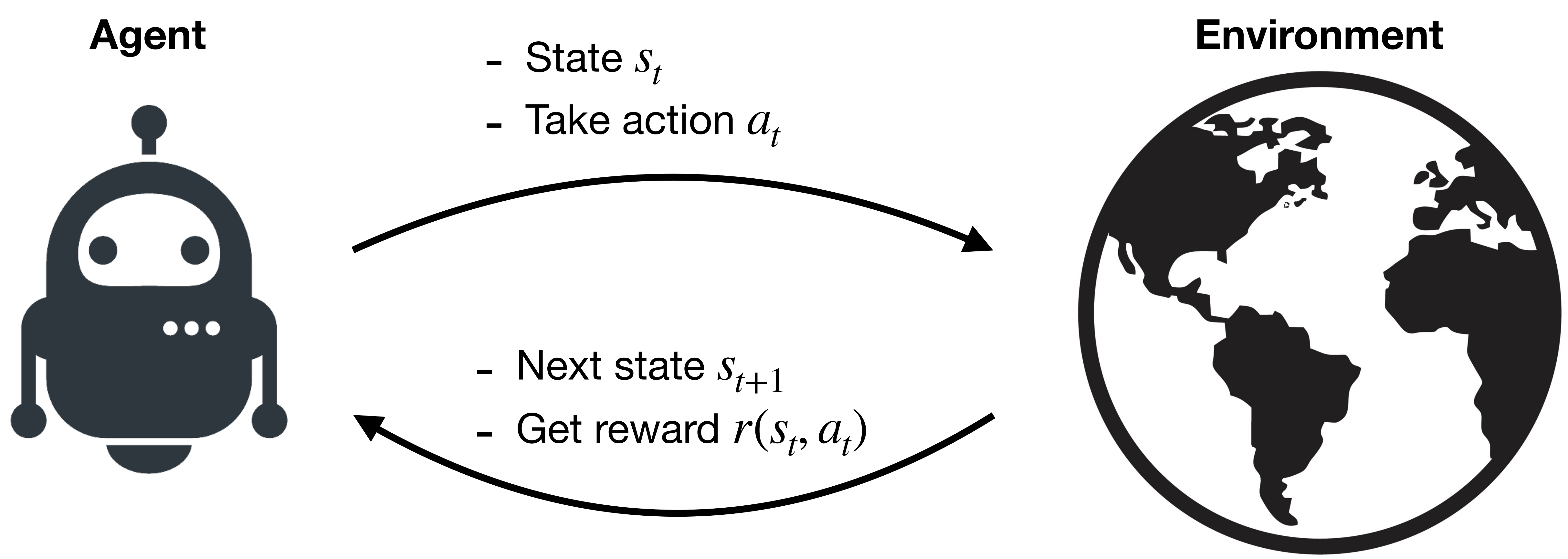
Introduction to Reinforcement Learning

What is reinforcement learning?

- **Sutton and Barto, 1998:** *"Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal".*

- **ChatGPT, 2022:** *"Reinforcement learning is a type of machine learning in which an* agent *learns to interact with its environment in order to maximize a reward signal".*



Recent advances

EPFL

2013

Atari

Deep Q-learning for Atari games [1].

2016

Energy saving

DeepMind AI reduces Google data centre cooling bill by 40% [3].

2017

AlphaGo/ AlphaZero

AI achieving grand master level in **chess, go,** and **shogi** [4,5].

2018

OpenAI Five

Training five artificial intelligence agents to play the **Dota 2** [6].

2019

Alpha Star

AI achieving grand master level in **StarCraft II** game [7].

Rubik's Cube

Solving **Rubik's Cube** with a human-like robot hand [8].

2022

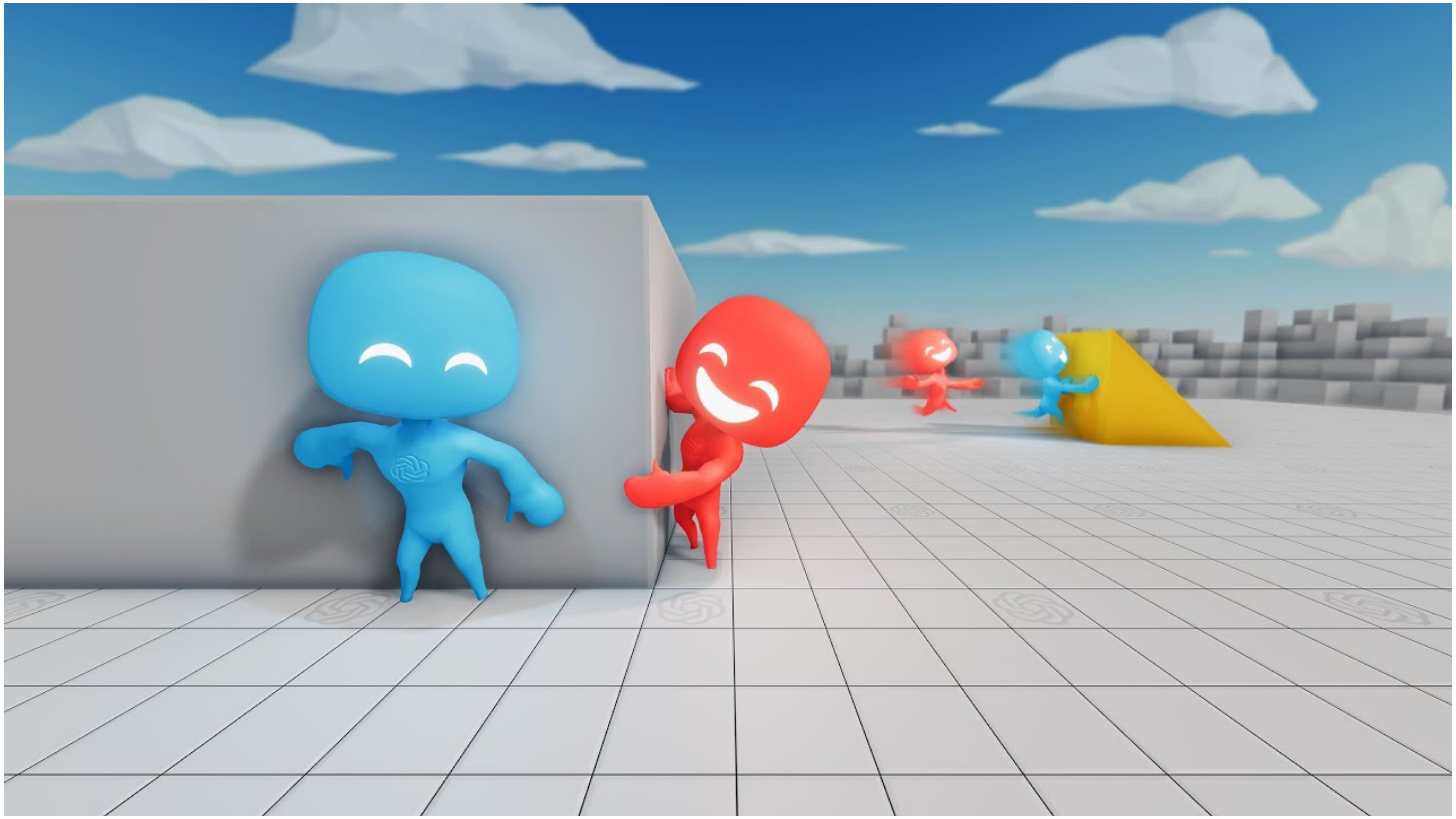
AlphaTensor

Discovering faster **matrix multiplication** algorithms [9].

ChatGPT

A **language model** trained to generate human-like responses to text input [10].

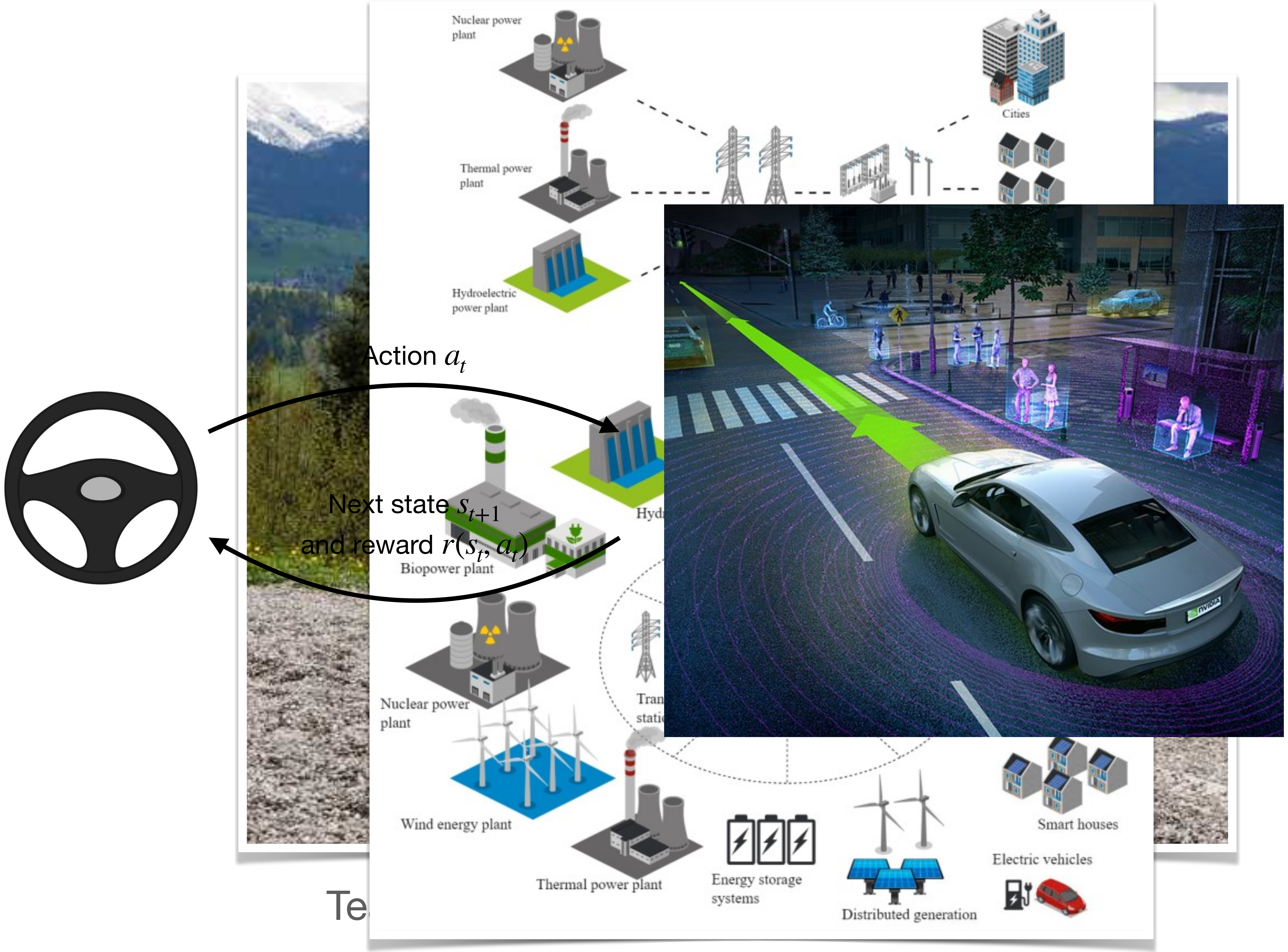
Example: Multi-agent game



2019: Learning to play hide and seek via multi-agent reinforcement learning [2].

(Potential) real world applications

- **Robotics**
- **Autonomous driving**
- **Control of power grids**



Te

Control of power grids [12].

Markov Decision processes

A **Markov decision process** is given by a tuple $\mathscr{M} = (\mathscr{S}, \mathscr{A}, P, \rho)$ where…

- $\mathscr{S}$ is the set of all possible states.
- $\mathscr{A}$ is the set of all possible actions.
- P is the transition law with $P(s' \mid s, a) = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$.
- ρ is the initial state distribution with $\rho(s) = Pr(s_0 = s)$.

Markov property

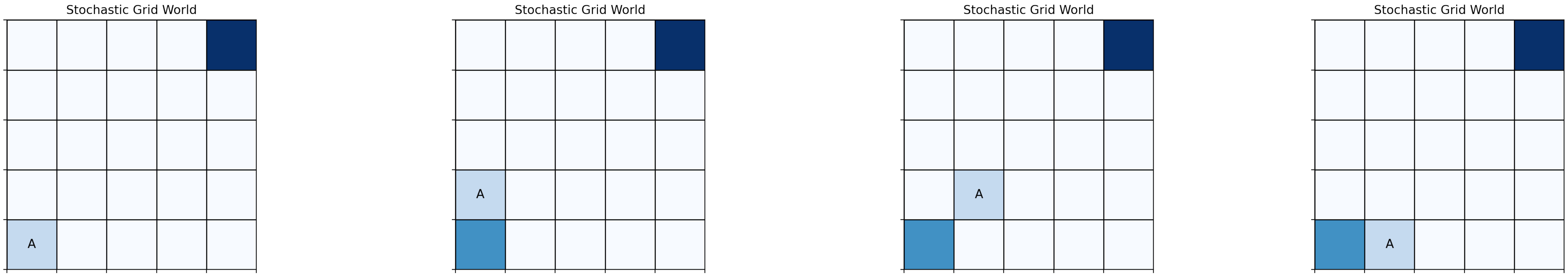
$Pr(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = Pr(s_{t+1} \mid s_t, a_t) \rightarrow$ stochastic dynamical system!

Example: Stochastic MDP (Gridworld)

EPFL

Time evolution

- Start in $s_0 \sim \rho$.
- At each time t:
    - Take action $a_t \in \mathscr{A}$.
    - End up in state $s_{t+1} \sim P(\,\cdot \mid s_t, a_t)$.



Objective

Reward and discount factor

- Reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
- Discount rate $\gamma \in (0,1)$.

Objective function

The goal is to find a policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ maximizing

$$\boxed{J(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi(\cdot \,|\, s_t), s_{t+1} \sim P(\cdot \,|\, s_t, a_t)\right]}$$

Reinforcement learning vs. optimal control

EPFL

Stochastic optimal control

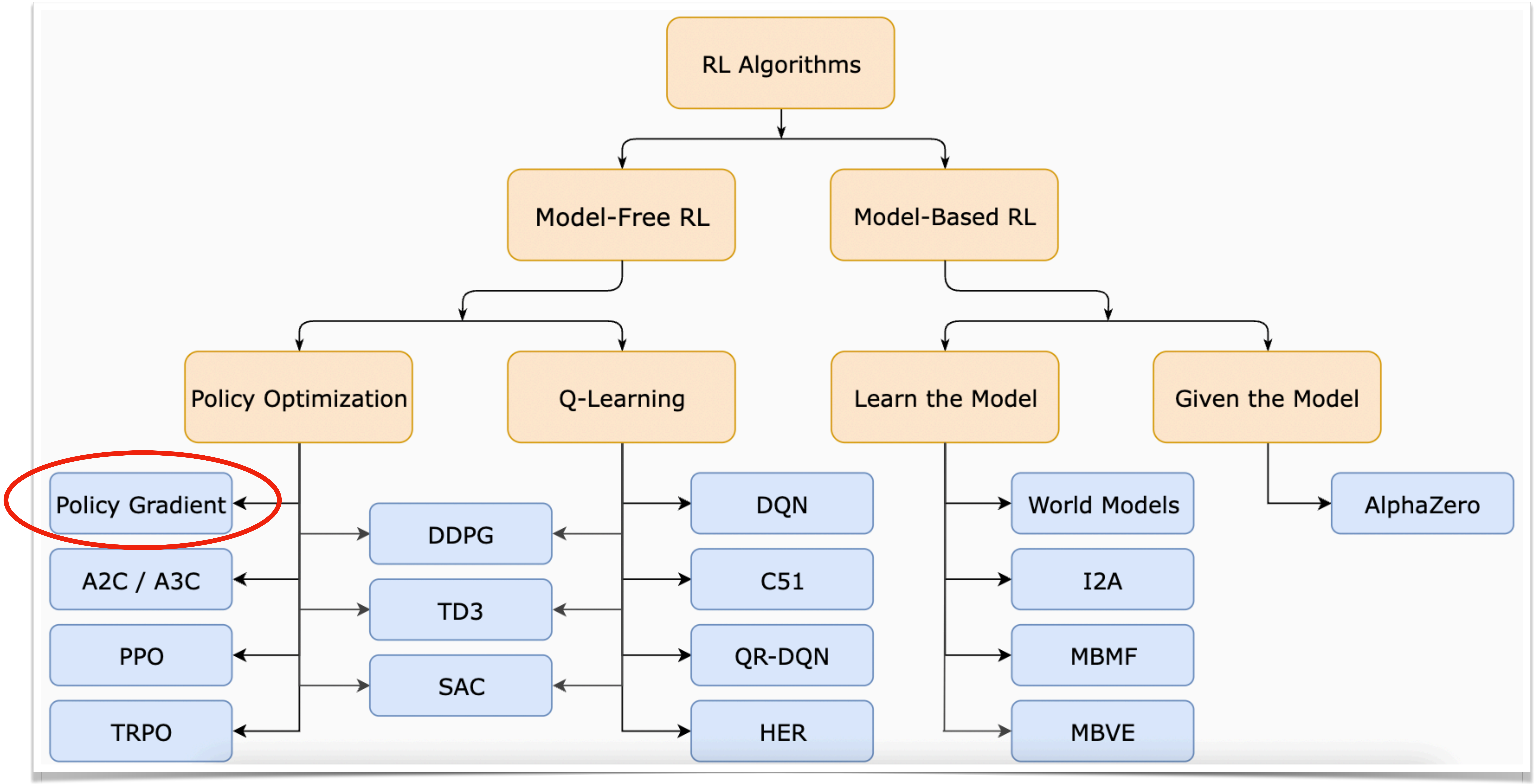
$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi(\,\cdot\,|\,s_t), s_{t+1} \sim P(\,\cdot\,|\,s_t, a_t)\right]$$

For **known transition dynamics** P, we can solve this via dynamic programming.

Reinforcement learning

- In RL we can only sample from the MDP, but do not assume to know P.
- We need to explore the environment.

Many different approaches



Taxonomy of reinforcement learning approaches [14].

Policy gradient method

$$\max_{\theta} J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\cdot \,|\, s_t)\right]$$

- Policy Optimization: Parameterize the policy as $\pi_\theta(a \,|\, s)$ and then find the best policy.

    - Direct parameterization

      $\pi_\theta(a \,|\, s) = \theta_{s,a}$, where $\theta \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$, $\theta_{s,a} \geq 0$ and $\sum_{a \in \mathcal{A}} \theta_{s,a} = 1$.

Policy Parameterization

- Softmax parameterization

$$\pi_\theta(a \mid s) = \frac{\exp\left(\theta_{s,a}\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\theta_{s,a'}\right)}, \text{ where } \theta \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$$

- Neural softmax parameterization

$$\pi_\theta(a \mid s) = \frac{\exp\left(f_\theta(s, a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(f_\theta(s, a')\right)}, \text{ where } f_\theta(s, a) \text{ represents a neural network.}$$

Policy gradient method

$$\max_{\theta} J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t)\right]$$

- Parameterize the policy as $\pi_\theta(a \,|\, s)$.
- Using gradient ascent method to find best policy π^*
- Pseudo-code for policy gradient method

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
    Calculate the gradient $\nabla_\theta J(\pi_\theta)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

Results for policy gradient method

EPFL

- Can we converge to the optimal policy when K is big enough?
  - Non-convexity may lead to sub-optimal policy (local minima)
- There are convergence guarantees for direct parametrization or softmax parametrization [15, 16]
  - Good

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
    Calculate the gradient $\nabla_\theta J(\pi_\theta)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

How to compute the gradient $\nabla_\theta J(\pi_\theta)$?

$$J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t)\right]$$

$$J(\pi_\theta) \approx J_H(\pi_\theta) \quad \Big\downarrow$$

$$J_H(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t)\right]$$

Computation of the gradient $\nabla_\theta J_H(\pi_\theta)$

- For every random trajectory $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$, the probability of choosing this trajectory as

$$p_\theta\left(\tau_H\right) := \rho\left(s_0\right)\prod_{t=0}^{H}\pi_\theta\left(a_t \mid s_t\right) P\left(s_{t+1} \mid s_t, a_t\right)$$

- And we set reward we get from this trajectory τ as

$$R\left(\tau_H\right) := \sum_{t=0}^{H}\gamma^t r(s_t, a_t)$$

- Then,

$$J_H(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{H}\gamma^t r(s_t, a_t) \mid s_0 = s, \pi\right] = \mathbb{E}_{\tau_H \sim p_\theta}\left[R(\tau_H)\right]$$

- Therefore,

$$\nabla_\theta J_H(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$$

Policy gradient theorem

Policy gradient theorem:

$$\nabla_\theta J_H(\pi_\theta) = \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

Proof: Because

$$p_\theta(\tau) = \rho(s_0) \prod_{t=0}^{H} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$$

$$R(\tau) = \sum_{t=0}^{H} r(s_t, a_t)\gamma^t$$

$$\nabla_x \log(f(x)) = \frac{\nabla_x f(x)}{f(x)}$$

$$\nabla_x f(x) = f(x) \nabla_x \log(f(x))$$

$$\begin{aligned}
\nabla_\theta J_H(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim p_\theta}[R(\tau_H)] \\
&= \nabla_\theta \sum_\tau R(\tau) p_\theta(\tau) \\
&= \sum_\tau R(\tau) \nabla_\theta p_\theta(\tau) \\
&= \sum_\tau R(\tau) p_\theta(\tau) \nabla_\theta \log(p_\theta(\tau)) \\
&= \mathbb{E}_{\tau \sim p_\theta}\left[R(\tau) \nabla_\theta \log(p_\theta(\tau))\right] \\
&= \mathbb{E}_{\tau \sim p_\theta}\left[R(\tau) \nabla_\theta \log(\rho(s_0) \prod_{t=0}^{H} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t))\right] \\
&= \mathbb{E}_{\tau \sim p_\theta}\left[R(\tau) \nabla_\theta \left(\sum_{t=0}^{H} \log(\pi_\theta(a_t \mid s_t)) + \log(\rho(s_0)) + \sum_{t=0}^{H} \log(P(s_{t+1} \mid s_t, a_t))\right)\right] \\
&= \mathbb{E}_{\tau \sim p_\theta}\left[R(\tau) \times \sum_{t=0}^{H} \nabla_\theta \log(\pi_\theta(a_t \mid s_t))\right]
\end{aligned}$$

How to estimate the gradient?

$$\nabla_\theta J_H(\pi_\theta) = \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

- Using Monte-Carlo method
  - Consider a random variable $X \sim q$.
  - Given independent and identically distributed $X_1, \ldots, X_N \sim q$, we can estimate

$$\mathbb{E}[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i)\,.$$

- We don't know P, but we can approximate $\nabla_\theta J_H(\pi_\theta)$ using the Monte-Carlo method.

Monte Carlo method for gradient estimation

EPFL

- We sample N trajectories by interacting with the environment

$$\left(s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_H^i, a_H^i\right), \quad i = \{1, \ldots, N\}$$

- Using the above samples, we estimate $\nabla_\theta J(\pi_\theta)$ as

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &\approx \nabla_\theta J_H(\pi_\theta) \\
&= \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H} \gamma^t r(s_t^i, a_t^i)\right)\left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)
\end{aligned}$$

Pros and cons of reinforcement learning

EPFL

Pros

- General methods for complex tasks
- Adapt to changing environments
- Model free: no need to know dynamic model

Cons

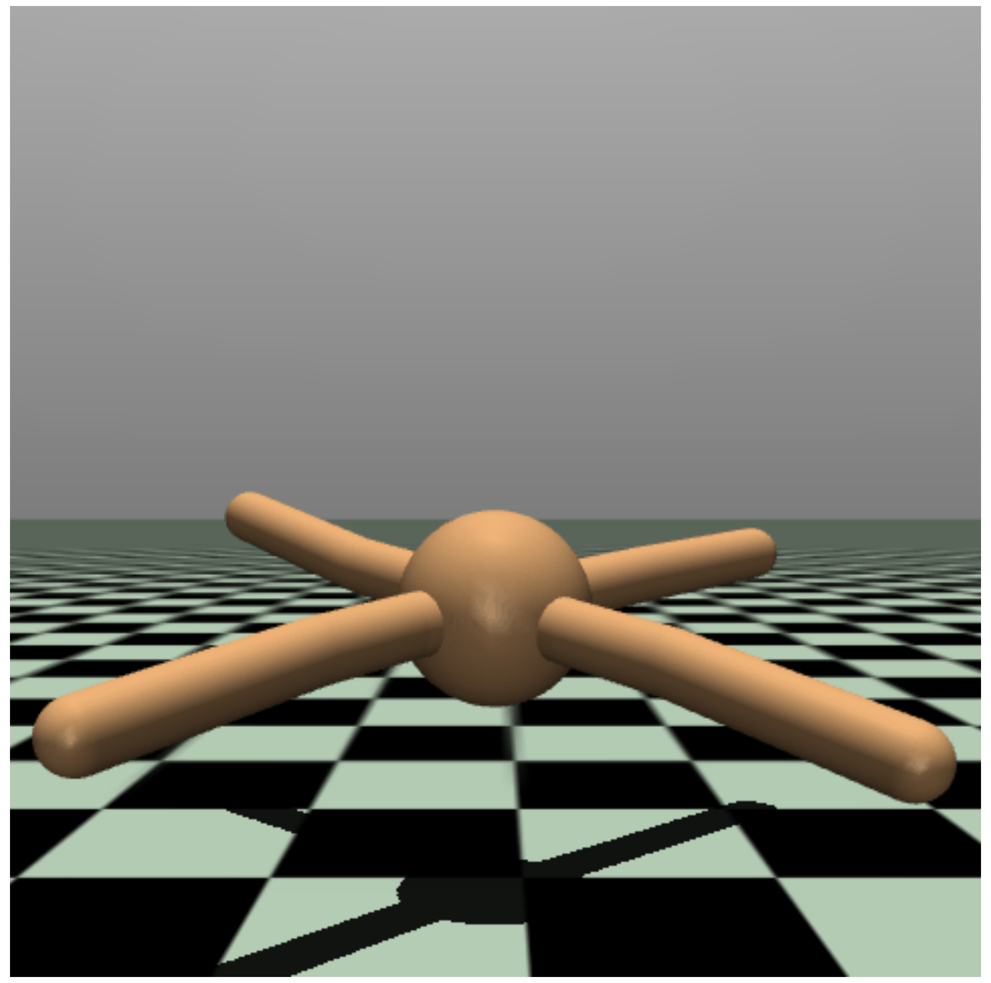
- Sample inefficiency for model free approach
- Lack of safety and convergence guarantees
- Hard to assign meaningful rewards

Reinforcement learning projects in Sycamore lab

EPFL

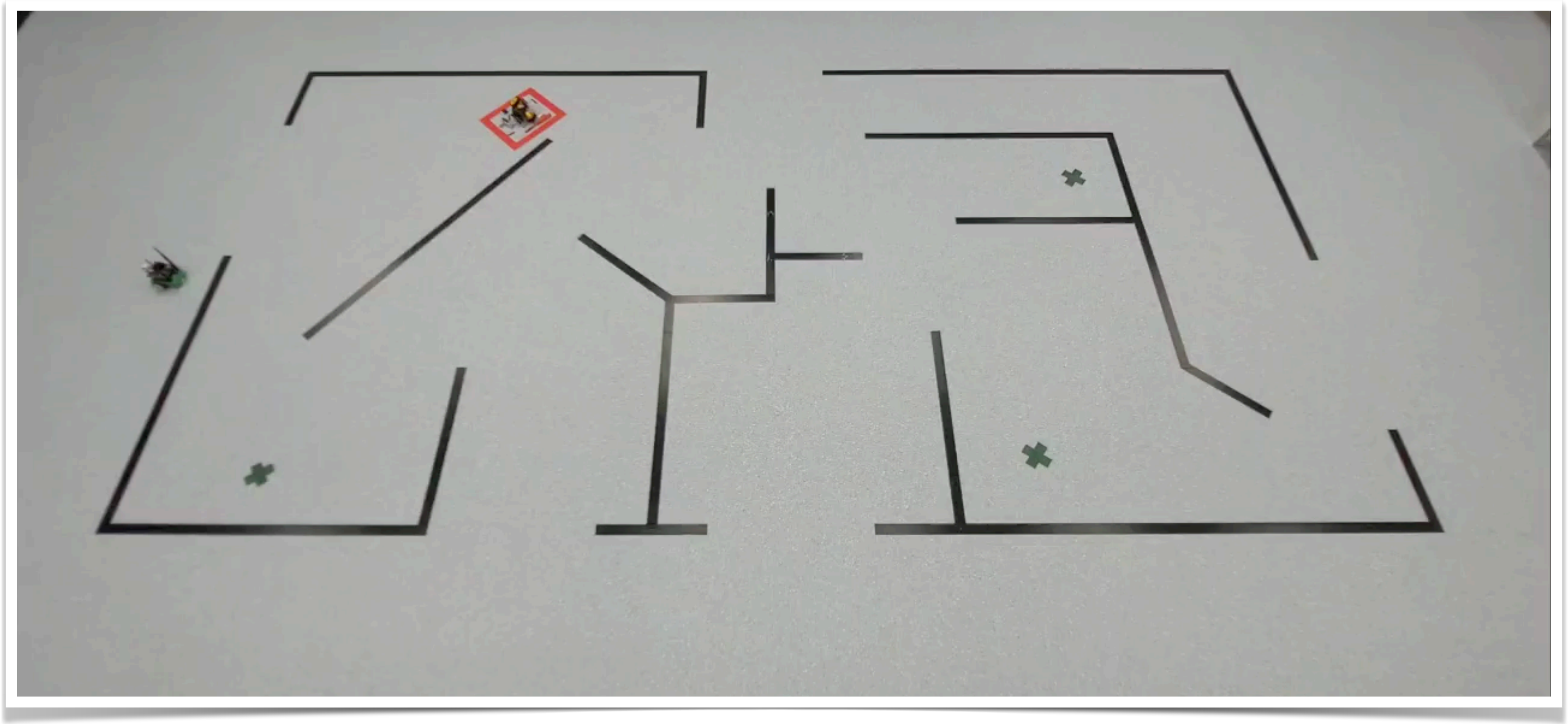
Bachelor projects

- Policy optimization for robotics



Semester and master projects

- Safe reinforcement learning
- Inverse reinforcement learning



References

- [1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
- [2] Baker, Bowen, et al. "Emergent tool use from multi-agent autocurricula." *arXiv preprint arXiv:1909.07528* (2019).
- [3] DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. https://www.deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40. Accessed 15 Dec. 2022.
- [4] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489
- [5] Silver, David, et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." *arXiv preprint arXiv:1712.01815* (2017).
- [6] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." arXiv preprint arXiv:1912.06680 (2019).
- [7] Arulkumaran, Kai, Antoine Cully, and Julian Togelius. "Alphastar: An evolutionary computation perspective." Proceedings of the genetic and evolutionary computation conference companion. 2019.
- [8] Akkaya, Ilge, et al. "Solving rubik's cube with a robot hand." arXiv preprint arXiv:1910.07113 (2019).
- [9] Fawzi, Alhussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning." Nature 610.7930 (2022): 47-53.
- [10] “ChatGPT: Optimizing Language Models for Dialogue.” OpenAI, 30 Nov. 2022, https://openai.com/blog/chatgpt/.
- [11] Miki, Takahiro, et al. "Learning robust perceptive locomotion for quadrupedal robots in the wild." *Science Robotics* 7.62 (2022): eabk2822.
- [12] Ibrahim, Muhammad Sohail, Wei Dong, and Qiang Yang. "Machine learning driven smart electric power systems: Current trends and new perspectives." *Applied Energy* 272 (2020): 115237.
- [13] Niao He, Lecture notes on “Introduction to Reinforcement Learning”, ETH Zurich, 2021, https://odi.inf.ethz.ch/files/zinal/Lecture-1-RL-introduction.pdf
- [14] Open AI, “Taxonomy of RL Algorithms”, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below
- [15] Agarwal, Alekh, et al. "Optimality and approximation with policy gradient methods in markov decision processes." *Conference on Learning Theory*. PMLR, 2020.
- [16] Mei, Jincheng, et al. "On the global convergence rates of softmax policy gradient methods." *International Conference on Machine Learning*. PMLR, 2020.