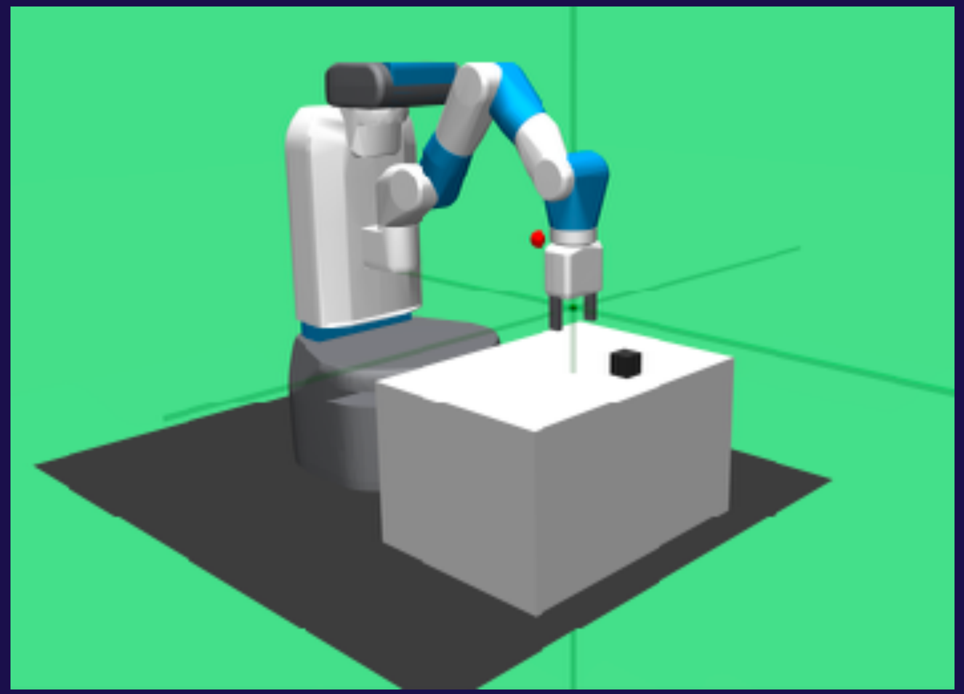
February 2018

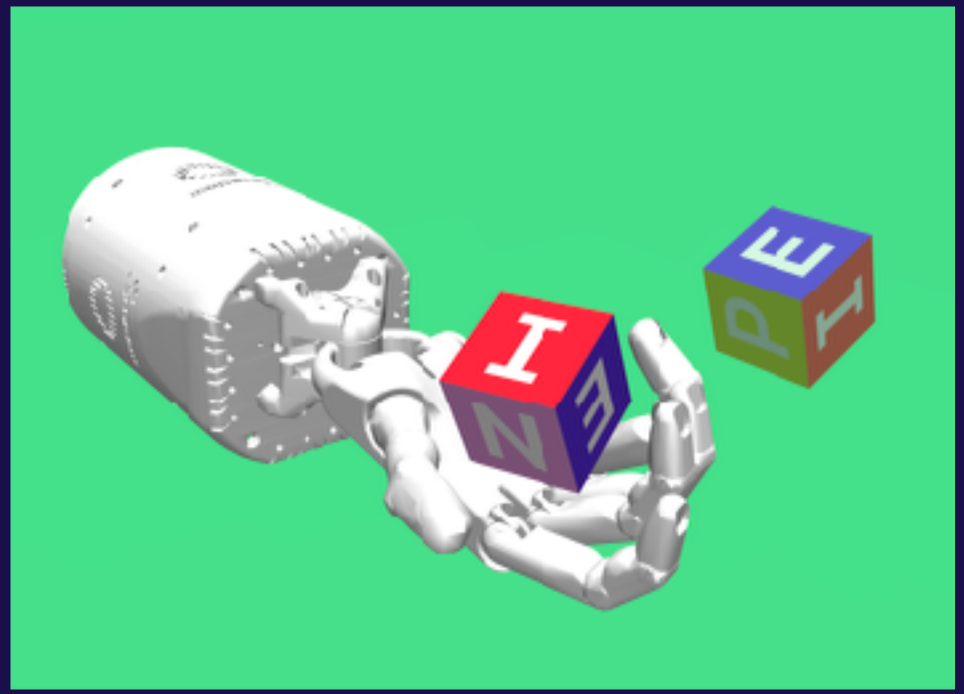
The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation

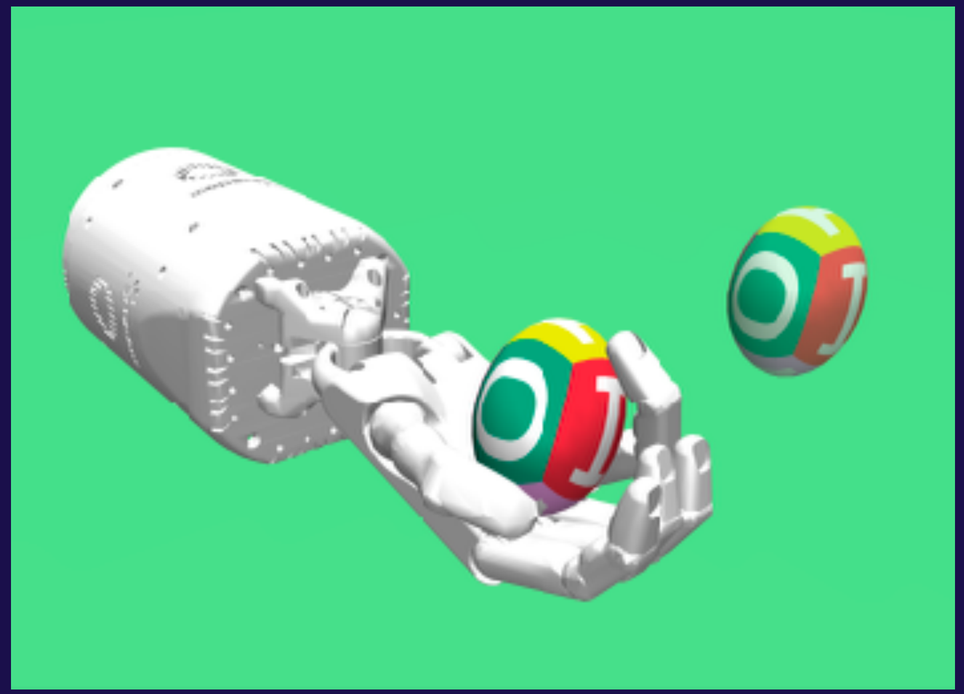
PREPARING FOR MALICIOUS USES OF AI

Robotics Release – Environments

- Realistic robot environments



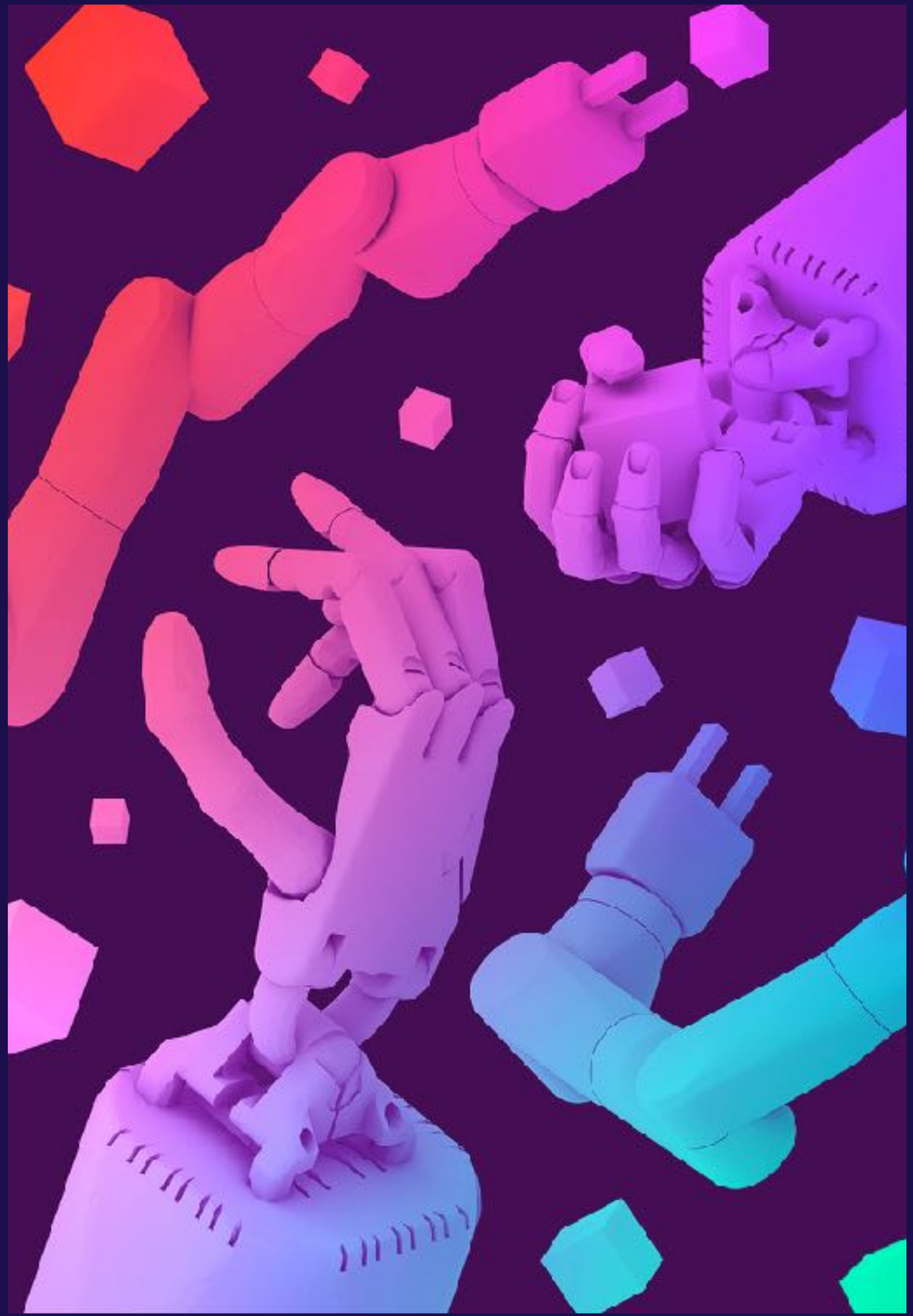




- Integrated with OpenAI Gym

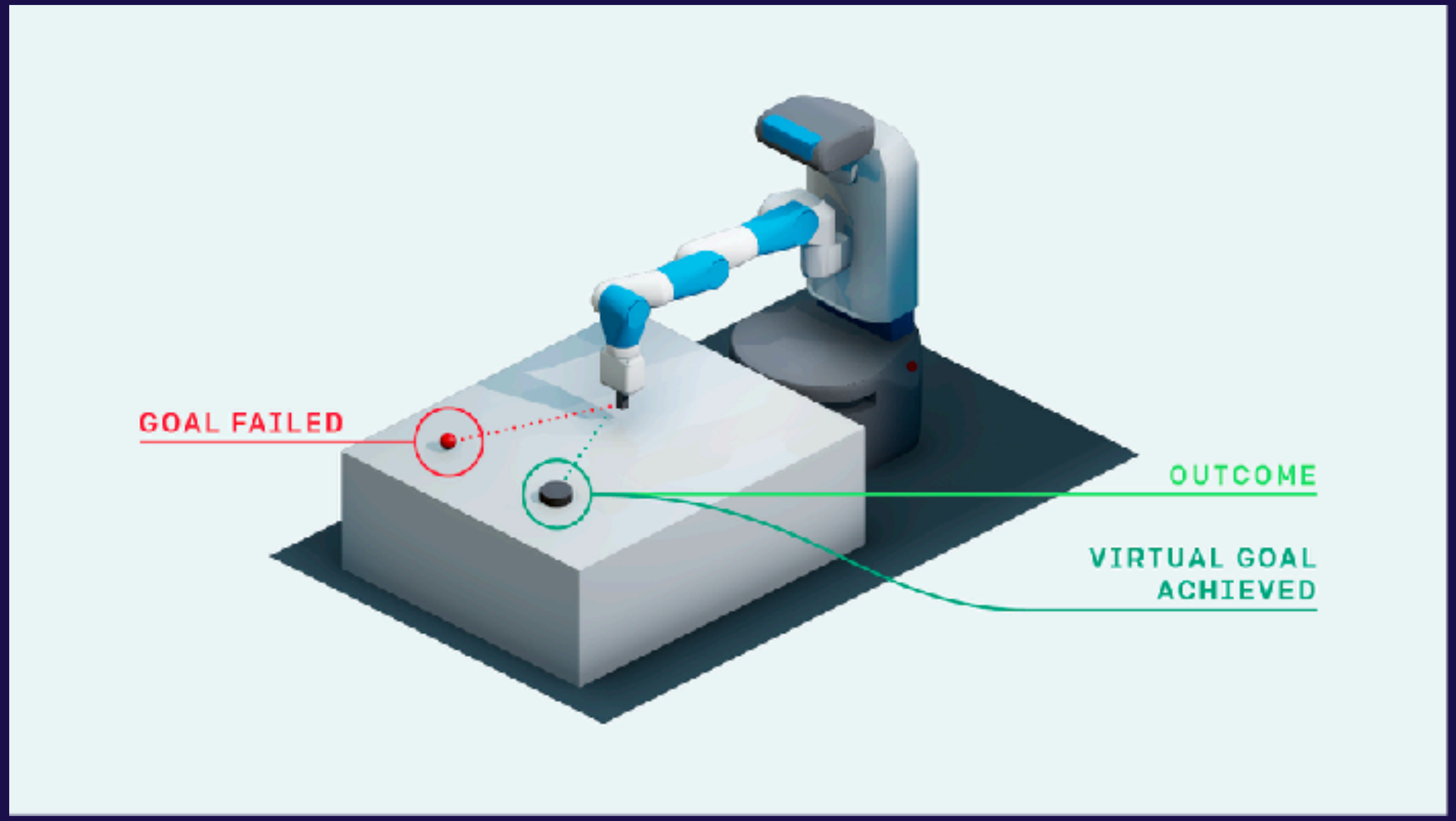
- Goal-based formulation

- Sparse rewards



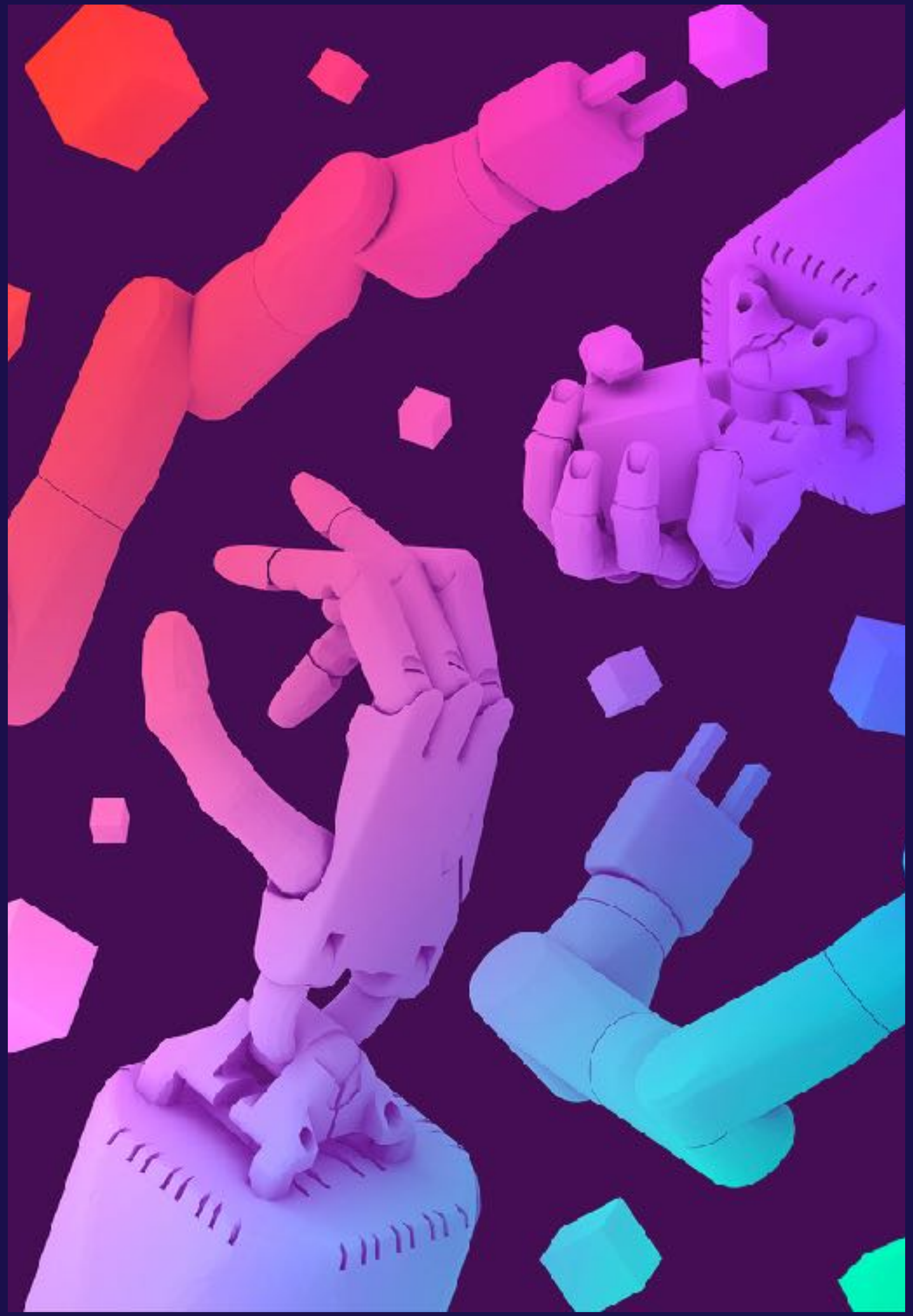
Robotics Release – Hindsight Experience Replay

- Learning from failed attempts

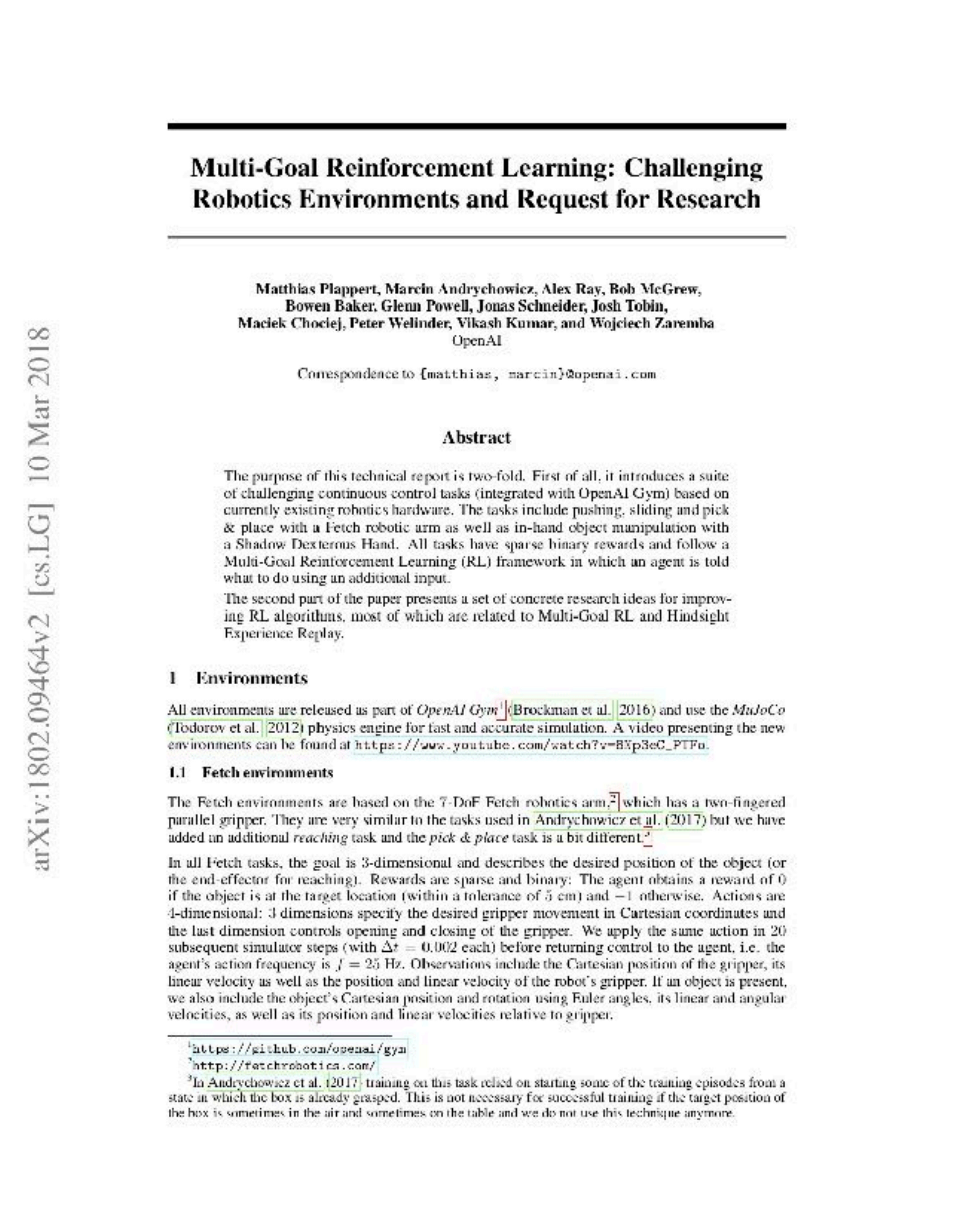


- Can be combined with any off-policy RL algorithm

- Allows us to solve sparse tasks



Robotics Release – Request for Research



arXiv:1802.09464v2 [cs.LG] 10 Mar 2018

Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba
OpenAI

Correspondence to {matthias, marcin}@openai.com

Abstract

The purpose of this technical report is two-fold. First of all, it introduces a suite of challenging continuous control tasks (integrated with OpenAI Gym) based on currently existing robotics hardware. The tasks include pushing, sliding and pick & place with a Fetch robotic arm as well as in-hand object manipulation with a Shadow Dexterous Hand. All tasks have sparse binary rewards and follow a Multi-Goal Reinforcement Learning (RL) framework in which an agent is told what to do using an additional input.

The second part of the paper presents a set of concrete research ideas for improving RL algorithms, most of which are related to Multi-Goal RL and Hindsight Experience Replay.

1 Environments

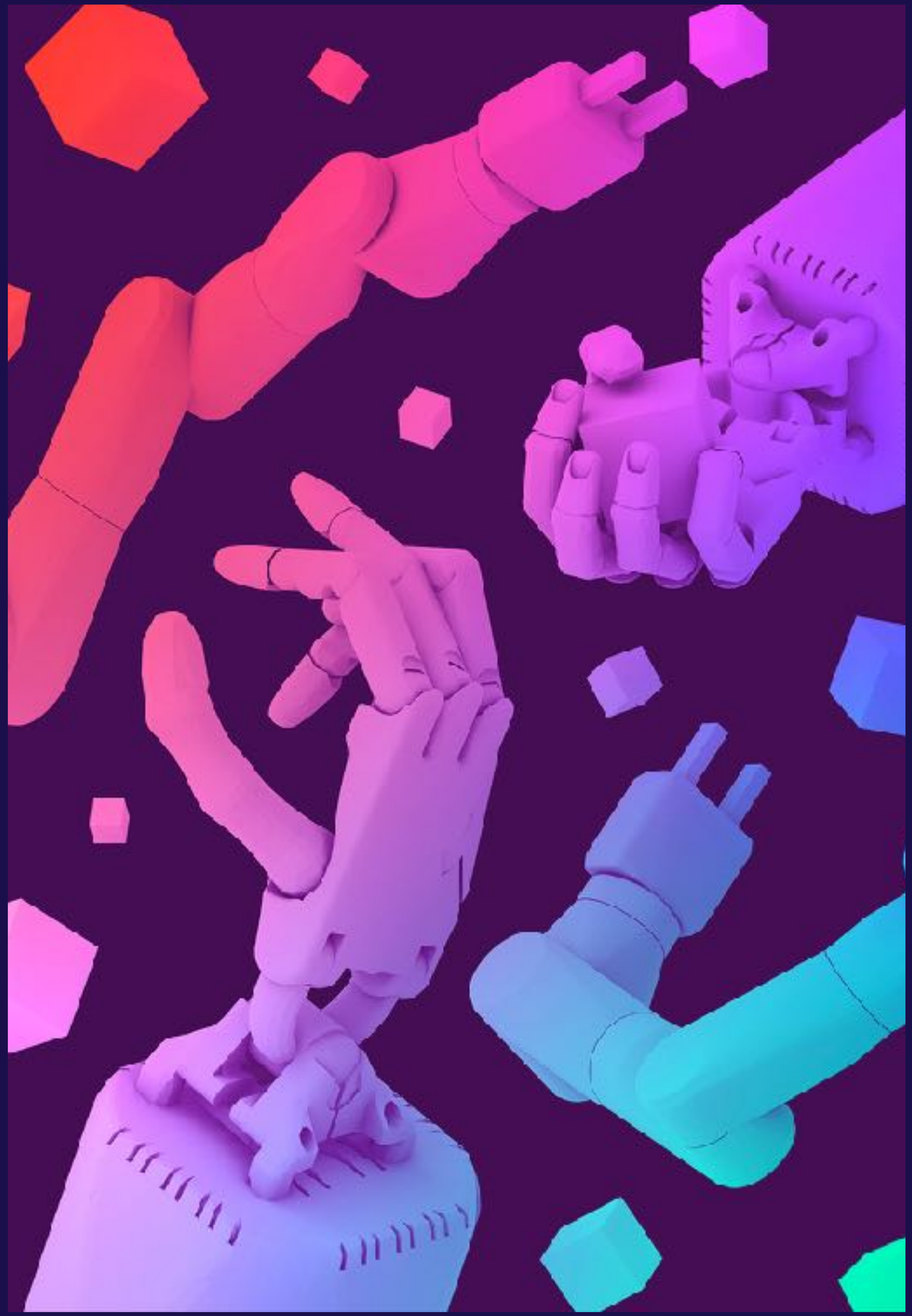
All environments are released as part of *OpenAI Gym*[1] (Brockman et al. 2016) and use the *MuJoCo* (Todorov et al. 2012) physics engine for fast and accurate simulation. A video presenting the new environments can be found at https://www.youtube.com/watch?v=8Xp3eC_PTFo.

1.1 Fetch environments

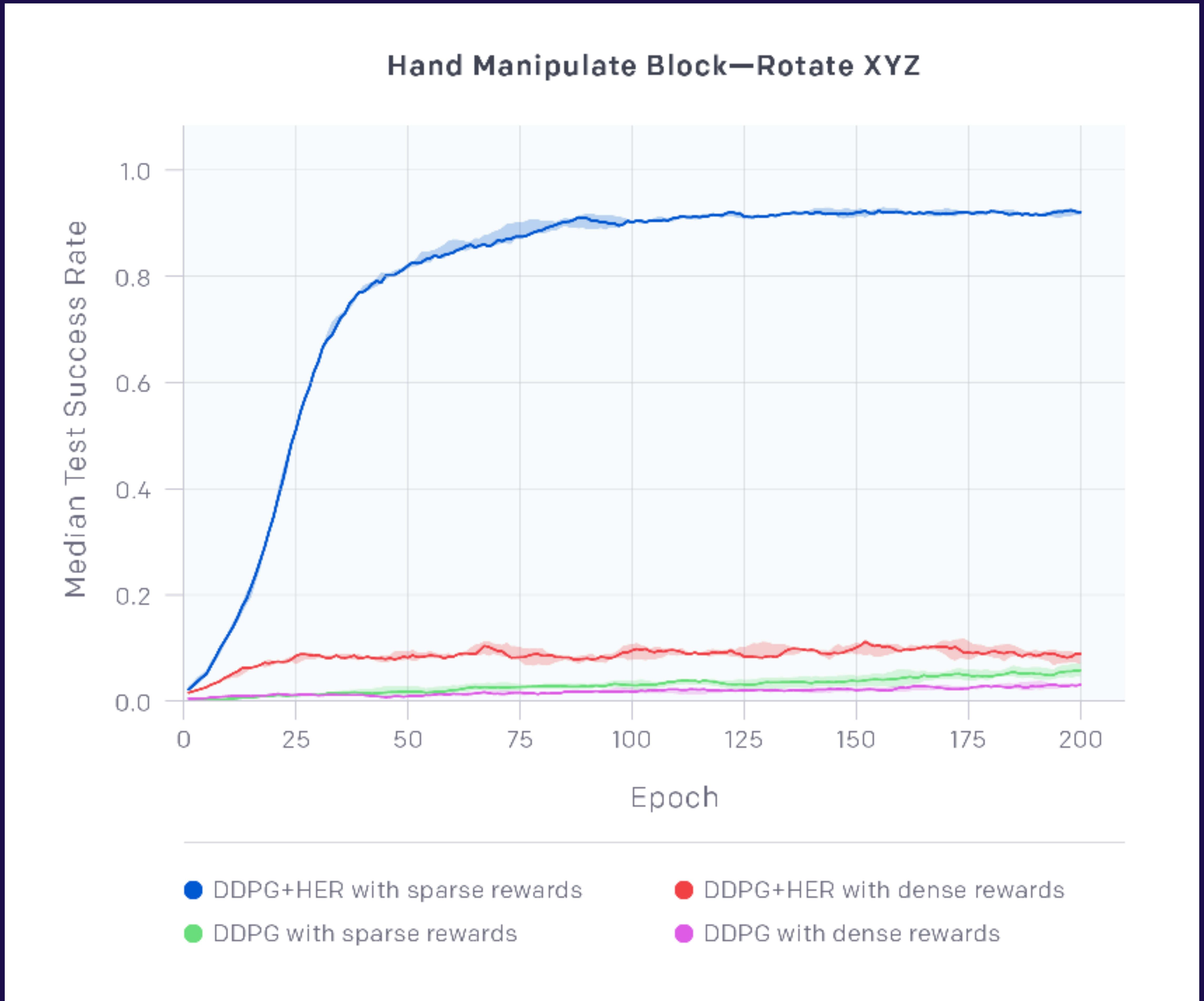
The Fetch environments are based on the 7-DoF Fetch robotics arm,[2] which has a two-fingered parallel gripper. They are very similar to the tasks used in Andrychowicz et al. (2017) but we have added an additional *reaching* task and the *pick & place* task is a bit different.[3]

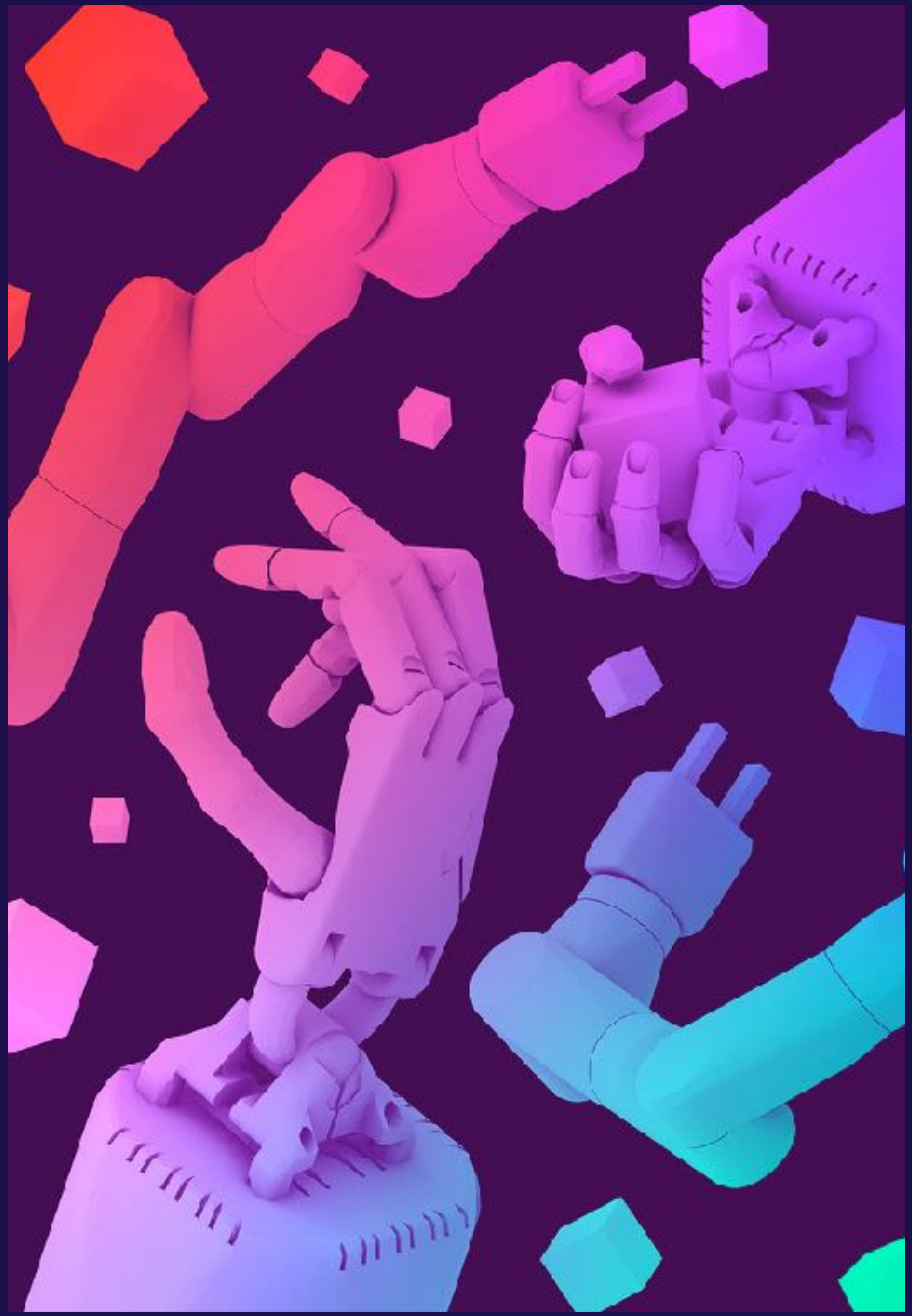
In all Fetch tasks, the goal is 3-dimensional and describes the desired position of the object (or the end-effector for reaching). Rewards are sparse and binary: The agent obtains a reward of 0 if the object is at the target location (within a tolerance of 5 cm) and -1 otherwise. Actions are 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates and the last dimension controls opening and closing of the gripper. We apply the same action in 20 subsequent simulator steps (with $\Delta t = 0.002$ each) before returning control to the agent, i.e. the agent's action frequency is $f = 25$ Hz. Observations include the Cartesian position of the gripper, its linear velocity as well as the position and linear velocity of the robot's gripper. If an object is present, we also include the object's Cartesian position and rotation using Euler angles, its linear and angular velocities, as well as its position and linear velocities relative to gripper.

[1] https://github.com/openai/gym
[2] http://fetchrobotics.com/
[3] In Andrychowicz et al. (2017) training on this task relied on starting some of the training episodes from a state in which the box is already grasped. This is not necessary for successful training if the target position of the box is sometimes in the air and sometimes on the table and we do not use this technique anymore.

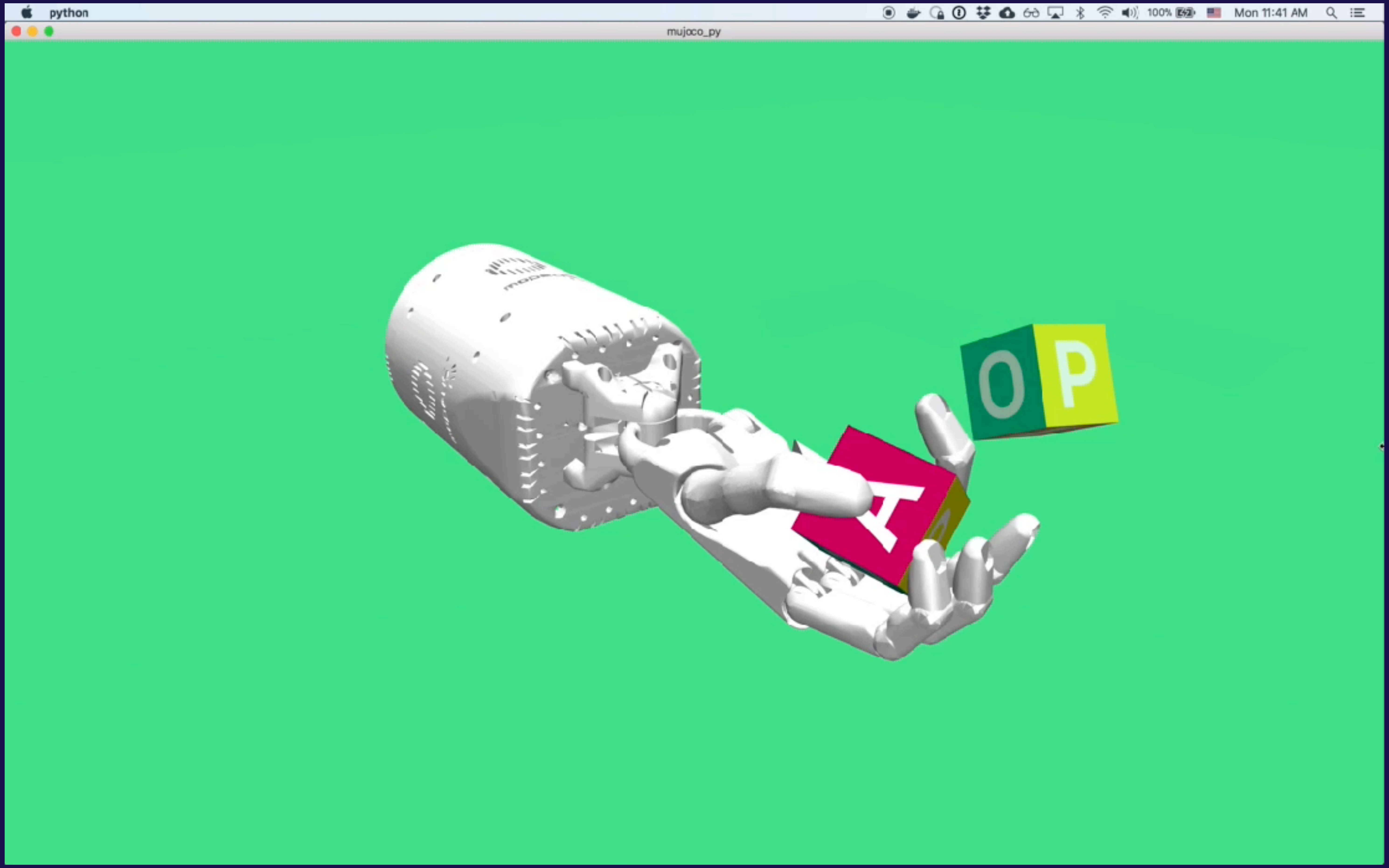


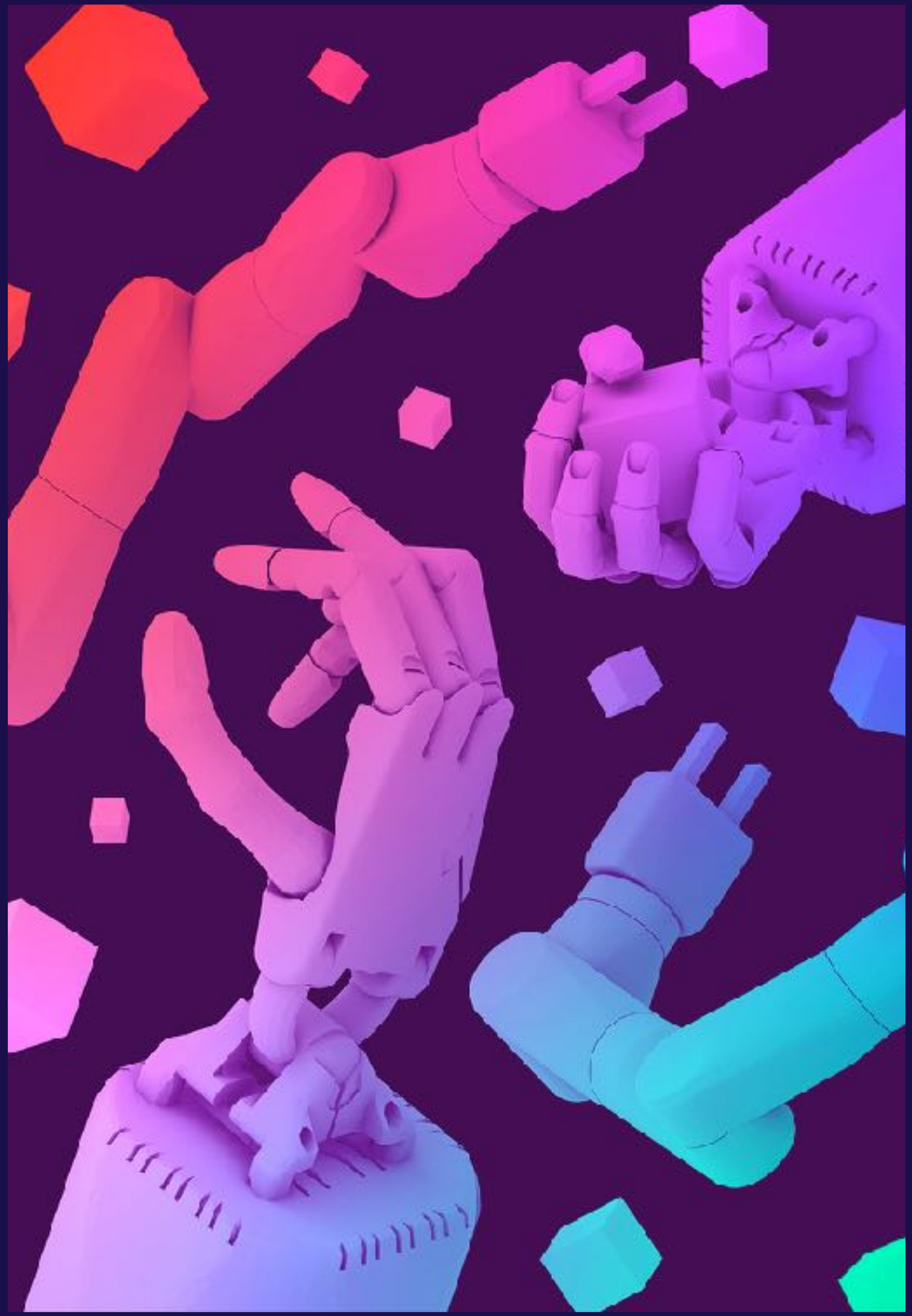
Robotics Release – Results





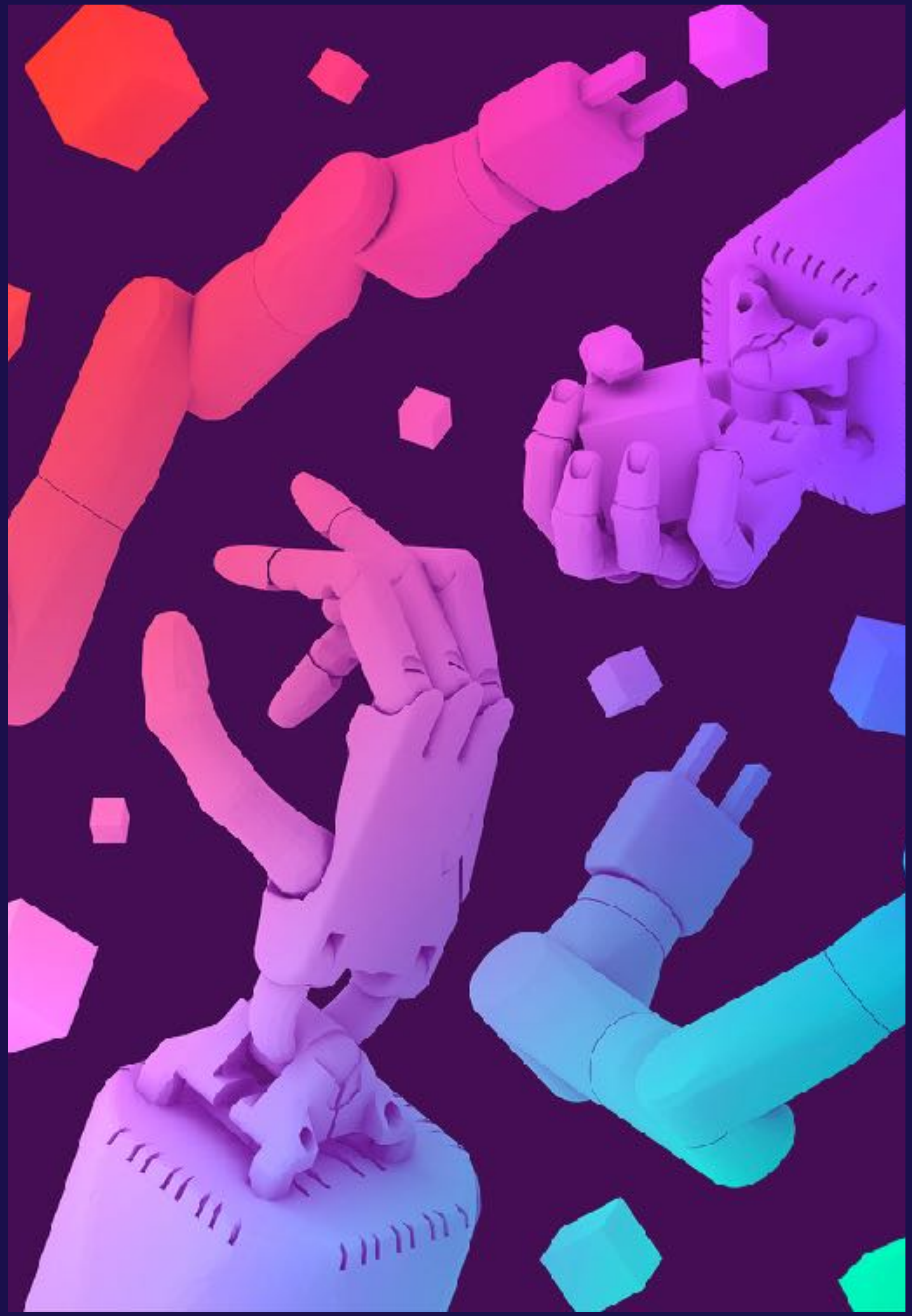
Robotics Release – Results



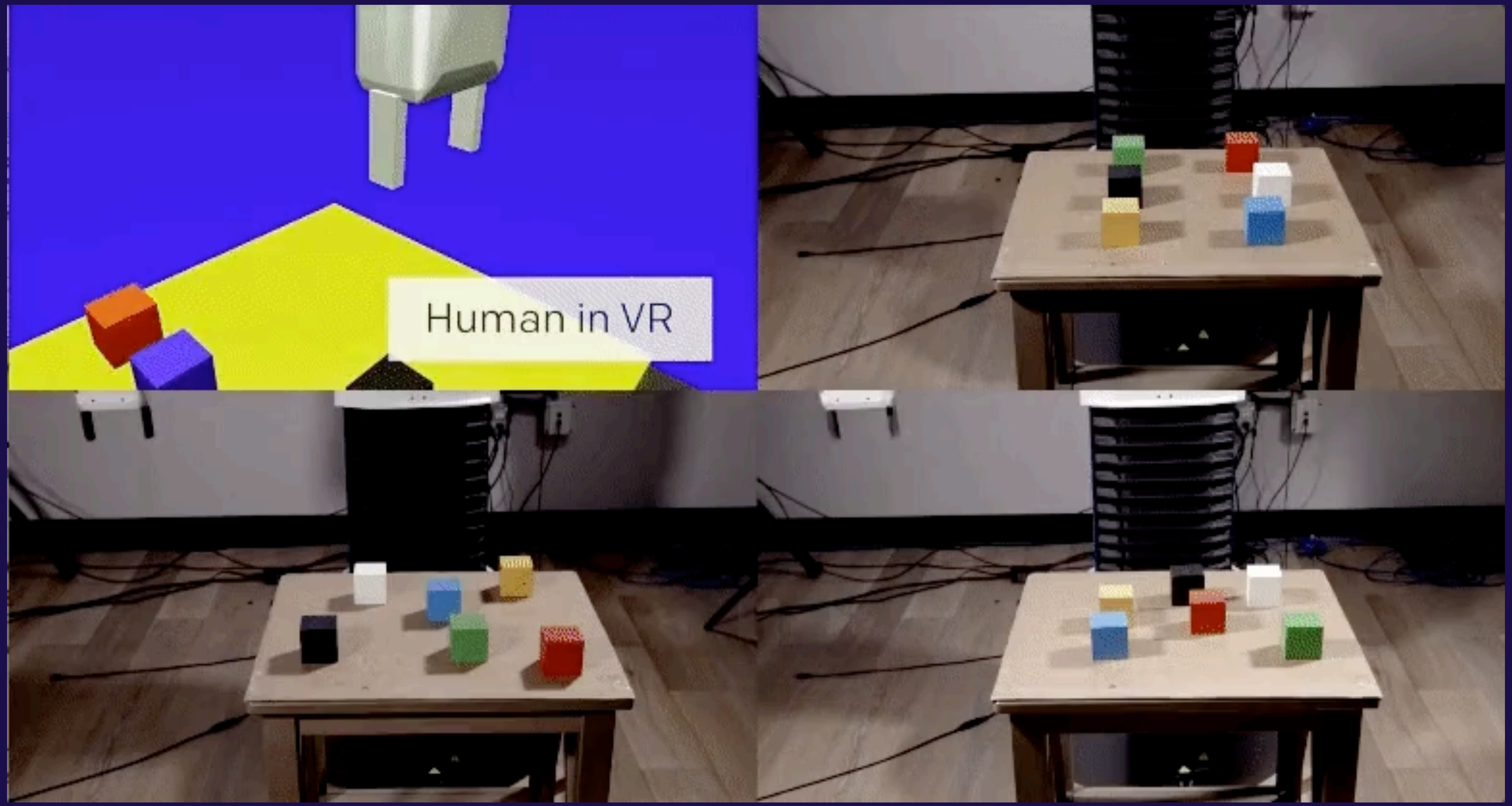


Robotics Release – Further Reading

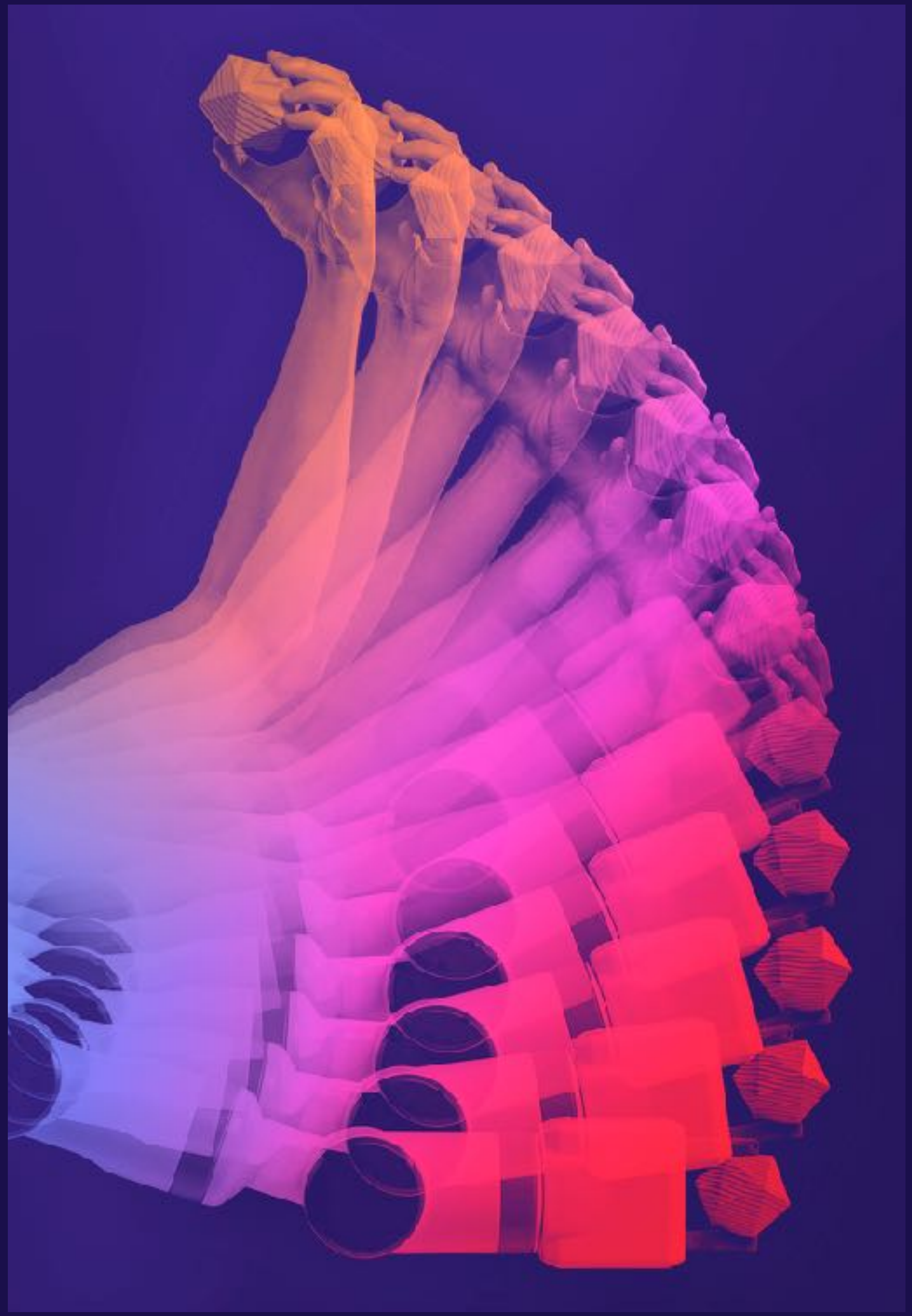
- Blog post https://blog.openai.com/ingredients-for-robotics-research/
- "Hindsight Experience Replay", Andrychowicz et al., 2017
- "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research", Plappert et al., 2018



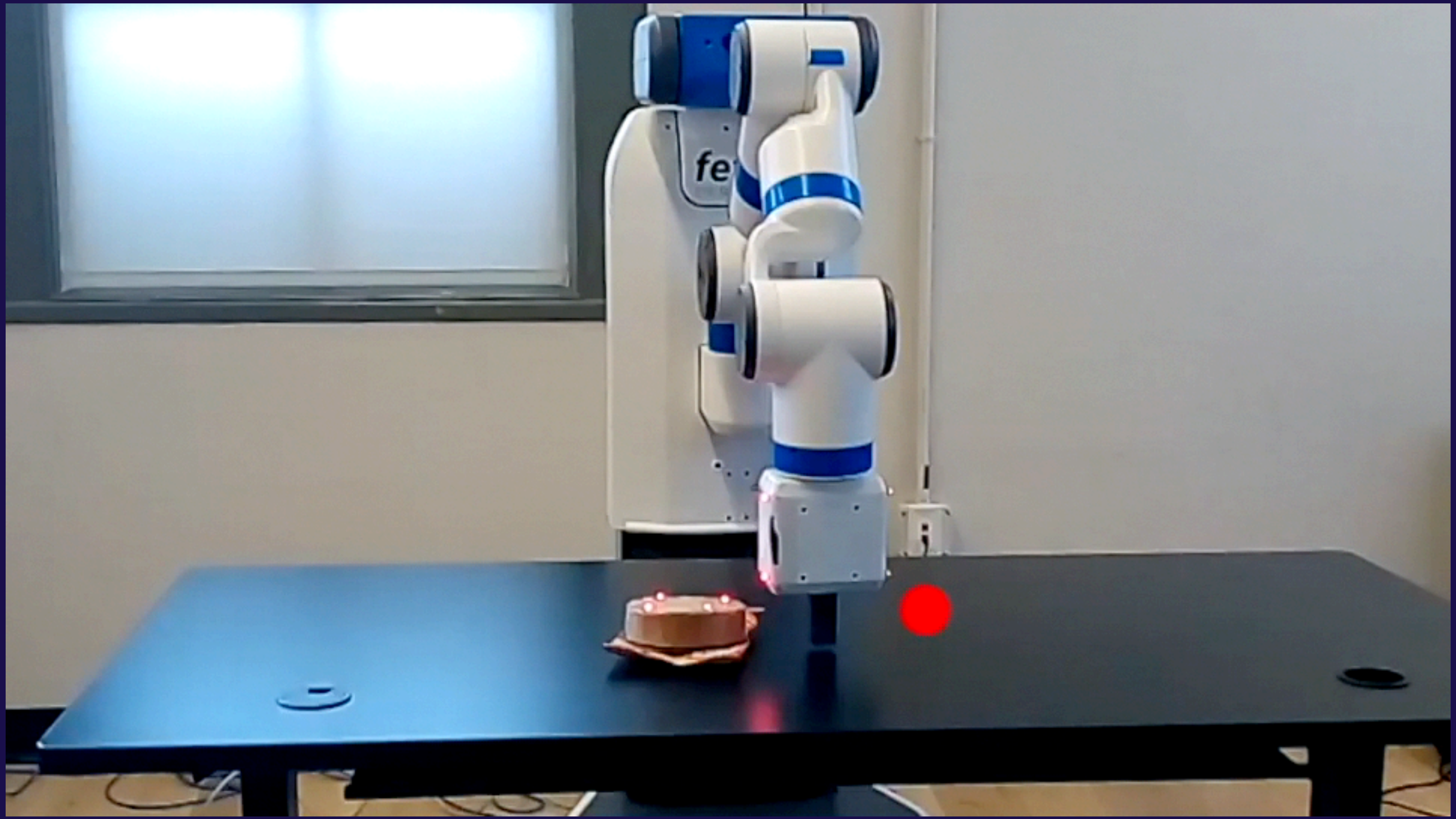
Robotics – Robots that Learn



- "One-Shot Imitation Learning", Duan et al., 2017
- "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World", Tobin et al., 2017



Robotics – Generalizing from Simulation

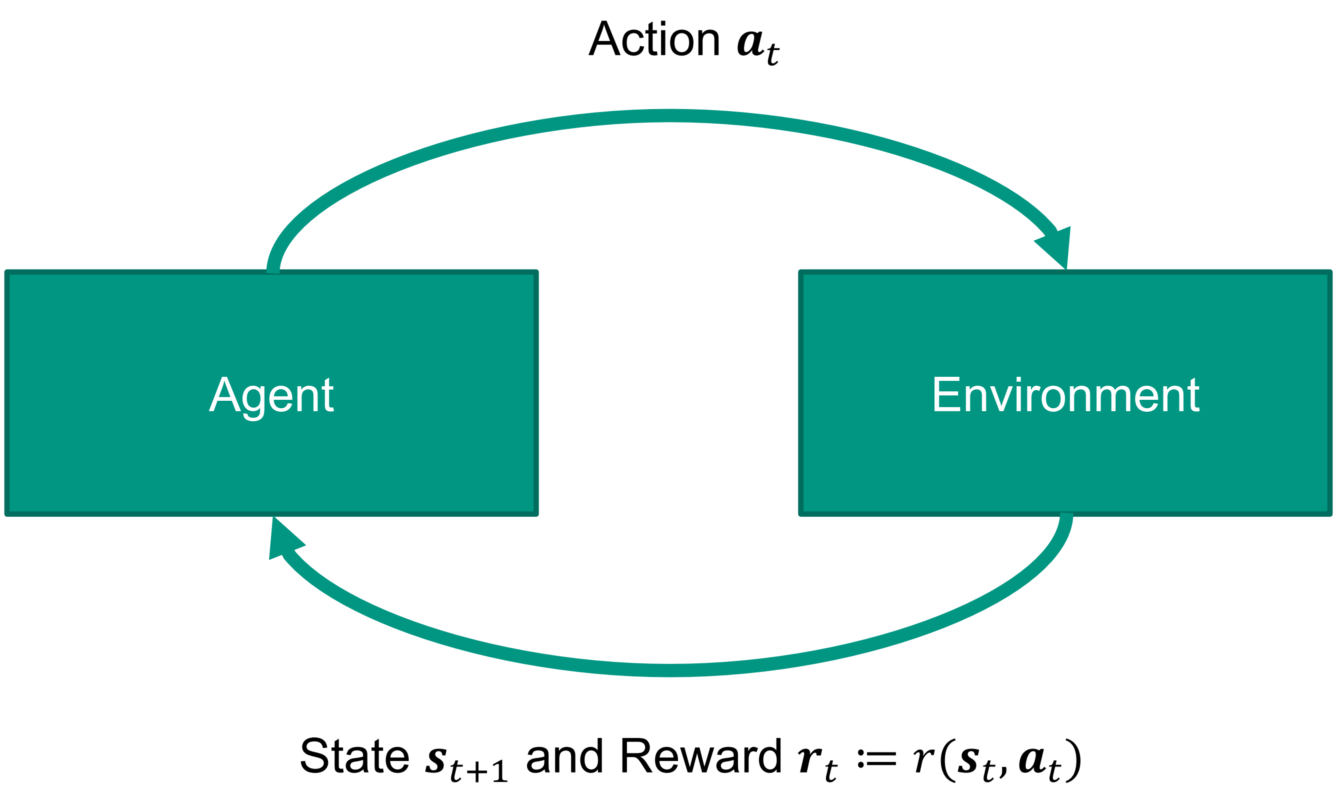


- "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization", Peng et al., 2017
- "Asymmetric Actor Critic for Image-Based Robot Learning", Pinto et al., 2017

Parameter Space Noise for Exploration

A Brief Introduction to Reinforcement Learning

Reinforcement Learning (1)



Reinforcement Learning (2)

- Formalize as Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, r)$ with
  - Set of states $\mathcal{S}$
  - Set of actions $\mathcal{A}$
  - Reward function r: $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
  - Initial state distribution $\boldsymbol{s}_0 \sim \rho(\cdot)$
  - State transition distribution: $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$

- Agent uses a policy to select actions:

$$\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$$

Reinforcement Learning (3)

- Let τ denote a trajectory with $\boldsymbol{s}_0 \sim \rho(\cdot)$, $\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$, $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$

- The discounted return is then defined as:

$$\mathrm{R}(\tau) \coloneqq \sum_t \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\,, \text{with } \gamma \in [0, 1)$$

- We wish to find a policy π^* that maximizes the expected discounted return:

$$\pi^* \coloneqq \mathrm{argmax}_\pi \mathbb{E}_\tau [R(\tau)]$$

- Also notice that we have to discover states with high rewards since we assume no a-priori information about our environment
  → requires exploration

Policy Gradients (1)

- Let's assume a parameterized policy $\pi_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ is some parameter vector

- Rewrite our optimization objective as:

$$\mathrm{J}(\boldsymbol{\theta}) = \mathbb{E}_{\tau}[R(\tau)],$$

  where $\mathbb{E}_{\tau}$ has a dependency on $\boldsymbol{\theta}$ through $\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)$ but, importantly, $R(\tau)$ has no dependence on $\boldsymbol{\theta}$ (it simply computes the return of any given trajectory)

- Simple idea: Let's compute the gradient and do gradient ascent on this objective!

Policy Gradients (2)

- We can expand the expectation as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathrm{J}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_\tau[R(\tau)] \\ &= \int R(\tau)\, \nabla_{\boldsymbol{\theta}} p(\tau) \mathrm{d}\tau\end{aligned}$$

- Furthermore, we can use the log derivative trick:

$$\nabla_{\boldsymbol{\theta}} \log p(\tau) = \frac{1}{p(\tau)} \nabla_{\boldsymbol{\theta}} p(\tau)$$

$$\nabla_{\boldsymbol{\theta}} p(\tau) = \nabla_{\boldsymbol{\theta}} \log p(\tau)\, p(\tau)$$

- Plugging this back in, we obtain:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathrm{J}(\boldsymbol{\theta}) &= \int R(\tau)\, \nabla_{\boldsymbol{\theta}} \log p(\tau)\, p(\tau) \\ &= \mathbb{E}_\tau[R(\tau) \nabla_{\boldsymbol{\theta}} \log p(\tau)]\end{aligned}$$

Policy Gradients (3)

- So we just have to worry about computing $\nabla_{\boldsymbol{\theta}} \log p(\tau)$!
- $p(\tau) = \rho(\boldsymbol{s}_0) \prod_t \mathcal{P}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)\, \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$
- Taking the log:
  $\log p(\tau) = \log \rho(\boldsymbol{s}_0) + \sum_t \log \mathcal{P}(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t) + \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t)$
- Only π has a dependence on $\boldsymbol{\theta}$, thus we obtain:
  $$\nabla_{\boldsymbol{\theta}} \log p(\tau) = \sum_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t)$$
- $\nabla_{\boldsymbol{\theta}} \mathrm{J}(\boldsymbol{\theta}) = \mathbb{E}_\tau[R(\tau) \nabla_{\boldsymbol{\theta}} \log p(\tau)]$
  $= \mathbb{E}_\tau[R(\tau) \sum_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t)]$

Policy Gradients (4)

- The expectation can be estimated using Monte Carlo sampling, which corresponds to rolling out the policy multiple times to collect N trajectories: $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(N)}$

- The estimate for the policy gradient is thus:

$$\hat{\boldsymbol{g}} = \frac{1}{N} \sum_{n} \left[R\left(\tau^{(n)}\right) \sum_{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{a}_t^{(n)} \middle| \boldsymbol{s}_t^{(n)} \right) \right]$$

Q-learning (1)

- Policy gradients work but …
  - gradient estimates have large variance
  - throw away all data after every policy update
- Introduce the state-action value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$
- Informally, $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ gives us the expected discounted return if we take action **a** in state **s** and follow π afterwards
- Define $\mathcal{T}^\pi$ as the Bellman operator:
$$(\mathcal{T}^\pi Q)(\boldsymbol{s}, \boldsymbol{a}) = r(s, a) + \gamma\, \mathbb{E}_\pi[Q(\boldsymbol{s}', \boldsymbol{a}')]$$
- $\mathcal{T}^\pi$ is a contraction and applying it iteratively convergences to the fixed point $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$, for any given initial Q! (Banach fixed-point theorem)

Q-learning (2)

- Assuming we have the optimal Q^*, the optimal policy is given by π^*:
$$\pi^*(\boldsymbol{s}) = \text{argmax}_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$$

- Conversely, $\mathcal{T}^*$ has the same properties as $\mathcal{T}^\pi$ and we can therefore compute Q^* by iteratively applying:

$$\begin{aligned}(\mathcal{T}^* Q)(\boldsymbol{s}, \boldsymbol{a}) &= r(s, a) + \gamma\, \mathbb{E}_{\pi^*}[Q(\boldsymbol{s}', \boldsymbol{a}')] \\ &= r(s, a) + \gamma\, \text{argmax}_{\boldsymbol{a}'} Q^*(\boldsymbol{s}', \boldsymbol{a}')\end{aligned}$$

- Still need to explore in order to “see” entire state space

- Tabular case breaks down as both state and action space become large
→ approximate Q, e.g. using neural networks

Deep Q-Networks

- DQN, Mnih et al., 2013 & 2015
- Parameterize Q-function using a deep neural network and learn:
  $$Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a}) = r(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} Q_{\boldsymbol{\theta}}(\boldsymbol{s}',\boldsymbol{a}')$$
  i.e. it uses Q-learning
- Policy is defined implicitly: $\pi(\boldsymbol{s}) \coloneqq \operatorname{argmax}_{\boldsymbol{a}'} Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a}')$
- Works with discrete action spaces
- Typically uses ε-greedy exploration:
  - With probability ε, select a random action
  - Otherwise, select $\pi(\boldsymbol{s})$
- No formal convergence guarantees anymore due to approximate nature

Deep Deterministic Policy Gradient

- DDPG, Lillicrap et al., 2015
- What if we want to use DQN but have a continuous action space?
- DDPG uses actor-critic architecture with two networks:

$$Q_{\boldsymbol{\phi}}(\boldsymbol{s},\boldsymbol{a}) = r(\boldsymbol{s},\boldsymbol{a}) + \gamma\, Q_{\boldsymbol{\phi}}(\boldsymbol{s}',\pi_{\boldsymbol{\theta}}(\boldsymbol{s}'))$$

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \arg\max_{\pi_{\boldsymbol{\theta}'}} Q_{\boldsymbol{\phi}}\big(\mathbf{s},\pi_{\boldsymbol{\theta}'}(\boldsymbol{s})\big)$$

- Typically uses additive Gaussian exploration:

$$\hat{\pi}(\boldsymbol{s}) \coloneqq \pi_{\boldsymbol{\theta}}(\boldsymbol{s}) + \mathcal{N}(\mathbf{0},\sigma^2\boldsymbol{I})$$

Further Reading

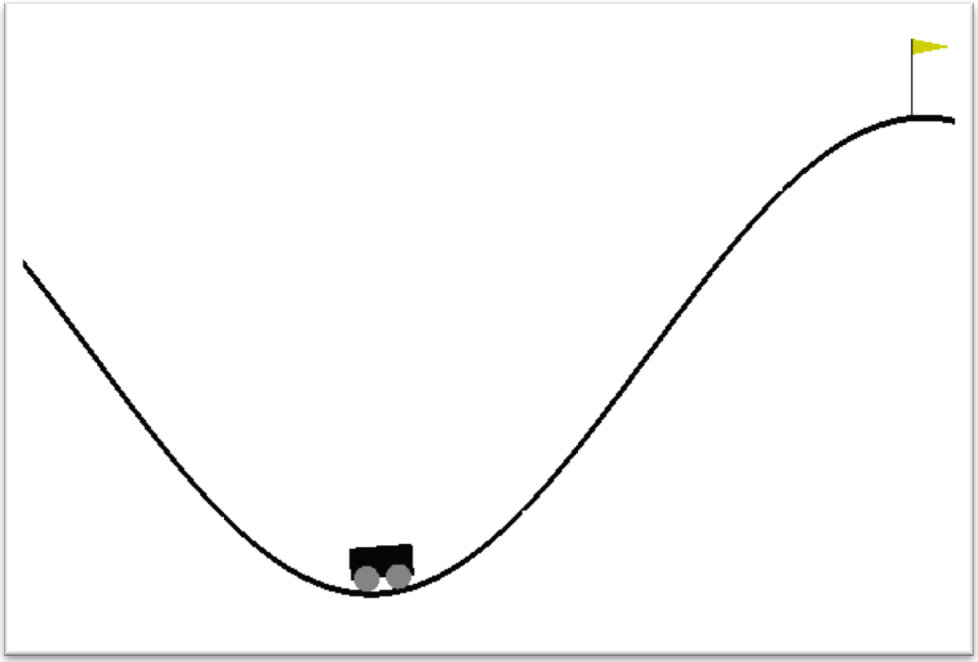
- “Reinforcement Learning: An Introduction”, Barto and Sutton, http://incompleteideas.net/book/the-book-2nd.html
- “Deep Reinforcement Learning”, Levine, http://rll.berkeley.edu/deeprlcourse/
- Selected deep RL papers
  - DQN: https://www.nature.com/articles/nature14236
  - DDPG: https://arxiv.org/abs/1509.02971
  - TRPO: https://arxiv.org/abs/1502.05477
  - PPO: https://arxiv.org/abs/1707.06347

Parameter Space Noise for Exploration

Motivation

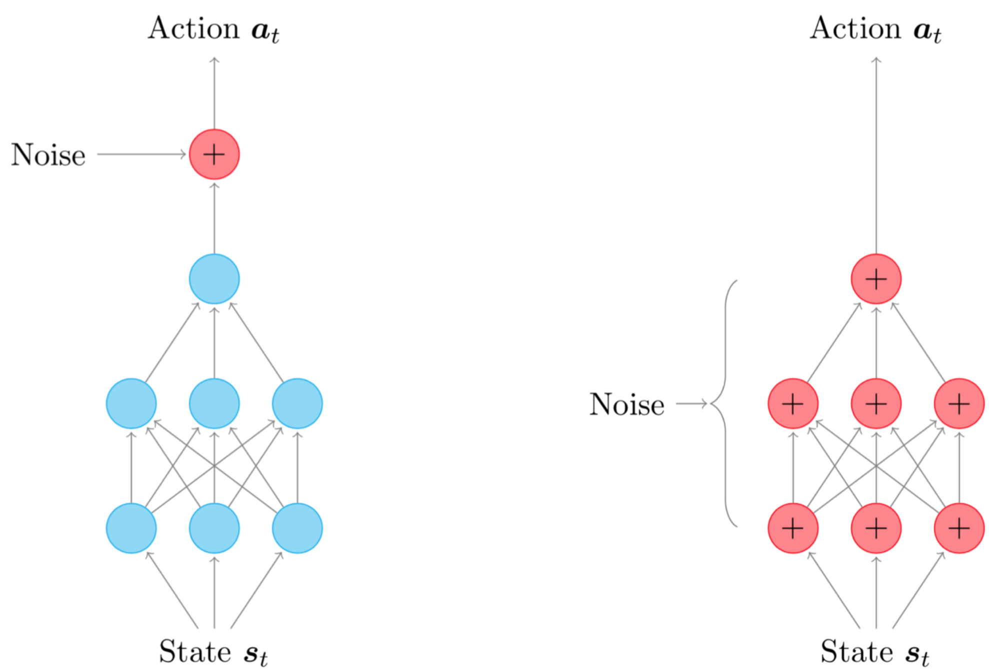
- Typically, exploration is realized in the action space:
$$\hat{\pi}(\boldsymbol{s}) \coloneqq \pi_{\boldsymbol{\theta}}(\boldsymbol{s}) + \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$$

- However, this leads to inconsistent exploration since the noise is not conditioned on the state



Formulation

- What if we apply noise to the parameters of the policy instead?



(a) *Action space noise* (b) *Parameter space noise*

- Define $\hat{\pi}(\boldsymbol{s}) := \pi_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{s})$ with $\widehat{\boldsymbol{\theta}} := \boldsymbol{\theta} + \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$

Problems

- Recall that $\widehat{\boldsymbol{\theta}} \coloneqq \boldsymbol{\theta} + \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$

- We use a scalar σ to perturb the weights of a deep network (Problem 1)
  - Such a network will likely have many layers
  - Each layer likely has different sensitivities to noise

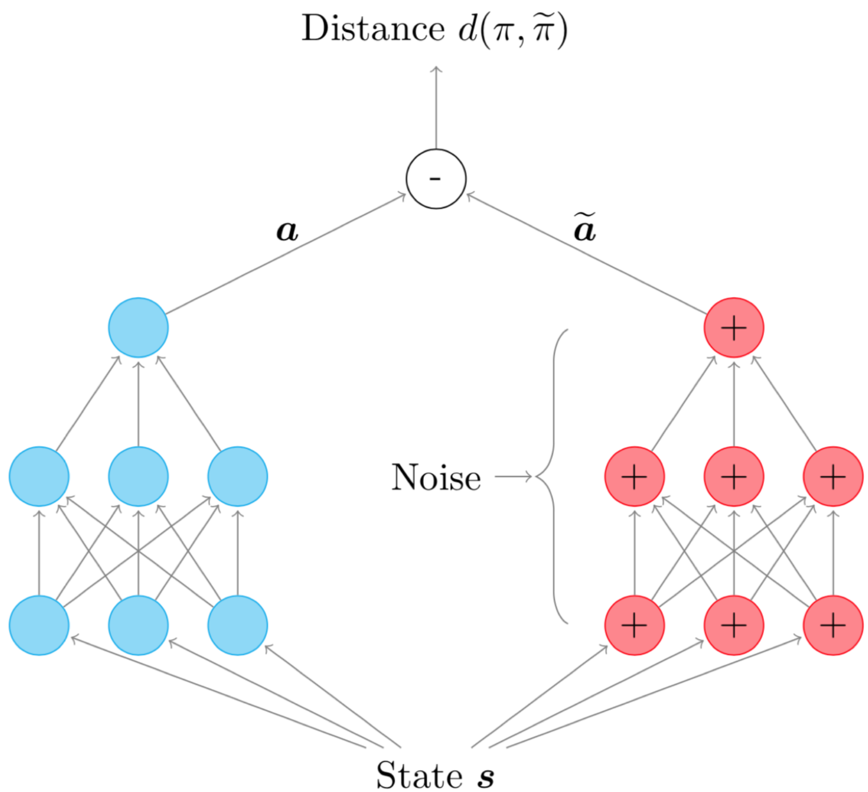
- We have to pick a suitable scalar σ (Problem 2)
  - In action space noise, the effect is intuitively understandable
  - In contrast, what does perturbing the weights of the policy mean?
  - Furthermore, the sensitivity of the policy to perturbations is likely changing as training progresses

Problem 1

- Use a similar re-parameterization as proposed in Salimans et al., 2017
- We use layer normalization (Ba et al., 2016)
$$\boldsymbol{h} = f\left[\frac{\boldsymbol{g}}{\sigma} \odot (\boldsymbol{a} - \mu) + \boldsymbol{b}\right]$$
with $\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x}$ and μ and σ are estimated over $\boldsymbol{a}$
- **h** ends up with approximately zero mean and unit variance
- Perturbation to $\boldsymbol{W}$ becomes invariant to sensitivity of that layer
- Hinges on assumption that units within layer are similar

Problem 2

- Reasoning about σ in parameter space is hard
- Idea: Think about the effect of a perturbation in action space:



Problem 2

- Reasoning about σ in parameter space is hard
- Idea: Think about the effect of a perturbation in action space:

$$d_k := \mathbb{E}_{\boldsymbol{s}}[d(\pi(\cdot \mid \boldsymbol{s}), \hat{\pi}(\cdot \mid \boldsymbol{s}))]$$

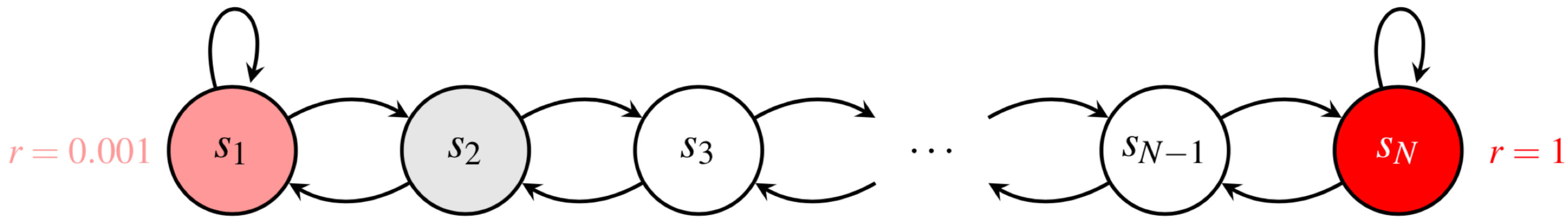
  using some distance / divergence measure $d(\cdot,\cdot)$

- Adaptively change σ:

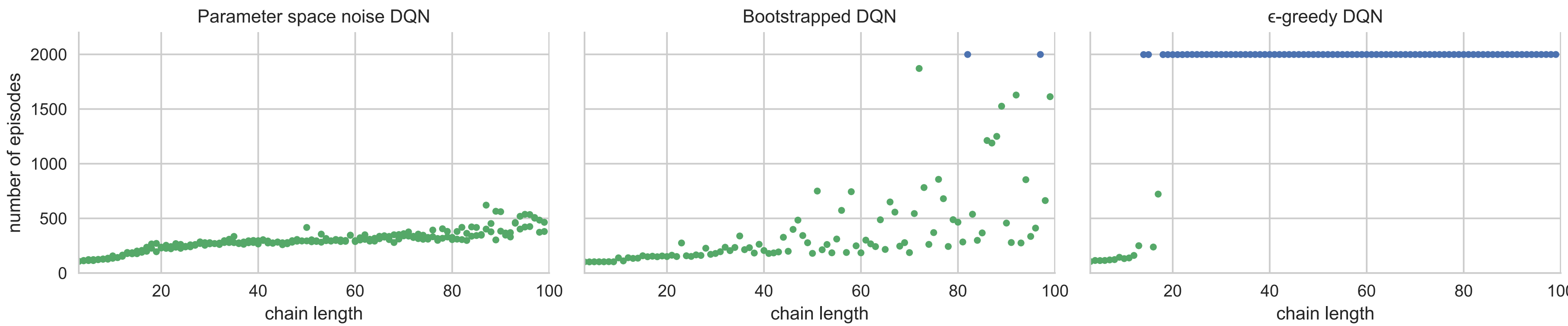
$$\sigma_{k+1} = \begin{cases} \alpha \sigma_k, & d_k \leq \delta \\ \frac{1}{\alpha} \sigma_k, & \text{otherwise} \end{cases}$$

Experiments (1)

- We test for exploration on a simple but scalable toy environment

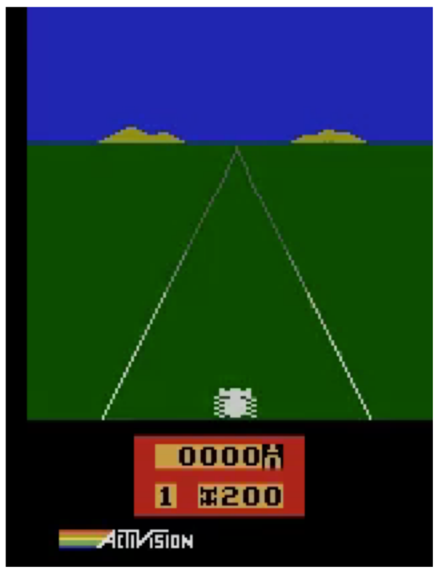


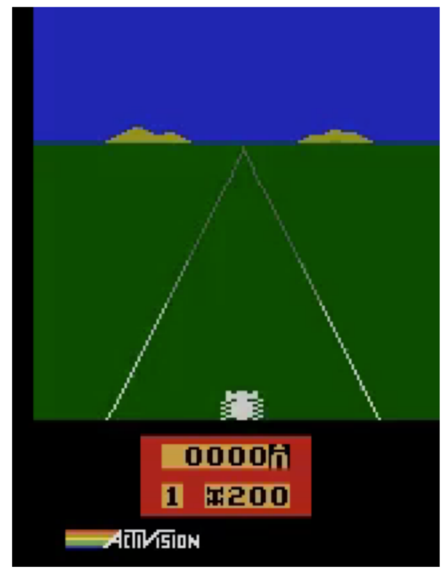
- Experiments on DQN with different exploration methods



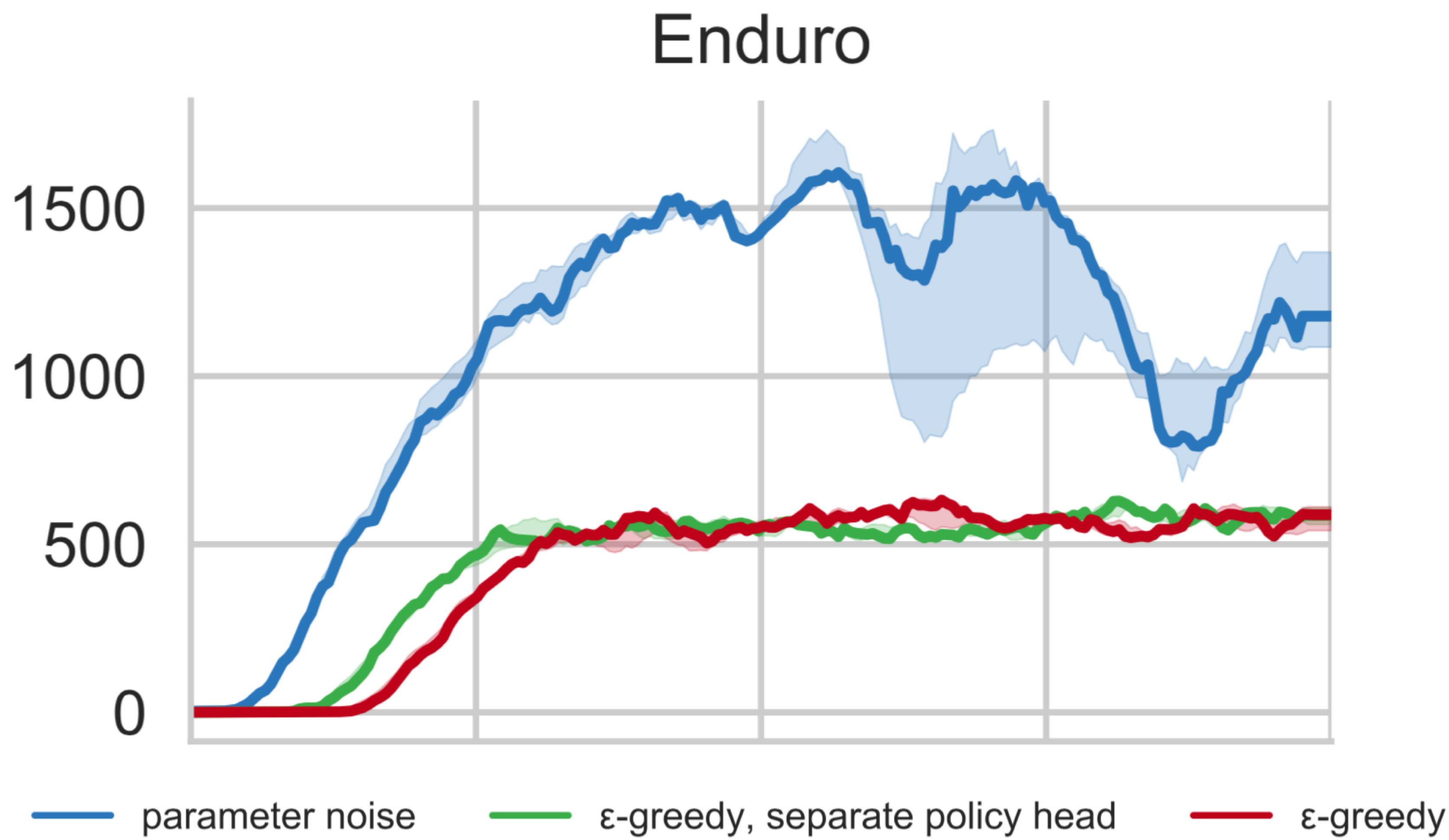
Experiments (2)

- Evaluation on 20 Atari games
- DQN with different exploration methods
- Exploration behavior of ε-greedy (left) vs. parameter space noise (right)



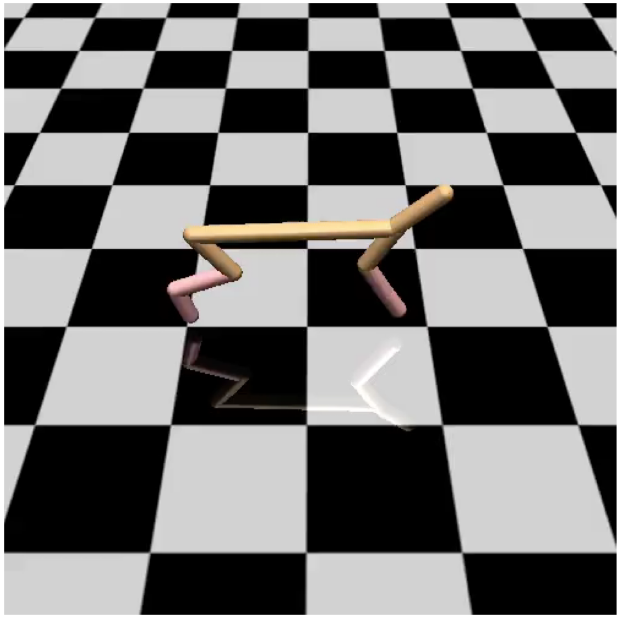


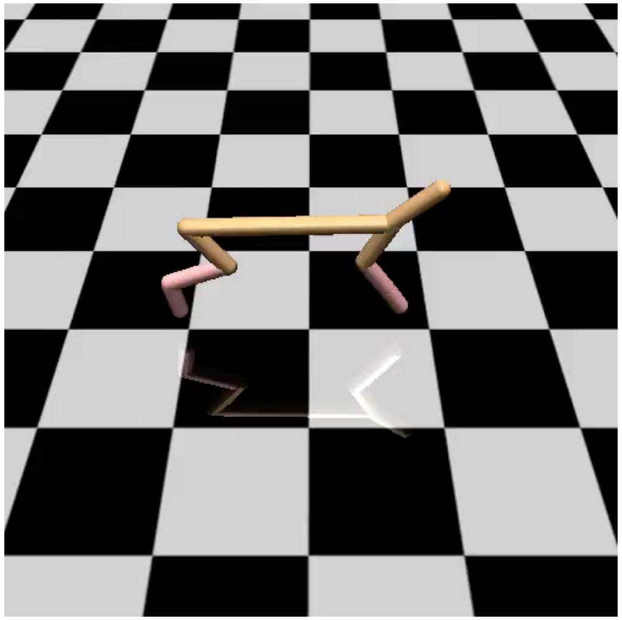
Experiments (3)



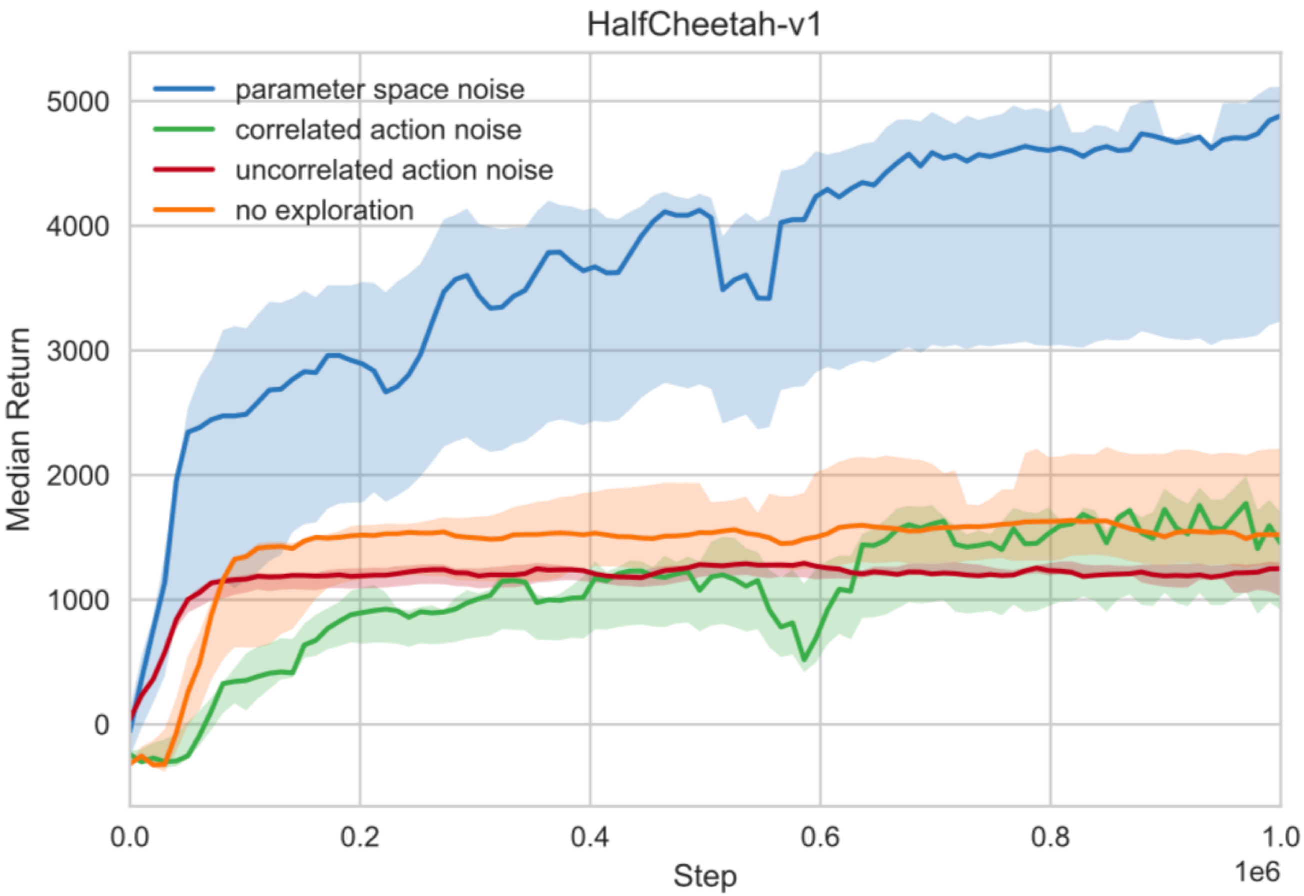
Experiments (4)

- Evaluation on 7 MuJoCo continuous control problems
- DDPG with different exploration methods
- Exploration of additive Gaussian noise (left) vs. parameter space noise (right)



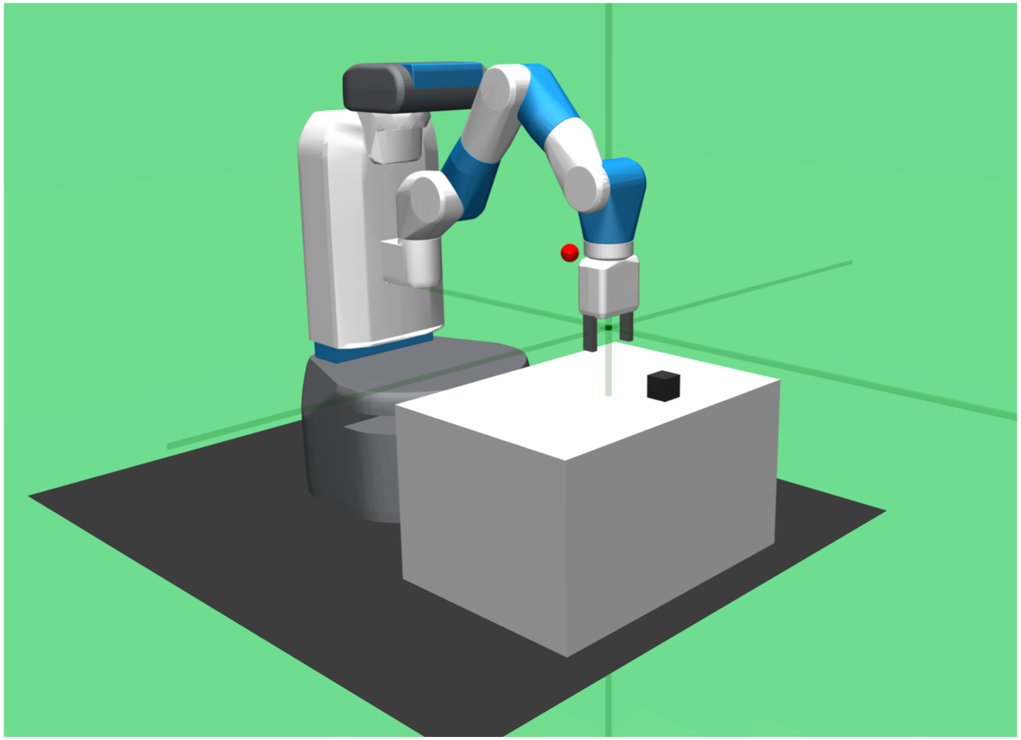


Experiments (5)

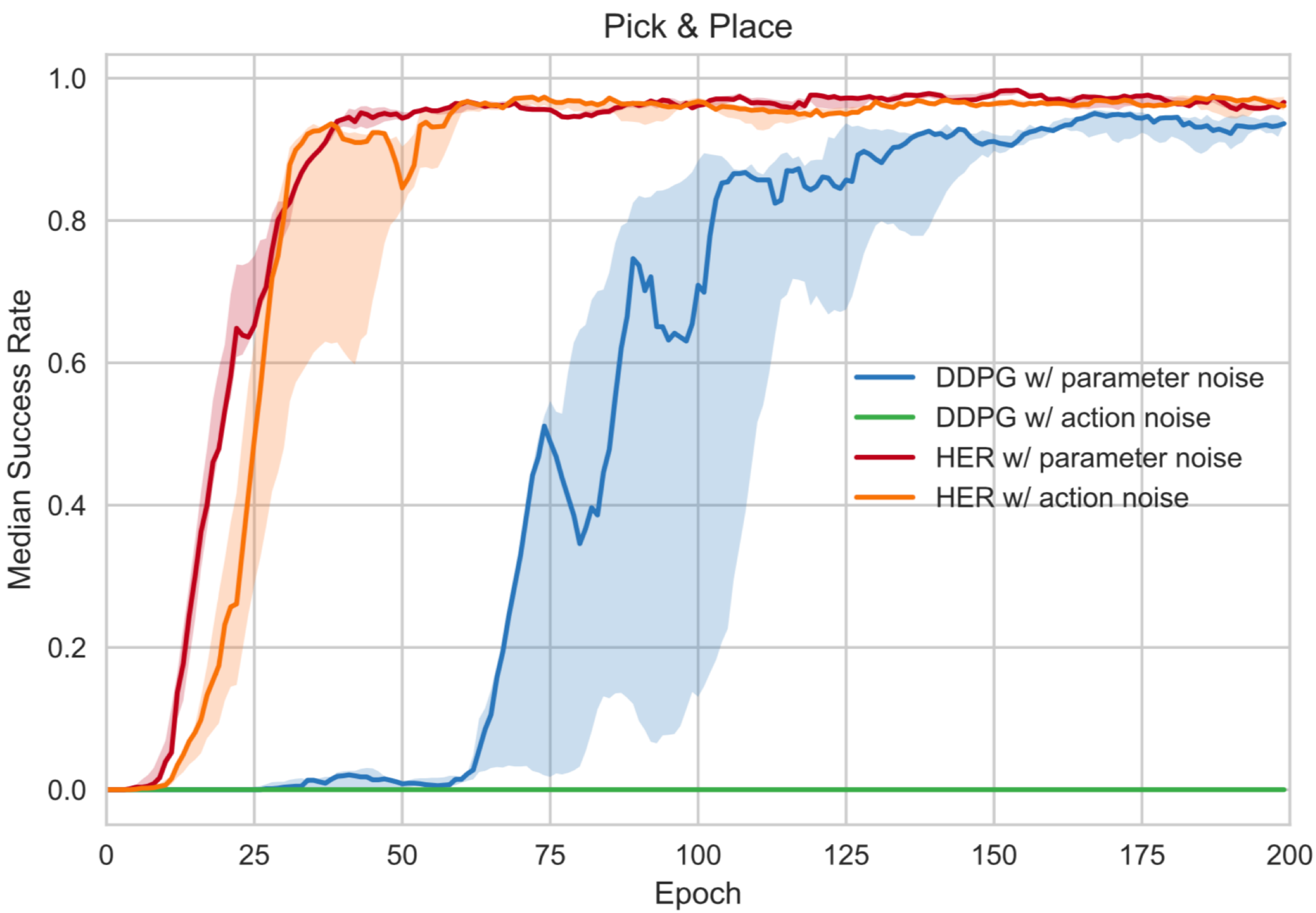


Experiments (6)

- Combine parameter space noise with Hindsight Experience Replay (HER)
- Evaluate on 3 robotics tasks with sparse rewards



Experiments (7)



Related Work

- Concurrently to our work, DeepMind has proposed “Noisy Networks for Exploration”, Fortunato et al., 2017
- “Deep Exploration via Bootstrapped DQN”, Osband et al., 2016
- “Evolution strategies as a scalable alternative to reinforcement learning”, Salimans et al., 2017
- “State-dependent exploration for policy gradient methods”, Rückstieß et al., 2008
- And a lot of other papers on the general topic of exploration in RL

Conclusion

- Conceptually simple and designed as a drop-in replacement for action space noise (or as an addition)
- Often leads to better performance due to better exploration
- Especially helps when exploration is really important (i.e. sparse rewards)
- Seems to escape local optima (e.g. HalfCheetah)
- Works for off- and on-policy algorithms for discrete and continuous action spaces

Future Directions

- More sophisticated adaption mechanisms
- Learned perturbations
- Combinations of action and parameter space noise
  - Both simulatenously
  - Switching between action and parameter noise
- Combination with Bayesian Neural Networks
  - Noise proportional to parameter uncertainty?

Collaborators

Rein Houthooft	*OpenAI*
Prafulla Dhariwal	*OpenAI*
Szymon Sidor	*OpenAI*
Richard Y. Chen	*OpenAI*
Xi Chen	*UC Berkeley / Embodied Intelligence*
Tamim Asfour	*KIT*
Pieter Abbeel	*UC Berkeley / Embodied Intelligence*
Marcin Andrychowicz	*OpenAI*

Future Reading

- “Parameter Space Noise for Exploration in Deep Reinforcement Learning“, Plappert, Master thesis, 2017, available online at matthiasplappert.com
- “Parameter Space Noise for Exploration“, Plappert et al., International Conference on Learning Representations, Vancouver, 2018

Thank you for your attention!

Learn more about OpenAI:
https://openai.com & https://blog.openai.com/

We're also hiring!
https://openai.com/jobs/