Trustworthy AI Autonomy

M5-1 Trustworthy RL-Generalization

Ding Zhao

Assistant Professor

Carnegie Mellon University

Plan for today

- Working on real robots
  - Setting of robots, human supervision etc
  - Continuous state actions (DDPG/SAC)
  - Delay-aware RL
  - Non-stationary context-aware Rl
  - Generalization and meta learning

Working with real robots

- Experiment design
  - Facilitating continuous operation
    Round-the-clock operation

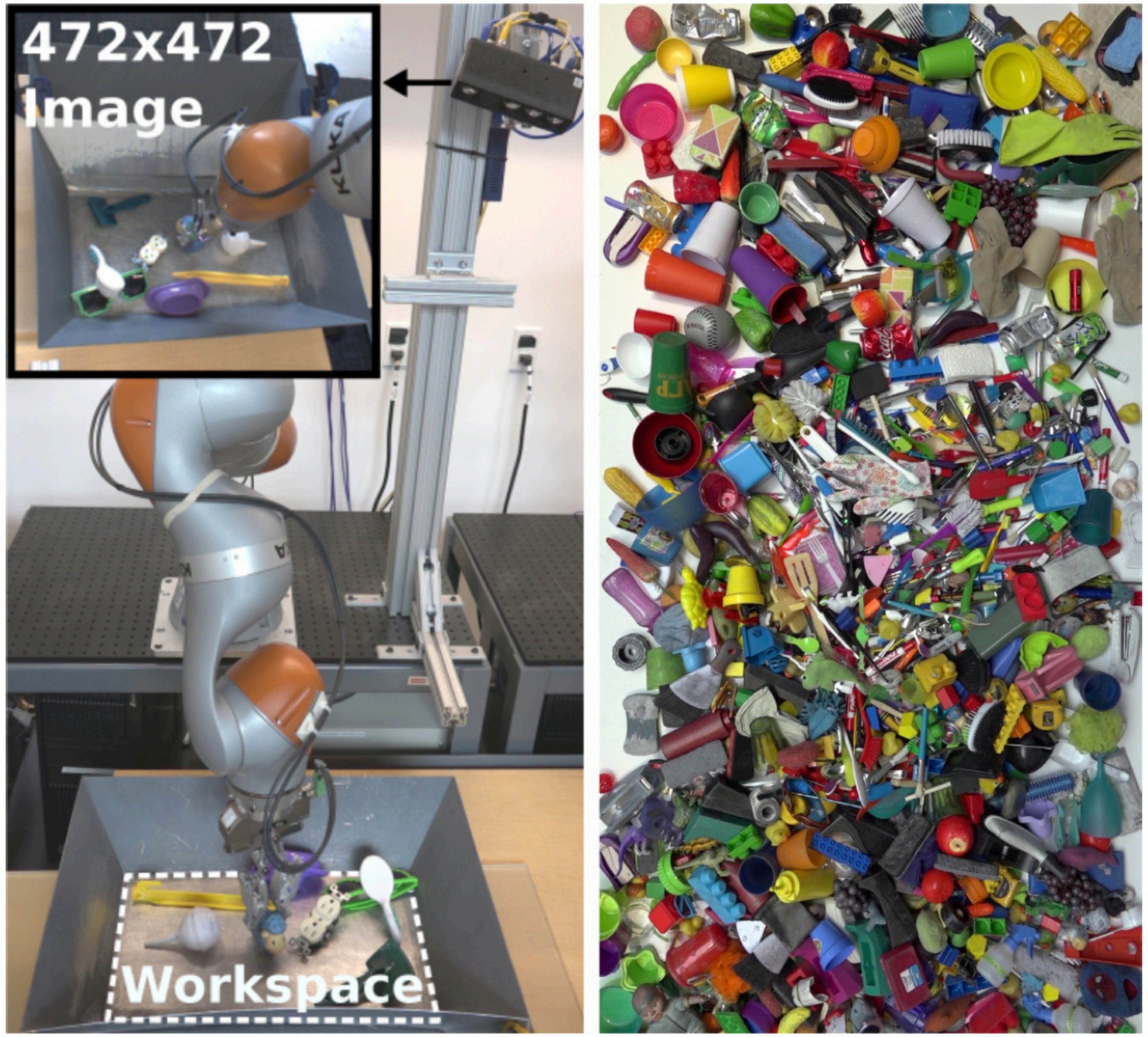
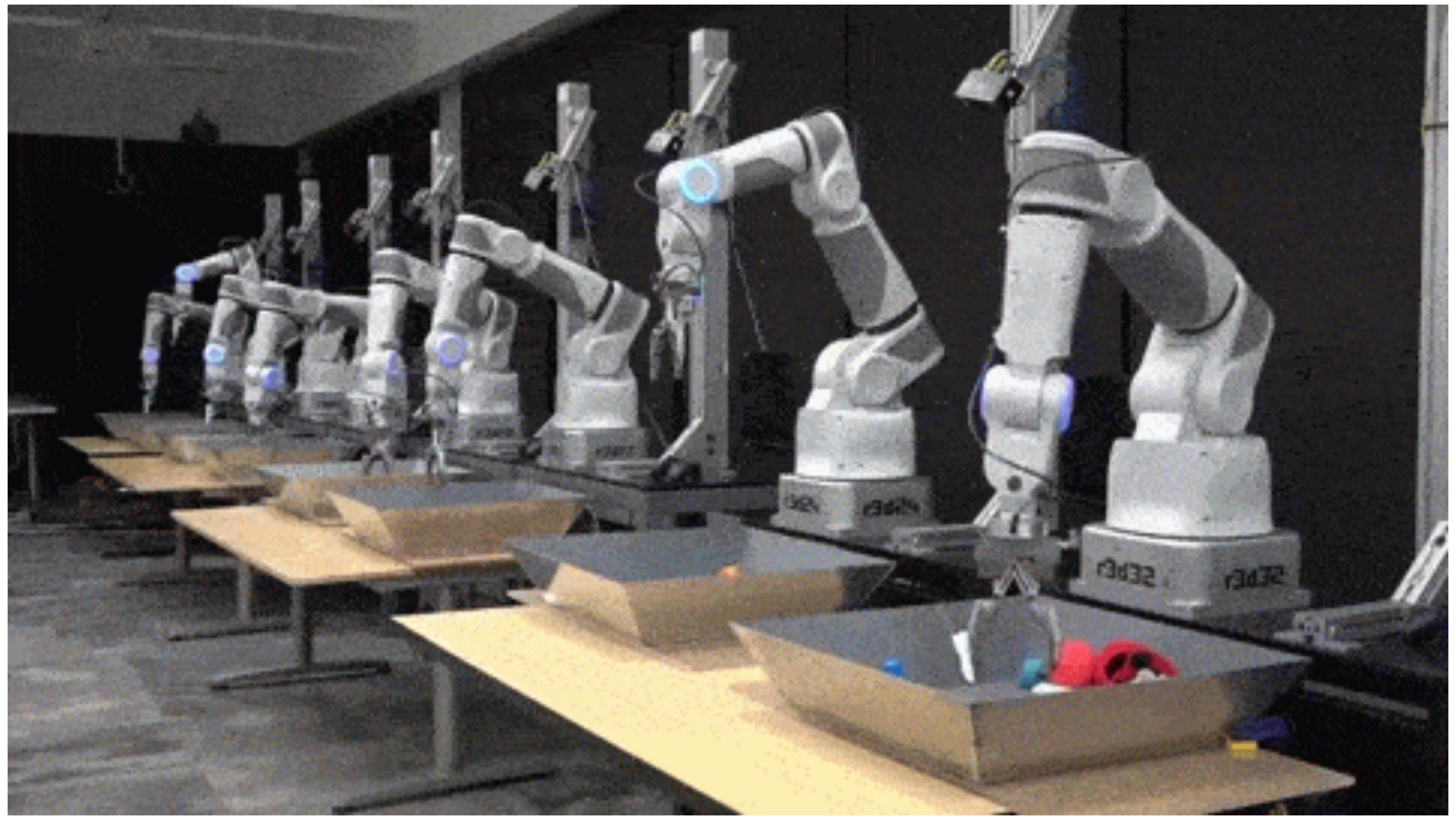


Figure 3. Close-up of our robot grasping setup in our setup (left) and about 1000 visually and physically diverse training objects (right). Each robot consists of a KUKA LBR IIWA arm with a two-finger gripper and an over-the-shoulder RGB camera.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, Sergey Levine, Sergay, "How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned," Journal of Robotics Research (IJRR), 2021

Working with real robots

- Experiment design
  - Facilitating continuous operation
    Round-the-clock operation
  - Non-stationarity due to environment changes

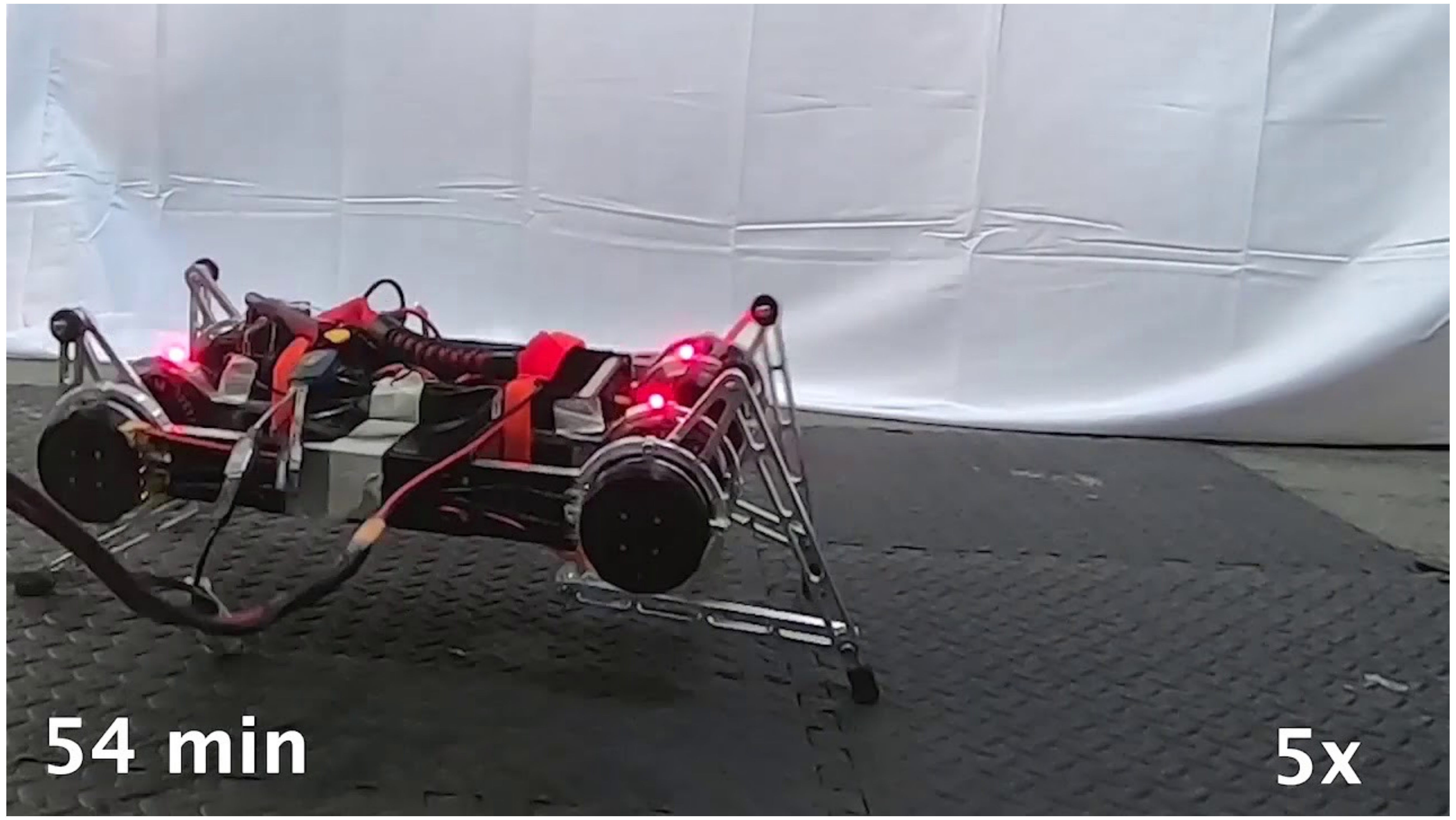


- a consistent performance drop of 5% in as little as 800 grasps executed on a single robot.

Levine, Sergey, et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." *The International Journal of Robotics Research* 37.4-5 (2018): 421-436.

Working with real robots

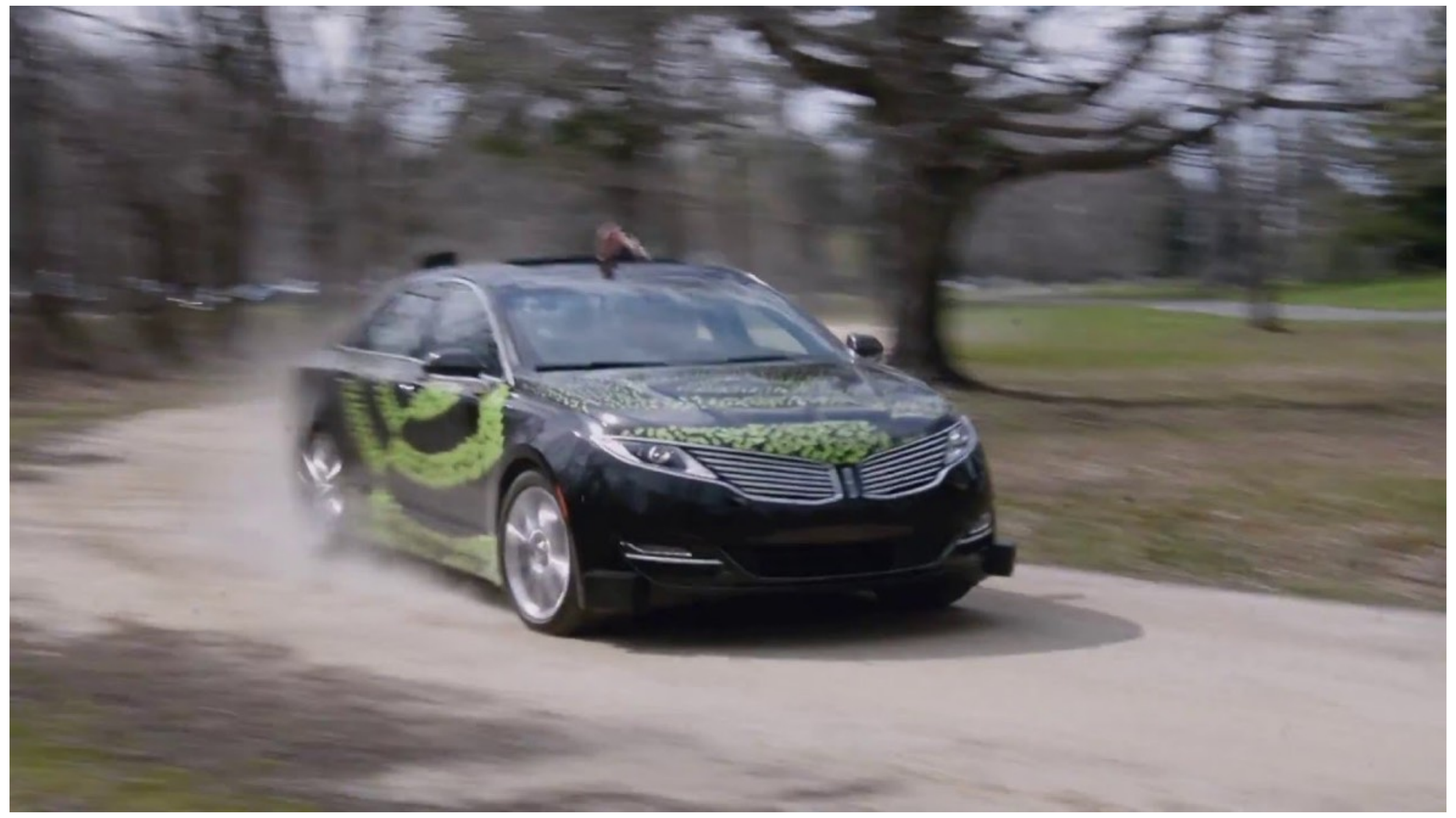
- Experiment design
  - Facilitating continuous operation
    Round-the-clock operation
  - Non-stationarity due to environment changes
  - The human can reset the scene, stop the robot in unsafe situations, and simply restart and reset the robot on failures.



https://www.youtube.com/watch?v=FmMPHL3TcrE&t=85s

Safe Reinforcement Learning via Human Intervention

- Trial without Error
- May not be scalable for real world implementation
  - However, people still do it e.g. self-driving companies



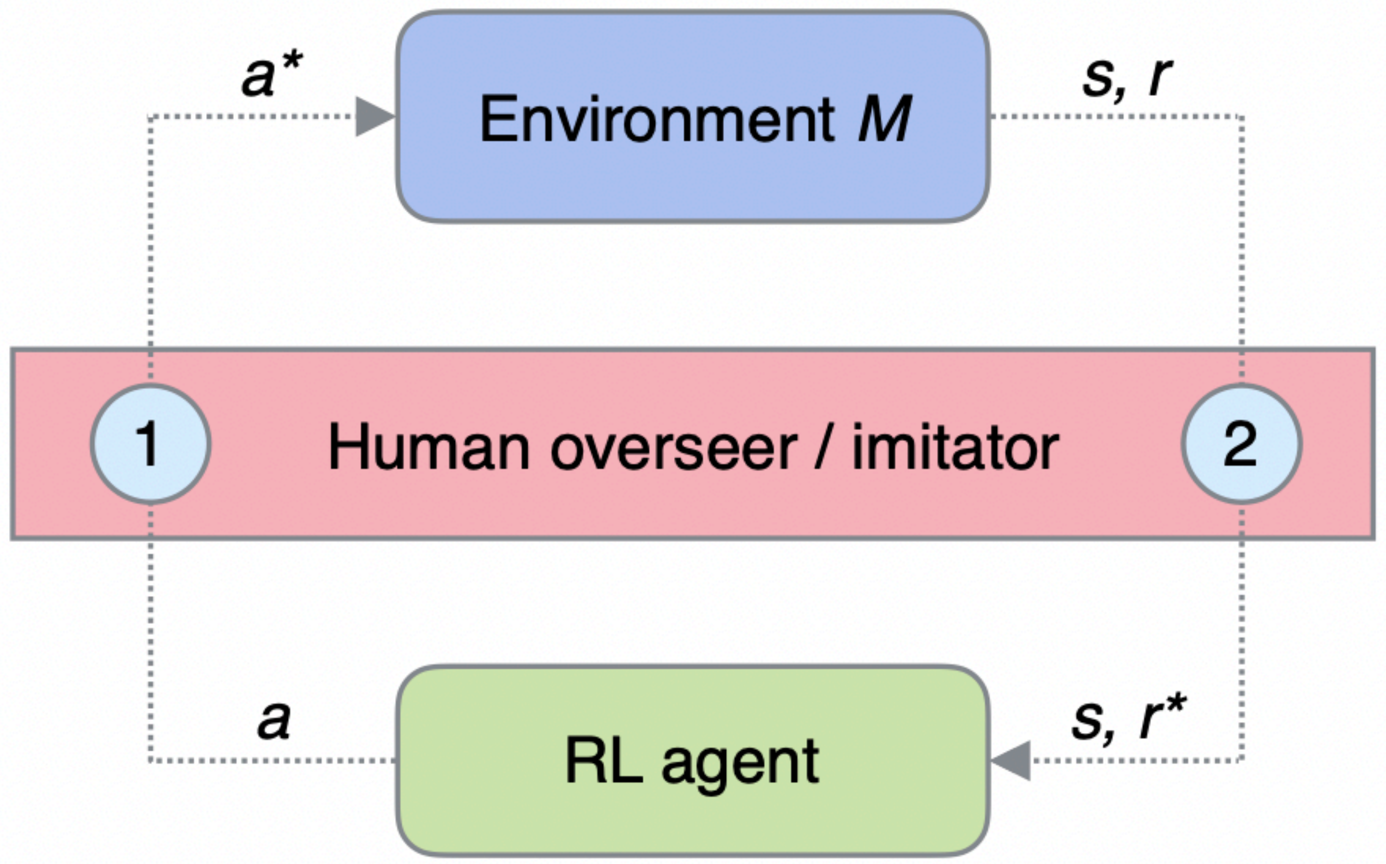


Figure 1: HIRL scheme. At (1) the human overseer (or Blocker imitating the human) can block/intercept unsafe actions a and replace them with safe actions a^*. At (2) the overseer can deliver a negative reward penalty r^* for the agent choosing an unsafe action.

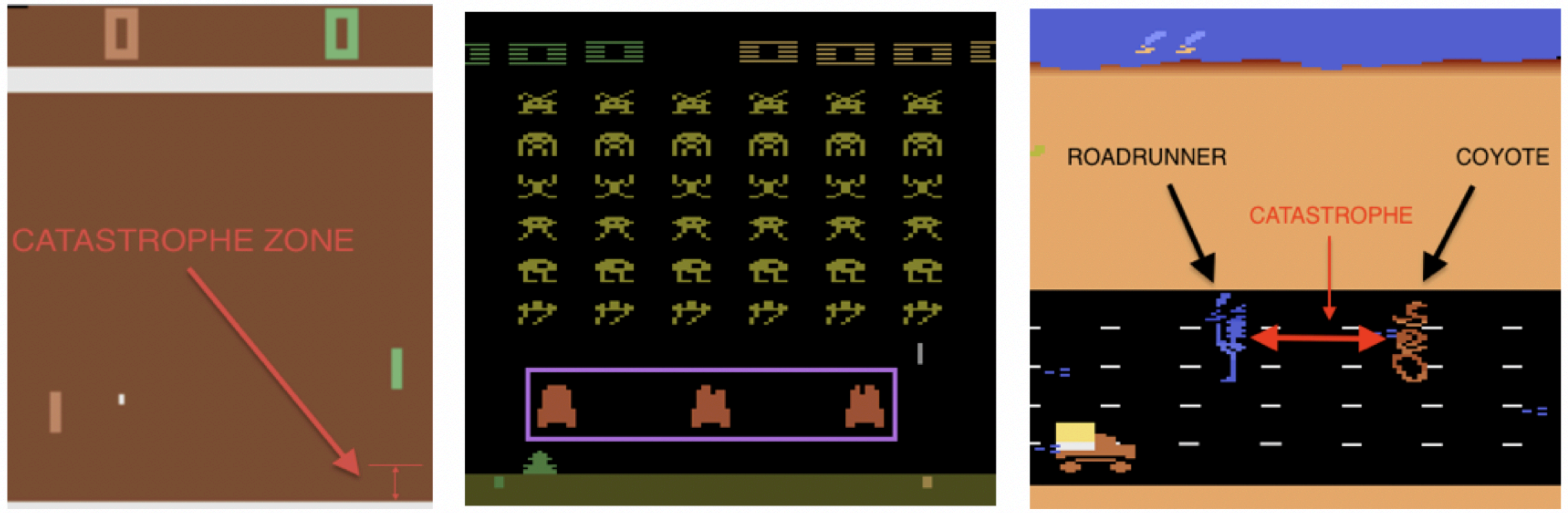


Figure 2: In Pong (left) it's a catastrophe if the agent (green paddle) enters the Catastrophe Zone. In Space Invaders (center), it's a catastrophe if the agent shoots their defensive barriers (highlighted in pink box). In Road Runner (right), it's a catastrophe if Road Runner touches the Coyote.

Saunders, William, et al. "Trial without error: Towards safe reinforcement learning via human intervention." *arXiv preprint arXiv:1707.05173* (2017).

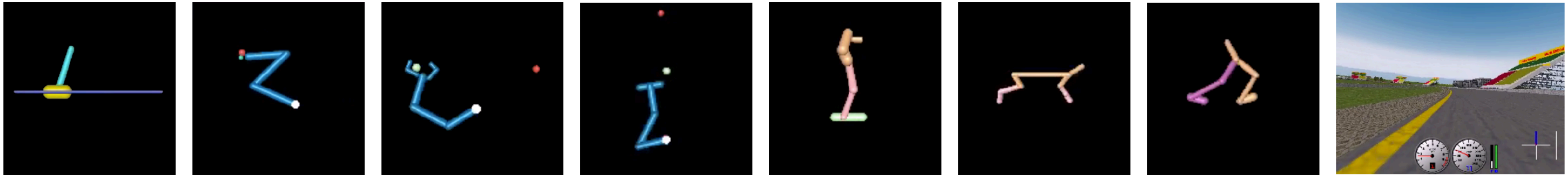
Recap: two ways to compute the optimal policy

- Parameterize the policy
  - Gradient ascent
- $J(\theta, \mathscr{D}_{\pi_\theta}) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t \,|\, \pi_\theta\right]$
- $\theta^* = \arg\max_\theta J(\theta, \mathscr{D}_{\pi_\theta})$
- $\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\theta)\,|_{\theta=\theta_i}, \theta_i \to \theta^*$
- $a_t \sim \pi_{\theta_i}(\,\cdot\,|\,s_t)$

- Parameterize the value function Q
  - Dynamic programming
- $Q_{\phi^*}^{\pi^*}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi^*}^{\pi^*}(s_{t+1}, a_{t+1})]$
- $e_\phi^\pi = Q_\phi^\pi(s_t, a_t) - \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_\phi^\pi(s_{t+1}, a_{t+1})]$
- $L(\phi, \pi) = \mathbb{E}\left[\frac{1}{2} e_\phi^{\pi 2}\right], (\phi^*, \pi^*) = \arg\min_{\phi,\pi} L(\phi, \pi)$
- With the“greedy method”, i.e., $\pi(a_t\,|\,s_t) = \max_a Q_\phi(s_t, a)$ ϕ of Q then can influence π.
- $\phi_{i+1} = \phi_i - \alpha \nabla_\phi L(\phi)\,|_{\phi=\phi_i}, \phi_i \to \phi^*, \pi_i \to \pi^*$

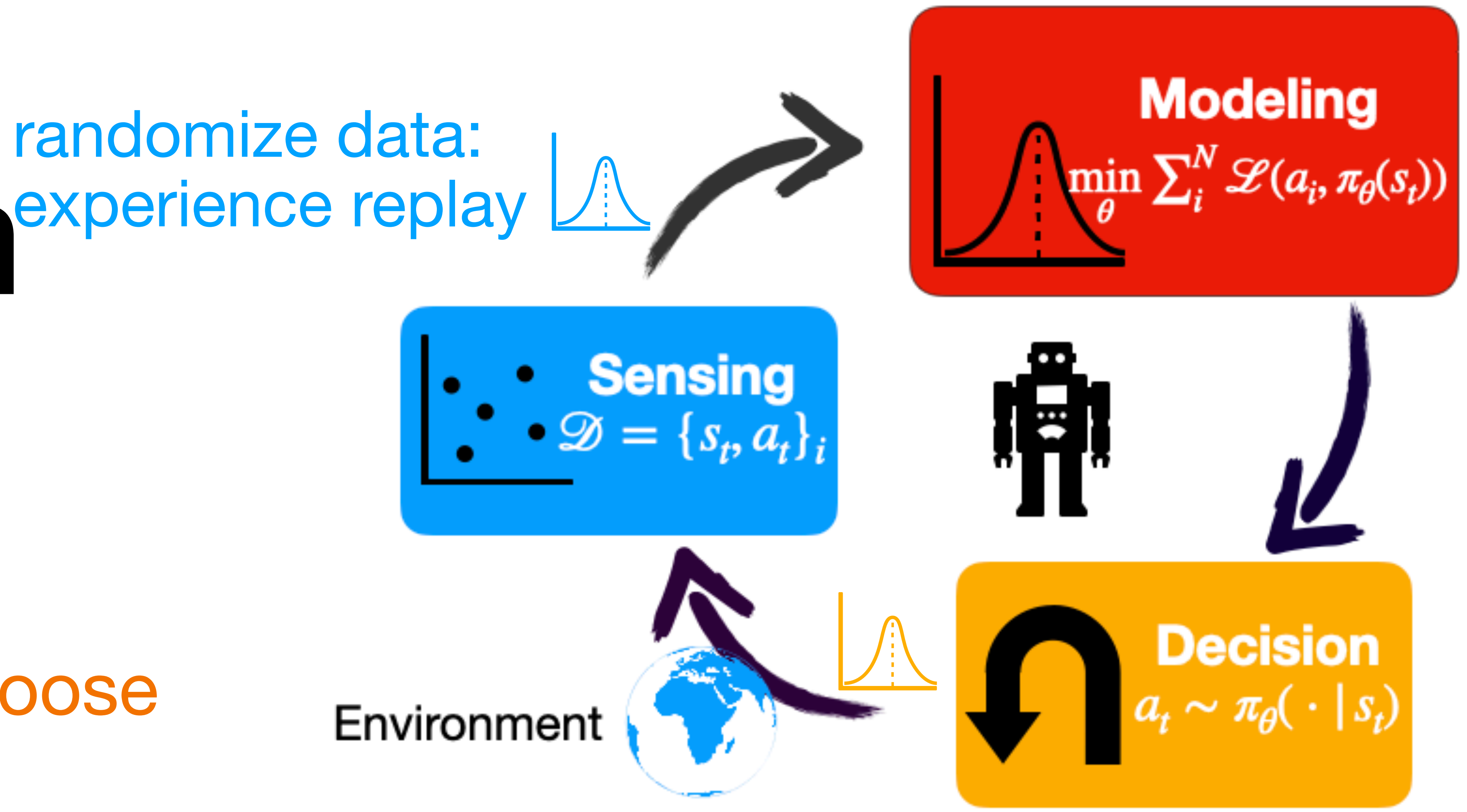
Handle continuous action space

- Issue of DQN: With continuous action space, we need to solve $a_t = \arg\max_a Q(s_t, a)$ every timestep in Q-learning.
- When there are a finite number of discrete actions, the max poses no problem, because we can just compute the Q-values for each action separately and directly compare them. (This also immediately gives us the action which maximizes the Q-value.)
- But when the action space is continuous, we can't exhaustively evaluate the space, and solving the optimization problem is highly non-trivial. Using a normal optimization algorithm would make calculating $\max_a Q^*(s, a)$ a painfully expensive subroutine. And since it would need to be run every time the agent wants to take an action in the environment, this is unacceptable.
- DDPG does not have this problem as it directly approximate $\arg\max_a Q^*(s, a)$ with $\mu_\theta(s)$. No optimization is needed.



https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Recap: DQN-3.0 algorithm



Randomize actions and training data

1. Take the ε-greedy method:

   $a_t = \max_a Q_{\phi_i}(s_t, a)$ with probability $1 - \varepsilon$, otherwise, choose a random action

   observe a dataset $\{(s_t, a_t, s_{t+1}, r_t)\}$ and add it to $\mathscr{D}$

   1. Randomly sample a mini batch from $\mathscr{D}$
   2. Calculate Bellman backup for this batch

      $y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1})$

      1. Update the Q function

         $\phi \leftarrow \phi - \alpha \sum_t (\nabla_\phi Q_\phi(s_t, a_t))(Q_\phi(s_t, a_t) - y_t)$

   3. Update Q function:

      Moving average: $\phi_{i+1} = \rho \phi_i + (1 - \rho)\phi$, e.g. $\rho = 0.999$

Also called low pass filter/Polyak averaging in literature

The only changes DDPG made

1. Use a deterministic policy to calculate

   arg max: $a_t = \mu_\theta(s_t) = \arg\max_a Q(s_t, a)$

   Q function:

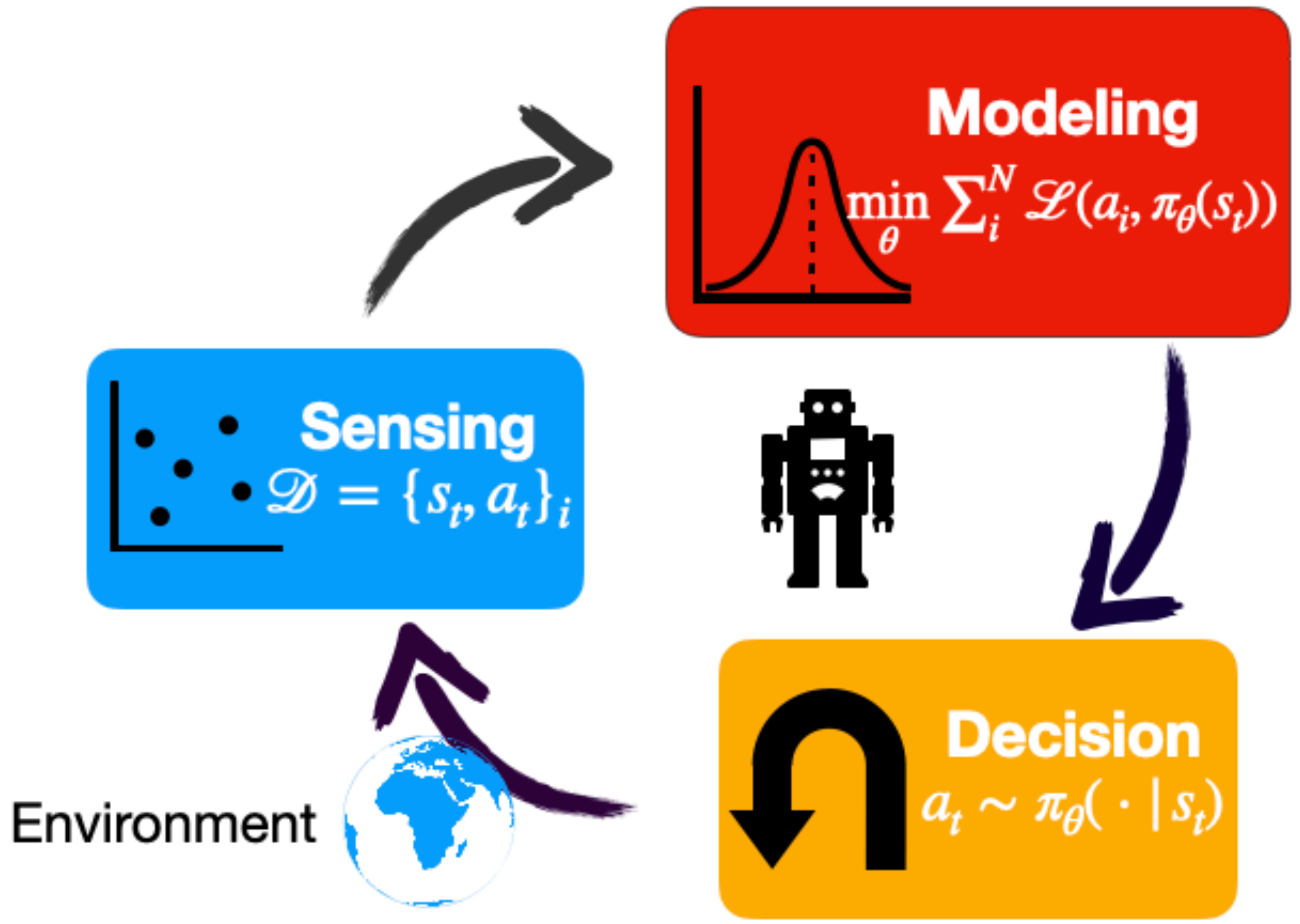
   $\max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1}) \rightarrow Q_{\phi_i}(s_{t+1}, \mu_\theta(s))$

2. $\theta_{i+1} = \theta_i + \nabla_\theta J(\theta)|_{\theta=\theta_i}$

$\nabla_\theta J(\theta) = \nabla_\theta Q(s_t, a_t) = \nabla_\theta Q(s_t, \mu_\theta(s_t))$

$= \mathbb{E}_{s_t \sim \mathscr{D}}[\nabla_\theta \mu_\theta(a_t | s_t) \nabla_a Q^\mu(s_t, a)|_{a=\mu_\theta(s_t)}]$

DDPG algorithms



Randomize actions and training data

1. Take the $a(s_t) = \mu_\theta(s_t)$
   observe a dataset $\{(s_t, a_t, s_{t+1}, r_i)\}$ and add it to $\mathscr{D}$
   1. Randomly sample a mini batch from $\mathscr{D}$
   2. Calculate Bellman backup for this batch
      $$y_t = r(s_t, a_t) + \gamma Q_{\phi_i}(s_{t+1}, \mu_\theta(s_t))$$
      1. Update the Q function
         $$\phi \leftarrow \phi - \alpha \sum_t (\nabla_\phi Q_\phi(s_t, a_t))(Q_\phi(s_t, a_t) - y_t)$$
      2. Update the policy
         $$\theta \leftarrow \theta + \beta \sum_t [\nabla_\theta \mu_\theta(a \mid s_t) \nabla_a Q^\mu_\phi(s_t, a) \mid_{a=\mu_\theta(s)}]$$
   3. Update Q function and policy
      $$\phi_{i+1} = \rho\phi_i + (1-\rho)\phi, \quad \theta_{i+1} = \rho\theta_i + (1-\rho)\theta,$$

Also called low pass filter/Polyak averaging in literature

Two tricks to enhance performance

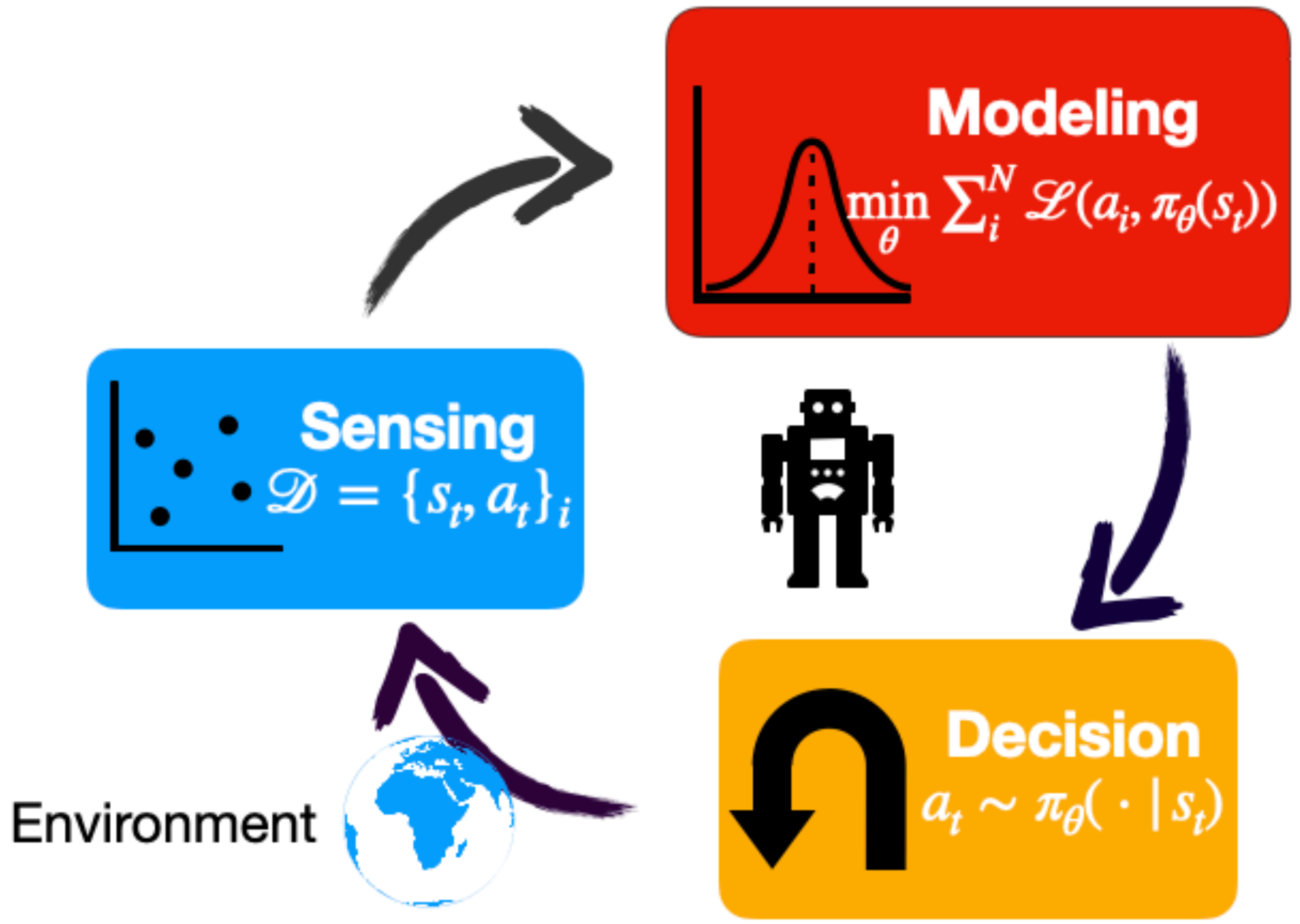
1. Increase stability: use a larger delay to stabilize the learning of Q function
   - use ϕ_{i-k} instead of ϕ_i to compute y_t

https://arxiv.org › cs

Continuous control with deep reinforcement learning

by TP Lillicrap · 2015 · Cited by 5390 — This **paper** has not been found in the **Papers** with Code database. If you are one of the registered authors of this **paper**, you can link your code on your arxiv user ...

DDPG algorithms



Randomize actions and training data

1. Take the $a(s_t) = \mu_\theta(s_t)$
   observe a dataset $\{(s_t, a_t, s_{t+1}, r_i)\}$ and add it to $\mathcal{D}$
    1. Randomly sample a mini batch from $\mathcal{D}$
    2. Calculate Bellman backup for this batch
       $$y_t = r(s_t, a_t) + \gamma Q_{\phi_i}(s_{t+1}, \mu_\theta(s_t))$$
        1. Update the Q function
           $$\phi \leftarrow \phi - \alpha \sum_t (\nabla_\phi Q_\phi(s_t, a_t))(Q_\phi(s_t, a_t) - y_t)$$
        2. Update the policy
           $$\theta \leftarrow \theta + \beta \sum_t [\nabla_\theta \mu_\theta(a \mid s_t) \nabla_a Q_\phi^\mu(s_t, a) \mid_{a=\mu_\theta(s)}]$$
    3. Update Q function and policy
       $$\phi_{i+1} = \rho\phi_i + (1-\rho)\phi, \; \theta_{i+1} = \rho\theta_i + (1-\rho)\theta,$$

Also called low pass filter/Polyak averaging in literature

Two tricks to enhance performance

1. Increase stability: use a larger delay to stabilize the learning of Q function
    - use ϕ_{i-k} instead of ϕ_i to compute y_t
2. Use add a Gaussian noise to the deterministic policy to make the data more i.i.d and covers more scenarios
    - $a(s_t) = \mu_\theta(s_t) + \mathcal{N}(0, \sigma^2)$

- Two approaches: TD3 and SAC

Improve the DDPG with TD3

- Overestimation problem
  - While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values (spikes), which then leads to the policy breaking. **Twin Delayed DDPG (TD3)** is an algorithm that addresses this issue by introducing three critical tricks:

1. Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the Bellman error loss functions. Particularly,
   $\phi^{(i)} \leftarrow \phi^{(i)} - \alpha \sum_t (\nabla_{\phi^{(i)}} Q_{\phi^{(i)}}(s_t, a_t))(Q_{\phi^{(i)}}(s_t, a_t) - y_t), i = \{1,2\}$. $y_t = r(s_t, a_t) + \gamma \min_i Q_{\phi^{(i)}}(s_{t+1}, \mu_\theta(s_t))$
2. "Delayed" Policy Updates.: use ϕ_{i-k} instead of ϕ_i to compute y_t. Usually $k = 1$ or 2.
3. Target Policy Smoothing: Use add a Gaussian noise to the deterministic policy
   $a(s_t) = \mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi, \xi \sim \mathcal{N}(0,I)$. Usually add clips to make the action range feasible. Or normalize it with $a_t = \tanh\left(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi\right)$

SAC

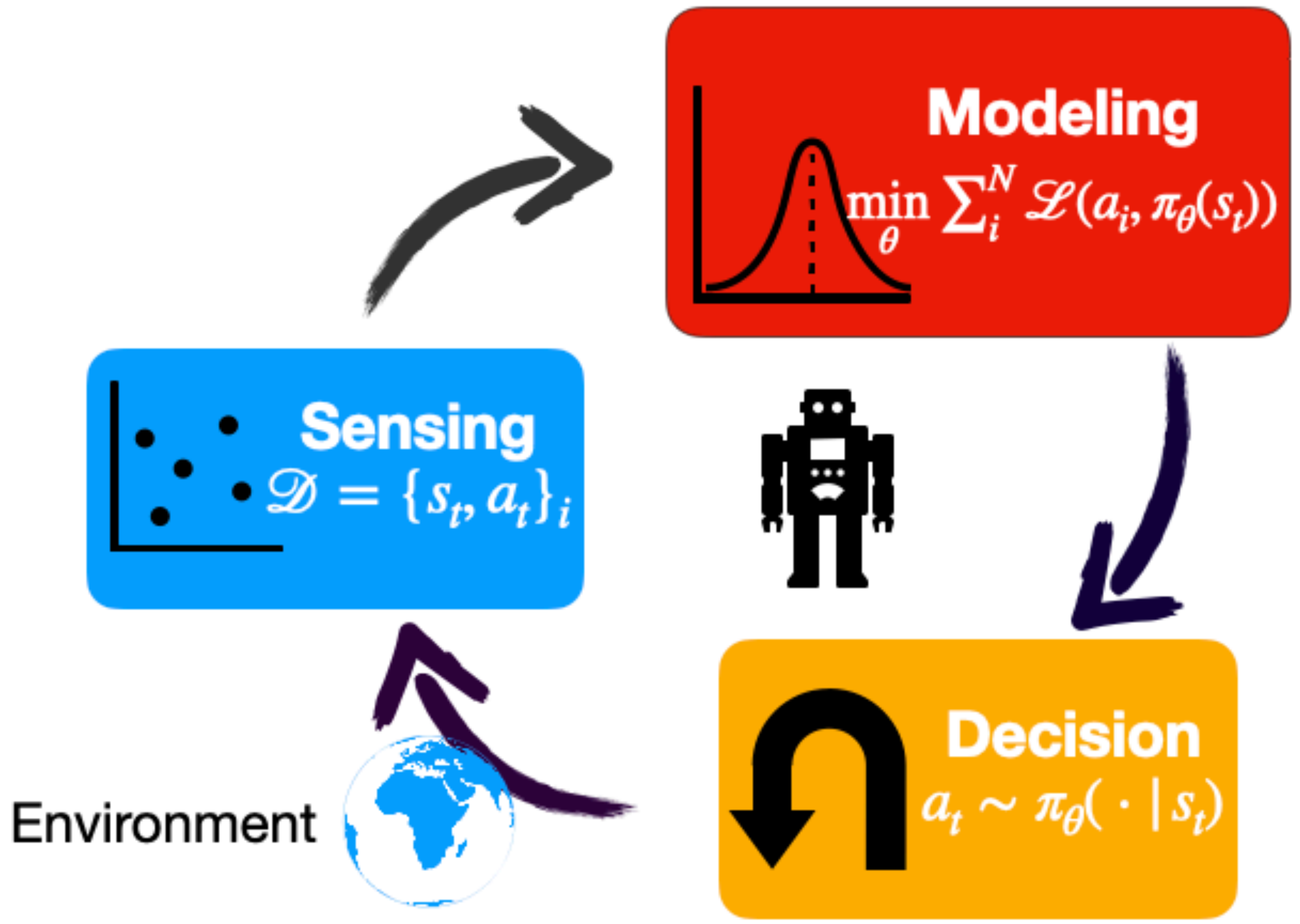
https://arxiv.org › cs

Soft Actor-Critic: Off-Policy Maximum Entropy Deep R...

by T Haarnoja · 2018 · Cited by 1399 — Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this **paper**, we propose soft actor-critic ...

- Idea 2: Increase the entropy of the policy distribution $\pi(a_t \mid s_t)$ to encourage exploration
  - Use a stochastic policy: $a_t \sim \pi_\theta(a_t | s_t) = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi), \xi \sim \mathcal{N}(0, I)$.
  - Entropy: $H(\pi_\theta) = \mathbb{E}_{\pi_\theta}[-\log \pi_\theta(a_t | s_t)]$
    - Entropy is a quantity which, roughly speaking, says how random a random variable is. If a coin is weighted so that it almost always comes up heads, it has low entropy; if it's evenly weighted and has a half chance of either outcome, it has high entropy.
    - Reward: $r(s_t, a_t) \Rightarrow r(s_t, a_t) + \alpha H(\pi(a_t \mid s_t))$
  - This has a close connection to exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on.
- The Bellman Equation becomes
  - $Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{t' \geq t} \gamma^{t'-t} r_{t'} + \alpha \sum_{t' \geq t+1} \gamma^{t'-t} H(\pi(a_{t'} | s_{t'})) \,|\, s_t, a_t \right] = \mathbb{E}_\pi \left[r_t + \gamma (Q^\pi(s_{t+1}, a_{t+1}) + \alpha H(\pi(a_{t'})) \right]$
  - The remaining is similar to TD3.

SAC algorithms



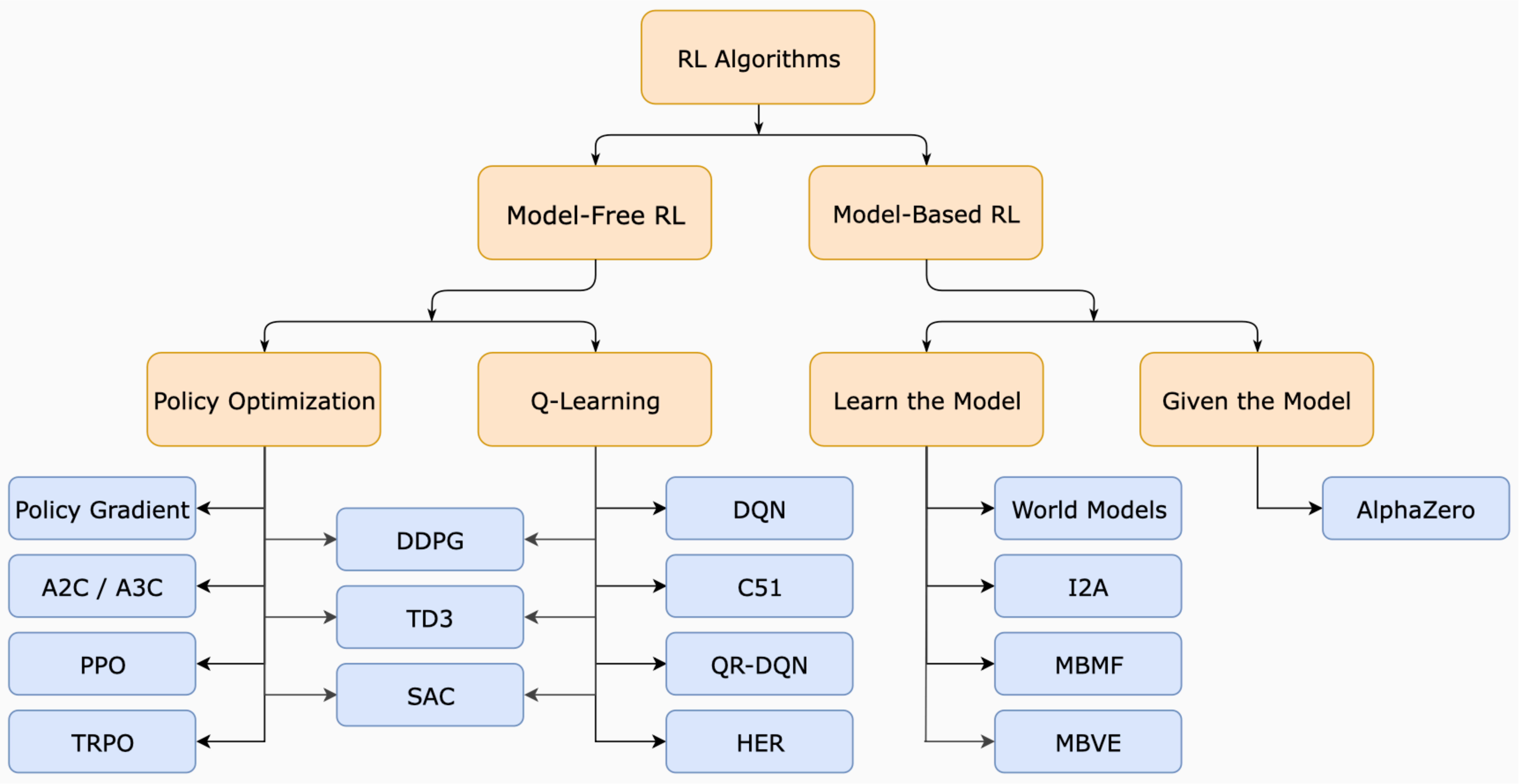
Randomize actions and training data

1. Take the $a_t = \tanh\left(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi\right), \quad \xi \sim \mathcal{N}(0, I)$
   observe a dataset $\{(s_t, a_t, s_{t+1}, r_i)\}$ and add it to $\mathcal{D}$

   1. Randomly sample a mini batch from $\mathcal{D}$
   2. Calculate Bellman backup for this batch
      $y_t = r(s_t, a_t) + \gamma(\min_{i=1,2} Q_{\phi^{(i)}}(s_{t+1}, a_{t+1}) + \alpha H(\pi(a_t \mid s_t))), a_{t+1} \sim \pi_\theta(\cdot \mid s_t)$
      1. Iteratively calculate ϕ from ϕ_i
         $\phi^{(i)} \leftarrow \phi^{(i)} - \alpha \sum_t (\nabla_{\phi^{(i)}} Q_{\phi^{(i)}}(s_t, a_t))(Q_{\phi^{(i)}}(s_t, a_t) - y_t), i = \{1,2\}$
      2. Iteratively calculate θ from θ_i
         $\theta \leftarrow \theta + \beta \nabla_\theta \sum_t [(\min_{i=1,2} Q_{\phi^{(i)}}(s_t, \pi_\theta(a_t \mid s_t)) + \alpha H(\pi(a_t \mid s_t))]$
   3. Update Q function and policy
      $\phi_{i+1} = \rho\phi_i + (1-\rho)\phi, \theta_{i+1} = \rho\theta_i + (1-\rho)\theta,$

Also called low pass filter/Polyak averaging in literature

Recap: popular RL algorithms



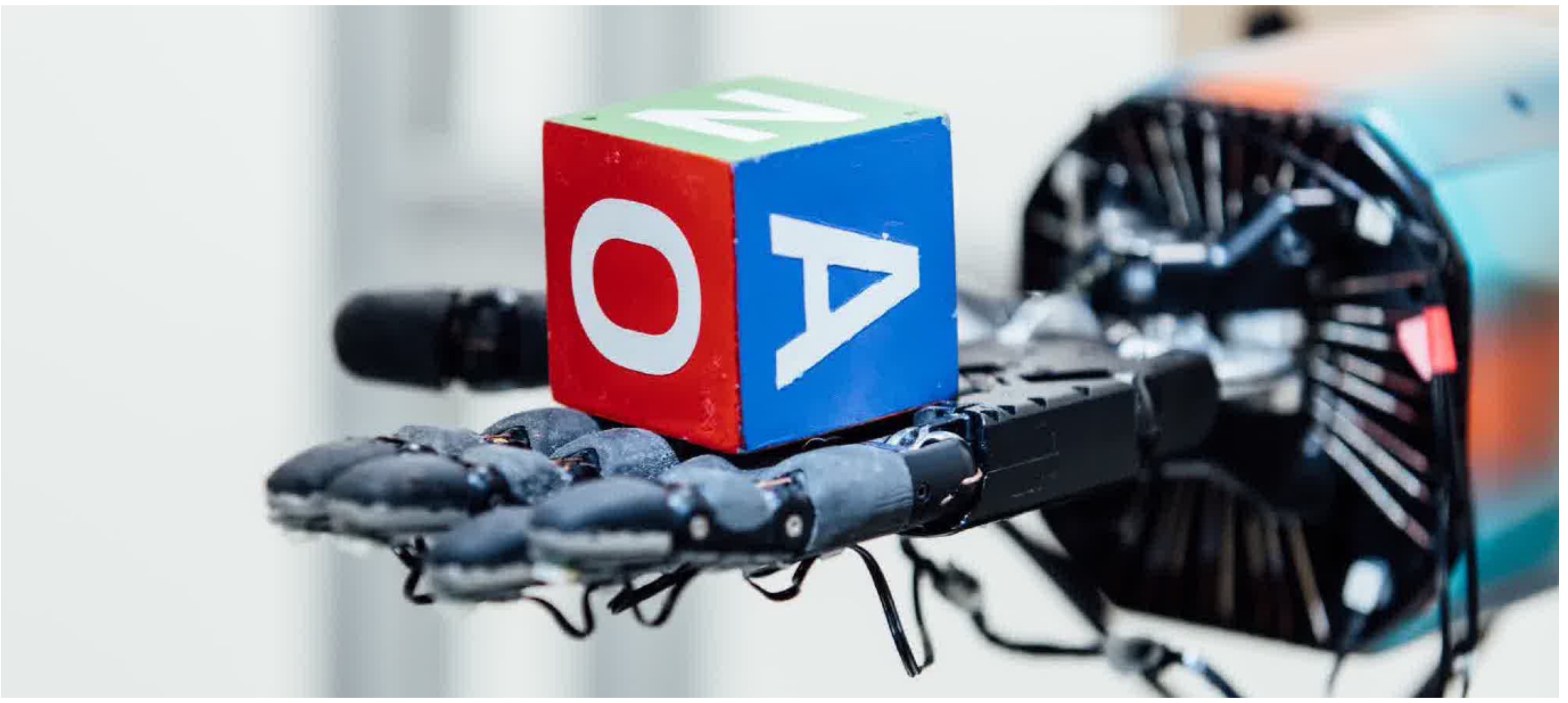
https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

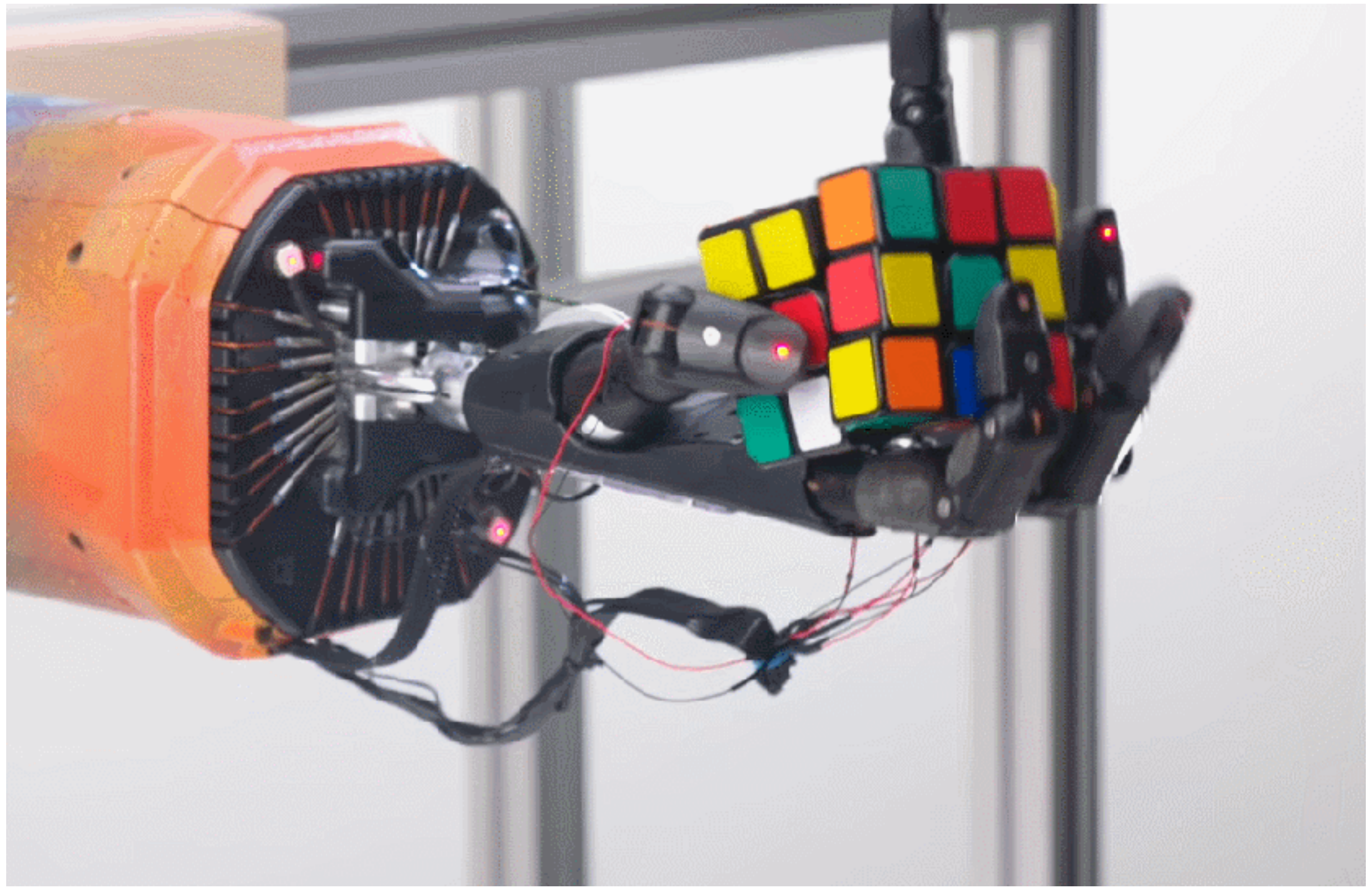
Algorithm	Agent type	Policy	Policy type	Monte Carlo (MC) or Temporal difference (TD)	Action space	State space
Tabular Q-learning (= SARSA max) Q learning lambda	Value-based	Off-policy	Pseudo-deterministic (epsilon greedy)	TD	Discrete only	Discrete only
SARSA SARSA lambda	Value-based	On-policy	Pseudo-deterministic (epsilon greedy)	TD	Discrete only	Discrete only
DQN N step DQN Double DQN Noisy DQN Prioritized Replay DQN Dueling DQN Categorical DQN Distributed DQN (C51)	Value-based	Off-policy	Pseudo-deterministic (epsilon greedy)		Discrete only	Discrete or continuous
NAF = continuous DQN	Value-based				Continuous	Continuous
CEM	Policy-based	On-policy		MC		
REINFORCE (Vanilla policy gradient)	Policy-based	On-policy	Stochastic	MC		
Policy gradient softmax	Policy-based		Stochastic			
Natural Policy Gradient	Policy-based		Stochastic			
TRPO	Actor-critic	On-policy (?)	Stochastic		Discrete or continuous	Discrete or continuous
PPO	Actor-critic	On-policy (?)	Stochastic		Discrete or continuous	Discrete or continuous
Distributed PPO	Actor-critic				Continuous	Continuous
A2C / A3C	Actor-critic	On-policy	Stochastic	TD	Discrete or continuous	Discrete or continuous
DDPG	Actor-critic	Off-policy	Deterministic		Continuous only	Discrete or Continuous
TD3	Actor-critic				Continuous only	Discrete or continuous
D4PG	Actor-critic				Continuous only	Discrete or continuous
SAC	Actor-critic	Off-policy			Continuous only	Discrete or continuous
ACER	Actor-critic				Discrete	Discrete or Continuous
ACKTR	Actor-critic				Discrete or Continuous	Discrete or Continuous

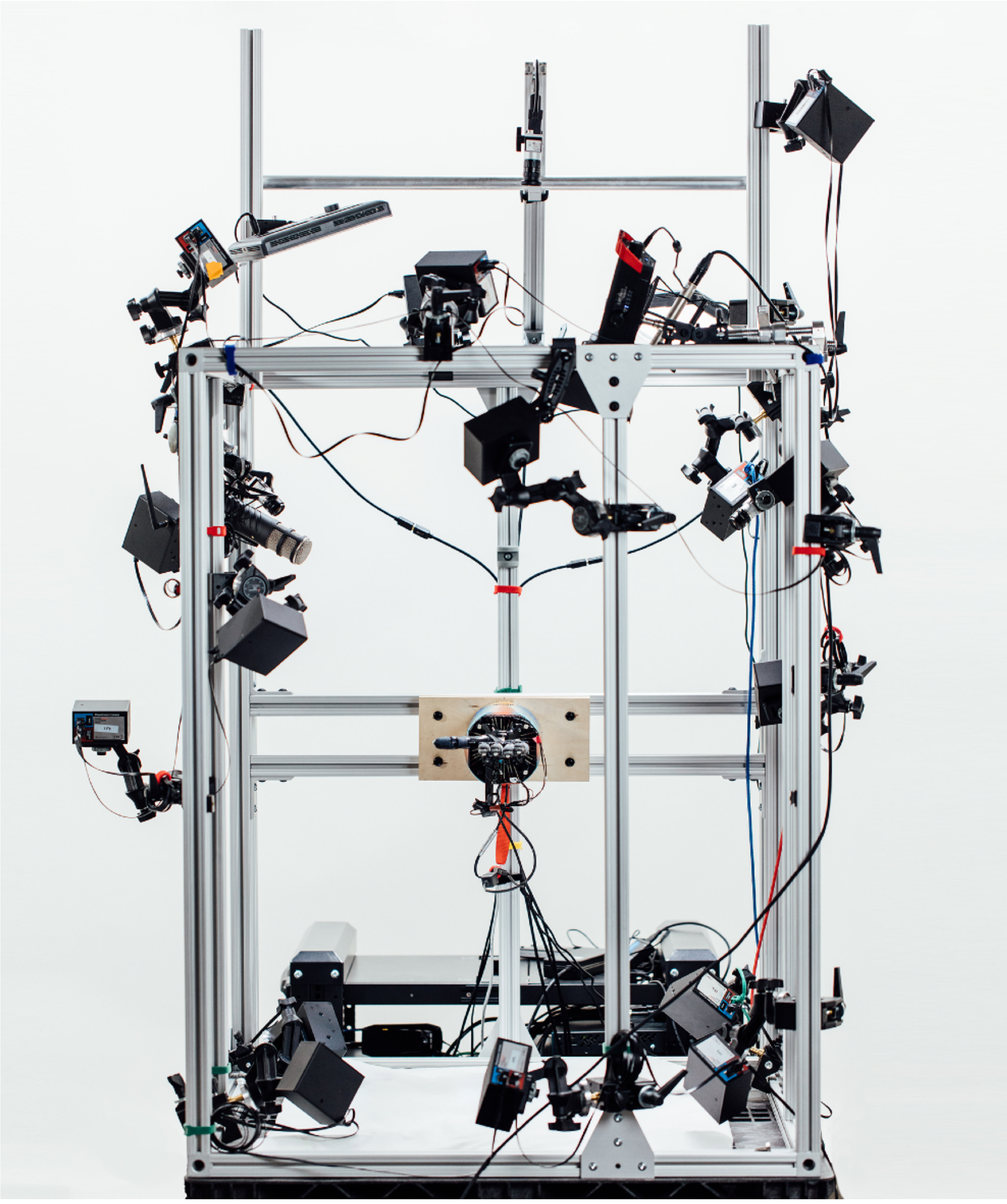
https://www.datamachinist.com/reinforcement-learning/part-2-reinforcement-learning-algorithms/

Domain randomization

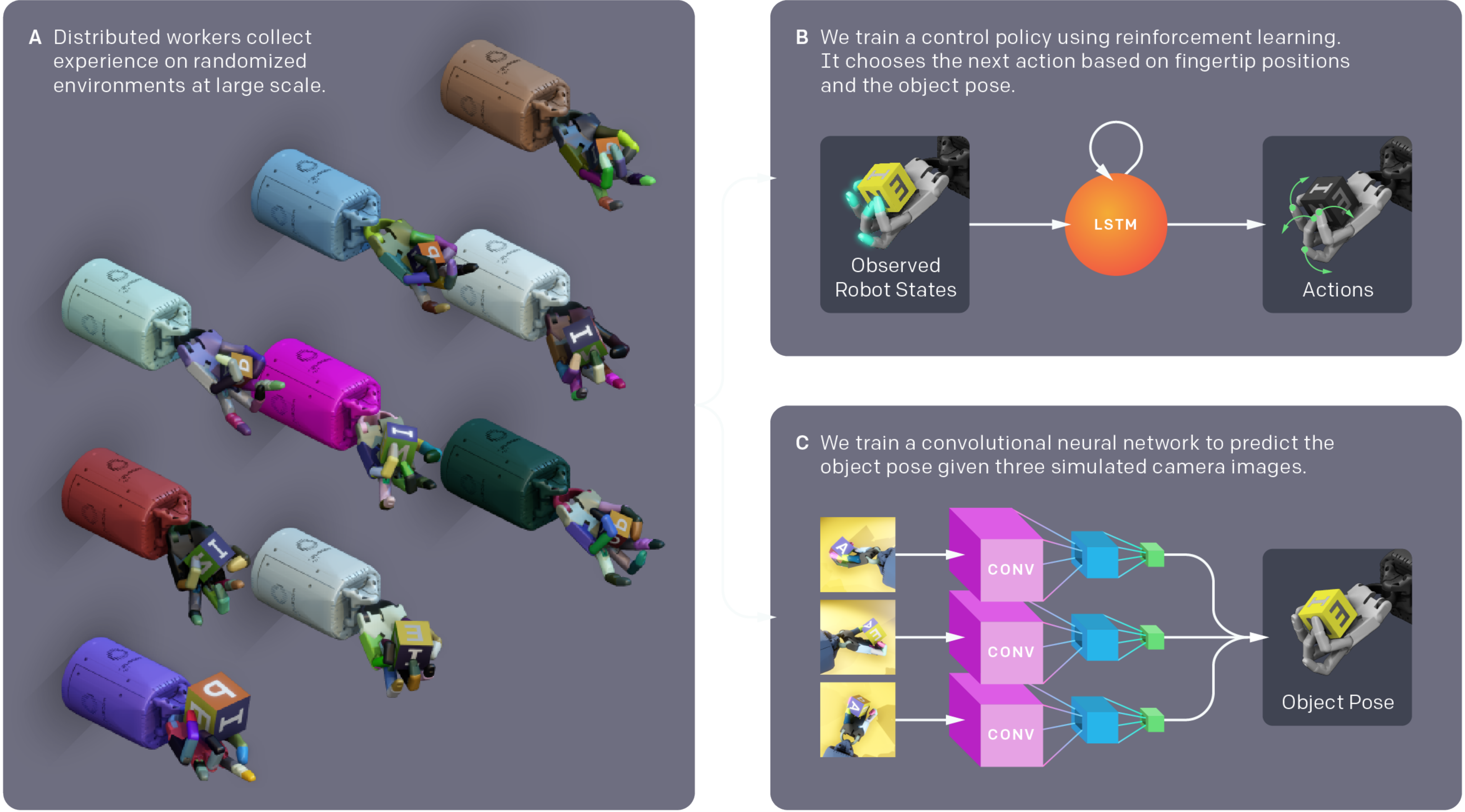
- Sim-to-real: Dactyl was trained entirely in simulation and transfers its knowledge to reality



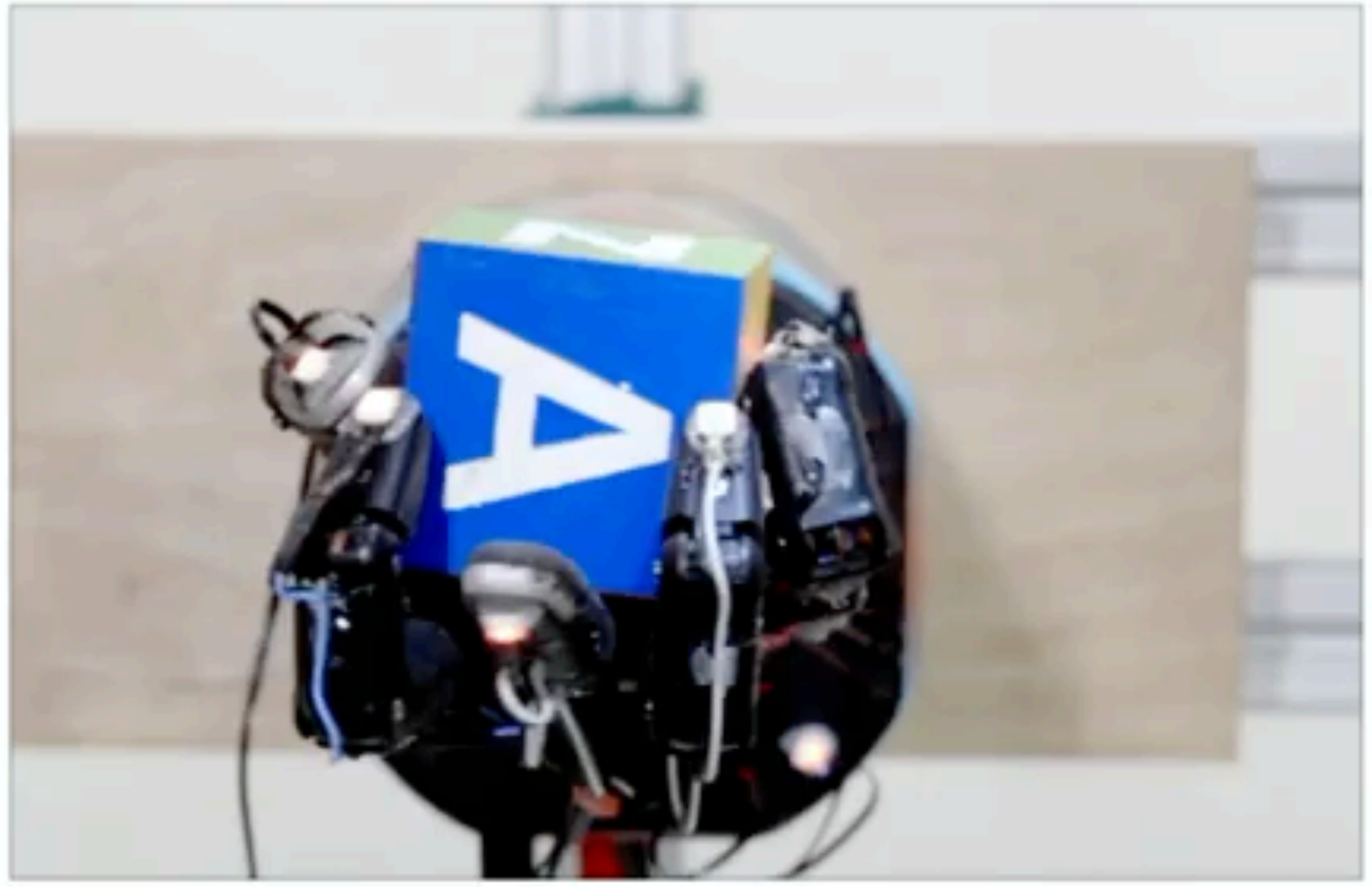




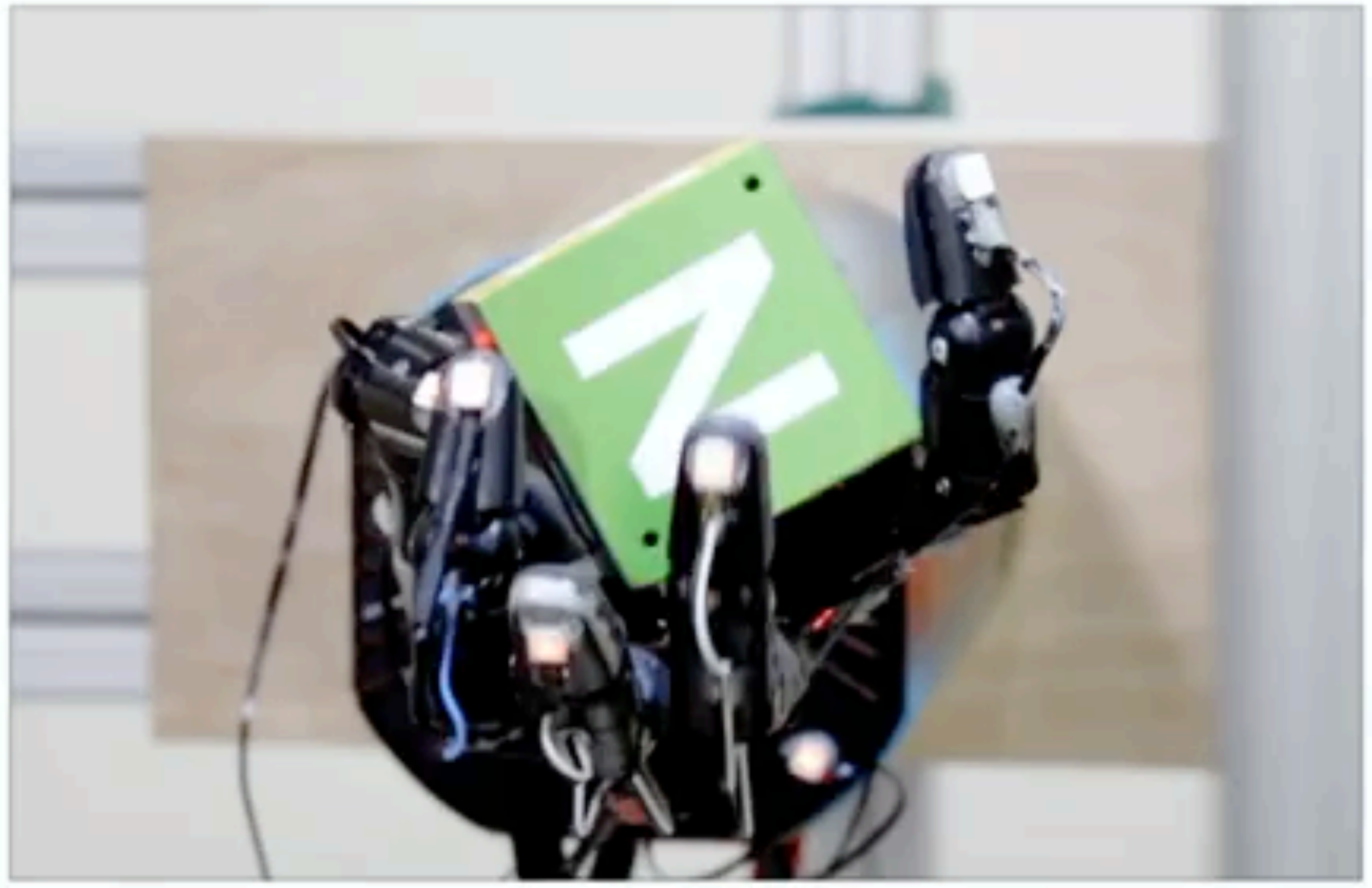
https://openai.com/blog/learning-dexterity/


A Distributed workers collect experience on randomized environments at large scale.
B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.
Observed Robot States
LSTM
Actions
C We train a convolutional neural network to predict the object pose given three simulated camera images.
CONV
CONV
CONV
Object Pose

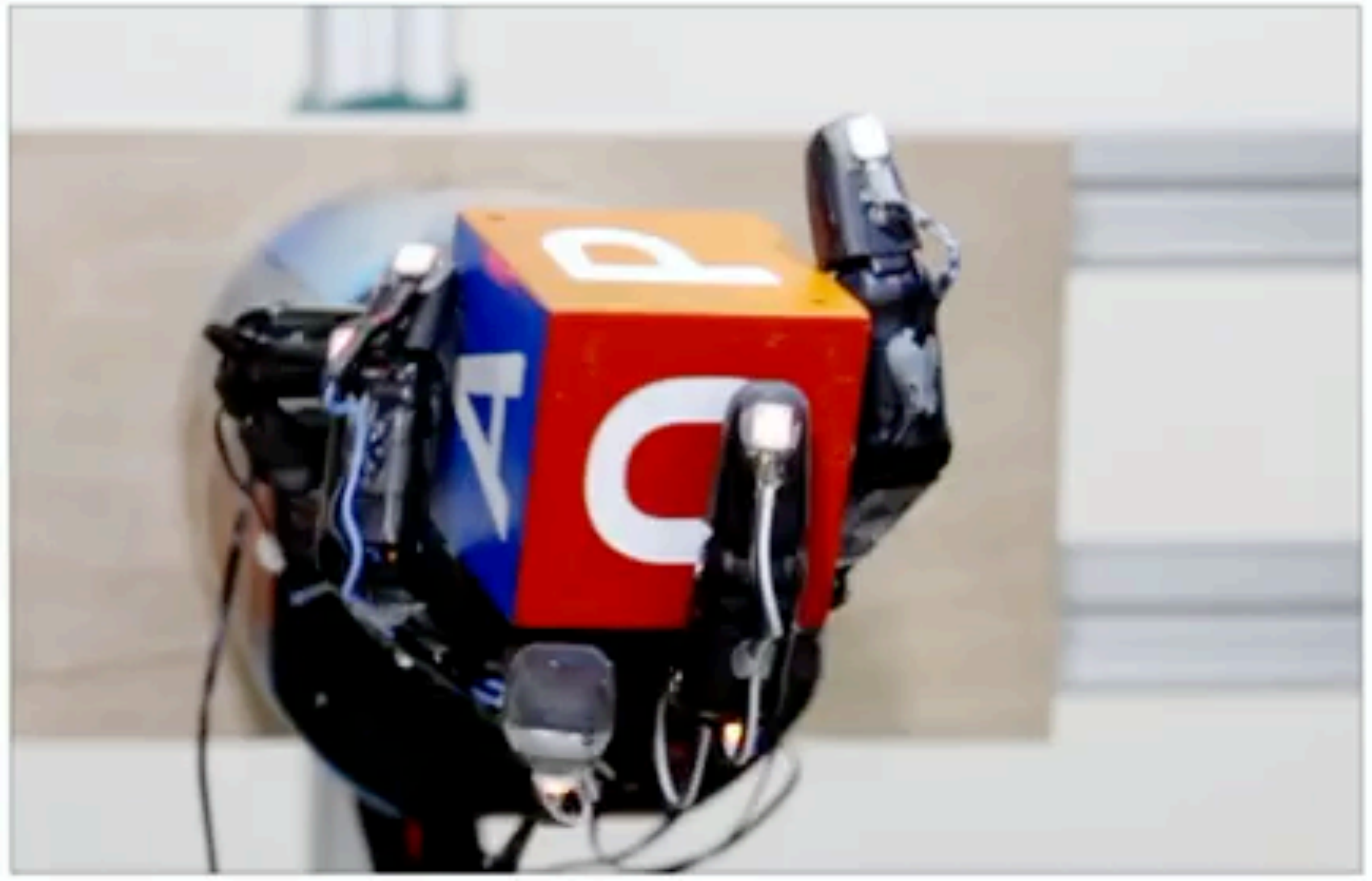
D We combine the pose estimation network and the control policy to transfer to the real world.



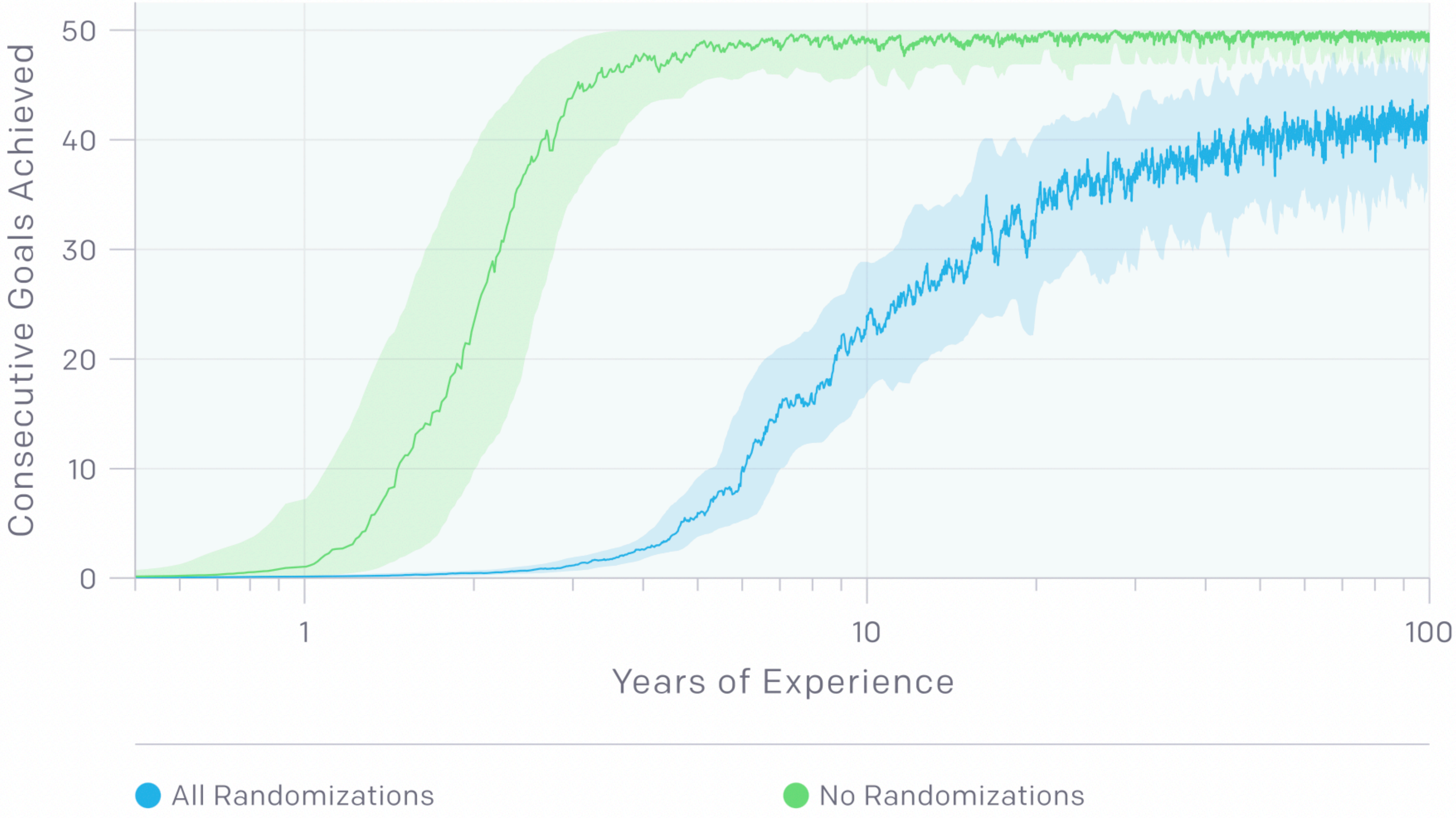
FINGER PIVOTING



SLIDING



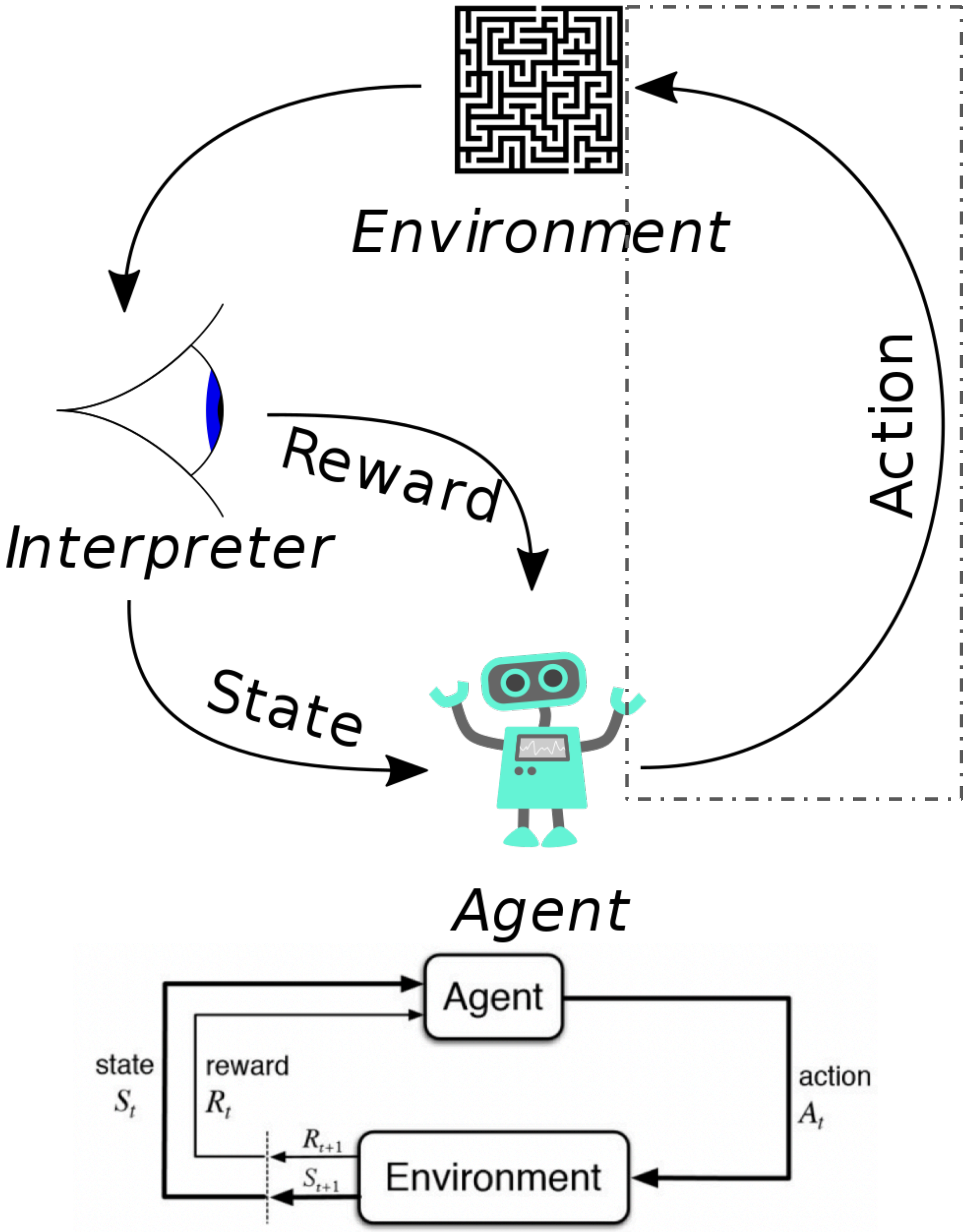
FINGER GAITING

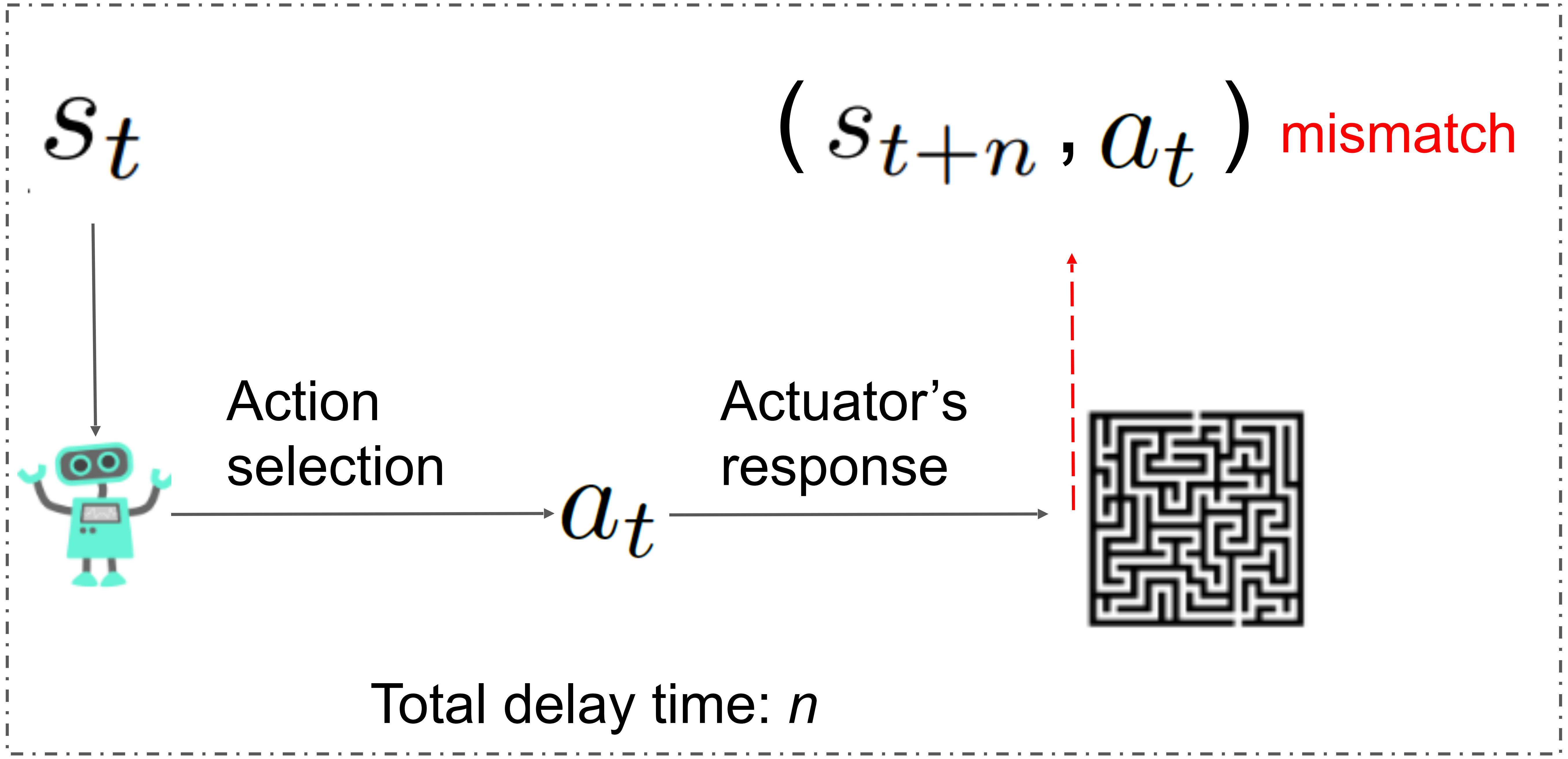


Learning progress with and without randomizations over years of simulated experience.

https://openai.com/blog/learning-dexterity/

Delay matters in real-world applications



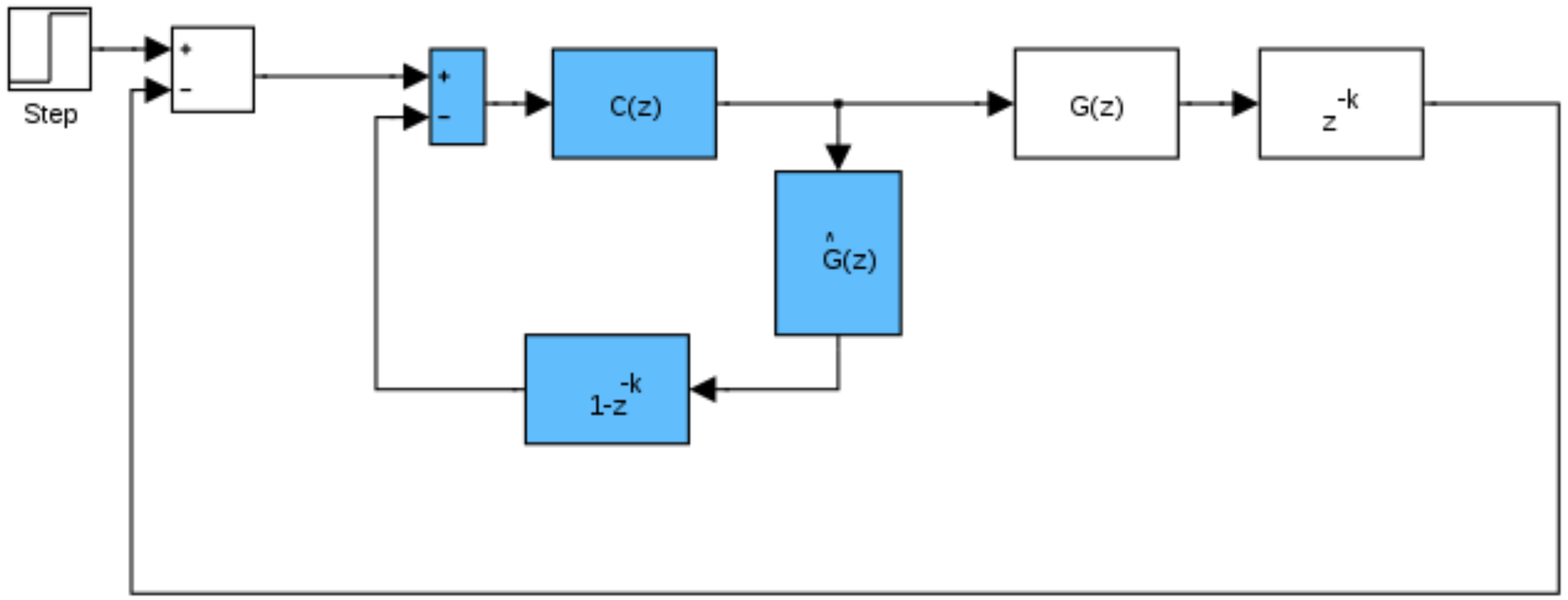


Delays are prevalent in the real world.
E.g., control freq of autonomous vehicles > 10 Hz,
While the hydraulic brake system delay > 0.4 seconds.

Chen, Baiming, et al. "Delay-aware model-based reinforcement learning for continuous control." *arXiv preprint arXiv:2005.05440* (2020).

Control of the delayed system

- Delays may not only **degrade the performance** of the agent but also **induce instability** to the dynamic systems. (Gu & Niculescu, 2003)
- The control community has proposed several methods to deal with delayed tasks. The most general approach is the Smith predictor (Astrom et al., 1994):

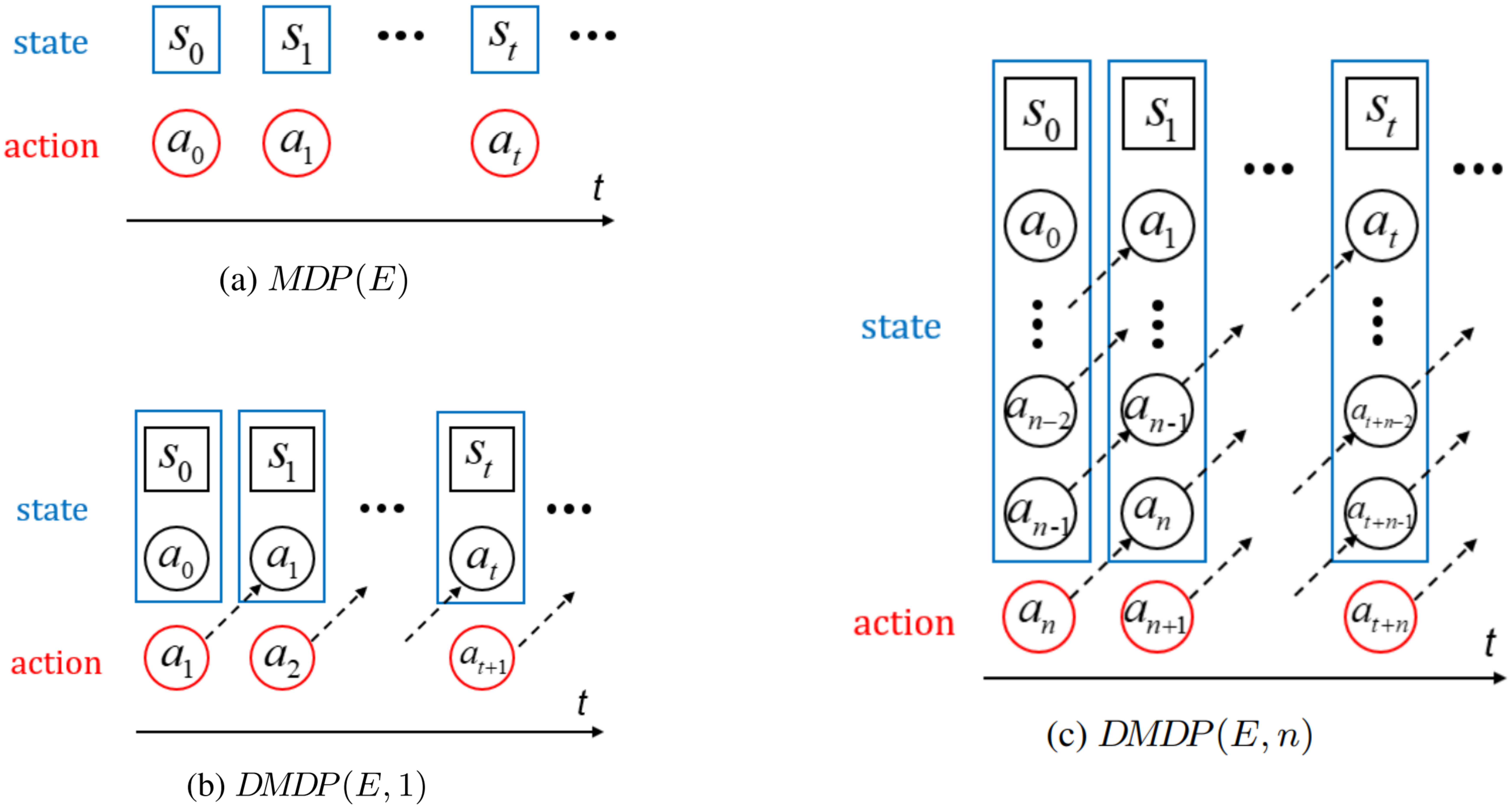


Basic idea: feedback control by predicting the future states.

Cons: requires a system model and is sensitive to model errors.

Chen, Baiming, et al. "Delay-aware model-based reinforcement learning for continuous control." *arXiv preprint arXiv:2005.05440* (2020).

Delayed Markov Decision Process (DMDP)



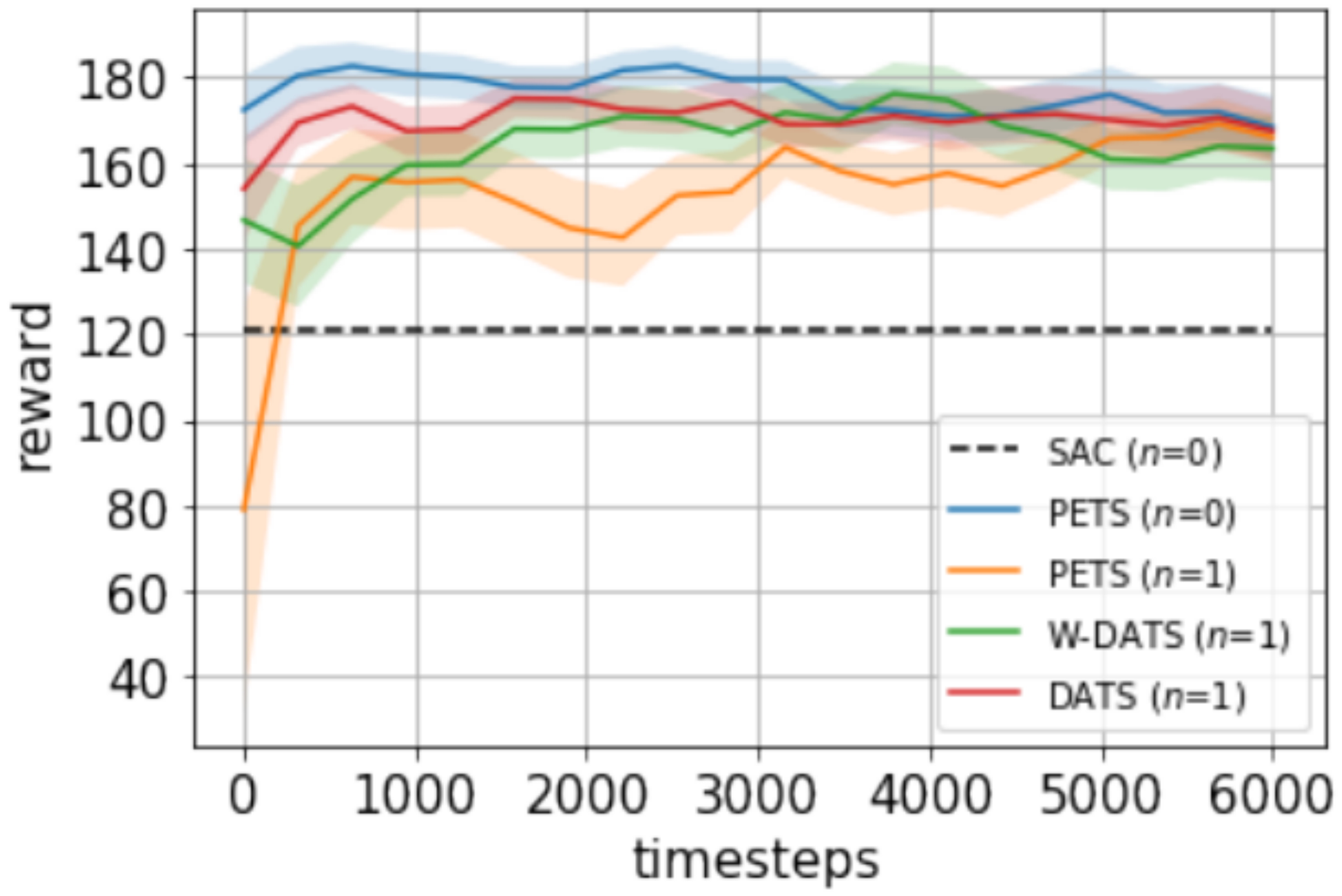
(a) $MDP(E)$

(b) $DMDP(E, 1)$

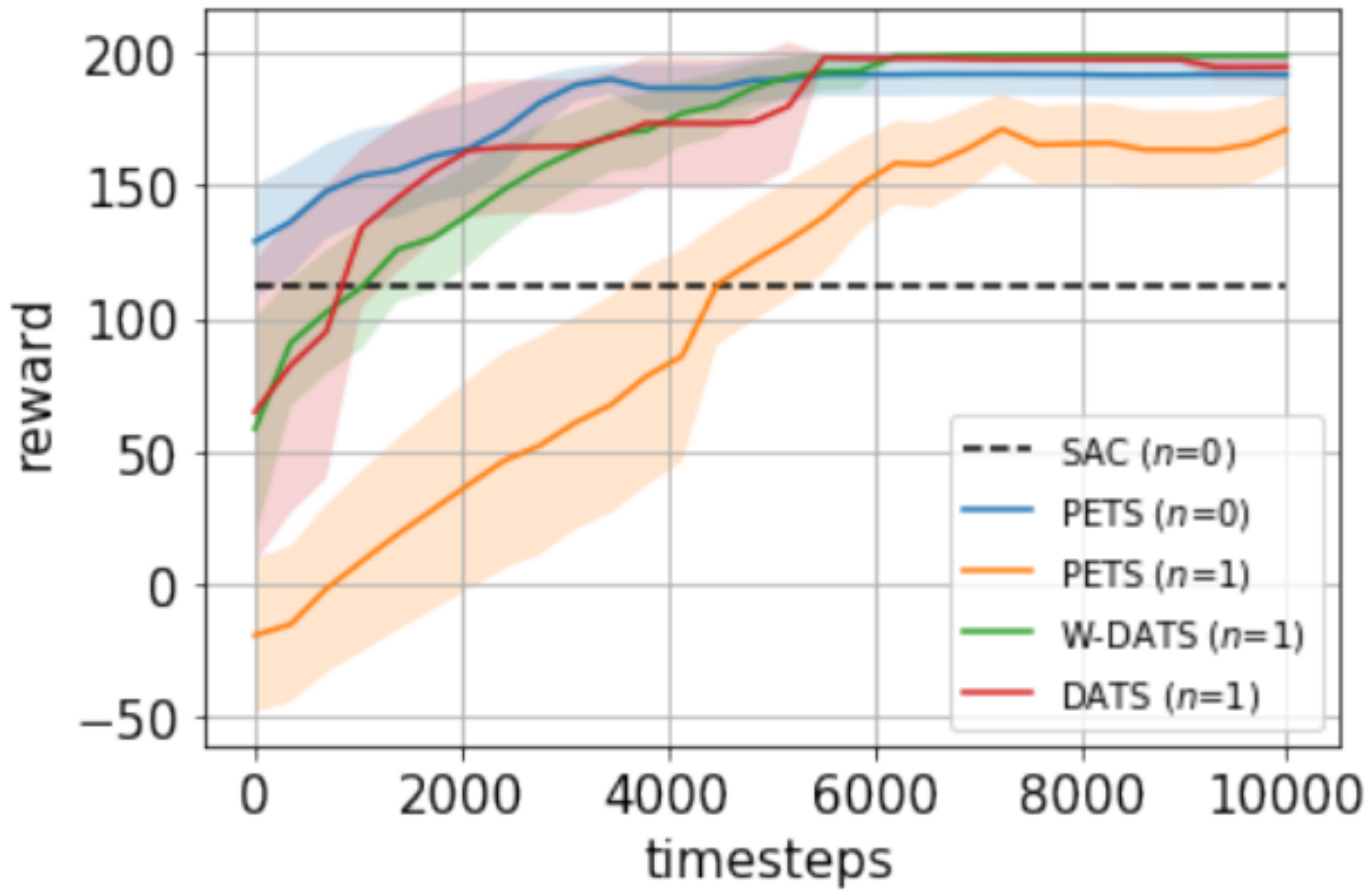
(c) $DMDP(E, n)$

Chen, Baiming, et al. "Delay-aware model-based reinforcement learning for continuous control." *arXiv preprint arXiv:2005.05440* (2020).

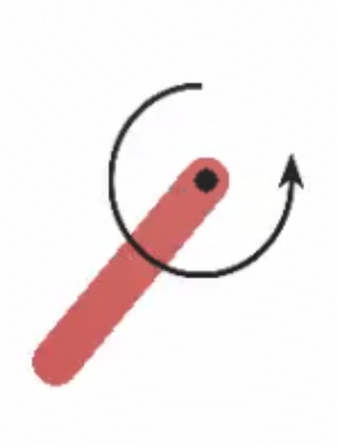
Experiment #1: the influence of delay

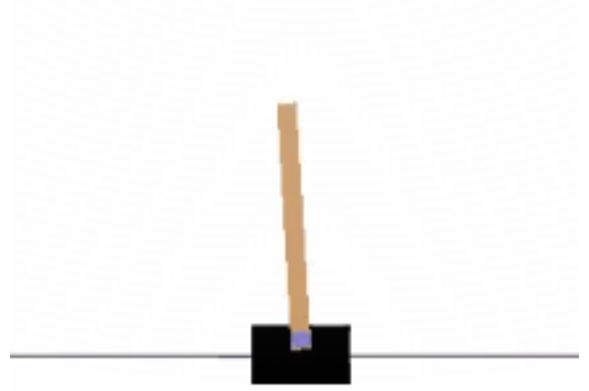


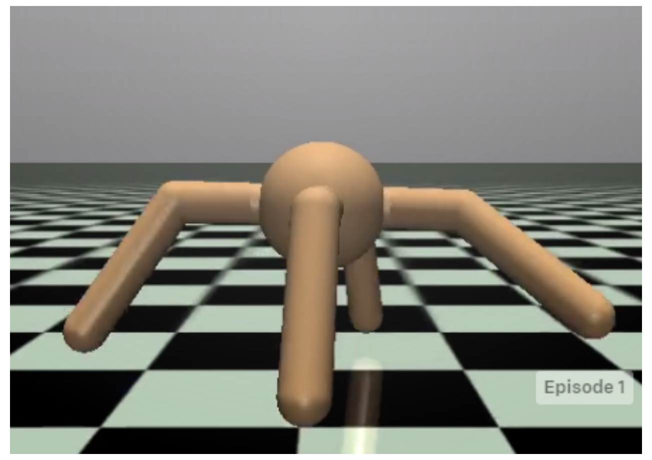
(a) Pendulum-v0

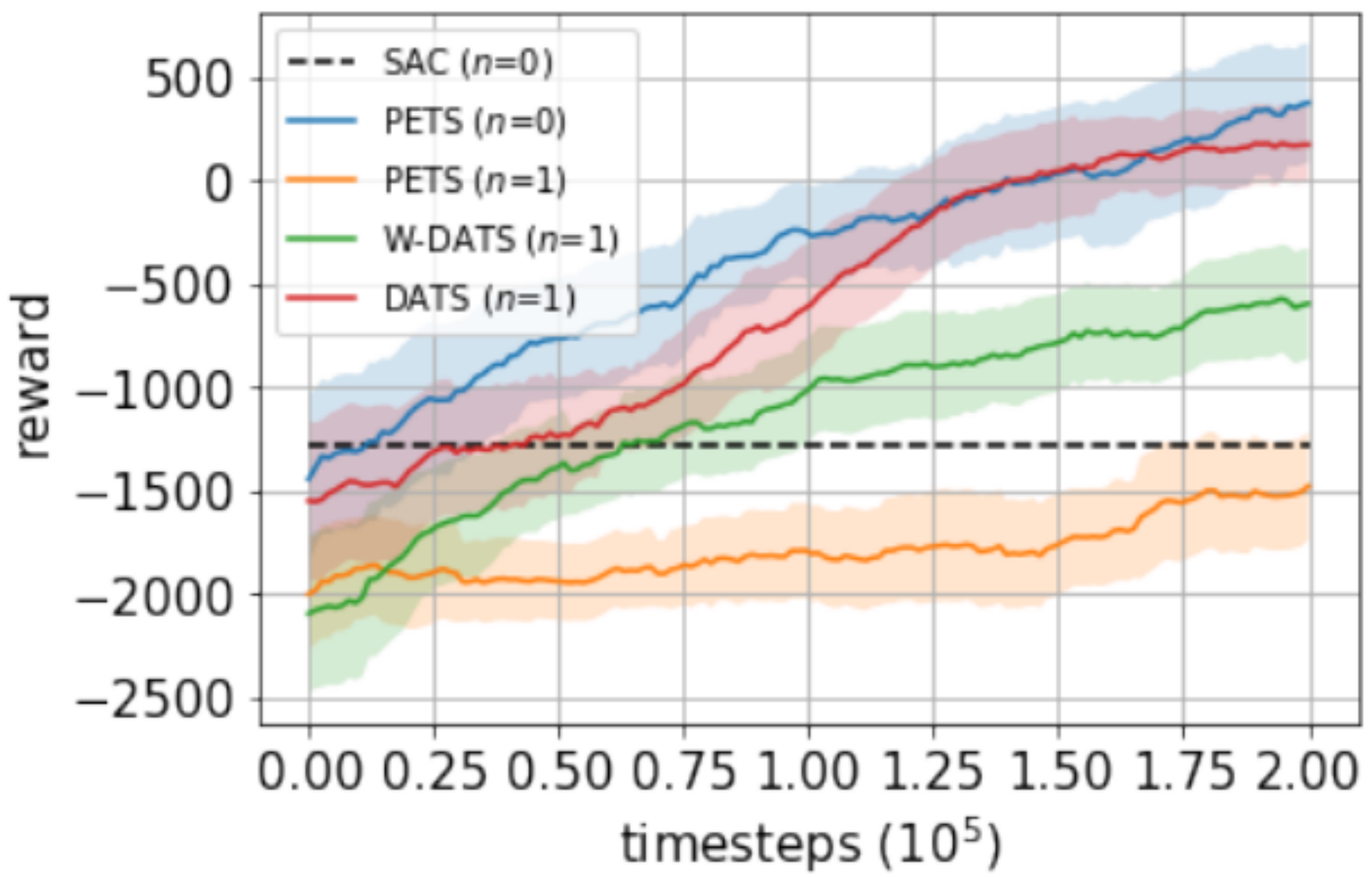


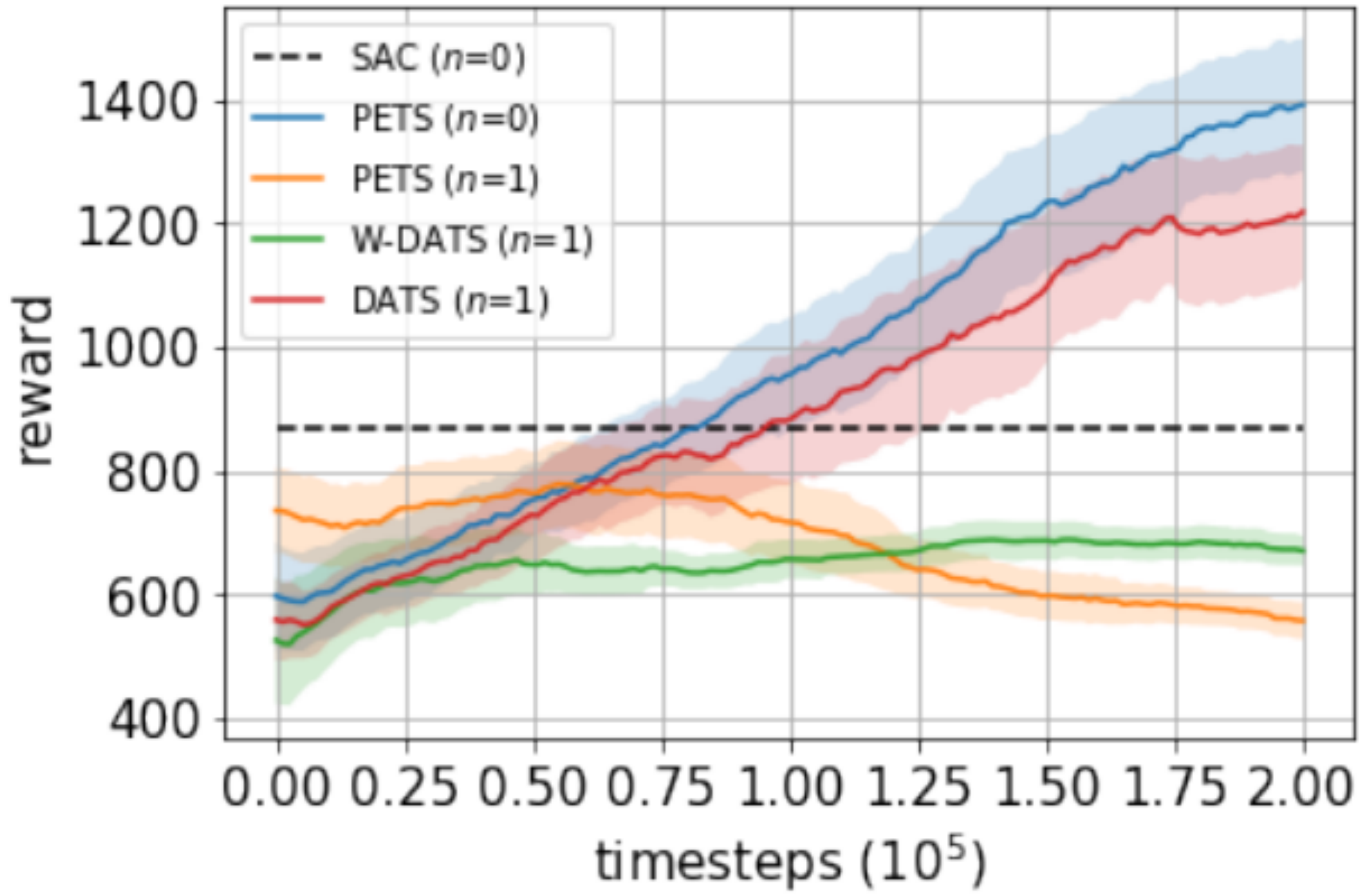
(b) CartPole-v1







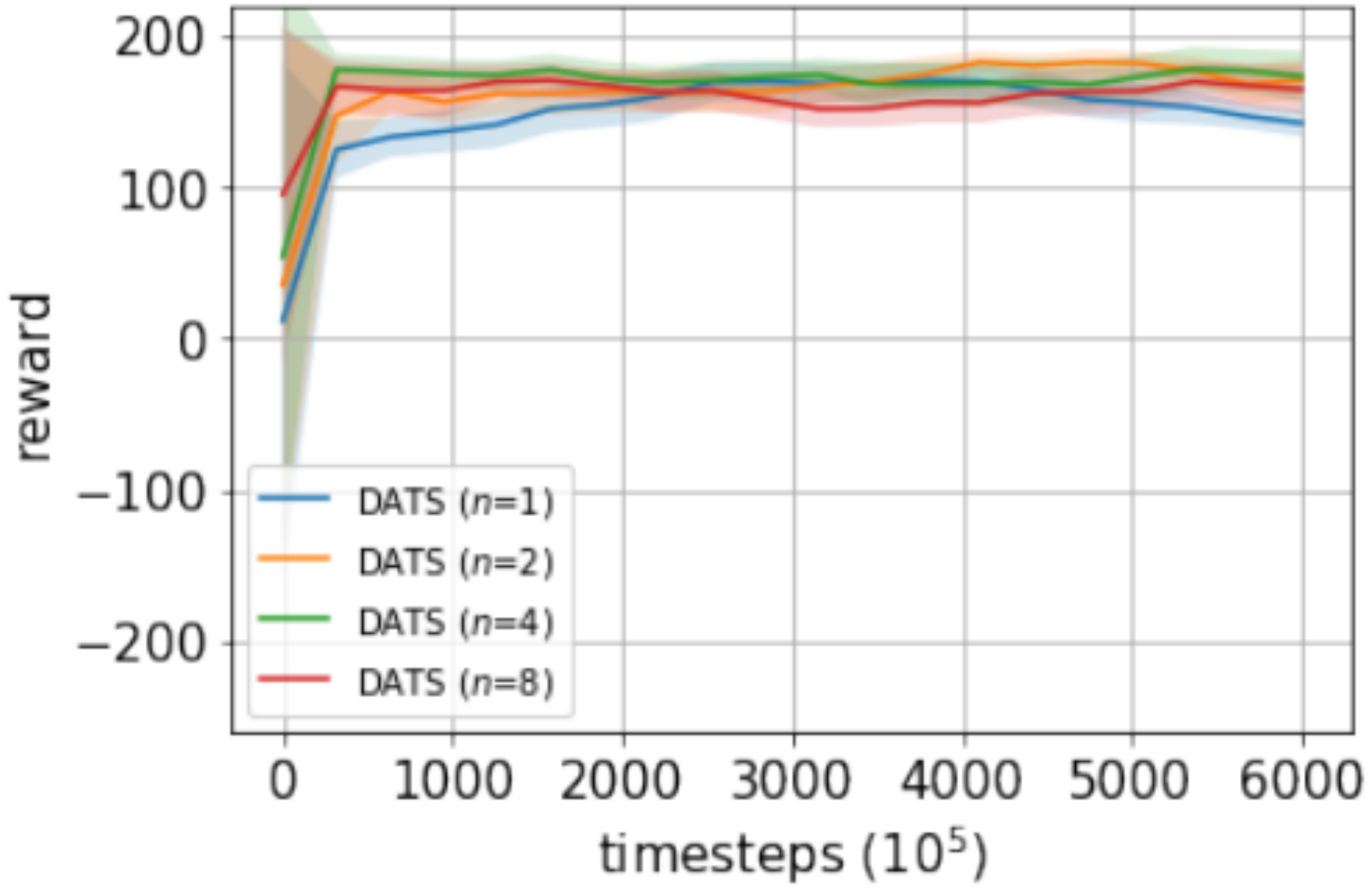




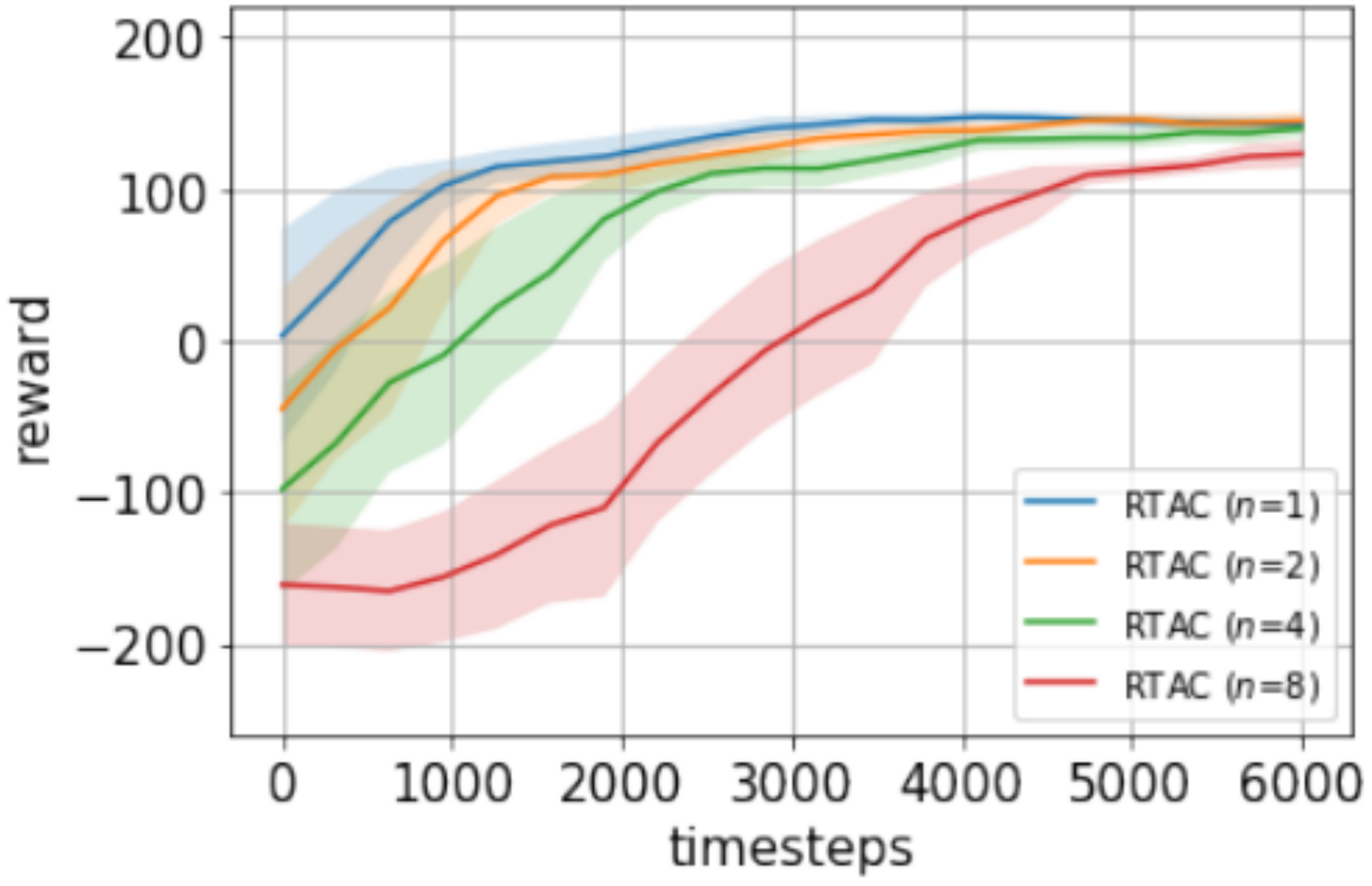
- SAC (n=0): soft actor-critic, SOTA model-free algorithm
- PETS (n=0): The original PETS algorithm in undelayed environment.
- PETS (n=1): The original PETS algorithm in the 1-step delayed environment but ignoring the action delay.
- W-DATS (n=1): PETS algorithm in the delayed environment that wastefully learns the whole dynamics of DMDPs.
- DATS (n=1): the proposed method introduced in Algorithm. 2.

Chen, Baiming, et al. "Delay-aware model-based reinforcement learning for continuous control." *arXiv preprint arXiv:2005.05440* (2020).

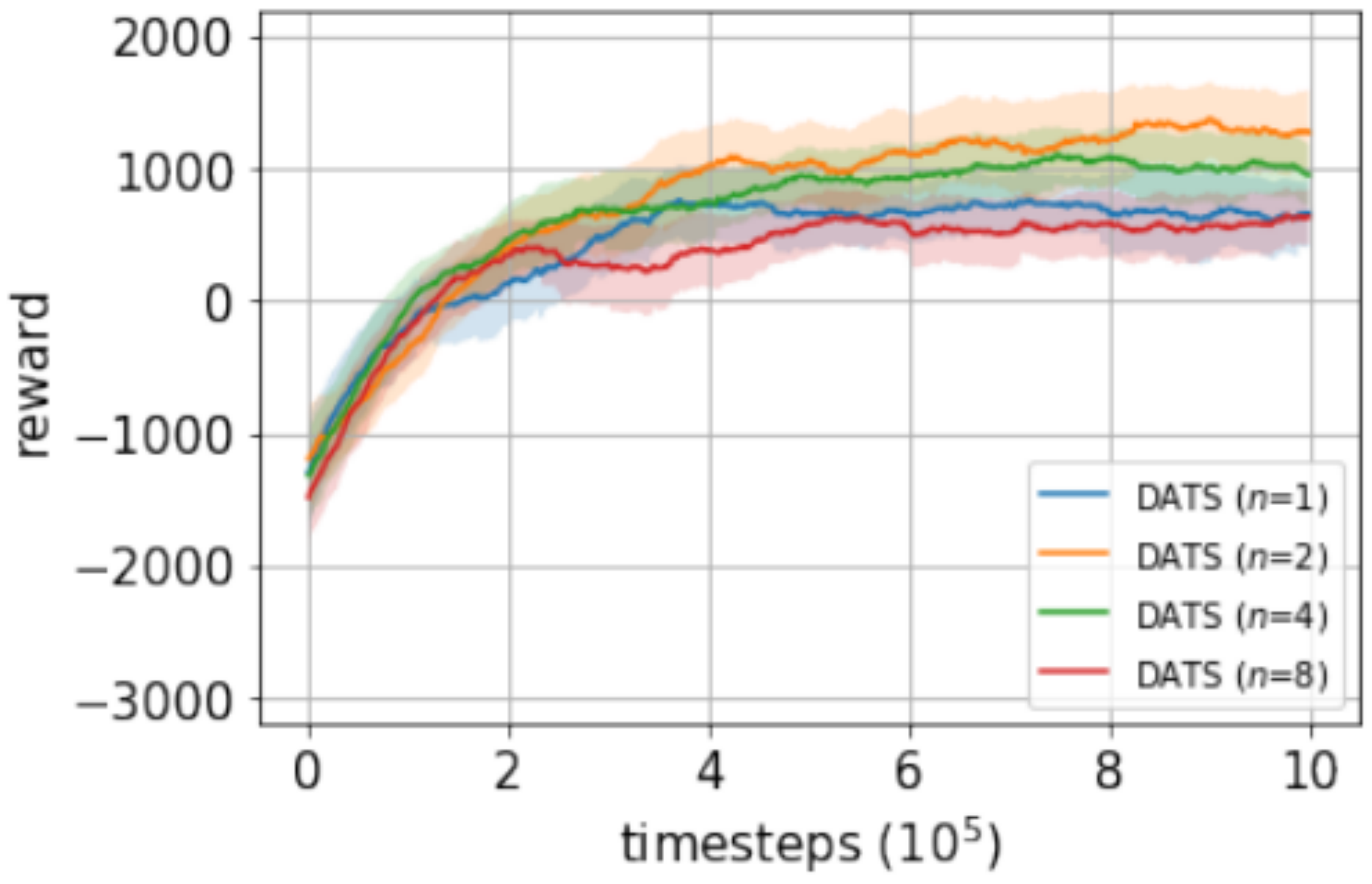
Experiment #2: model-based vs model-free



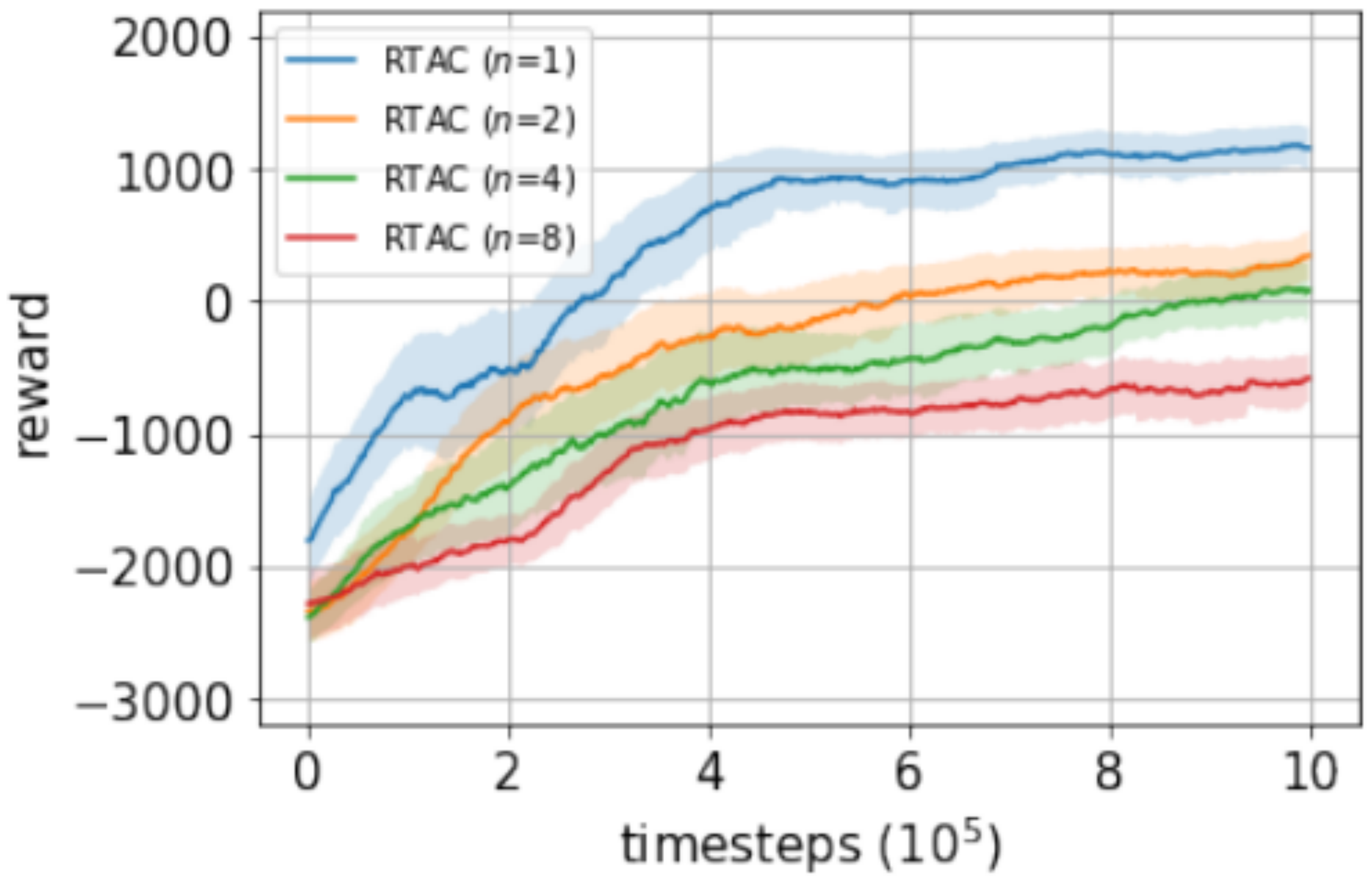
(a) DATS in Pendulum-v0



(b) RTAC in Pendulum-v0



(c) DATS in Walker2d-v1



(d) RTAC in Walker2d-v1

- DATS has stable performance when the delay step increases.
- RTAC degrades significantly as the delay step increases.

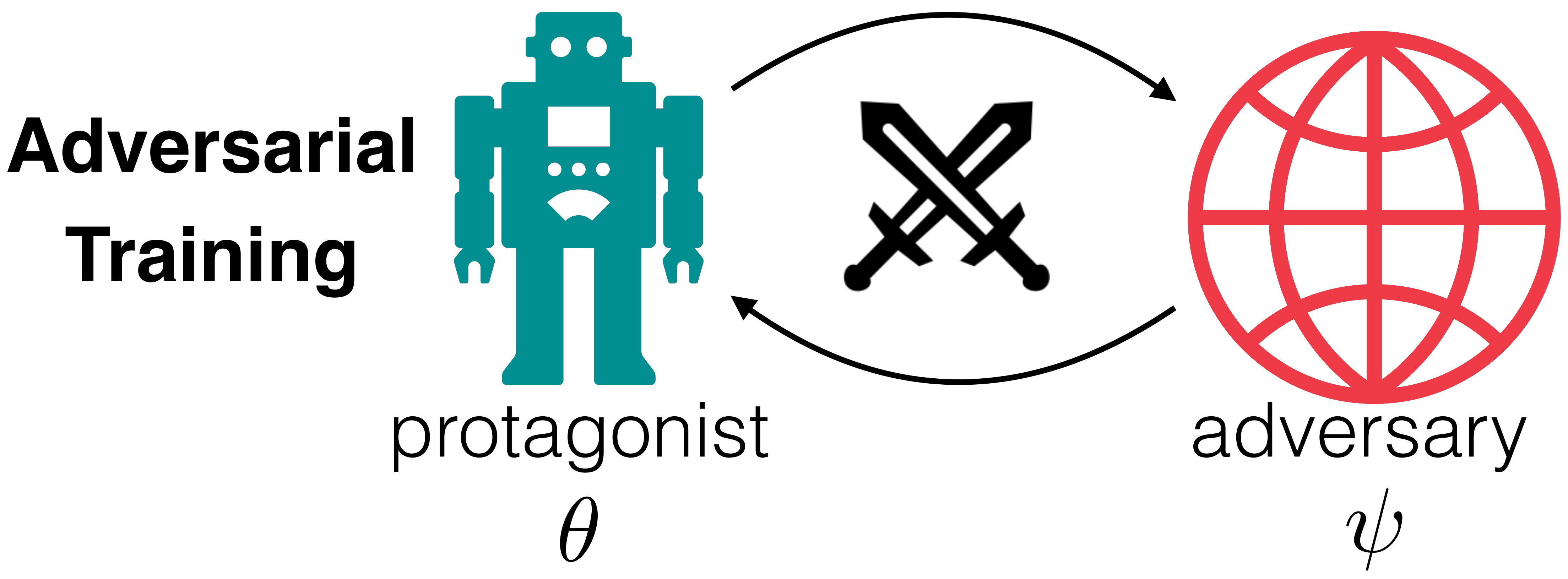
DATS: the proposed model-based method.

RTAC: model-free method, the updated SAC. (Ramstedt & Pal, 2019)

Chen, Baiming, et al. "Delay-aware model-based reinforcement learning for continuous control." *arXiv preprint arXiv:2005.05440* (2020).

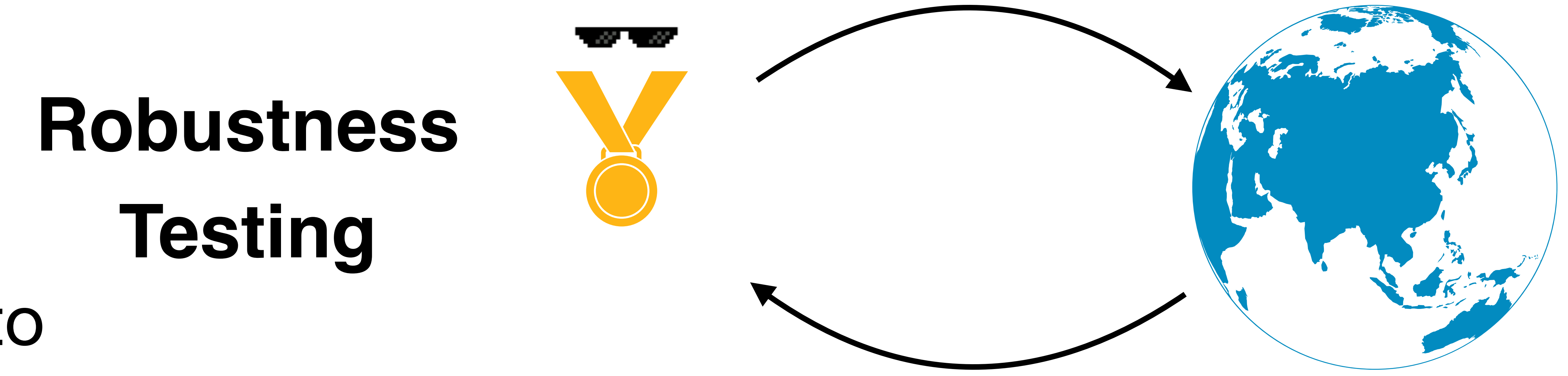
Robust Adversarial Reinforcement Learning (RARL)

- Main agent: *protagonist*
- Environment agent: *adversary*
    - *Consider env as the adversary*



Key idea:

- Adversarial training as a two-player zero-sum game
    - The protagonist maximizes $\mathbb{E}_{\tau\sim\theta,\psi}\left[R\left(\tau\right)\right]$
    - The adversary minimizes $\mathbb{E}_{\tau\sim\theta,\psi}\left[R\left(\tau\right)\right]$
    - Use gradient-descent-ascent-based algorithm to train the protagonist and adversary



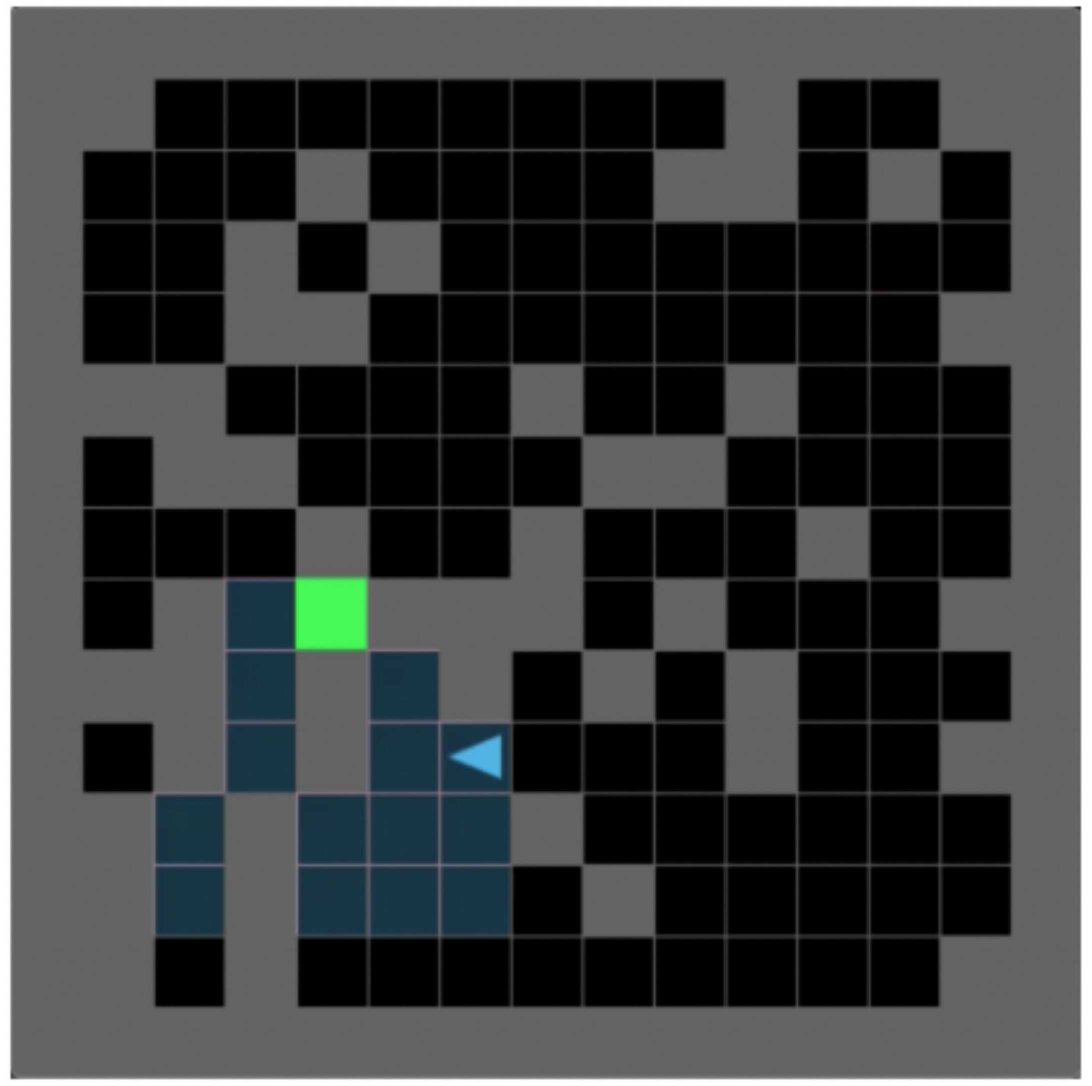
τ is the trajectories sampled using policy θ and ψ
$R(\tau)$ is the return of the protagonist

[1] Pinto et al., 2017

Limitations of existing RARL Methods

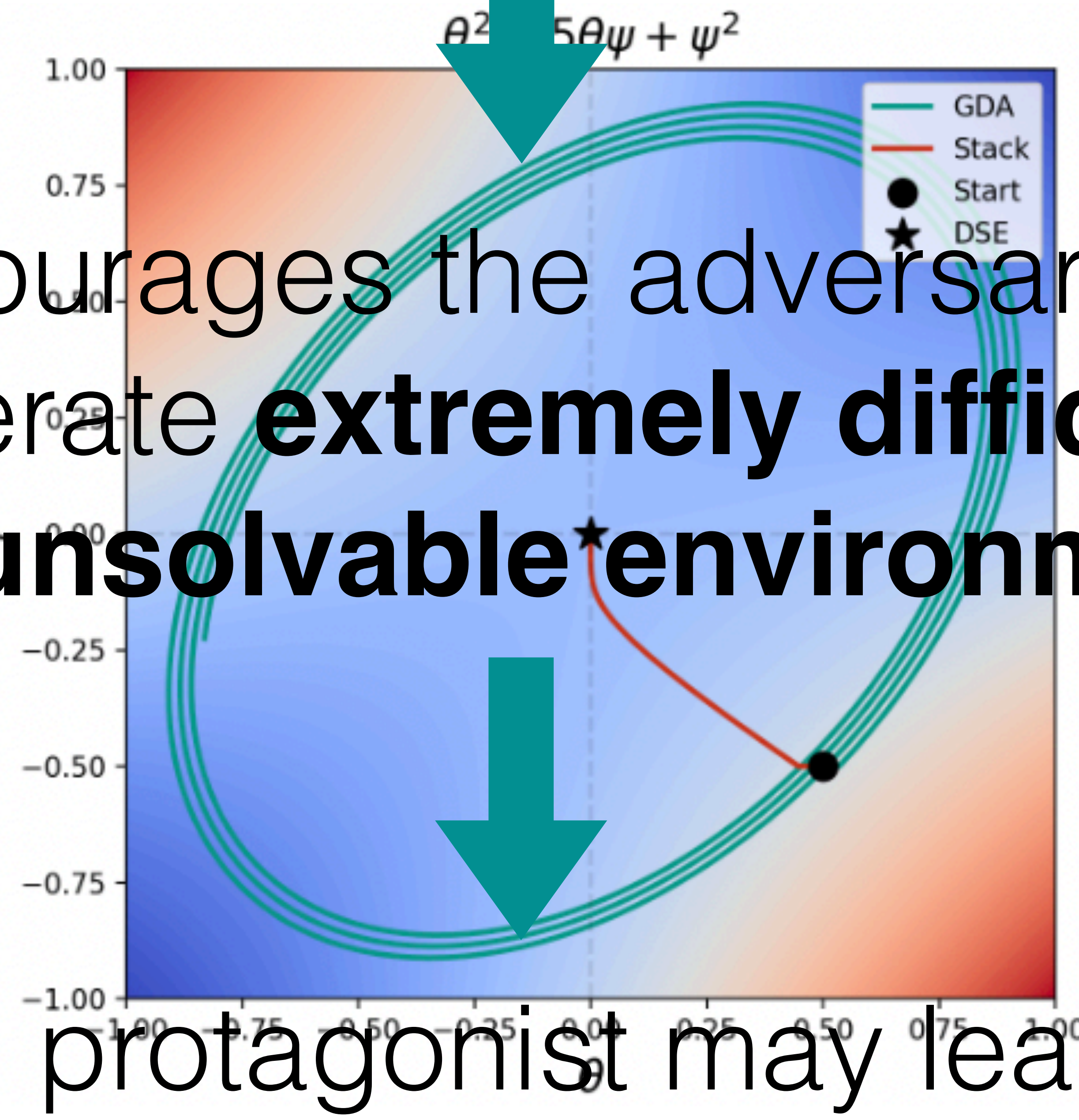
①

[Dennis et al., 2020]



Existing works: **Zero-sum**

②



Encourages the adversary to generate **extremely difficult**, even **unsolvable environments**

The protagonist may learn an **overly conservative** strategy or even **not be able to learn**

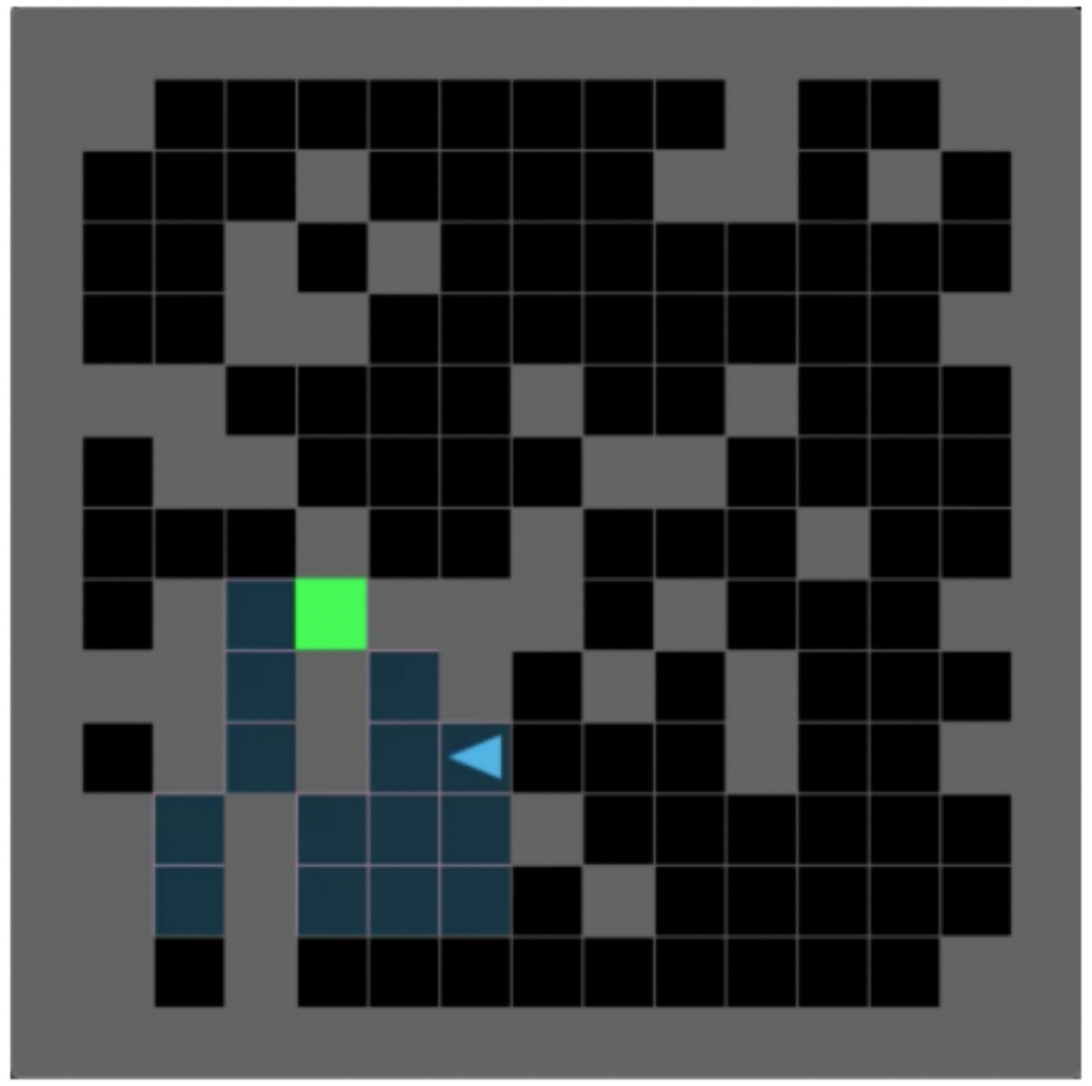
Unstable training

Liu, Z., Cen, Z., Isenbaev, V., Liu, W., Wu, Z.S., Li, B., & Zhao, D. (2022). Constrained Variational Policy Optimization for Safe Reinforcement Learning. *ArXiv, abs/2201.11927*.

Limitations of existing RARL methods

①

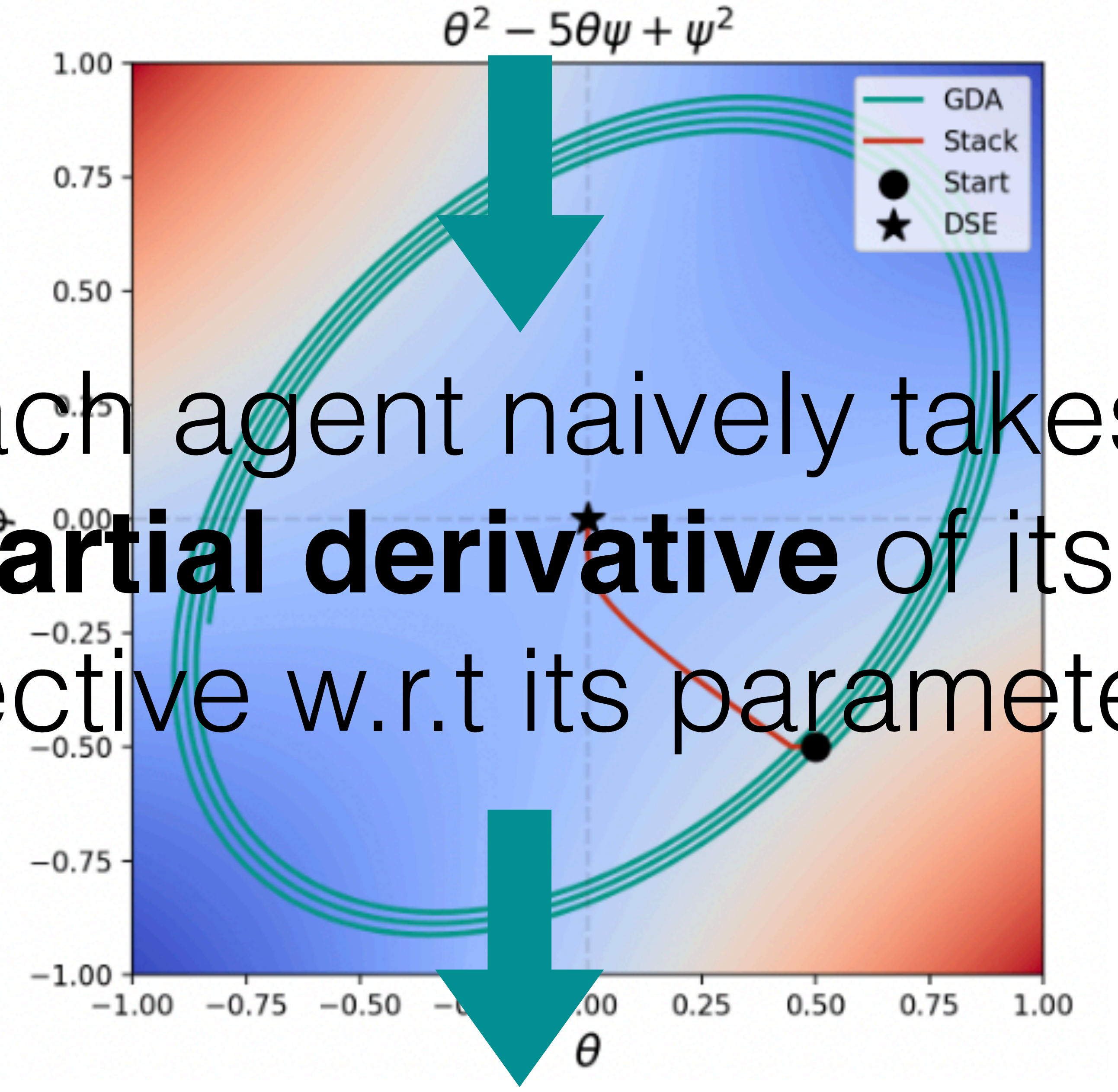
[Dennis et al., 2020]



②

Existing work:

gradient-descent-ascent

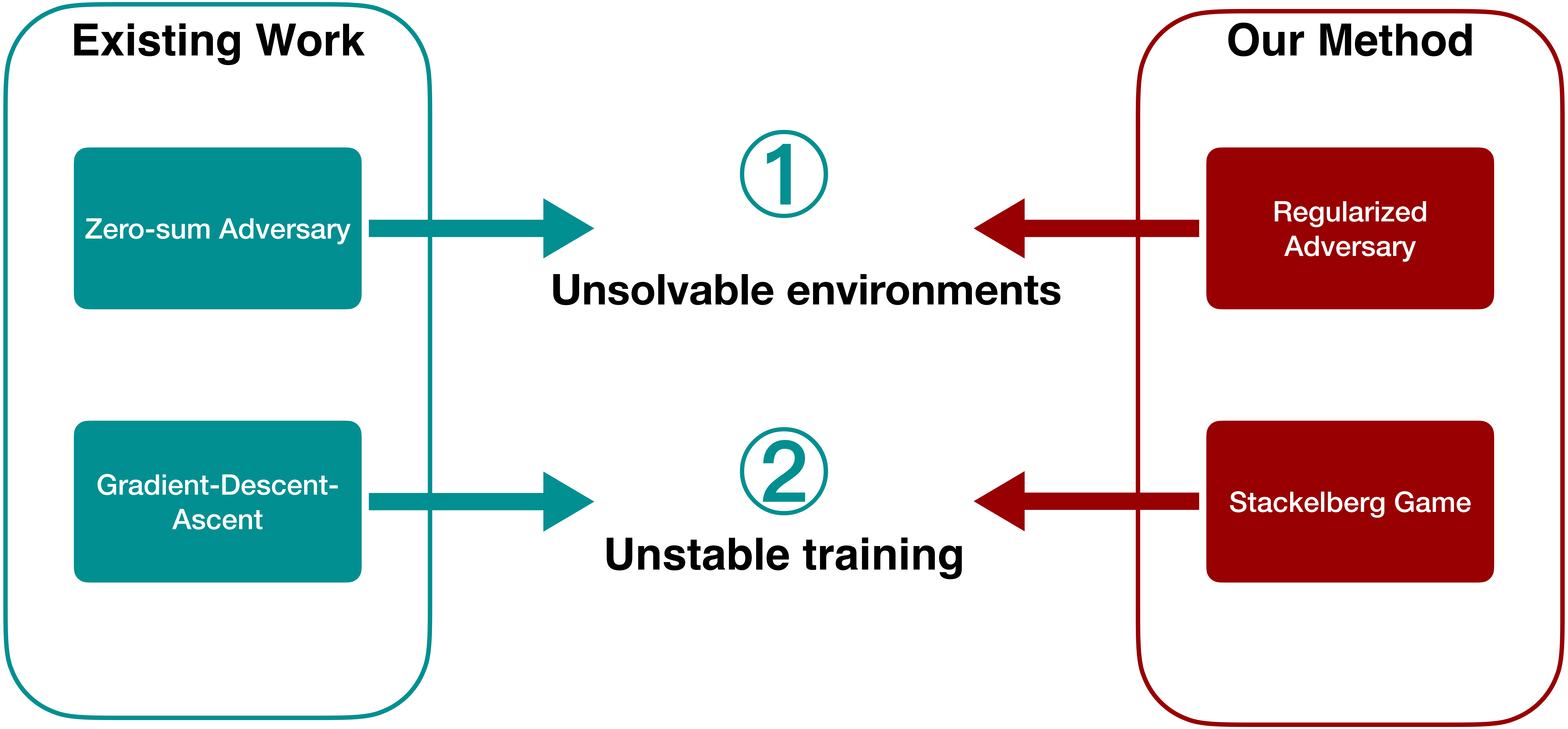


Each agent naively takes **partial derivative** of its objective w.r.t its parameters

Unstable Training

Unstable training

Robust Reinforcement Learning as a Stackelberg Game via Adaptively-Regularized Adversarial Training

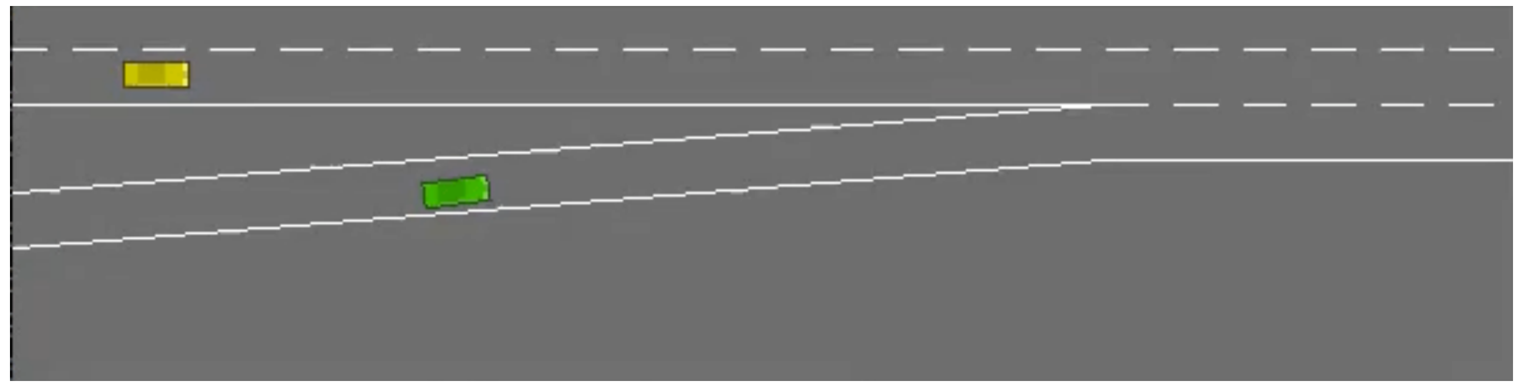


Huang P, Xu M, Fang F, Zhao D. Robust Reinforcement Learning as a Stackelberg Game via Adaptively-Regularized Adversarial Training. arXiv preprint arXiv:2202.09514. 2022

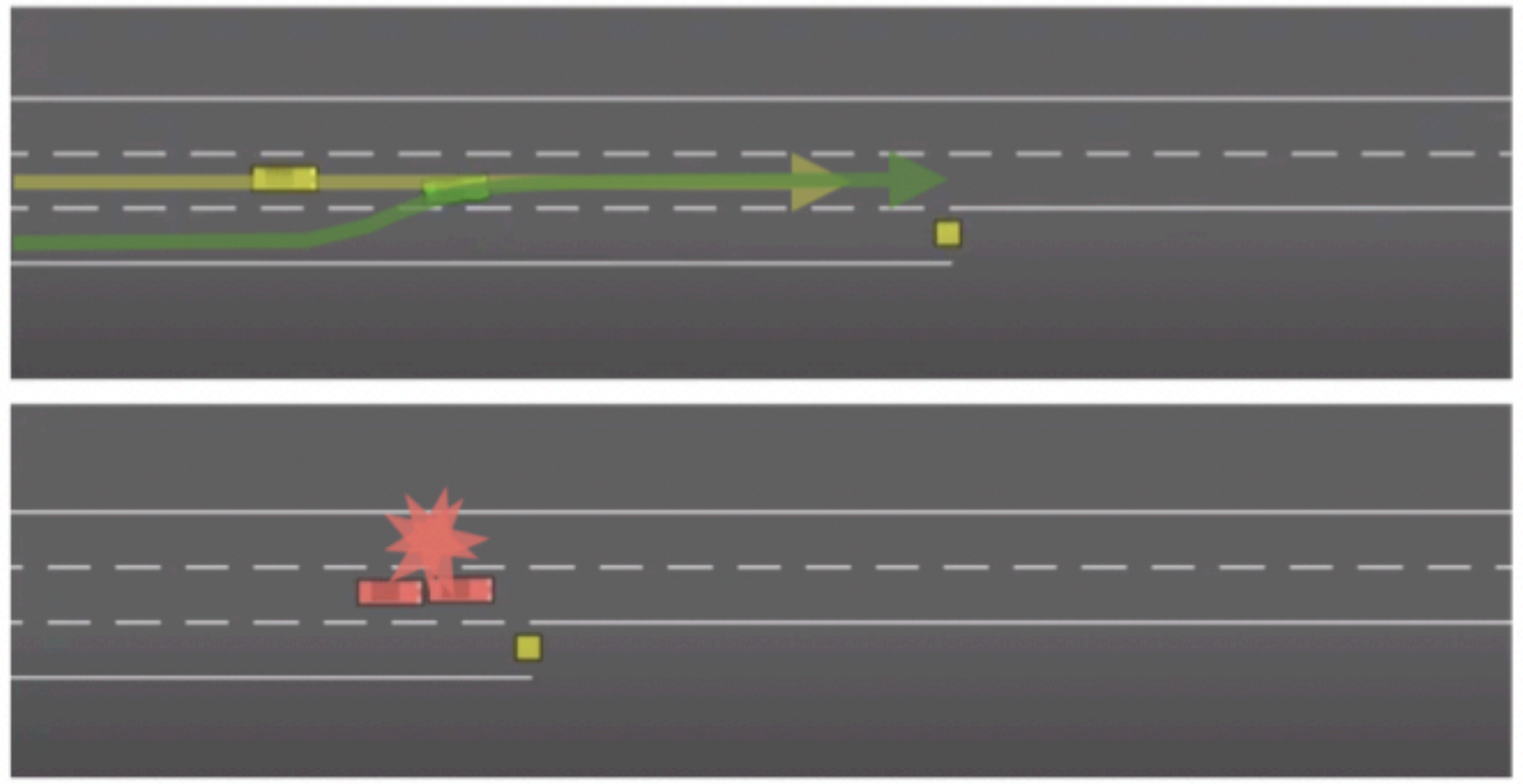
Encourage Challenging but Solvable Environments

- No-Adv: the protagonist is not aware of the danger
- RARL: the adversary generates unsolvable environments
- RRL-Stack: the adversary generates challenging but solvable environments

: Protagonist **: Adversary**



No-Adv

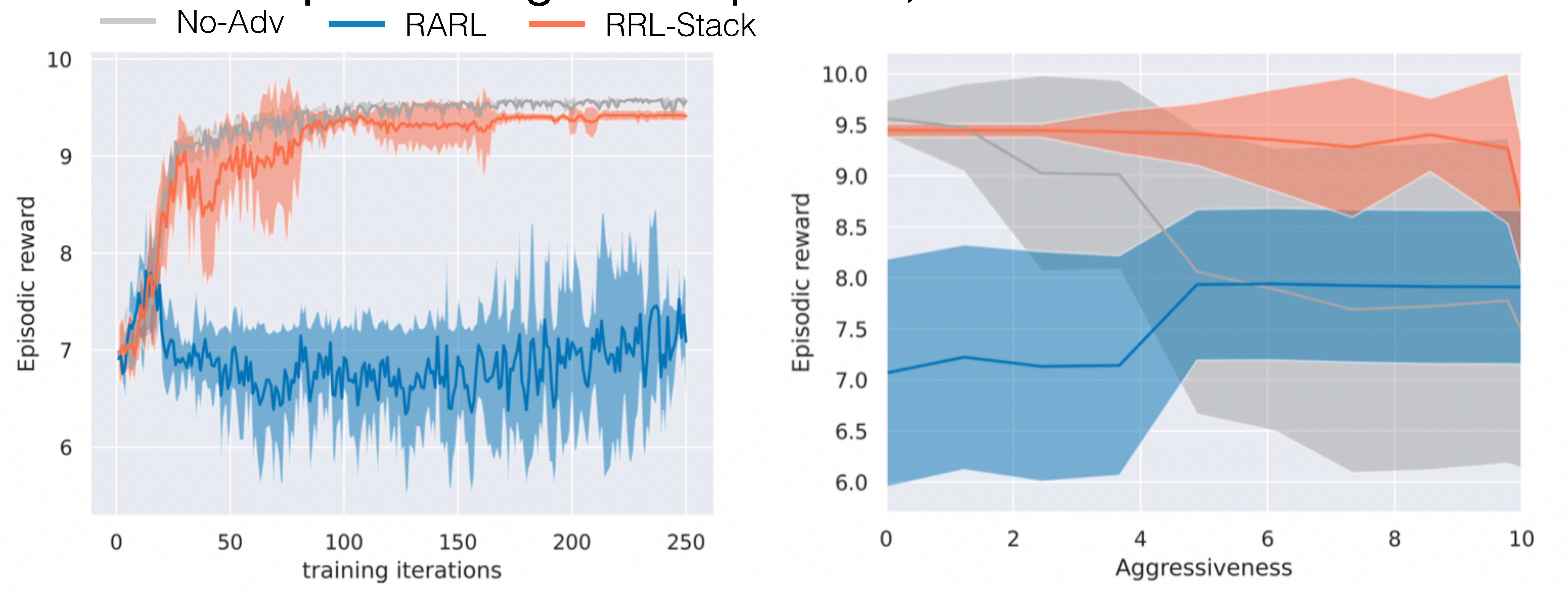


Huang P, Xu M, Fang F, Zhao D. Robust Reinforcement Learning as a Stackelberg Game via Adaptively-Regularized Adversarial Training. arXiv preprint arXiv:2202.09514. 2022

Improve Training Stability and Robustness

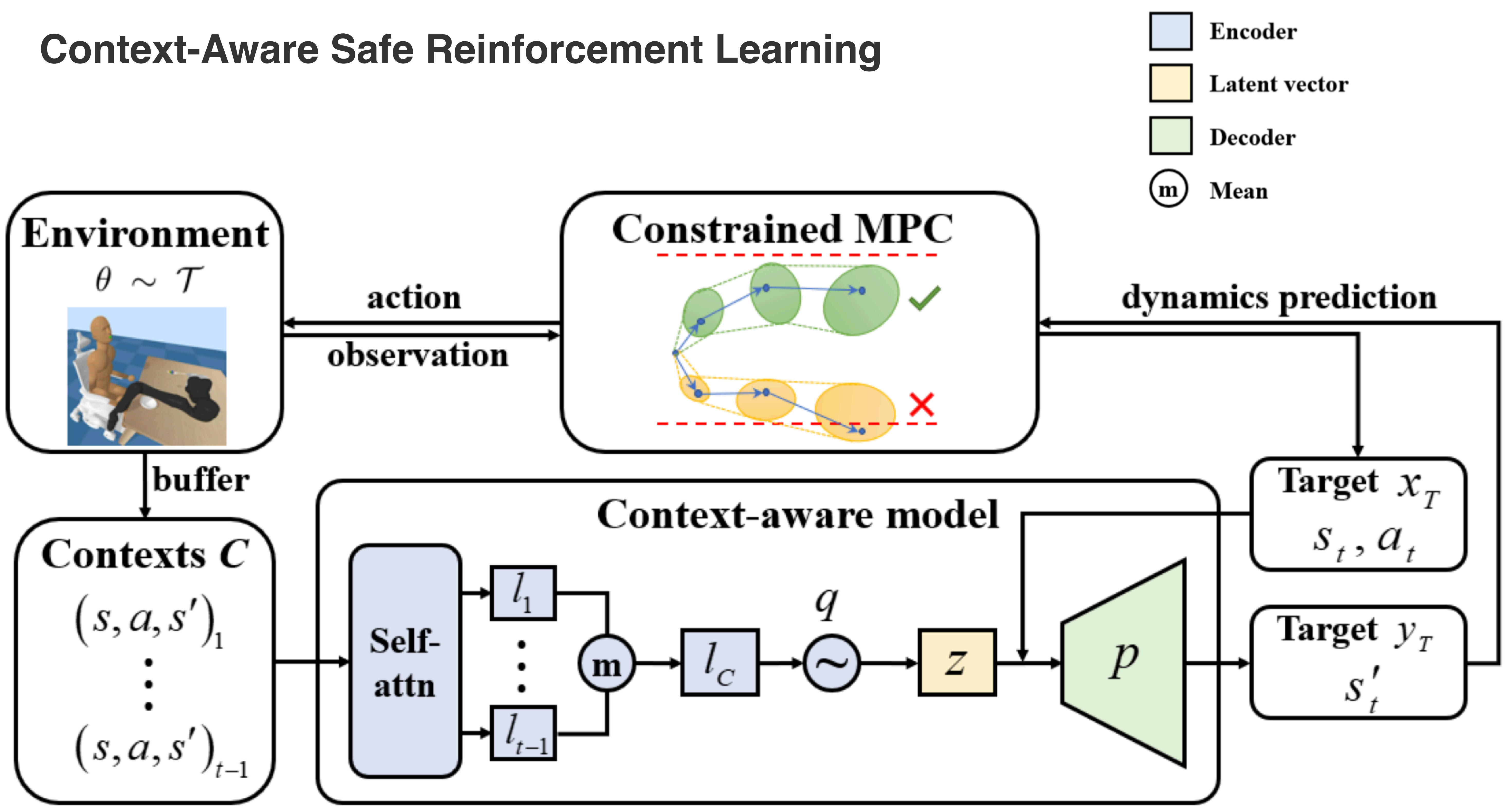
- No-Adv: is stable; not robust
- RARL: diverges at around 20 iterations; not robust
- RRL-Stack: keeps learning robust policies; robust

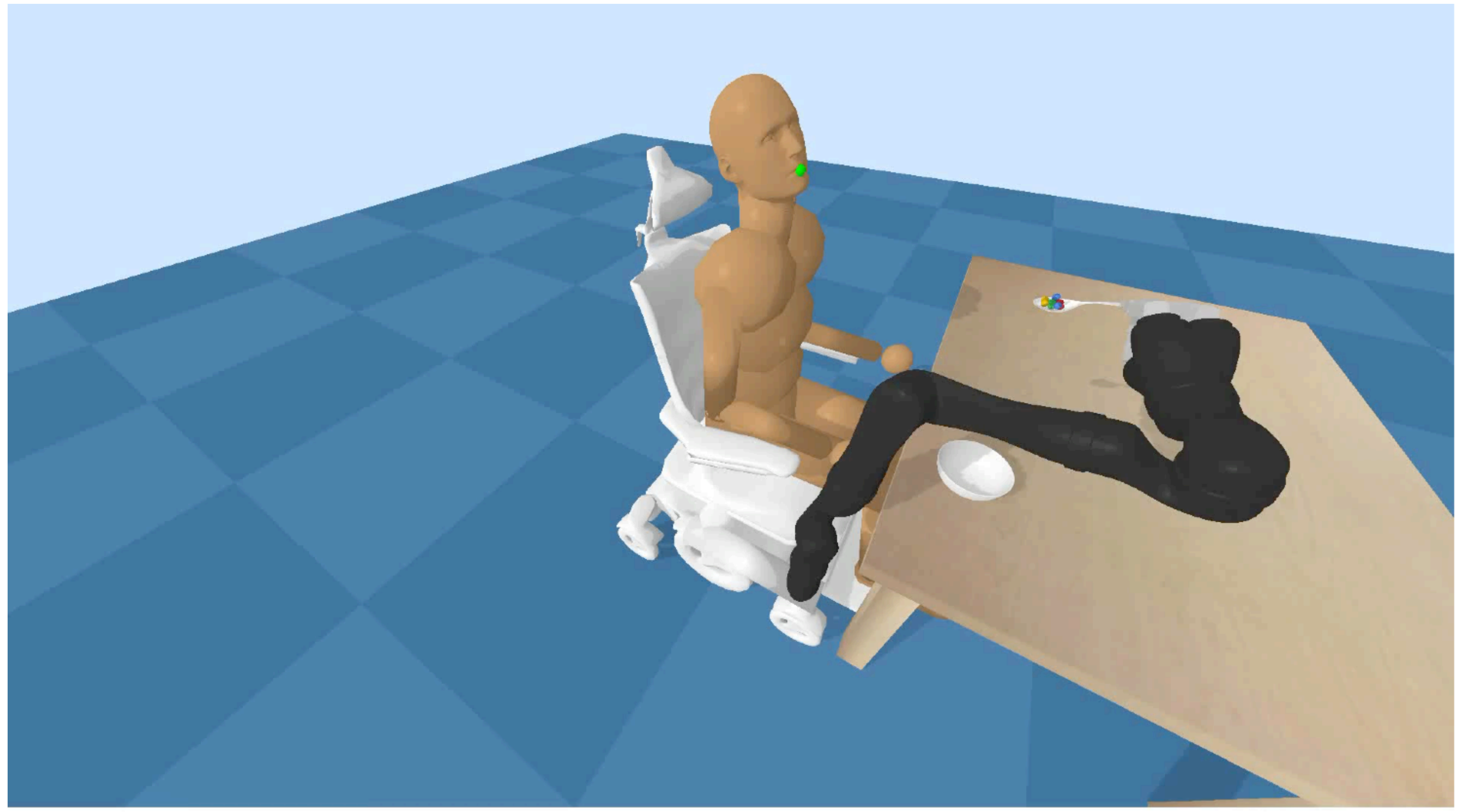
How to deal with a changing environment?



Huang P, Xu M, Fang F, Zhao D. Robust Reinforcement Learning as a Stackelberg Game via Adaptively-Regularized Adversarial Training. arXiv preprint arXiv:2202.09514. 2022

Safe reinforcement learning for non-stationary environments





What if the contexts are very different?

B. Chen *et al.*, "Context-Aware Safe Reinforcement Learning for Non-Stationary Environments," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021

Worth reading

- Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, Sergey Levine, Sergay, “How to Train Your Robot with Deep Reinforcement Learning – Lessons We’ve Learned,” Journal of Robotics Research (IJRR), 2021

- Kirk R, Zhang A, Grefenstette E, Rocktäschel T. A survey of generalisation in deep reinforcement learning. arXiv preprint arXiv:2111.09794. 2021 Nov 18.