Robotic reinforcement learning

Arthur Louette (arthur.louette@uliege.be)

April 14, 2026

Outline

Introduction

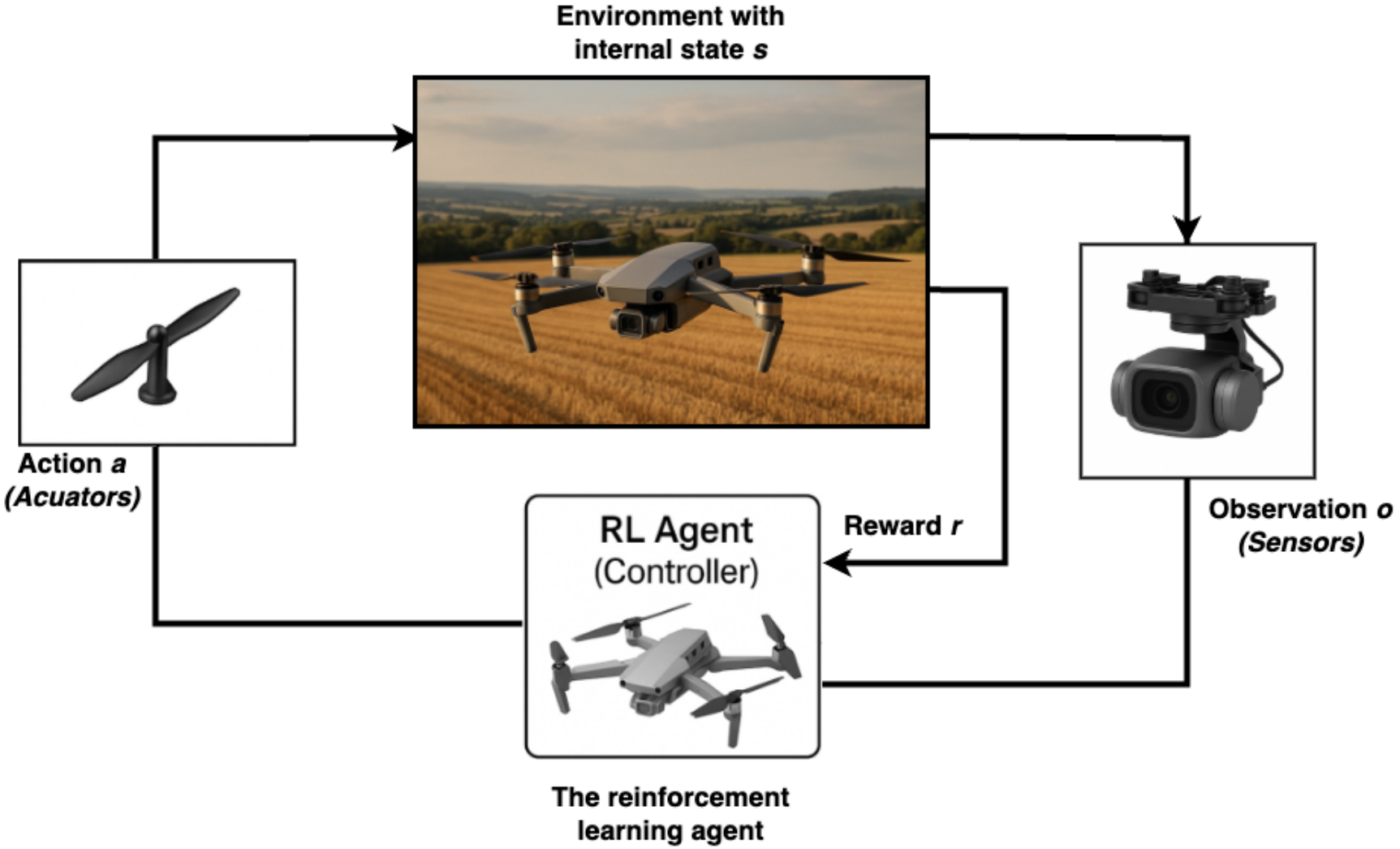
What is a robotic agent?

A robot is a goal oriented machine that can sense, plan, and act [Corke, 2017].

This lecture focuses on reinforcement learning to control robots.

A robotic agent is an autonomous agent that perceives its environment via sensors, makes decisions using a learned controller, and executes actions through actuators to achieve specific goals. It learns and adapts its behavior over time through interaction with the environment.

Robotic agent interaction loop



Why reinforcement learning for robotics?

The classical approach is optimal control: given a model $\dot{x} = f(x, u)$ and a cost, compute $u^*(\cdot)$. It gives us LQR, iLQR, MPC, ...

Where optimal control is limited for robotics:

- Long-horizon planning is expensive online.
- Contacts and discontinuities are hard to model and differentiate through.
- Pixel-level sensing has no closed-form cost.
- Every new robot requires re-modeling and re-tuning.

Reinforcement learning instead learns a controller from interaction, without requiring a model. Planning cost is amortized into a fast policy $\eta(a|h)$ evaluated at control rate.

POMDP

A robotic agent only perceives partial information about its state through sensors.

This lecture formalizes the problems with a single robotic agent that has to solve one task in the real world as POMDPs.

A POMDP is represented by its model $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, P, \gamma)$:

- States $s_t \in \mathcal{S}$,
- Actions $a_t \in \mathcal{A}$,
- Observations $o_t \in \mathcal{O}$,
- Transition distribution $T(s_{t+1}|s_t, a_t)$,
- Reward function $r_t = R(s_t, a_t)$,
- Observation distribution $O(o_t|s_t)$,
- Initial distribution $P(s_0)$,
- Discount factor $\gamma \in [0, 1[$.

Challenges for solving POMDPs in the real world

- Safety: exploration could break the robot.

Challenges for solving POMDPs in the real world

- Safety: exploration could break the robot.
- Sample efficiency: real-world interaction is bounded by real time.

Challenges for solving POMDPs in the real world

- Safety: exploration could break the robot.
- Sample efficiency: real-world interaction is bounded by real time.
- Real-time decision: sensing, inference and control must fit within the loop deadline.

Challenges for solving POMDPs in the real world

- Safety: exploration could break the robot.
- Sample efficiency: real-world interaction is bounded by real time.
- Real-time decision: sensing, inference and control must fit within the loop deadline.
- Diversity of deployed conditions: lighting, surfaces, object layout, all vary unpredictably.

Challenges for solving POMDPs in the real world

- Safety: exploration could break the robot.
- Sample efficiency: real-world interaction is bounded by real time.
- Real-time decision: sensing, inference and control must fit within the loop deadline.
- Diversity of deployed conditions: lighting, surfaces, object layout, all vary unpredictably.
- Sensor noise and uncertainty: real sensors are noisy, biased and sometimes drop data.

Scope of this lecture

We consider a single robotic agent solving one task. Other entities (humans, other robots) are treated as stationary parts of the environment, so the POMDP formulation above is sufficient.

Three complementary ways to obtain a policy:

1. Online RL directly in the real world,
2. Learning in simulation and transferring to reality,
3. Offline RL: Learning from dataset.

We start with the setting closest to the real world and progressively step back to larger-scale data sources.

Online RL in the real world

Motivation

Why learn directly on the robot?

- No reality gap, by construction.
- Needed when no faithful simulator exists: soft robots, deformables, ...

The price to pay:

- Real-world interaction is precious.
- Safety and reset must be handled carefully.
- Off-policy, sample-efficient, and stable algorithms must be considered.

Off-policy actor-critic algorithms

Actor-critic algorithms interleave between critic and actor updates.

Critic Update:

$$J_Q(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - y\right)^2\right]$$

$$y = R(s,a) + \gamma Q_{\hat{\phi}}(s', a'), \quad a' \sim \eta_\theta(\cdot|s')$$

ϕ' is an exponentially averaged target network.
$\mathcal{D}$ is an experience replay buffer.

Actor Update:

$$J_\eta(\theta) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\eta_\theta(\cdot|s)}\left[Q_\phi(s,a)\right]$$

Soft Actor-Critic

SAC [Haarnoja et al., 2019] augments the standard RL objective with an entropy bonus:

$$\eta^* = \arg\max_{\eta} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t) + \alpha\, \mathcal{H}\big(\eta(\cdot|s_t)\big) \Big)\right]$$

where $\mathcal{H}\big(\eta(\cdot|s)\big) = -\mathbb{E}_{a\sim\eta}[\log \eta(a|s)]$ is the entropy of the policy at state s.

The temperature $\alpha > 0$ controls the trade-off between reward maximization and entropy. Higher α means more exploration.

SAC losses

Critic loss (ϕ_1, ϕ_2): Bellman error with entropy-augmented target.

$$\mathcal{L}_Q(\phi_i) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})} \left[\frac{1}{2} \left(Q_{\phi_i}(s_t, a_t) - y_t \right)^2 \right]$$

$$y_t = r_t + \gamma \left(\min_j Q_{\hat{\phi}_j}(s_{t+1}, a_{t+1}) - \alpha \log \eta_\theta(a_{t+1}|s_{t+1}) \right), \quad a_{t+1} \sim \eta_\theta(\cdot|s_{t+1})$$

Policy loss (θ): maximize Q while keeping entropy high.

$$\mathcal{L}_\eta(\theta) = \mathbb{E}_{s_t, \varepsilon_t} \left[\alpha \log \eta_\theta(a_t|s_t) - \min_j Q_{\phi_j}(s_t, a_t) \right]$$

Actions are sampled via the reparameterization trick: $a_t = f_\theta(\varepsilon_t; s_t)$, $\varepsilon_t \sim \mathcal{N}(0, I)$.

Temperature loss (α): auto-tune α to maintain a target entropy $\bar{\mathcal{H}}$.

$$\mathcal{L}(\alpha) = \mathbb{E}_{a_t \sim \eta_\theta} \left[-\alpha \log \eta_\theta(a_t|s_t) - \alpha \bar{\mathcal{H}} \right]$$

SAC algorithm

Algorithm 0: Soft Actor-Critic [Haarnoja et al., 2019]

Initialize critics Q_{ϕ_1}, Q_{ϕ_2}, policy η_θ, targets $\hat{\phi}_i \leftarrow \phi_i$, buffer $\mathcal{D} \leftarrow \emptyset$
for *each iteration* **do**
  for *each environment step* **do**
    Sample $a_t \sim \eta_\theta(\cdot \mid s_t)$, step $s_{t+1} \sim T(\cdot \mid s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  for *each gradient step* **do**
    Update critics: $\phi_i \leftarrow \phi_i - \lambda_Q \nabla_{\phi_i} \mathcal{L}_Q(\phi_i)$ for $i \in \{1, 2\}$
    Update policy: $\theta \leftarrow \theta - \lambda_\eta \nabla_\theta \mathcal{L}_\eta(\theta)$
    Update temperature: $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \mathcal{L}(\alpha)$
    Update targets: $\hat{\phi}_i \leftarrow \tau\, \phi_i + (1 - \tau)\, \hat{\phi}_i$
Return η_θ

Update-to-data ratio

Off-policy algorithms decouple data collection from training. We can reuse buffer data to update the critic and actor multiple times per environment step.

Definition (Update-to-data ratio (UTD))

The UTD is the ratio between the number of gradient updates and the number of environment steps collected.

- Increasing the UTD improves sample efficiency.
- But re-using the same data too often leads the critic to overfit and Q-values to diverge.

Regularizing a high-UTD critic

To benefit from a high UTD, the critic must be regularized:

- Dropout: reduces variance.
- Ensembling: explicit, take min or mean over N critics to fight overestimation.
- Layer normalization: stabilizes training and breaks gradient spikes.

Two canonical high-UTD SAC variants

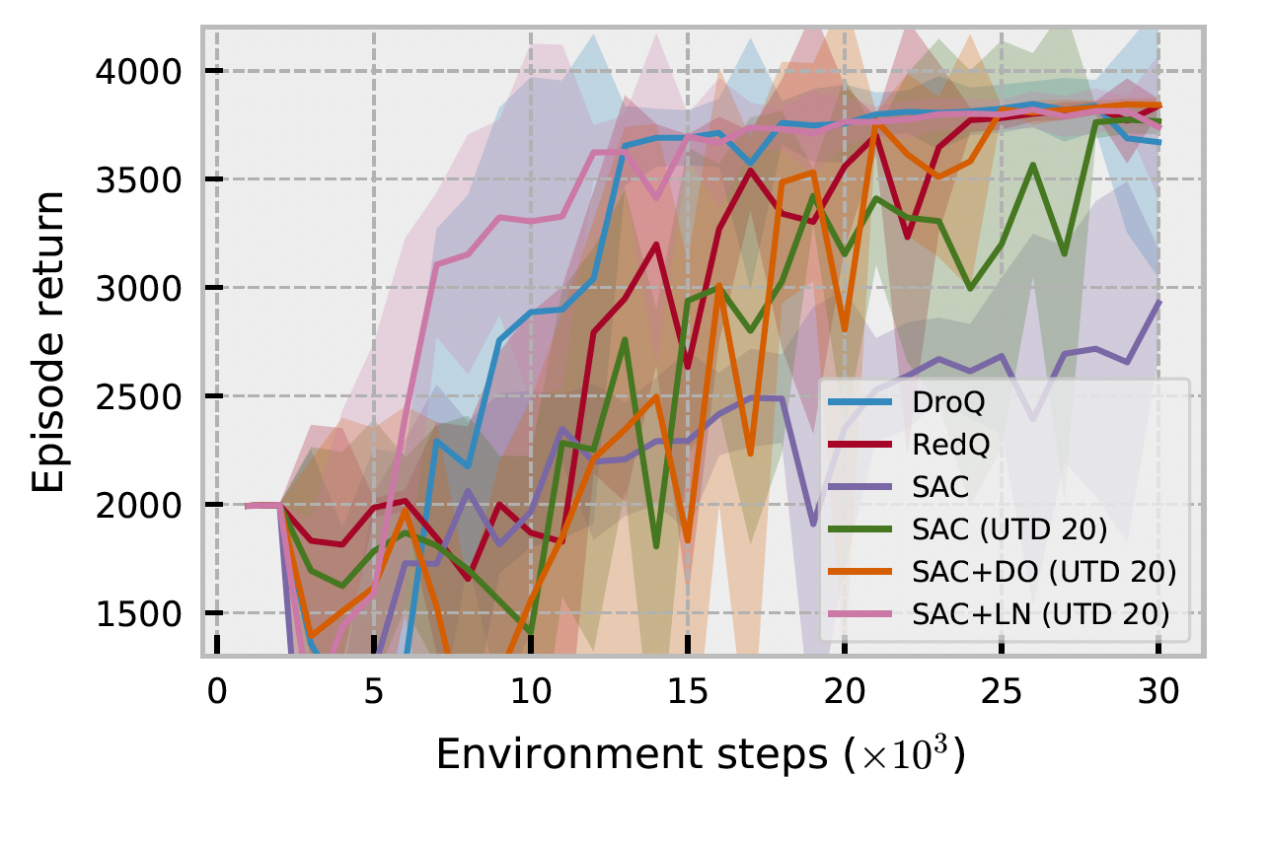
- REDQ [Chen et al., 2021]: large critic ensemble. TD target uses the min over a random subset.
- DroQ [Hiraoka et al., 2022]: dropout + layer norm inside each critic.

Locomotion task in real world

https://www.youtube.com/watch?v=YO1USfn6sHY

Smith, Kostrikov, Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 2023

Regularization and higher update-to-data ratio



Regularization and a higher UTD together unlock real-world sample efficiency: a quadruped learns to walk in 20 minutes, from scratch, with no simulator.

Smith, Kostrikov, Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 2023

Beyond DroQ: recent progress (2024–2026)

- BRO [Nauman et al., 2024] (*Bigger, Regularized, Optimistic*): a recipe to scale the critic without losing stability.
- SimBa / SimBaV2 [Lee et al., 2024]: Scales to wider and deeper networks for RL.
- CrossQ, TD7, MR.Q: batch normalization, representation learning, and state-action embeddings to improve sample efficiency.

Take-away: a modern real-world SAC is DroQ + a SimBa-style critic + high UTD. These improvements stack.

Smooth exploration for real hardware

Real motors do not survive bang-bang exploration. A policy that is optimal in simulation may burn the hardware in the real world.

Practical regularization:

- Penalize consecutive action differences: $\mathcal{L}_{\text{smooth}} = \lambda\, \mathbb{E}[\|\mu_\theta(s_t) - \mu_\theta(s_{t-1})\|^2]$ [Raffin et al., 2021].
- Rate-limit and clip actions. Model communication and sensor delays during training.
- Hard-coded safety envelopes (torque caps, virtual walls, dead-man switches).

Limitations of real-world learning

Learning in the real world is impressive but remains limited:

- Safety: for some tasks, safety and reset are difficult.
- Real-time decision: some tasks blow up the inference and learning time.
- Sample efficiency: training from scratch is still a challenge for many tasks.

$\Rightarrow$ When interaction on the real robot is too slow, too dangerous or too expensive, we move to simulation.

Learning in simulation

Motivations

Learning in simulation has several advantages:

- Safety: no risk of damaging the robot.
- Massive data acquisition: parallelized simulators yield thousands of rollouts per second.
- Realism: modern engines model lighting, textures, contact, fluids.
- Easy reset: no manual intervention required.

Robotic simulator

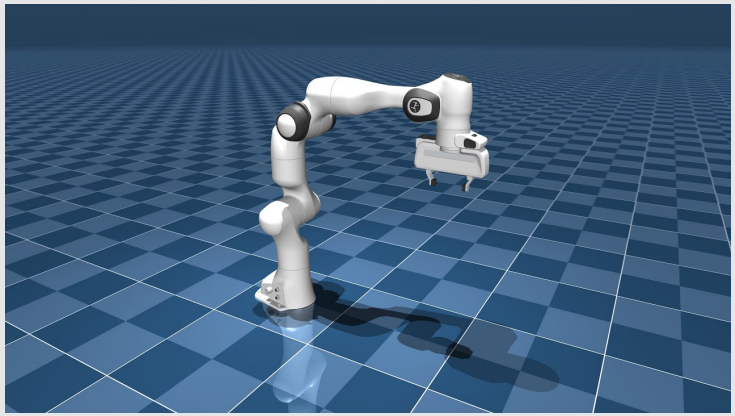
A physics simulator replicates the behavior of real-world physics within a virtual environment.

A robotic simulator builds on a physics engine and adds everything needed for robot learning: collision detection, friction models, rendering, scene and mesh import, API support, and a library of joints, actuators, and sensors.

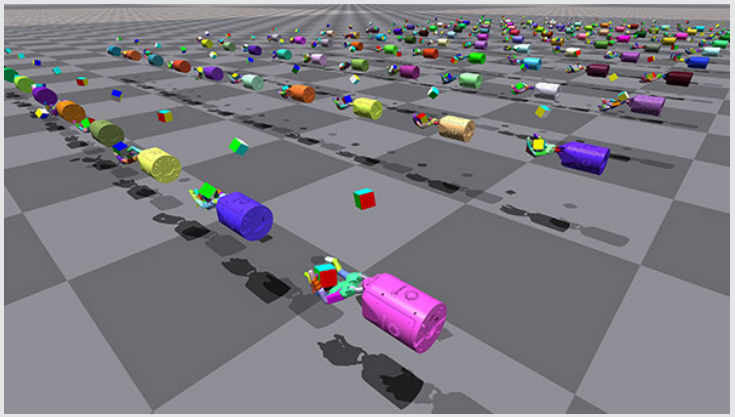
It enables learning in simulation and subsequent transfer of learned policies to reality.

Some robotic simulators

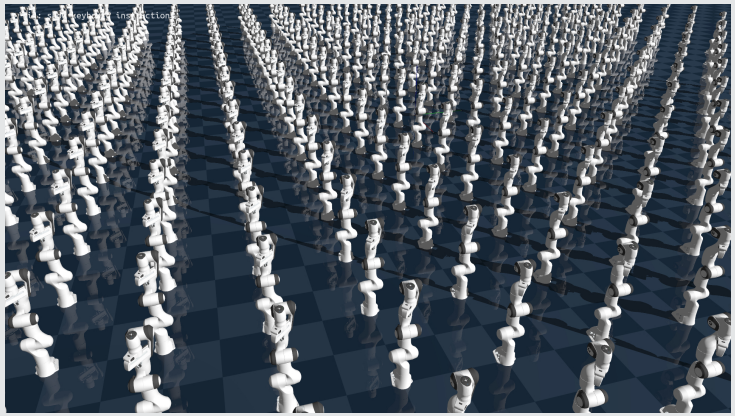
MuJoCo / MJX



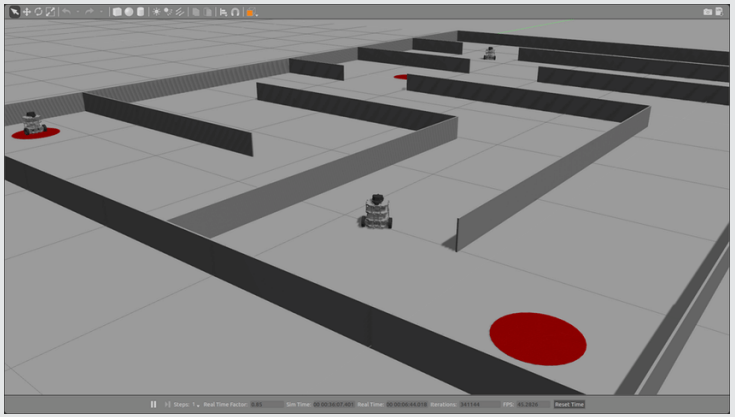
Isaac Lab (NVIDIA)



Genesis



Gazebo



Among others: CoppeliaSim, CARLA, SOFA, Webots, Brax, Warp...

Formulation: simulated and real POMDPs

Following Aljalbout et al. [2026], we model simulation and reality as two POMDPs over the same spaces $(\mathcal{S}, \mathcal{A}, \mathcal{O})$.

Definition (Context variables)

Context variables $\psi \in \Psi$ are environment parameters (masses, friction coefficients, drag, motor delays, textures, ...) that parametrize the transition and observation functions but are not part of the agent's state. They are fixed for a given episode.

Definition (Simulated and real POMDPs)

A simulated POMDP with context ψ: $\mathcal{P}_s(\psi) = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T_\psi, R_\psi, O_\psi, P_\psi, \gamma)$.
The real-world POMDP: $\mathcal{P}_r = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T_r, R_r, O_r, P_r, \gamma)$.

T_r and O_r are unknown and not necessarily in the simulator family: no value of ψ may perfectly reproduce reality (unmodelled effects, numerical errors). The real world also has its own variability across episodes (weather, wear, lighting) baked into T_r and P_r.

Return of a policy in a POMDP

Let $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t) \in \mathcal{H}$ denote the history at time step t.

Definition (Expected return)

The expected return of a policy $\eta : \mathcal{H} \to \mathcal{A}$ in a POMDP $\mathcal{P}$ is

$$J_{\mathcal{P}}(\eta) = \mathop{\mathbb{E}}_{\substack{s_0 \sim P(\cdot) \\ o_t \sim O(\cdot|s_t) \\ a_t \sim \eta(\cdot|h_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

We write $J_{\mathcal{P}_s}(\eta)$ and $J_{\mathcal{P}_r}(\eta)$ for the returns in the simulated and real POMDPs.

Reality gap and performance gap

Following Aljalbout et al. [2026], we distinguish two notions of gap.

Definition (Reality gap)

The reality gap of a simulator $\mathcal{P}_s(\psi)$ with respect to reality $\mathcal{P}_r$:

$$G_{\text{dyn}}(\psi) = \mathbb{E}_{(s,a)}\left[D(T_\psi(\cdot \mid s,a) \parallel T_r(\cdot \mid s,a))\right]$$

$$G_{\text{perc}}(\psi) = \mathbb{E}_{s}\left[D(O_\psi(\cdot \mid s) \parallel O_r(\cdot \mid s))\right]$$

Definition (Performance gap)

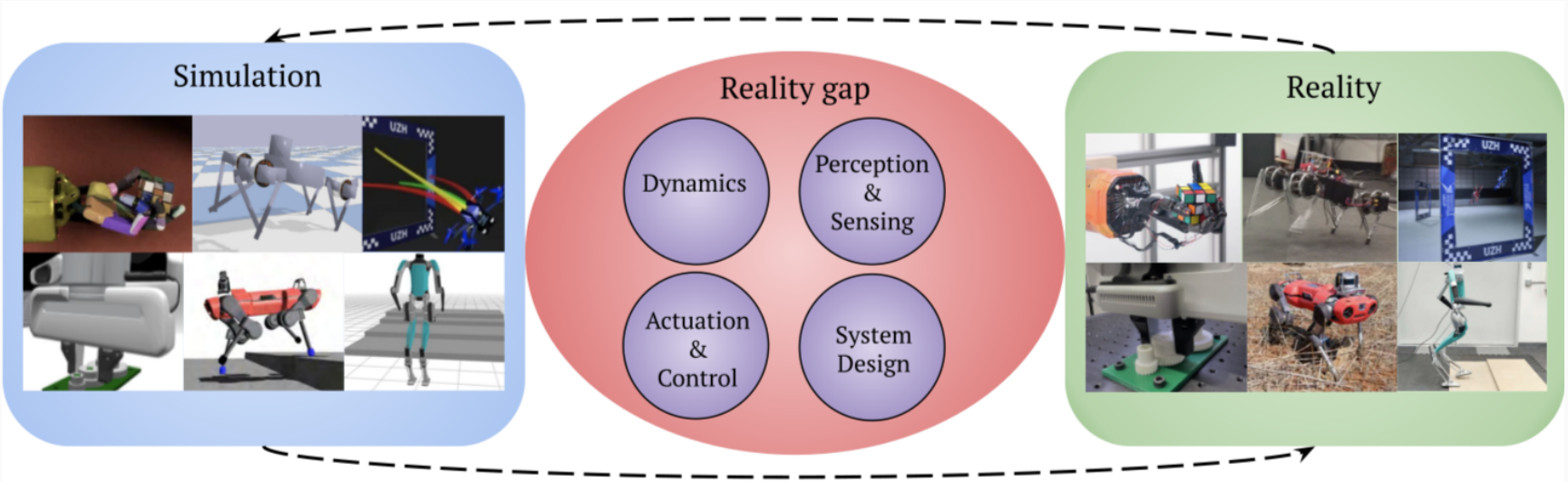
The performance gap for a policy η trained under distribution $p(\psi)$:

$$G_{\text{perf}}(p,\eta) = \left|\mathbb{E}_{\psi\sim p}\left[J_{\mathcal{P}_s(\psi)}(\eta)\right] - J_{\mathcal{P}_r}(\eta)\right|$$

The goal of sim-to-real transfer is to minimize G_{perf}.

A large reality gap is acceptable if the performance gap is small.

Sources of the reality gap



The four families of reality gap sources. Figure from Aljalbout et al. [2026].

Sources of the reality gap: dynamics

Dynamics gap ($G_{\mathbf{Dyn}}$):

- Modeling: rigid-body assumptions break for deformable objects, compliant joints, and chaotic phenomena (e.g. turbulence).
- Parameterization: friction, aerodynamics, mass, inertia are hard to measure precisely and may change over time.
- Contact dynamics: friction varies with velocity, contact states alternate between sticking and slipping. Simplified contact models cause non-physical artifacts.
- Numerical integration: discretization (Euler, RK4) introduces errors that accumulate over long horizons.
- Unmodelled effects: battery voltage drop, thermal drift, wear and fatigue are rarely simulated.

Following Aljalbout et al. [2026], Section 3.1.

Sources of the reality gap: perception, actuation and system design

Perception gap (G_{perc}):

- Sensor models omit lens flares, rolling shutter, depth artifacts, LiDAR beam divergence, and non-Gaussian noise.
- Rendered images lack photorealism. Environment assets simplify geometry and materials.

Actuation gap:

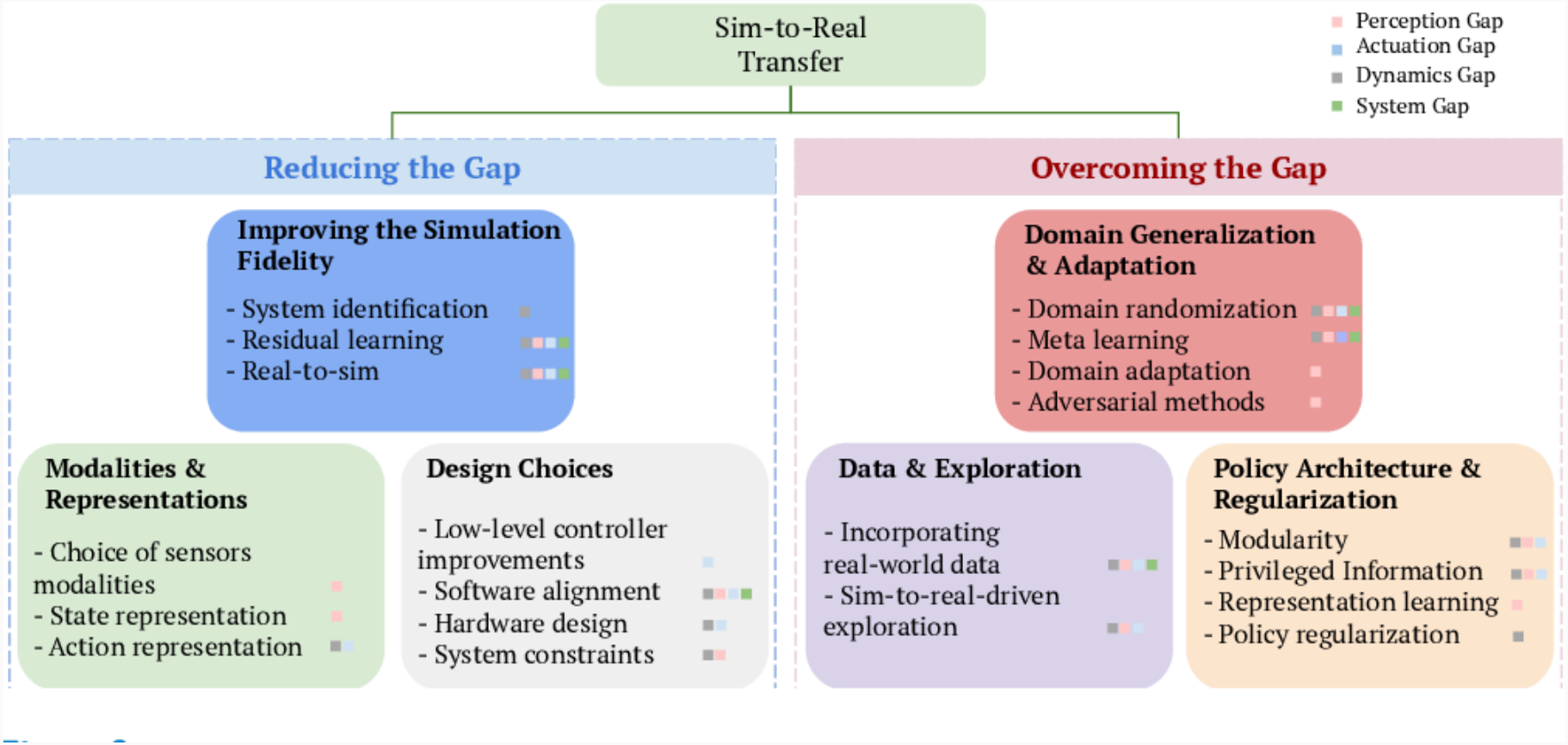
- Real motors have dead-zones, backlash, slew-rate limits, time constants, and hysteresis.
- Low-level controllers (hidden PID, anti-windup filters) are vendor-specific and not replicated.
- Power electronics introduce PWM quantization and current/voltage caps.

System design gap:

- Communication delays, packet loss, and clock drift between modules.
- Safety mechanisms and execution order differ between sim and real.

Following Aljalbout et al. [2026], Sections 3.2–3.4.

Reducing and overcoming the reality gap



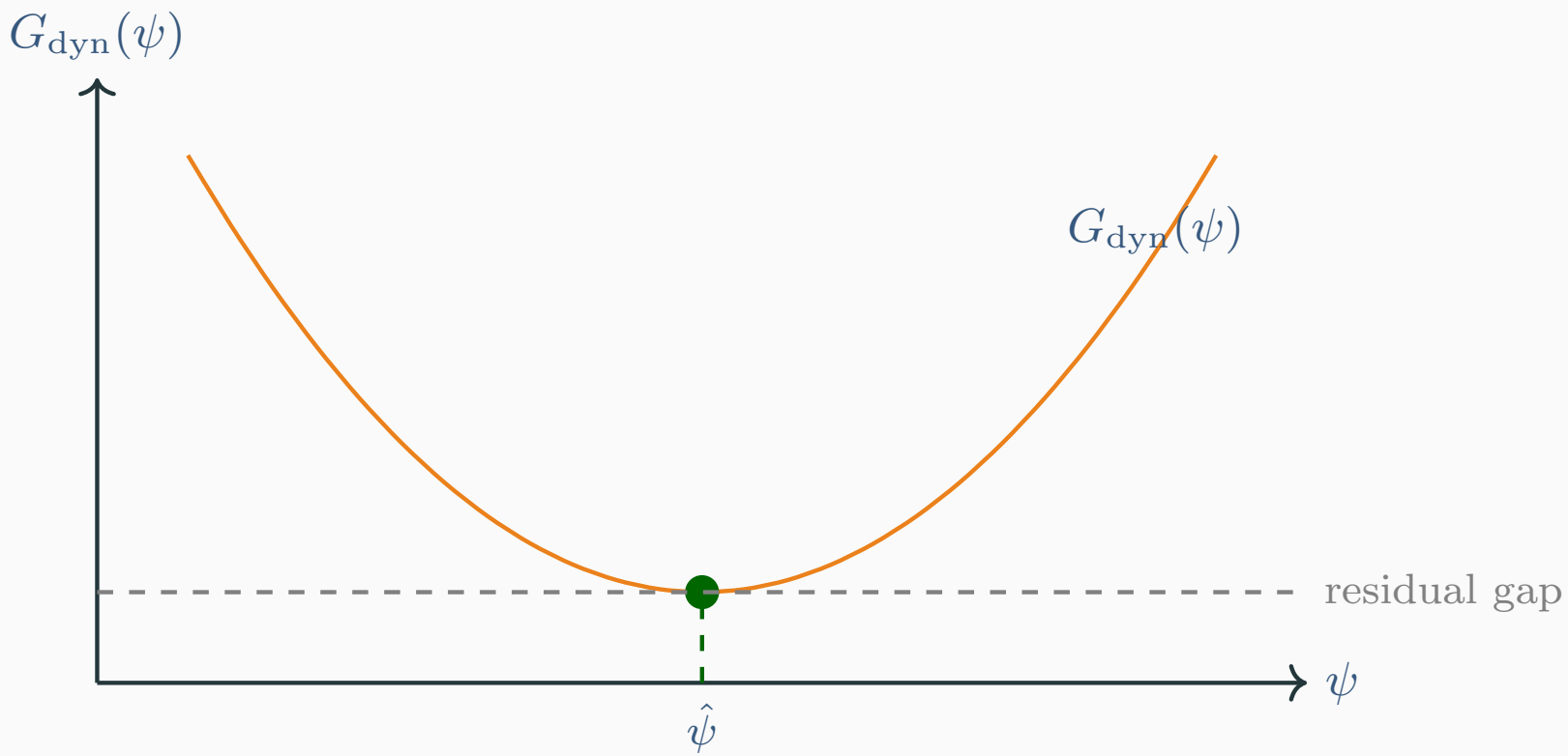
A taxonomy of sim-to-real transfer methods, distinguishing clearly between approaches and techniques that are designed to reduce the reality gap and those designed to overcome it. The colored squares indicate which reality gap each method addresses. From Aljalbout et al. [2026].

System identification

System identification finds $\hat{\psi}$ that makes $\mathcal{P}_s(\hat{\psi})$ as close as possible to $\mathcal{P}_r$.

System ID targets the dynamics gap G_{dyn}: fit physical parameters from real trajectories.

1. Collect real-world trajectories and solve $\hat{\psi} = \arg\min_{\psi} G_{\text{dyn}}(\psi)$.
2. Train the policy in $\mathcal{P}_s(\hat{\psi})$.



Even at the best $\hat{\psi}$, a residual gap remains (structural mismatch).

System identification: limitations

Problems:

- A single $\hat{\psi}$ works for one operating condition. Real conditions change across episodes (lighting, wear, surface properties), so the best ψ shifts over time.
- Some quantities vary within an episode (battery drain, thermal drift, wind gusts). These are better modeled as unobserved state, not as episode-level context variables.
- The real world may contain effects outside the parametrization Ψ entirely (unmodelled contacts, sensor artifacts). The residual gap $G_{\text{dyn}}(\hat{\psi}) > 0$.

$\Rightarrow$ Randomize $\psi \sim p(\psi)$ over a plausible range so the policy is robust to real-world variability.

Domain randomization

Definition (Domain Randomization)

At the start of each training episode, sample $\psi \sim p(\psi)$ and roll out in $\mathcal{P}_s(\psi)$. The training distribution $p(\psi)$ is chosen so the resulting policy performs well in $\mathcal{P}_r$:

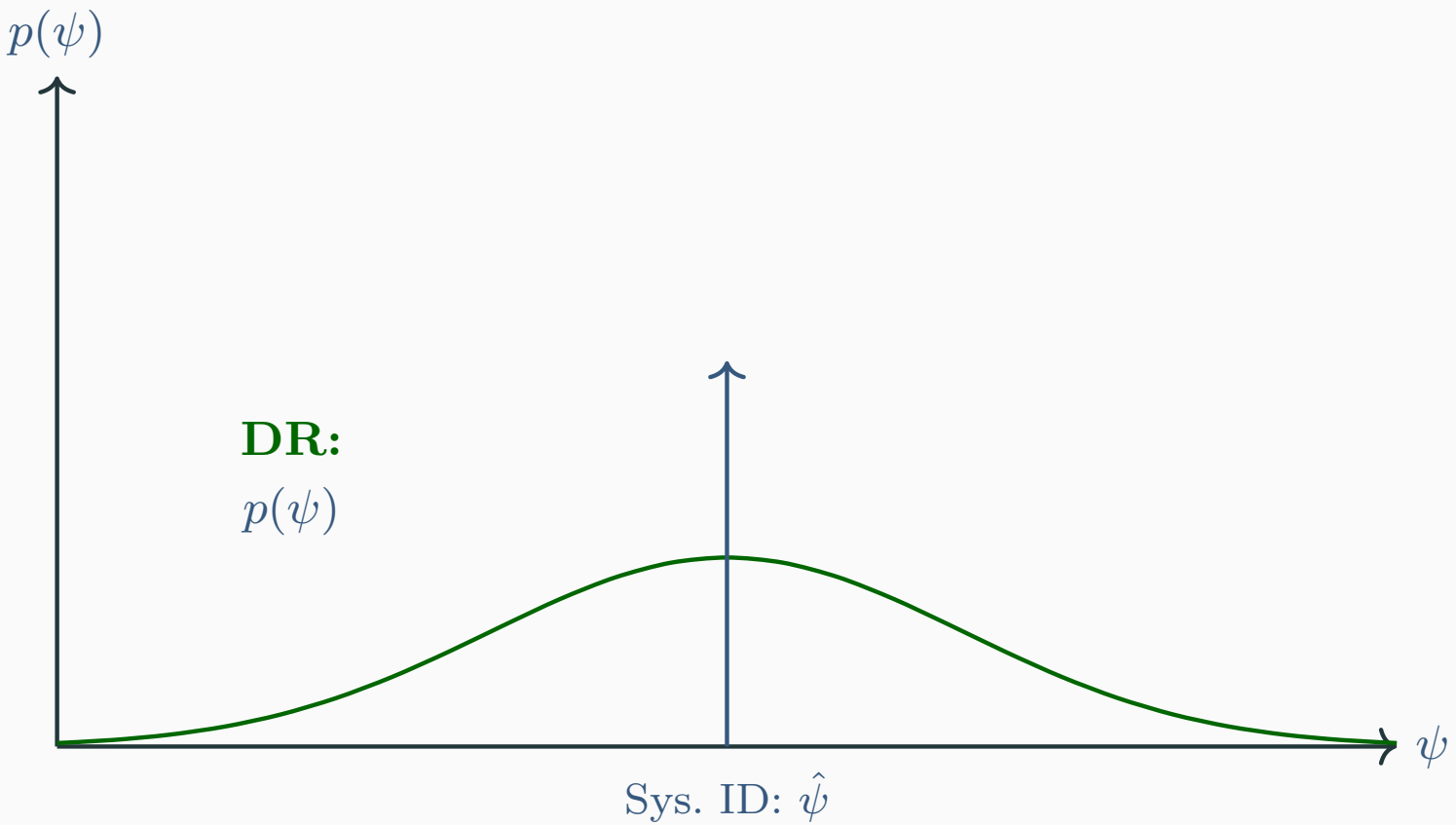
$$p^* = \arg\min_p G_{\text{perf}}\left(p,\, \eta_p^*\right), \quad \eta_p^* = \arg\max_\eta \mathbb{E}_{\psi\sim p}\left[J_{\mathcal{P}_s(\psi)}(\eta)\right]$$

$p(\psi)$ is a training distribution we control. We hope that training over a broad enough range of ψ produces a policy robust enough to handle $\mathcal{P}_r$, even though $\mathcal{P}_r$ may not correspond to any single ψ.

In practice, $p(\psi)$ is uniform or Gaussian over hand-tuned ranges. Automated schemes (ADR) widen the range during training.

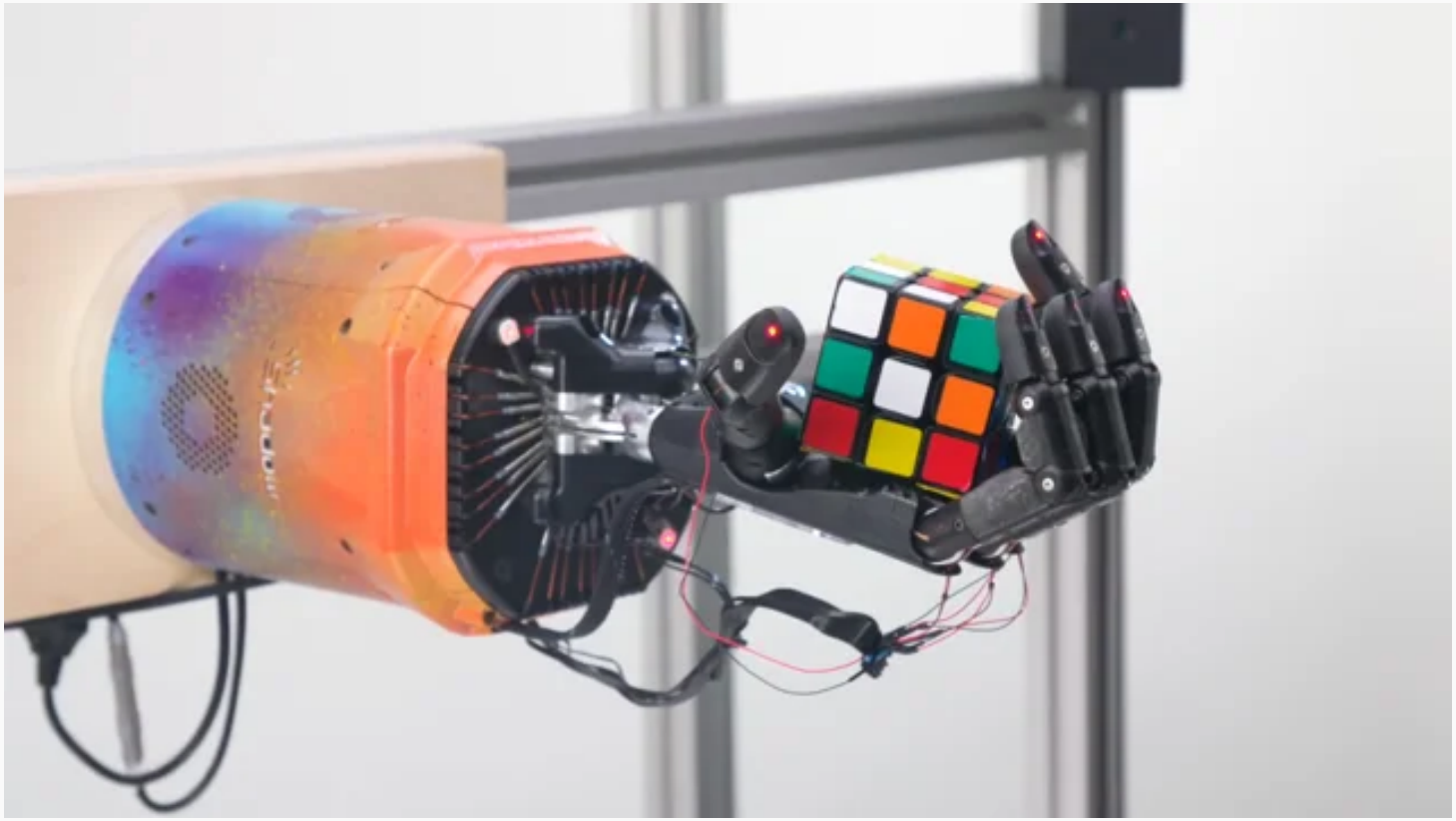
Domain randomization: intuition

System identification overfits to one setting: train at $\hat{\psi}$, hope it matches $\mathcal{P}_r$.
Domain randomization takes the opposite approach: train over a broad $p(\psi)$ so the policy becomes invariant to what it cannot control.



Sys. ID commits to a point. DR trains over a broad range: the policy's robustness to variation is what transfers to reality.

Domain randomization: example



https://openai.com/research/solving-rubiks-cube

Exploiting privileged information: informed POMDPs

In simulation we know much more than the policy will see at deployment: object poses, velocities, contacts, ψ itself, ...

Definition (Informed POMDP [Lambrechts et al., 2024])

An informed POMDP augments a POMDP with a training information signal i_t such that o_t is conditionally independent of s_t given i_t:

$$\widetilde{\mathcal{P}} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{O}, T, R, \widetilde{I}, \widetilde{O}, P, \gamma), \qquad s_t \to i_t \to o_t.$$

- At training time: both o_t and i_t are available.
- At execution time: only o_t.
- The policy $\eta(a|h)$ still reads o_t only, so it remains deployable.
- But the training objective can use i_t to supervise representations, critics, or world models.

Informed objective for learning statistics

Under mild conditions, a statistic $f(h)$ is sufficient for the optimal control if it is recurrent and predictive-sufficient for the reward and next information:

$$p(r, i' \mid h, a) \;=\; p(r, i' \mid f(h), a).$$

This suggests training f_θ jointly with a model q_θ by likelihood maximization:

$$\max_\theta \;\; \mathbb{E}_{p(h,a,r,i')} \left[\log q_\theta(r, i' \mid f_\theta(h), a) \right].$$

The data processing inequality gives $I(s'; i' \mid h, a) \geq I(s'; o' \mid h, a)$: information is more informative than observation.

$\Rightarrow$ richer supervision, faster convergence to a sufficient statistic.

Teacher-student: privileged information as a training-time tool

A widely used pattern that fits the informed POMDP view:

1. Teacher (in simulation, privileged): train $\eta_T(a \mid s_t, \psi)$ with standard RL. Fast, because it sees ψ.
2. Student (deployable): learn a recurrent encoder that infers a latent $\hat{z}_t$ of ψ from the history of observations and actions, and train $\eta_S(a \mid o_t, \hat{z}_t)$ to imitate the teacher.

Rapid Motor Adaptation (RMA) [Kumar et al., 2021] is the canonical example: the student performs online system identification from its own motion history.

PPO vs SAC in simulation

In simulation, we have unlimited cheap samples. This changes the algorithmic trade-off.

PPO (on-policy)

- Uses fresh rollouts each update.
- No replay buffer needed.
- Scales naturally to thousands of parallel envs.
- Stable, simple to tune.

SAC (off-policy)

- Reuses past data via replay buffer.
- Much higher sample efficiency.
- Harder to parallelize naively.
- Better for real-world learning where samples are precious.

Data collection and policy update

On-policy RL algorithms alternate between data collection and policy update.

- Data collection: policy inference, simulation, reward and observation.
- Policy update: gradient steps on the GPU.

Data collection and policy update

On-policy RL algorithms alternate between data collection and policy update.

- Data collection: policy inference, simulation, reward and observation.
- Policy update: gradient steps on the GPU.

In many current pipelines, simulation runs on the CPU while the network runs on the GPU.

This creates a PCIe communication bottleneck, up to 50× slower than GPU compute [Gregg and Hazelwood, 2011].

Overcoming bottlenecks with parallel simulation

CPU-based simulation has limited parallelism, constrained by core count and memory.

Move both simulation and policy updates to the GPU.

GPU/TPU-accelerated simulators:

- Isaac Sim / Isaac Lab (NVIDIA): general robotics.
- Brax, MJX, Genesis, Warp: written in JAX, for TPUs and GPUs.

JIT compilation matters too. JAX and `torch.compile` rewrite the whole update step into a single GPU kernel, giving roughly a 10× speedup on SAC and PPO [Raffin, 2025].

Case study: HOVER (humanoid versatile controller)

Pipeline [He et al., 2025]:

1. Oracle policy: PPO in IsaacGym for full-body motion imitation from human motion data.
2. Domain randomization over physical parameters.
3. Teacher-student distillation (DAgger): oracle sees full rigid-body state, student uses joint positions/velocities, base angular velocity, gravity vector, and action history.
4. Zero-shot transfer to Unitree H1.

Key idea: mode-specific masks let a single policy handle navigation, loco-manipulation, and teleoperation by selectively activating upper/lower body commands.

https://hover-versatile-humanoid.github.io/

Differentiable simulation

Classical simulators are black boxes: we can query $s_{t+1} = f(s_t, a_t)$ but cannot compute $\nabla_\psi f$.

Differentiable simulators allow backpropagation through the physics engine:

- Gradient-based system identification: directly minimize $\|T_s(\cdot \mid s, a, \psi) - T_r(\cdot \mid s, a)\|$ with respect to ψ.
- End-to-end trajectory optimization: differentiate the return J through the dynamics to update the policy.
- Faster convergence: orders of magnitude fewer samples than black-box RL for smooth tasks.

Key platforms:

- Newton (NVIDIA / DeepMind / Disney, 2025): 65% faster for dexterous manipulation, full differentiability, open-source.
- Genesis: 430,000× real-time, locomotion policy trained in 26 seconds.
- Warp, Brax, MJX: JAX-based differentiable backends.

Model-based RL in simulation

Instead of training a model-free RL agent that uses the simulator only, we can learn a world model from data and then plan or dream inside it.

The Dreamer family [Hafner et al., 2023]:

- A recurrent state-space model (RSSM) encodes history into a latent state z_t.
- Reward, observation and termination are predicted from z_t.
- The actor-critic is trained entirely on imagined latent rollouts.

Informed Dreamer: decode privileged information

Core idea [Lambrechts et al., 2024]: in a Dreamer-style world model, decode to the privileged information i_t instead of, or in addition to, the observation.

- The encoder path $o \to z$ remains usable at deployment.
- The decoder $z \to (r, i)$ is used only at training time to supervise the latent state.
- The world model learns task-relevant, Markov-sufficient latents faster.

A pure training-time upgrade to DreamerV3, with empirical gains on Velocity Control, Pop Gym and Flickering Atari.

Case study: SkyDreamer

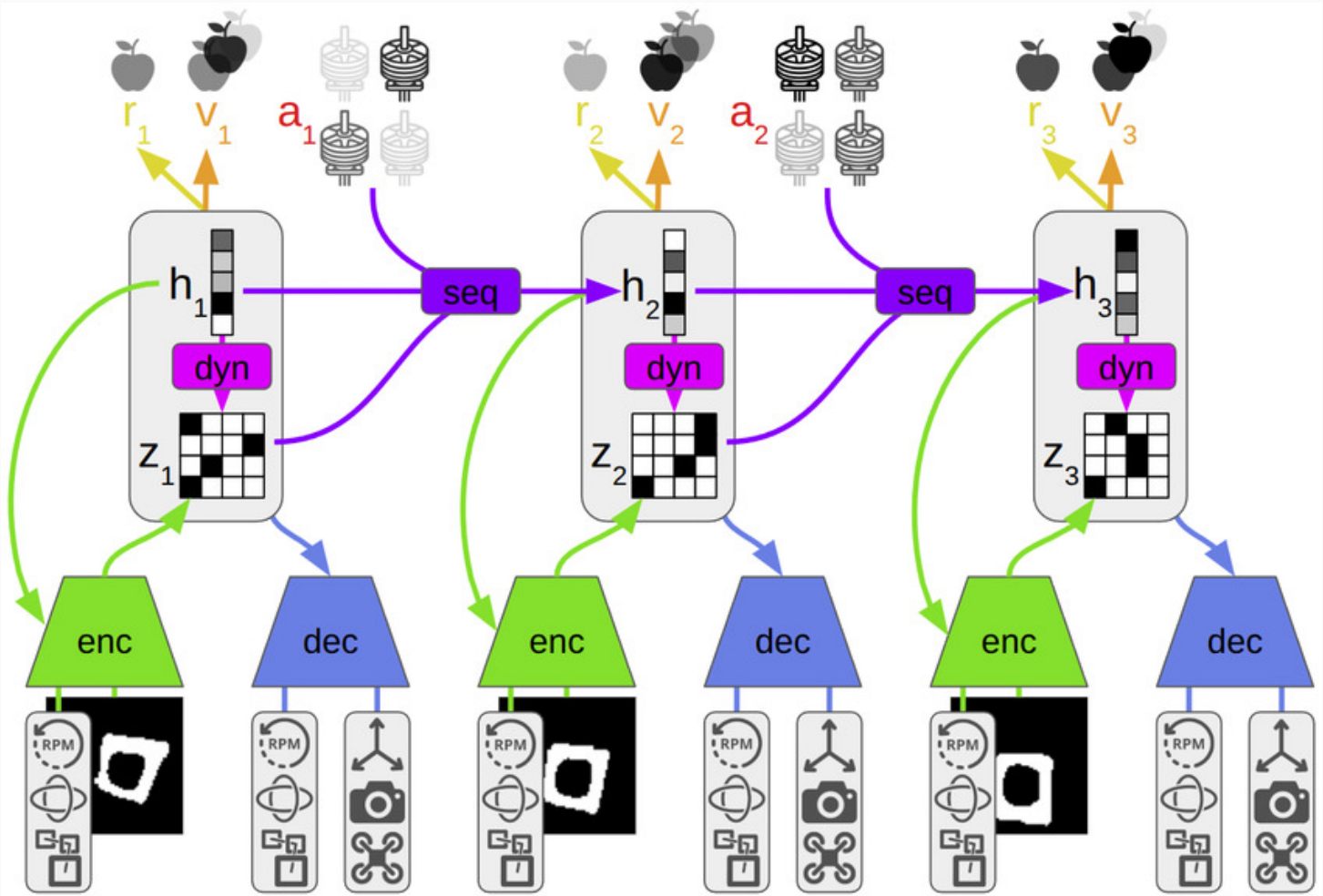
Problem. End-to-end vision-based drone racing: pixels → motor commands, onboard, no external aid, up to 21 m/s and 6 g.

SkyDreamer [Verraest et al., 2025] = DreamerV3 + Informed POMDP + massive randomization over ψ.

- Observations: binary gate masks, body rates, motor RPMs.
- Privileged information i_t: pose, velocity, attitude, camera extrinsics, dynamics parameters.
- Actions: four normalized motor commands, no inner-loop PID.

Outcome. The world model behaves as an implicit state and parameter estimator: it re-infers $\omega_{\max}$ as the battery drains and the policy compensates online.

SkyDreamer: architecture



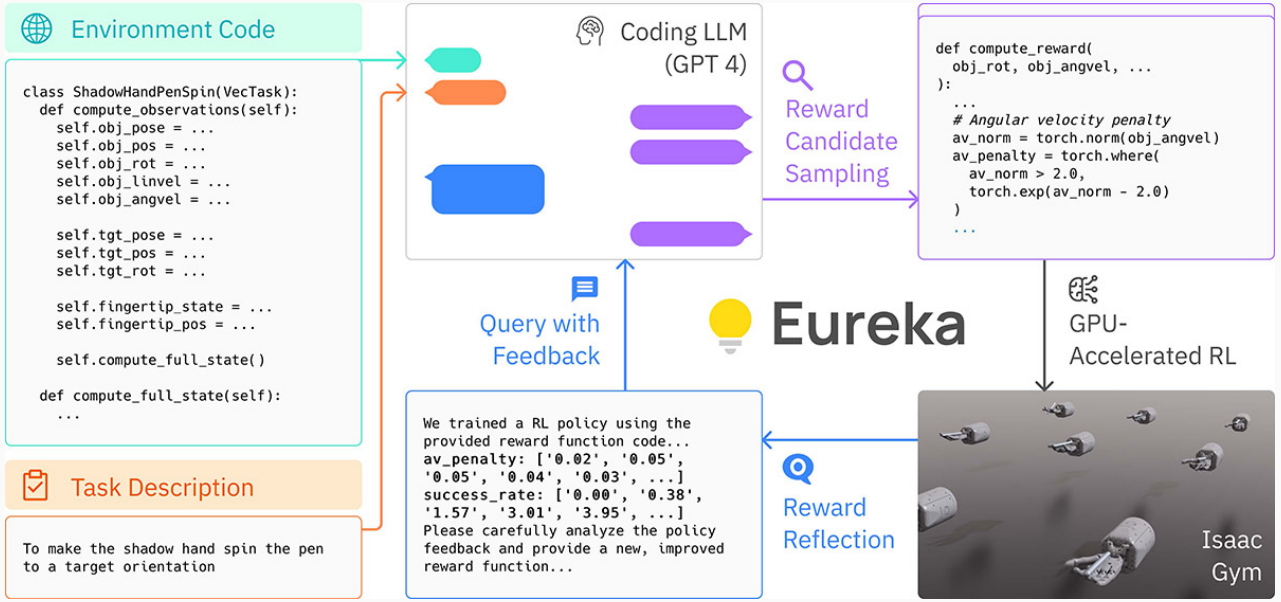
The world model decodes to privileged information i_t. At deployment, only the encoder path $o \to z$ is used. Figure from Verraest et al. [2025].

Reward design

So far we assumed $R(s, a)$ is given. In practice, reward design is one of the hardest steps, especially in robotics.

How would you design a reward for this task?

Eureka



Automated approach: Eureka [Ma et al., 2023] uses an LLM (GPT-4) to generate reward function code from a task description, then iteratively improves it via evolutionary search + GPU-accelerated evaluation.

- Outperforms human experts on 83% of 29 tasks across 10 robot morphologies.
- Average 52% normalized improvement over human-designed rewards.

Limitations of learning in simulation

Learning in simulation:

- Safety: learning in simulation avoids breaking robots during the learning phase.
- Real-time decision: RL shifts computation to training, not execution.

Limitations of learning in simulation

Learning in simulation:

- Safety: learning in simulation avoids breaking robots during the learning phase.
- Real-time decision: RL shifts computation to training, not execution.
- Sample efficiency: handled by GPU simulation and high parallelism.
- Diversity of deployed conditions: handled by the parametrized distribution of the context variables, simulated dynamics, and perception.
- Sensor noise: simulated models but some sensors are hard to replicate.

Open question: Could we avoid the reality gap and online learning limitations with large-scale datasets?

Offline RL and generalist robot policies

A third regime: learn from data

- **Part 1.** Online interaction in the real world.
- **Part 2.** Online interaction in a simulator.
- **Part 3.** Offline: learn from a fixed dataset of real-world trajectories.

Where does the data come from?

Most robotic datasets are collected via teleoperation: a human operator controls the robot through a leader device while the robot records observations and actions.

Offline RL

Definition (Offline RL)

Given a fixed dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$ collected by an unknown behavior policy η_β, find a policy η^* maximizing the expected return without any new environment interaction.

Central difficulty: distributional shift. Standard off-policy methods query $Q(s, a)$ at actions the dataset never saw. The critic extrapolates, becomes optimistic, and training diverges.

Three families of offline RL algorithms

The central trade-off: improve over η_β while staying close to η_β.

- Policy constraint: penalize deviation from η_β.
  Examples: BCQ, BEAR, TD3+BC [Fujimoto and Gu, 2021], AWAC.

- Value pessimism: push the critic down on out-of-distribution actions.
  Example: CQL [Kumar et al., 2020].

- In-sample learning: never query Q off-support.
  Example: IQL [Kostrikov et al., 2022]. Fits V with an expectile of Q, then extracts the policy by advantage-weighted behavior cloning.

Expressive policies: diffusion and flow matching

Robot datasets contain multimodal action distributions: left hand or right hand, approach from front or side, different speeds...

A unimodal Gaussian policy under-fits. We need expressive policy classes.

- Diffusion policies [Chi et al., 2023]: represent $\eta(a|s)$ as the denoising process of a diffusion model. Naturally handles multimodality.
- Flow matching: a more efficient alternative that learns a velocity field mapping a base distribution to actions in few integration steps.

These classes compose with offline RL (Diffusion-QL, IDQL) and are the backbone of modern VLAs.

From task-specific RL to foundation policies

Traditional robotic RL trains one policy per task from scratch. This limits generalization.

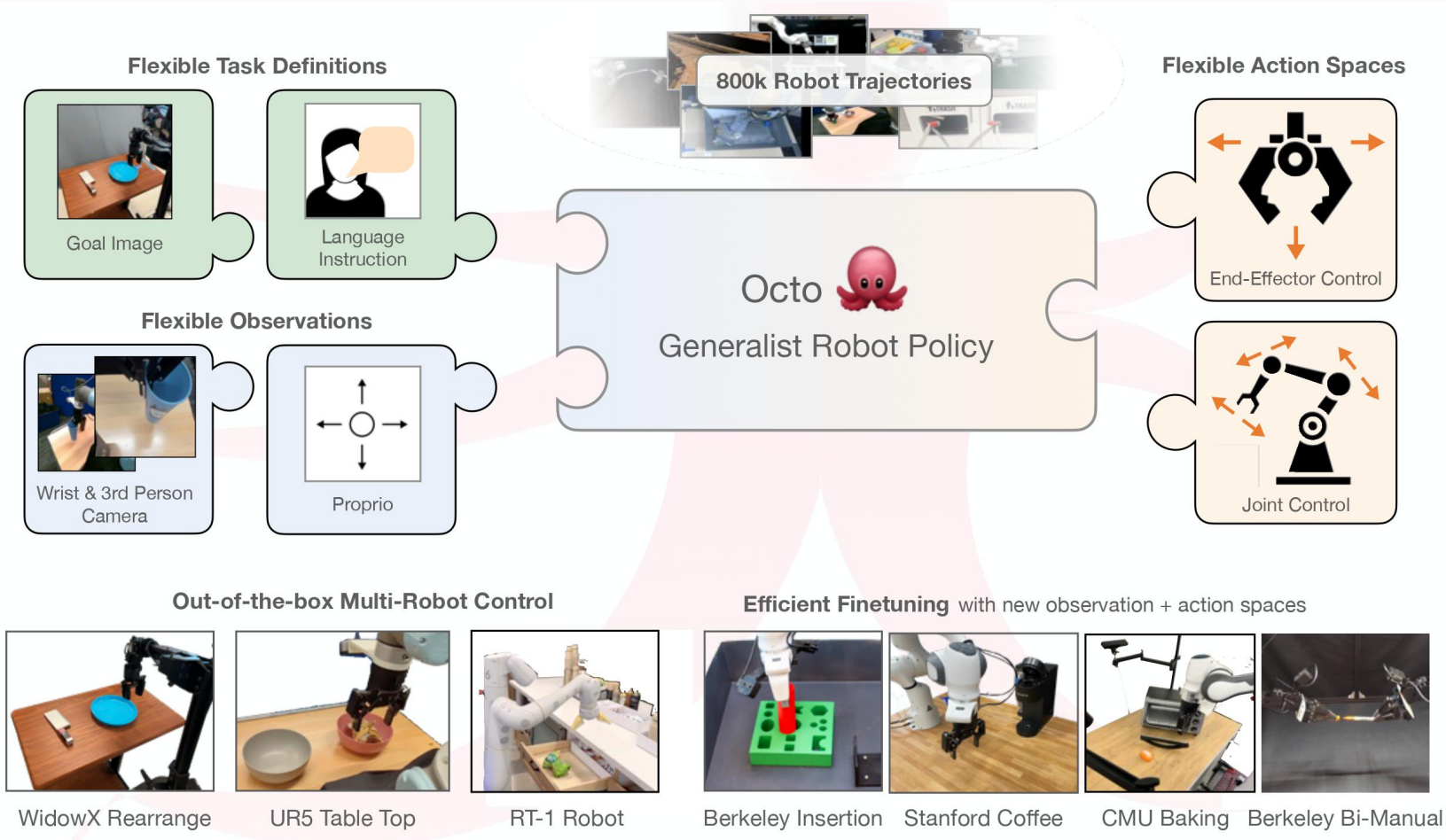
The foundation model paradigm:

1. Pretrain a large policy on diverse robot data (imitation, play, teleoperation) [Collaboration et al., 2024].
2. Fine-tune with RL on specific tasks or with real-world experience.

Reinforcement learning enters at two levels:

- Offline RL for pretraining on heterogeneous data (mixed quality, multiple sources).
- Online RL fine-tuning to surpass the imitation ceiling and adapt to deployment.

Octo: An Open-Source Generalist Robot Policy



Octo is a transformer-based policy pretrained on 800k diverse robot episodes from the Open X-Embodiment dataset [Collaboration et al., 2024]. It supports flexible task and observation definitions and can be quickly finetuned to new observation and action spaces.

Octo: architecture

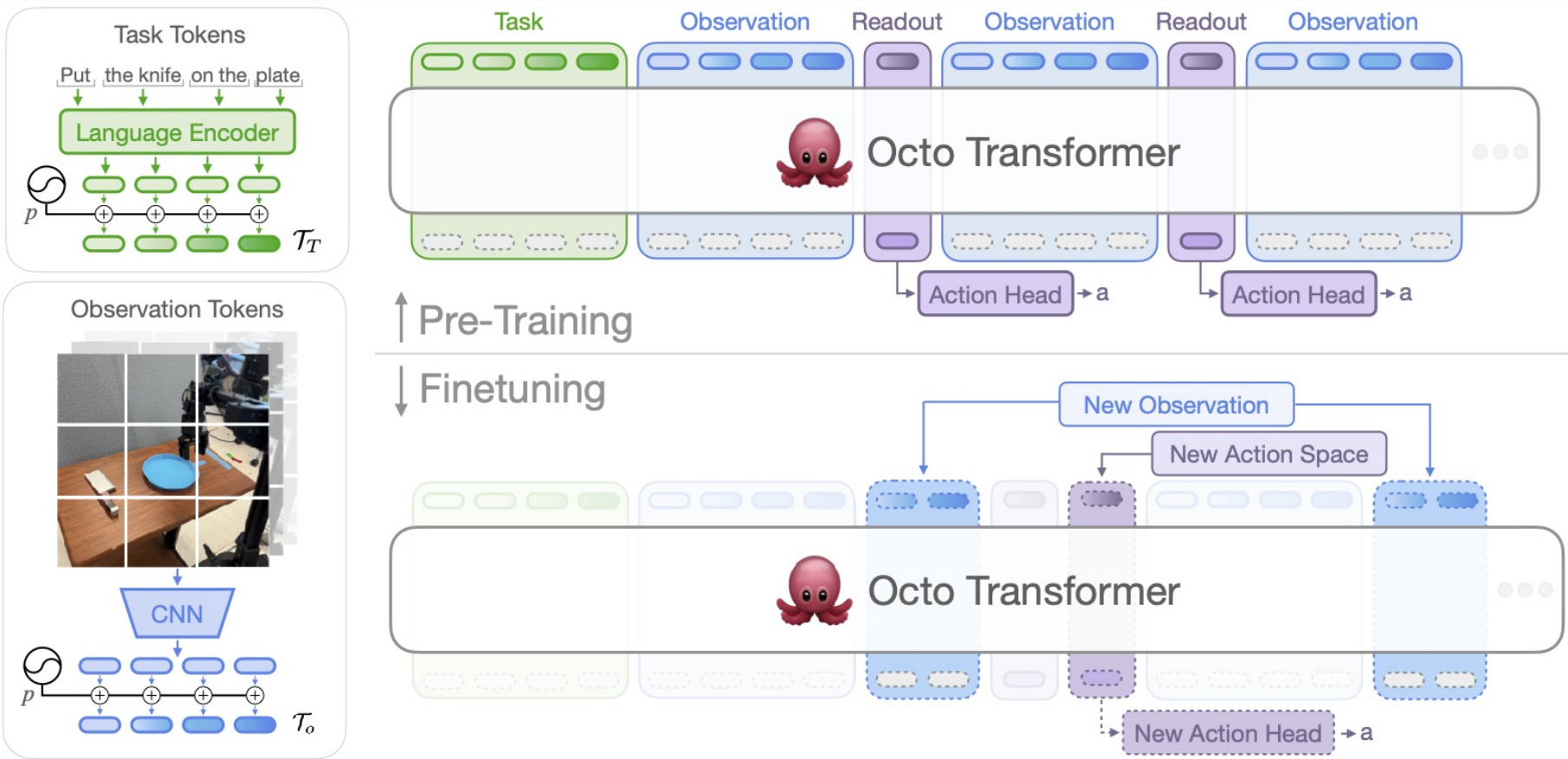


Figure from Octo Model Team et al. [2024].

Cross-embodiment learning

Cross-embodiment learning: a single policy performs across different robots.

Data from different robots and internet-scale vision-language data can be combined.

In practice: a shared high-level action space and an abstraction layer (often a control-theory-based low-level controller).

RL implication: the policy must generalize across embodiments. Offline RL on pooled multi-robot data naturally handles this if the action space is unified.

π_0: generalist robot policy for dexterous manipulation

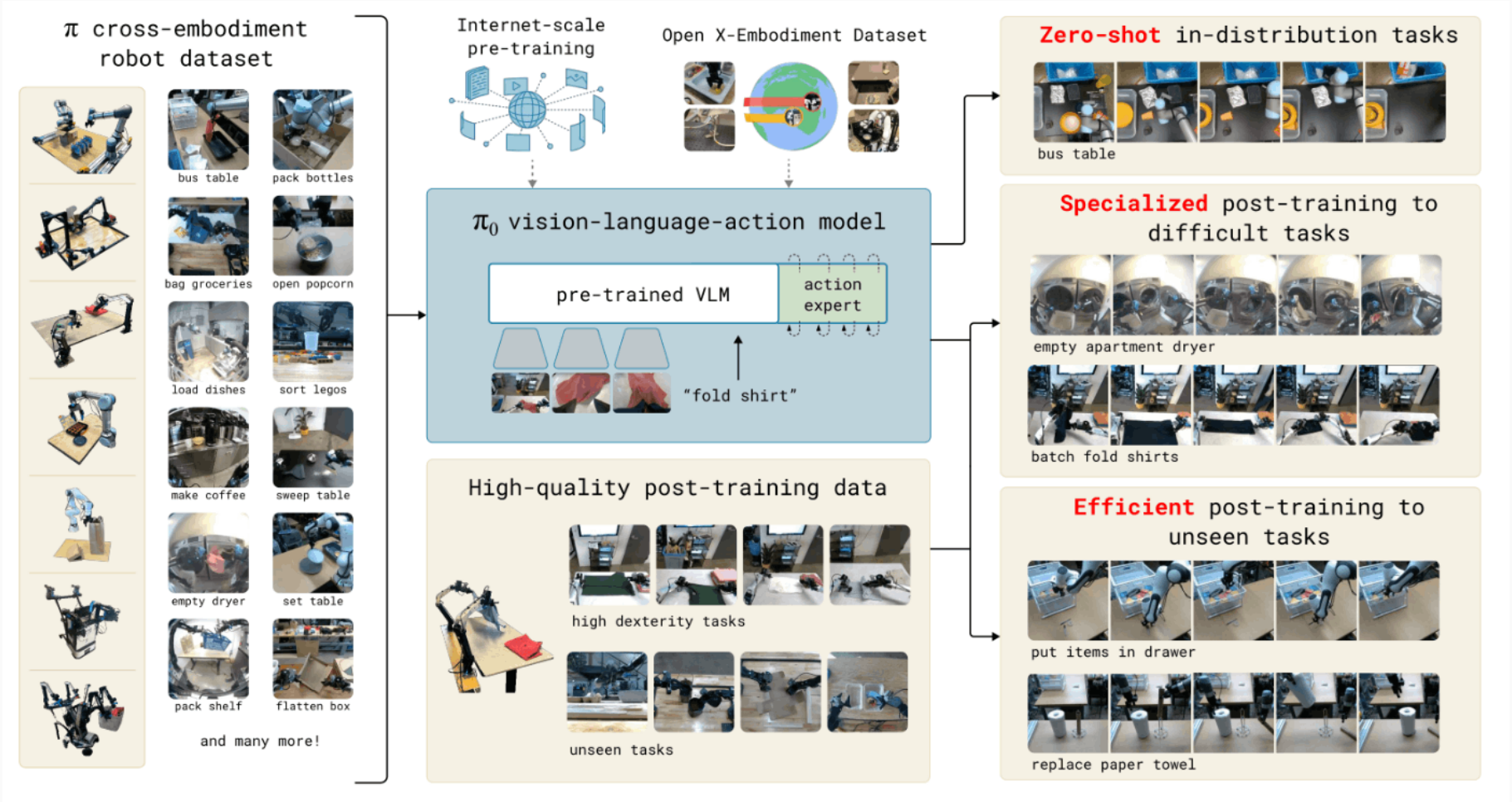


Figure from Black et al. [2024]

https://www.physicalintelligence.company/blog/pi0

From imitation to experience: $\pi^*_{0.6}$

π_0 and $\pi_{0.5}$ are trained by imitation on demonstrations. Their performance plateaus at the data. They have no way to learn from their own failures.

$\pi^*_{0.6}$ [Physical Intelligence, 2025] is the first VLA that learns from its own real-world experience via reinforcement learning. The recipe is called RECAP (RL with Experience and Corrections via Advantage-conditioned Policies).

Three stages:

1. Offline RL pretraining on demonstrations, past deployments and interventions.
2. Autonomous on-robot collection: successes and failures both re-enter the dataset.
3. Expert interventions: brief teleoperation corrections during autonomous execution, folded into the objective through advantage conditioning.

On the hardest tasks (folding laundry in real homes, box assembly, espresso), RECAP doubles task throughput and halves failure rate.

https://www.pi.website/blog/pistar06

Why advantage conditioning?

Instead of imitating the data or querying Q at unseen actions, train the policy conditioned on the advantage:

$$\eta_\theta(a \mid s, A) \;\approx\; p(a \mid s, \hat{A}(s,a) = A).$$

At deployment, condition on a high advantage.

Why it works for VLAs:

- The policy stays in the data distribution. No OOD queries, training stable at foundation-model scale.
- Handles heterogeneous data naturally: each source (demo, autonomous, intervention) just gets a different estimated advantage.
- Composes with any expressive head (flow matching, diffusion).

RL is the key to surpassing imitation

Current foundation models (π_0, Octo) are bounded by the demonstration data. RL is what breaks through this ceiling.

Evidence:

- $\pi^*_{0.6}$ (RECAP): RL fine-tuning doubles throughput and halves failure rate on the hardest tasks.
- Offline RL handles mixed-quality data better than pure imitation.
- Advantage conditioning lets RL scale to foundation-model-sized policies.

Open directions:

- Efficient RL fine-tuning of billion-parameter VLAs.
- LLM-designed rewards (Eureka) to specify new tasks without manual engineering.
- Combining sim pretraining + real-world RL in a single loop.

Putting it all together

In practice, state-of-the-art systems combine all three paradigms:

1. Pretrain in simulation.
2. Collect real-world data.
3. Fine-tune with RL on the real robot.

Example: $\pi^*_{0.6}$ pretrains on demonstrations (offline RL), then collects autonomous experience and expert corrections on the real robot (online RL). SkyDreamer trains entirely in simulation with domain randomization and deploys zero-shot.

The boundaries between the three paradigms are blurring. The best approach depends on the task, the available simulator, and the data budget.

When to use which approach?

	Online RL	Simulation	Offline / VLA
Simulator needed?	No	Yes	No
Real data needed?	Yes	No (or minimal)	Yes
Safe during training?	Risk	Safe	Safe
Sample efficiency	High (SAC)	Low (PPO)	N/A
Generalization	Specific robot	Sim distribution	Broad
Best for	Simple tasks, specific robot	easy-to-model tasks	Manipulation, open-world
Possible algorithms	SAC + high UTD	PPO + DR	IQL, Diff-QL, RECAP

Key takeaways

1. Real-world RL is possible but limited to simple tasks. SAC + high UTD + regularization can learn locomotion in 20 minutes.

2. Simulation + domain randomization is the current workhorse for locomotion and flight. PPO on GPU-parallel sims dominates. Differentiable simulators are the next frontier.

3. Privileged information (teacher-student, informed POMDPs) bridges the sim-to-real gap by using training-time signals unavailable at deployment.

4. Offline RL and VLAs are the path to general-purpose manipulation. RL fine-tuning ($\pi^*_{0.6}$) is what breaks through the imitation ceiling.

5. The three paradigms combine: sim pretraining $\rightarrow$ real data collection $\rightarrow$ RL fine-tuning. The boundaries are blurring.

References

Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second edition, 2017. ISBN 978-3-319-54413-7.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. URL https://arxiv.org/abs/1812.05905.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model, 2021. URL https://arxiv.org/abs/2101.05982.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning, 2022. URL https://arxiv.org/abs/2110.02034.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: Scaling for compute and sample-efficient continuous control, 2024.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity bias for scaling up parameters in deep reinforcement learning, 2024.

Antonin Raffin, Jens Kober, and Freek Stulp. Smooth exploration for robotic reinforcement learning, 2021.

Elie Aljalbout, Jiaxu Xing, Angel Romero, Iretiayo Akinola, Caelan Reed Garrett, Eric Heiden, Abhishek Gupta, Tucker Hermans, Yashraj Narang, Dieter Fox, Davide Scaramuzza, and Fabio Ramos. The reality gap in robotics: Challenges, solutions, and best practices, 2026. To appear in Annual Review of Control, Robotics, and Autonomous Systems.

References iii

Gaspard Lambrechts, Adrien Bolland, and Damien Ernst. Informed POMDP: Leveraging additional information in model-based RL. *Reinforcement Learning Journal*, 2:763–784, 2024.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid motor adaptation for legged robots. *Robotics: Science and Systems*, 2021.

Chris Gregg and Kim Hazelwood. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144. IEEE, 2011.

Antonin Raffin. Getting SAC to work on a massive parallel simulator. https://araffin.github.io/post/sac-massive-sim/, 2025.

Tairan He, Wenli Luo, Jiashun He, Ruisi Zhang, and Kris Kitani. HOVER: Versatile Neural Whole-Body Controller for Humanoid Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. arXiv:2410.21229.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

Aderik Verraest, Stavrow Bahnam, Robin Ferede, Guido de Croon, and Christophe De Wagter. SkyDreamer: Interpretable end-to-end vision-based drone racing with model-based reinforcement learning, 2025.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2023.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *NeurIPS*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *ICLR*, 2022.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Embodiment Collaboration, Abby O'Neill, Abdul Rehman, and Abhiram Maddukuri. Open x-embodiment: Robotic learning datasets and rt-x models, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

Physical Intelligence. $\pi^*_{0.6}$: a VLA that learns from experience, 2025.