Robot Learning: A Tutorial

Francesco Capuano ... **Adil Zouitine** **Pepijn Kooijmans** **Thomas Wolf** **Michel Aractingi**

École Normale Supérieure Paris-Saclay, Hugging Face

Abstract

Recent advancements in foundation models have enabled unprecedented progress in robotics, particularly in the development of increasingly more capable generalist robotics models. Leveraging modern learning-based approaches, the robotics community produced the first general models for tasks including locomotion and manipulation, and many said results have been openly released to the open-source community. The premise of generalist models is to learn control strategies across tasks and different robot embodiments, a long-standing challenge in robot learning. This work aims at being a tutorial on the most common techniques used for modern robot learning, covering both the conceptual underpinnings of the most prominent approaches, and pairing them with reproducible implementations using `lerobot`, the full-stack open-source robotics library developed by Hugging Face. This tutorial is designed to serve as a self-contained, structured reference for robotics practictioners working with `lerobot`, as well as a practical guide for researchers in the field of robot learning.
Code: `https://github.com/huggingface/lerobot`

Contents

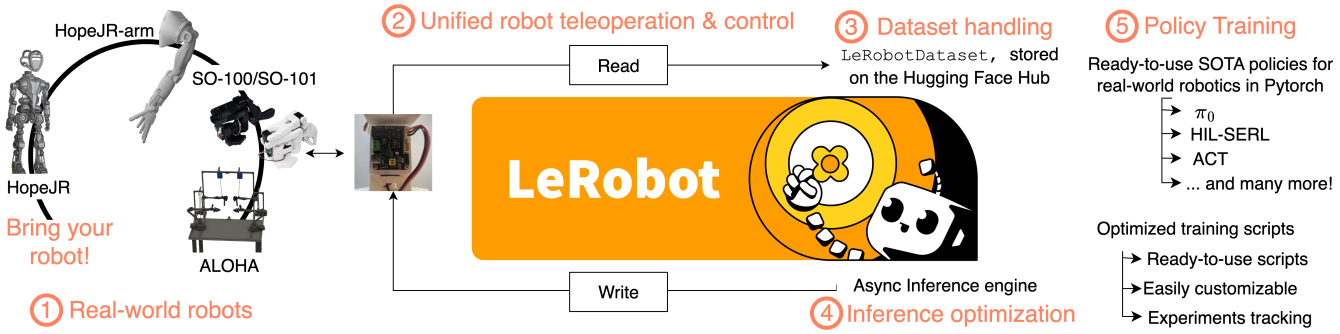


Figure 1 | `lerobot` is the open-source library for end-to-end robotics developed by Hugging Face. The library is vertically integrated on the entire robotics stack, supporting low-level control of real-world robot devices, advanced data and inference optimizations, as well as SOTA robot learning methods ported in pure Pytorch.

1 Introduction

Autonomous robotics holds the premise of relieving humans from repetitive, tiring or dangerous manual tasks. Consequently, the field of robotics has been widely studied since its first inception in the 1950s. Lately, advancements in Machine Learning (ML) have sparked the development of a relatively new class of methods used to tackle robotics problems, leveraging large amounts of computation and data rather than human expertise and modeling to develop autonomous systems.

Robotics in 2025 sits is increasingly moving away from classical model-based control paradigm, embracing the advancements made in ML, thus unlocking (1) monolithic perception-to-action action pipelines and (2) multi-modal data-driven feature extraction strategies, together with (3) reduced reliance on precise models of the world and (4) a better positioning to benefit from the growing availability of robotics data openly available. While central problems in manipulation, locomotion and whole-body control demand knowledge of rigid-body dynamics, contact modeling, planning under uncertainty, recent results seem to indicate learning can prove as effective as explicit modeling, sparking interest in the field of *robot learning*. This interest can be largely justified considering the significant challenges related to deriving accurate models of robot-environment interactions. Also, because end-to-end learning on large and increasing amounts of data produced *foundation models* capable to semantically reason over multiple input modalities (images, text, audio, etc.), deriving methods for robotics based on learning seems particularly promising, given the current trends related to the growth of openly available datasets.

At its core, robotics is also an inherently multi-disciplinary field, to the very least considering the diverse skills required in terms of *software* and *hardware*. Integrating learning-based techniques increases the heterogeneity of skills needed to tackle robotics, whether in applications or research. `lerobot`is an open-source library end-to-end integrated with the entire robotics stack. With a strong focus on accessible, real-world robots, (1) `lerobot`supports many, openly available, robotic platforms for manipulation, locomotion and even whole-body control. `lerobot`'s (2) unified, low-level approach to reading/writing robot configurations also allows to extend support for other robots relatively easily. The library also supports (3) a native robotics dataset's format—`LeRobotDataset`—that is used to efficiently share datasets. `lerobot`also supports many state-of-the-art (SOTA) algorithms in robot learning—Reinforcement Learning (RL) and Behavioral Cloning (BC)—with efficient implementations in Pytorch and extended support to experimentation and experiments tracking. Lastly, `lerobot`defines a custom, optimized inference stack for robotic policies decoupling action planning from action execution, proving effective in guaranteeing more adaptability at runtime.

This tutorial serves the double purpose of providing useful references for the science and practical use of common robot learning techniques. To this aim, we strike to provide rigorous yet concise overviews of the core concepts behind the techniques presented, paired with practical examples of how to use these in applications, for practitioners and researchers in the field of robot learning. This tutorial is structured as follows:

- Section 2 reviews classical robotics foundations, introducing the limitations of dynamics-based approaches to robotics.
- Section **??** elaborates on the limitations of dynamics-based methods, and introduces learning-based techniques, their upsides and potential limitations.

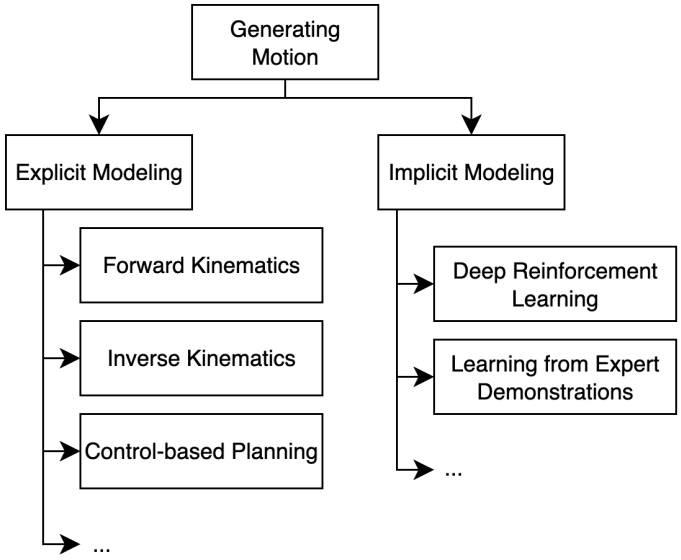


Figure 2 | Overview of methods to generate motion (clearly non-exhausitve, see Bekris et al. (2024)). The different methods can be grouped based on whether they explicitly (*dynamics-based*) or implicitly (*learning-based*) model robot-environment interactions.

- Section **??** further describes robot learning techniques that aim at solving single tasks learning from specific expert demonstrations.
- Section **??** presents recent contributions towards generalist models for robotics applications learning from large corpora of multi-robot multi-task data.
- Lastly, Section **??** covers emerging directions in robot learning research, introducing recent works in post-training techniques for robotics foundation models.

Lastly, we complement our presentation of the most common and recent approaches in robot learning with practical code implementations using `lerobot`.

2 Classical Robotics

Know your enemy [...]

Sun Tzu

TL;DR

Learning-based approaches to robotics are motivated by the need to (1) generalize across tasks and embodiments (2) reduce dependency on human expertise (3) leverage historical trends on the production of data—all traditionally overlooked by dynamics-based techniques.

2.1 Explicit and Implicit Models

Robotics is concerned with producing artificial motion in the physical world in useful, reliable and safe fashion. Thus, robotics is an inherently multi-disciplinar domain: producing autonomous motion in the physical world requires, to the very least, interfacing different software (motion planners) and hardware (motion executioners) components. Further, knowledge of mechanical, electrical, and software engineering, as well as rigid-body mechanics and control theory have therefore proven quintessential in robotics since the field first developed in the 1950s. More recently, Machine Learning (ML) has also proved effective in robotics, complementing these more traditional disciplines (Connell and Mahadevan, 1993). As a direct consequence of its multi-disciplinar nature, robotics has developed as a rather wide array of methods, all concerned with the main purpose of producing artificial motion in the physical world.

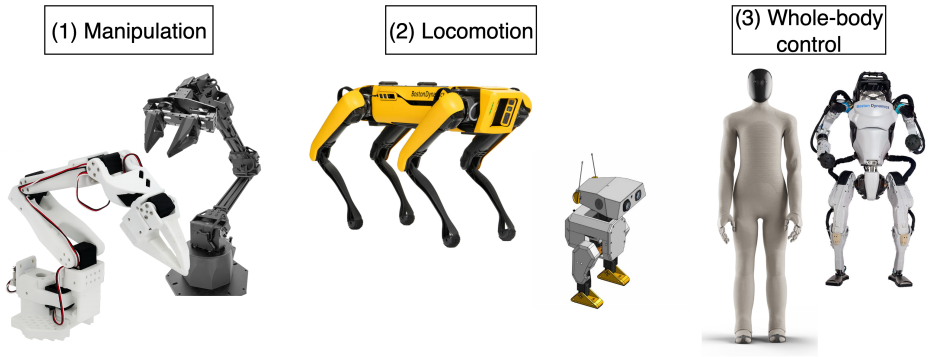


Figure 3 | Different kinds of motions are achieved with potentially very different robotic platforms. From left to right, top to bottom: ViperX, SO-100, Boston Dynamics' Spot, Open-Duck, 1X's NEO, Boston Dynamics' Atlas. This is an example list of robotic platforms and is (very) far from being exhaustive.

Methods to produce robotics motion range from traditional *explicit* models—dynamics-based methods, leveraging precise descriptions of the mechanics of robots' rigid bodies and their interactions with eventual obstacles in the environment—to *implicit* models—learning-based methods, treating artificial motion as a statistical pattern to learn given multiple sensorimotor readings (Agrawal; Bekris et al., 2024). A variety of methods have been developed between these two extrema. For instance, Hansen et al. (2022) show how learning-based systems can benefit from information on the physics of problems, complementing a traditional learning method such as Temporal Difference (TD)-learning Sutton and Barto (2018) with Model-Predictive Control (MPC). Conversely, as explicit models may be relying on assumptions proving overly simplistic—or even unrealistic—in practice, learning can prove effective to improve modeling of complex phenomena or complement perception (McCormac et al., 2016). Such examples aim at demonstrating the richness of approaches to robotics, and Figure 2 graphically illustrates some of the most relevant techniques. Such a list is clearly far from being exhaustive, and we refer to Bekris et al. (2024) for a more comprehensive overview of both general and application-specific methods for motion generation. In this section, we wish to introduce the inherent benefits of learning-based approaches to robotics—the core focus on this tutorial.

2.2 Different Types of Motion

At its very core, robotics deals with producing motion via actuating joints connecting nearly entirely-rigid links. A key distinction between focus areas in robotics is based on whether the generated motion modifies (1) the absolute state of the environment (via dexterity), (2) the relative state of the robot with respect to its environment (exercising mobility skills), or (3) a combination of the two (Figure 3).

Effects such as (1) are typically achieved *through* the robot, i.e. generating motion to perform an action inducing a desirable modification, effectively *manipulating* the environment (manipulation). Motions like (2) may result in changes in the robot's physical location within its environment. Generally, modifications to a robot's location within its environment may be considered instances of the general *locomotion* problem, further specified as *wheeled* or *legged* locomotion based on whenever a robot makes use of wheels or leg(s) to move in the environment. Lastly, an increased level of dynamism in the robot-environment interactions can be obtained combining (1) and (2), thus designing systems capable to interact with *and* move within their environment. This category is problems is typically termed *whole-body control*, and is characterized by a typically much larger set of control variables compared to either locomotion or manipulation alone.

The traditional body of work developed since the very inception of robotics is increasingly more complemented by learning-based approaches. ML has indeed proven particularly transformative across the entire robotics stack, first empowering planning-based techniques with improved state estimation used for traditional planning (Tang et al., 2023) and then end-to-end replacing controllers, effectively yielding perception-to-action methods (Kober et al.). Work in producing robots capable of navigating a diverse set of terrains demonstrated the premise of both dynamics and learning-based approaches for locomotion (Griffin et al., 2017; Ji et al., 2023; Lee et al., 2020; Margolis et al., 2022), and recent works on whole-body control indicated the premise of learning-based approaches to generate rich motion on complex robots, including humanoids (Zhang et al., 2024; **?**). Manipulation has also been widely studied, particularly considering its relevance for many impactful applications ranging from high-risk applications for humans (Fujita et al., 2020; Alizadeh and Zhu, 2024; Fujita et al., 2020) to manufacturing (Sanneman et al., 2020). While explicit models have proven fundamental in achieving important milestones towards the development of modern robotics, recent

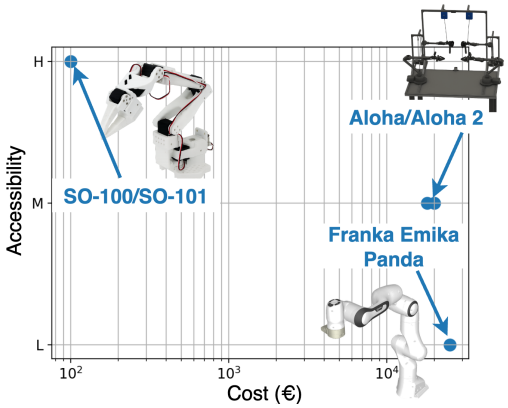


Figure 4 | Cheaper, more accessible robots are starting to rival traditional platforms like the Panda arm platforms in adoption in resource-constrained scenarios. The SO-100, in particular, has a cost in the 100s of Euros, and can be entirely 3D-printed in hours, while the industrially-manufactured Panda arm costs tens of thousands of Euros and is not openly available.

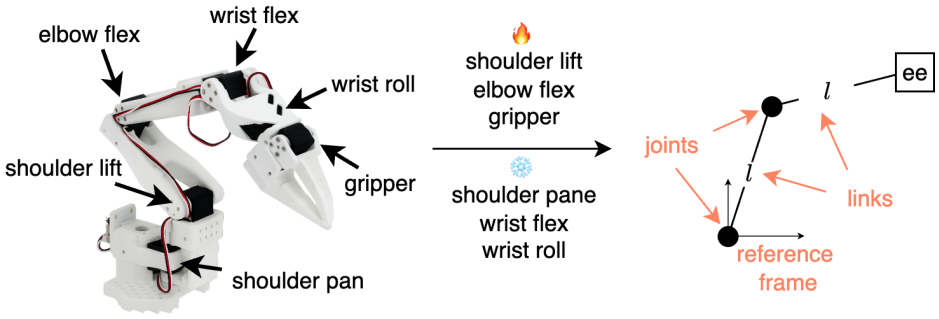


Figure 5 | The SO-100 arm is a 6-dof manipulator arm. Preventing some of its joints (shoulder pane, wrist flex and wrist roll) from actuating, it can be represented as a traditional 2-dof planar manipulator (the gripper joint in the end-effector is not considered towards the count of the degrees of freedom used to produce motion).

works leveraging implicit models proved particularly promising in surpassing scalability and practical applicability challenges via learning (Kober et al.).

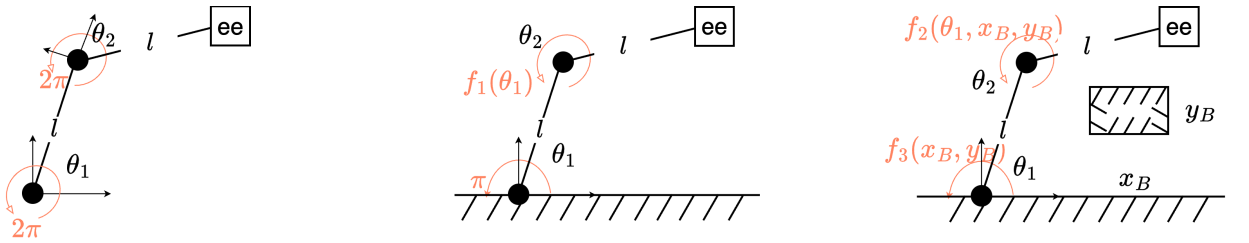
2.3 Example: Planar Manipulation

Robot manipulators typically consist of a series of links and joints, articulated in a chain finally connected to an *end-effector*. Links and joints are considered responsible for generating motion, while the end effector is instead used to perform specific actions at the target location (e.g., grasping/releasing objects via closing/opening a gripper end-effector, using a specialized tool like a screwdriver, etc.).

Recently, the development of low-cost manipulators like the Aloha (Zhao et al., 2023) Aloha-2 (Aldaco et al.) and SO-100/SO-101 (Knight et al.) platforms significantly lowered the barrier to entry to robotics, considering the increased accessibility of these robots compared to more traditional platforms like the Franka Emika Panda arm (Figure 4).

Deriving an intuition as per why learning-based approaches are gaining popularity in the robotics community requires briefly analyzing traditional approaches for manipulation, leveraging tools like forward and inverse kinematics (FK, IK) and control theory. Providing a detailed overview of these methods falls (well) out of the scope of this tutorial, and we refer the reader to works including Siciliano and Khatib (2016); Lynch and Park (2017); Tedrake (a,b) for a much more comprehensive description of these techniques. Here, we mostly wish to highlight the benefits of ML over these traditional techniques

Consider the (simple) case where a SO-100 is restrained from actuating (1) the shoulder pane and (2) the wrist flex and roll motors. This effectively reduces the degrees of freedom of the SO-100 from the original 5+1 (5 joints + 1 gripper) to 2+1 (shoulder lift, elbow flex + gripper). As the end-effector does not impact motion in this model, the SO-100 is effectively reduced to the planar manipulator robot presented in Figure 5, where spheres represent



(a) | Free to move

(b) | Constrained by the surface

(c) | Constrained by surface and (fixed) obstacle

Figure 6 | Planar, 2-dof schematic representation of the SO-100 manipulator under diverse deployment settings. From left to right: completely free of moving; constrained by the presence of the surface; constrained by the surface and presence of obstacles. Circular arrows around each joint indicate the maximal rotation feasible at that joint.

actuators, and solid lines indicate links from the base of the SO-100 to the end-effector (ee).

Further, let us make the simplifying assumption that actuators can produce rotations up to 2π radians. In practice, this is seldom the case due to movement obstructions caused by the robot body itself (for instance, the shoulder lift cannot produce counter-clockwise movement due to the presence of the robot's base used to secure the SO-100 to its support and host the robot bus), but we will introduce movement obstruction at a later stage.

All these simplifying assumptions leave us with the planar manipulator of Figure 6a, free of moving its end-effector by controlling the angles θ_1 and θ_2, jointly referred to as the robot's *configuration*, and indicated with $q = [\theta_1, \theta_2] \in [-\pi, +\pi]^2$. The axis attached to the joints indicate the associated reference frame, whereas circular arrows indicate the maximal feasible rotation allowed at each joint. In this tutorial, we do not cover topics related to spatial algebra, and we instead refer the reader to Lynch and Park (2017, Chapter 2) and Tedrake (a, Chapter 3) for excellent explanations of the mechanics and theoretical foundations of producing motion on rigid bodies.

Considering the (toy) example presented in Figure 6a, then we can analytically write the end-effector's position $p \in \mathbb{R}^2$ as a function of the robot's configuration, $p = p(q), p : \mathcal{Q} \mapsto \mathbb{R}^2$. In particular, we have:

$$p(q) = \begin{pmatrix} p_x(\theta_1, \theta_2) \\ p_y(\theta_1, \theta_2) \end{pmatrix} = \begin{pmatrix} l\cos(\theta_1) + l\cos(\theta_1 + \theta_2) \\ l\sin(\theta_1) + l\sin(\theta_1 + \theta_2) \end{pmatrix} \in S^{n=2}_{l_1+l_2} = \{p(q) \in \mathbb{R}^2 : \|p(q)\|_2^2 \leq (2l)^2, \ \forall q \in \mathcal{Q}\}$$

Deriving the end-effector's *pose* in some m-dimensional space $\boldsymbol{p} \in \mathcal{P} \subset \mathbb{R}^m$ (position and orientation) starting from the configuration $\mathrm{q} \in \mathcal{Q} \subset \mathbb{R}^n$ of a n-joints robot is referred to as *forward kinematics* (FK), whereas identifying the configuration corresponding to any given target pose is termed *inverse kinematics* (IK). In that, FK is used to map a robot configuration into the corresponding end-effector pose, whereas IK is used to reconstruct the configuration(s) given an end-effector pose.

In the simplified case here considered (for which $\boldsymbol{p} \equiv p$, as the orientation of the end-effector is disregarded for simplicity), one can solve the problem of controlling the end-effector's location to reach a goal position p^* by solving analytically for $q : p(q) = f_{\text{FK}}(q) = p^*$. However, in the general case, one might not be able to solve this problem analytically, and can typically resort to iterative optimization methods comparing candidate solutions using a loss function (in the simplest case, $\|p(q) - p^*\|_2^2$ is a natural candidate), yielding:

$$\min_{q \in \mathcal{Q}} \|p(q) - p^*\|_2^2 \tag{1}$$

Analytical solutions are even less appealing when one considers the presence of obstacles in the robot's workspace, resulting in constraints on the possible values of $q \in \mathcal{Q} \subseteq [-\pi, +\pi]^n \subset \mathbb{R}^n$ in the general case of n-links robots.

For instance, the robot in Figure 6b is (very naturally) obstacled by the presence of the surface upon which it rests: θ_1 can now exclusively vary within $[0, \pi]$, while possible variations in θ_2 depend on θ_1 (when $\theta_1 \to 0$ or $\theta_1 \to \pi$, further downwards movements are restricted). Even for a simplified kinematic model, developing techniques to solve 1 is in general non-trivial in the presence of constraints, particularly considering that the feasible set of solutions $\mathcal{Q}$ may change across problems. Figure 6c provides an example of how the environment influences the feasible set considered, with a new set of constraints deriving from the position of a new obstacle.

Further, IK—solving 1 for a feasible q—only proves useful in determining information regarding the robot's configuration in the goal pose, and crucially does not provide information on the *trajectory* to follow over time to reach a target pose. Expert-defined trajectories obviate to this problem providing a length-K succession of goal poses $\tau_K = [p_0^*, p_1^*, \dots p_K^*]$ for tracking. In practice, trajectories can also be obtained automatically through *motion planning* algorithms, thus avoiding expensive trajectory definition from human experts. However, tracking τ_K via IK can prove prohibitively expensive, as tracking would require K resolutions of 1 (one for each target pose). *Differential* inverse kinematics (diff-IK) complements IK by enabling adaptive trajectory tracking. Let $J(q)$ denote the Jacobian matrix of (partial) derivatives of the FK-function $f_{\text{FK}} : \mathcal{Q} \mapsto \mathcal{P}$, such that $J(q) = \frac{\partial f_{FK}(q)}{\partial q}$. Then, one can apply the chain rule to any $p(q) = f_{\text{FK}}(q)$, deriving $\dot{p} = J(q)\dot{q}$, and thus finally relating variations in the robot configurations to variations in pose, thereby providing a platform for control.

Given a desired end-effector trajectory $\dot{p}^*(t)$ (1) indicating ancor regions in space and (2) how much time to spend in each region, diff-IK finds $\dot{q}(t)$ solving for joints' *velocities* instead of *configurations*,

$$\dot{q}(t) = \arg\min_{\nu} \|J(q(t))\nu - \dot{p}^*(t)\|_2^2 \tag{2}$$

Unlike 1, solving for $\dot{q}$ is much less dependent on the environment (typically, variations in velocity are constrained by physical limits on the actuators represented with box-like constraints in 2). Conveniently, 2 also often admits the closed-form solution $\dot{q} = J(q)^+\dot{p}^*$, where $J^+(q)$ denotes the Moore-Penrose pseudo-inverse of $J(q)$. Finally, discrete-time joint configurations q can be reconstructed from joint velocities $\dot{q}$ using forward-integration on the continuous-time joint velocity , $q_{t+1} = q_t + \Delta t\, \dot{q}_t$ for a given Δt, resulting in tracking via diff-IK.

Following trajectories with diff-IK is a valid option in well-controlled and static environments (e.g., industrial manipulators in controlled manufacturing settings), and relies on the ability to define a set of target velocities to track $[\dot{p}_0^*, \dot{p}_1^*, \dots, \dot{p}_k^*]$—an error-prone task largely carried out by human experts. Furthermore, diff-IK relies on the ability to (1) access $J(q)\,\forall q \in \mathcal{Q}$ and (2) compute its pseudo-inverse at every iteration of a given control cycle—a challenging assumption in highly dynamical settings, or for complex kinematic chains, or nonlinear dynamics.

2.3.1 Adding Feedback Loops

While very effective when a goal trajectory has been well specified, the performance of diff-IK can degrade significantly in the presence of modeling/tracking errors, or in the presence of non-modeled dynamics in the environment.

One such case is presented in Figure 7, where another rigid body other than the manipulator is moving in the environment along the horizontal axis, with velocity $\dot{x}_B$. Accounting analytically for the presence of this disturbance—for instance, to prevent the midpoint of l_1 from ever colliding with the object—requires access to $\dot{x}_B$ at least, to derive the equation characterizing the motion of the environment.

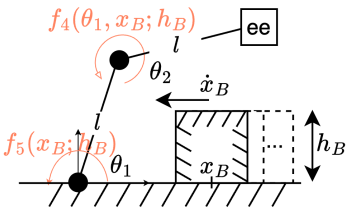


Figure 7 | Planar manipulator robot in the presence of a moving obstacle.

Less predictable disturbances however (e.g., $\dot{x}_B \leftarrow \dot{x}_B + \varepsilon, \varepsilon \sim N(0,1)$) may prove challenging to model analytically, and one could attain the same result of preventing link-object collision by adding a condition on the distance between the midpoint of l_1 and x_B, enforced through a feedback loop on the position of the robot and object at each control cycle.

To mitigate the effect of modeling errors, sensing noise and other disturbances, classical pipelines indeed do augment diff-IK with feedback control looping back quantities of interest. In practice, following a trajectory with a closed feedback loop might consist in backwarding the error between the target and measured pose, $\Delta p = p^* - p(q)$, hereby modifying the control applied to $\dot{q} = J(q)^+(\dot{p}^* + k_p\Delta p)$, with k_p defined as the (proportional) gain.

More advanced techniques for control consisting in feedback linearization, PID control, Linear Quatratic Regulator (LQR) or Model-Predictive Control (MPC) can be employed to stabilize tracking and reject moderate perturbations ((Siciliano and Khatib, 2016, Chapter 8) for in-detail explanation of these concepts, or (Tedrake, a, Chapter 8) for a simple, intuitive example in the case of a point-mass system). Nonetheless, feedback control presents its challenges as well: tuning gains remains laborious and system-specific. In particular, manipulation tasks present intermittent contacts inducing hybrid dynamics (mode switches) and discontinuities in the effective Jacobian, challenging the stability guarantees of the controller and thus often necessitating rather conservative gains and substantial hand-tuning.

We point the interested reader to Siciliano and Khatib (2016, Chapter 2,7,8), Lynch and Park (2017, Chapter 6,11),

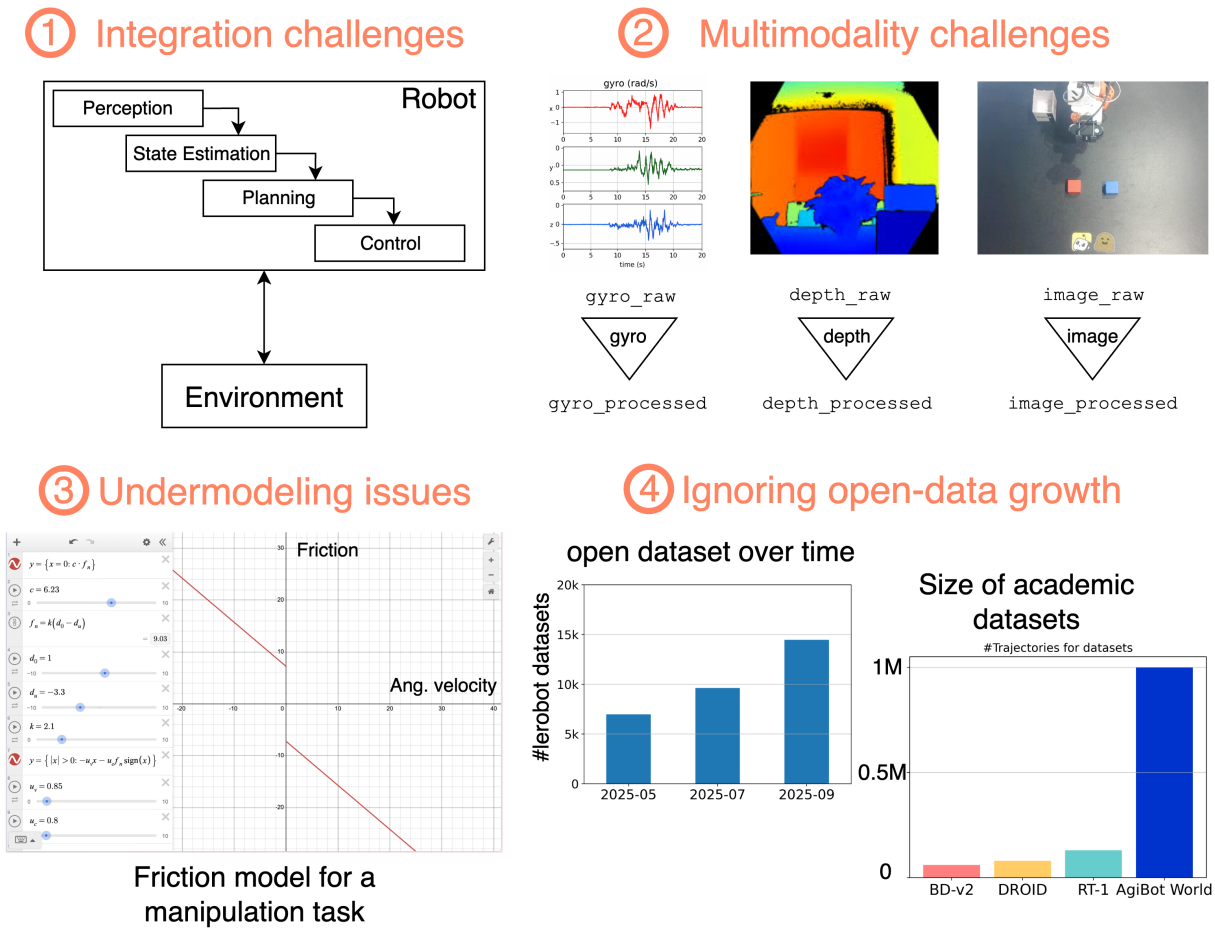


Figure 8 | Dynamics-based approaches to robotics suffer from several limitations: (1) orchestrating multiple components poses integration challenges; (2) the need to develop custom processing pipelines for the sensing modalities and tasks considered hinders scalability; (3) simplified analytical models of physical phenomena (here friction at the gripper; credits to Antonova et al. (2017)) limit real-world performance. Lastly, (4) dynamics-based methods overlook trends in the availability and growth of robotics data.

and Tedrake (a, Chapter 3,8) for extended coverage of FK, IK, diff-IK and control for (diff-)IK.

2.4 Limitations of Dynamics-based Robotics

Despite the last 60+ years of robotics research, autonomous robots are still largely incapable of performing tasks at human-level performance in the physical world generalizing across (1) robot embodiments (different manipulators, different locomotion platforms, etc.) and (2) tasks (tying shoe-laces, manipulating a diverse set of objects). While essential in the early development of robotics, the aforementioned methods require significant human expertise to be used in practice, and are typically specific to a particular applicative problem.

Dynamics-based robotics pipelines have historically been developed sequentially, engineering the different blocks now within most architectures for specific purposes. That is, sensing, state estimation, mapping, planning, (diff-)IK, and low-level control have been traditionally developed as distinct modules with fixed interfaces. Pipelining these specific modules proved error-prone, and brittleness emerges—alongside error compounding—whenever changes incur (e.g., changes in lighting for sensing, occlusion/failure of sensors, control failures). Adapting such a stack to new tasks or robotic platforms often entails re-specifying objectives, constraints, and heuristics at multiple stages, incurring significant engineering overhead.

Moreover, classical planners operate on compact, assumed-sufficient state representations; extending them to reason directly over raw, heterogeneous and noisy data streams is non-trivial. This results in a limited scalability to multimodal data and multitask settings, as incorporating high-dimensional perceptual inputs (RGB, depth, tactile, audio) traditionally required extensive engineering efforts to extract meaningful features for control. Also, the large

number of tasks, coupled with the adoption of *per-task* planners, goal parameterizations, and safety constraints, results in an explosion in design and validation options, with little opportunity to reuse solutions across tasks.

Setting aside integration and scalability challenges: developing accurate modeling of contact, friction, and compliance for complicated remains difficult. Rigid-body approximations are often insufficient in the presence of deformable objects, and relying on approximated models hinders real-world applicability of the methods developed. In the case of complex, time-dependent and/or non-linear dynamics, even moderate mismatches in parameters, unmodeled evolutions, or grasp-induced couplings can qualitatively affect the observed dynamics.

Lastly, dynamics-based methods (naturally) overlook the rather recent increase in availability of openly-available robotics datasets. The curation of academic datasets by large centralized groups of human experts in robotics (Ope; DRO; AgiBot-World-Contributors et al., 2025) is now increasingly complemented by a growing number of robotics datasets contributed in a decentralized fashion by individuals with varied expertises. If not tangentially, dynamics-based approaches are not posed to maximally benefit from this trend, which holds the premise of allowing generalization in the space of tasks and embodiments, like data was the cornerstone for advancements in vision (Alayrac et al., 2022) and natural-language understanding (Brown et al., 2020).

Taken together, these limitations (Figure 8) motivate the exploration of learning-based approaches that can (1) integrate perception and control more tightly, (2) adapt across tasks and embodiments with reduced expert modeling interventions and (3) scale gracefully in performance as more robotics data becomes available.

3 Robot (Reinforcement) Learning

Approximate the solution, not the problem [...]

Richard Sutton

TL;DR

The need for expensive high-fidelity simulators can be obviated by learning from real-world data, using sample-efficient algorithms that can safely train directly on hardware.

Learning-based techniques for robotics naturally address the limitations presented in 2 (Figure 9). Learning-based techniques typically rely on prediction-to-action (*visuomotor policies*), thereby directly mapping sensorimotor inputs to predicted actions, streamlining control policies by removing the need to interface multiple components. Mapping sensorimotor inputs to actions directly also allows to add diverse input modalities, leveraging the automatic feature extraction characteristic of most modern learning systems. Further, learning-based approaches can in principle entirely bypass modeling efforts and instead rely exclusively on interactions data, proving transformative when dynamics are challenging to model or even entirely unknown. Lastly, learning for robotics (*robot learning*) is naturally well posed to leverage the growing amount of robotics data openly available, just as computer vision first and natural language processing later did historically benefit from large scale corpora of (possibly non curated) data, in great part overlooked by dynamics-based approaches.

Being a field at its relative nascent stages, no prevalent technique(s) proved distinctly better better in robot learning. Still, two major classes of methods gained prominence: reinforcement learning (RL) and Behavioral Cloning (BC) (Figure 10). In this section, we provide a conceptual overview of applications of the former to robotics, as well as introduce practical examples of how to use RL within `lerobot`. We then introduce the major limitations RL suffers from, to introduce BC techniques in the next sections (**??**).

In Figure 10 we decided to include generalist robot models (Black et al., 2024; Shukor et al., 2025) alongside task-specific BC methods. While significant different in spirit—*generalist* models are language-conditioned and use instructions to generate motion valid across many tasks, while *task-specific* models are typically not language-conditioned and used to perform a single task—foundation models are largely trained to reproduce trajectories contained in a large training set of input demonstrations. Thus, we argue generalist policies can indeed be grouped alongside other task-specific BC methods, as they both leverage similar training data and schemas.

Figure 10 illustrates this categorization graphically, explicitly listing all the robot learning policies currently available in `lerobot`: Action Chunking with Transformers (ACT) (Zhao et al., 2023), Diffusion Policy (Chi et al., 2024), Vector-Quantized Behavior Transformer (VQ-BeT) (Lee et al., 2024), π_0 (Black et al., 2024), SmolVLA (Shukor et al.,

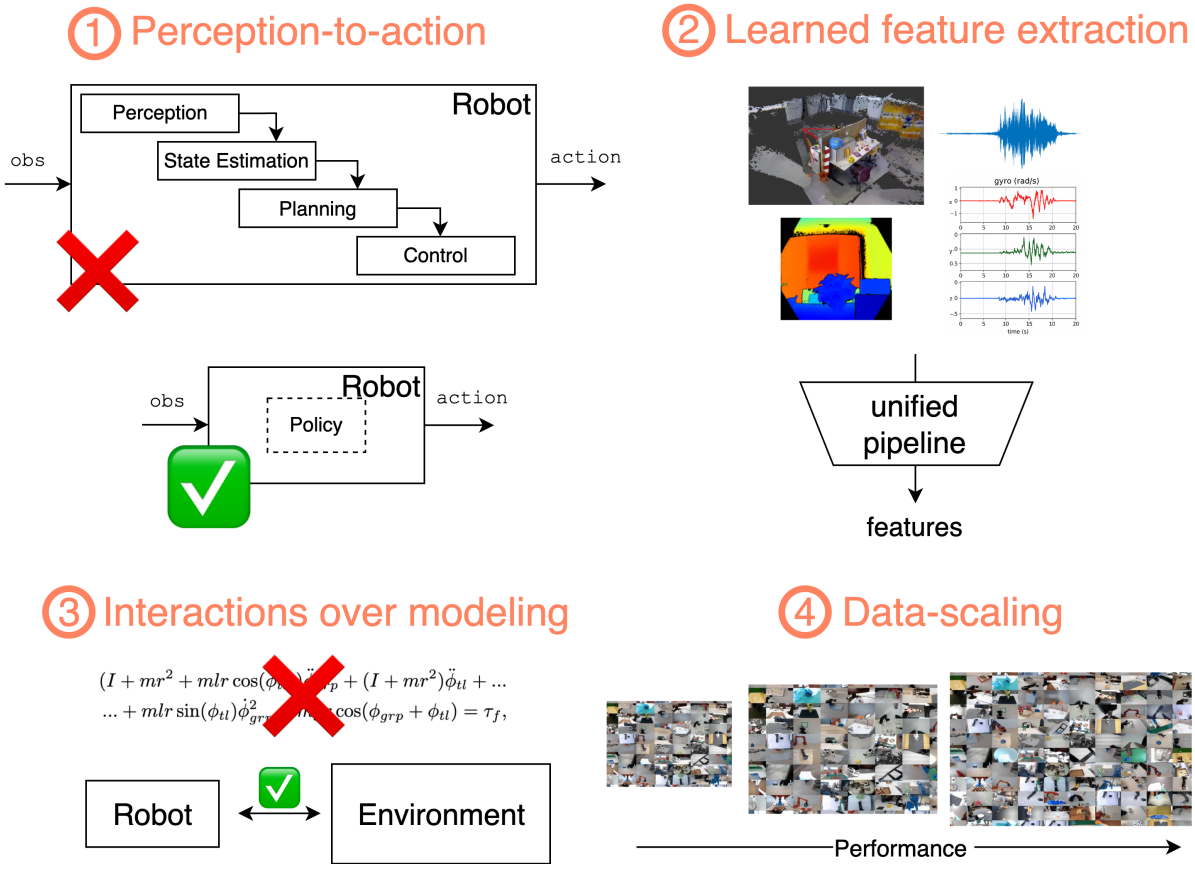


Figure 9 | Learning-based robotics streamlines perception-to-action by learning a (1) unified high-level controller capable to take (2) high-dimensional, unstructured sensorimotor information. Learning (3) does not require a dynamics model and instead focuses on interaction data, and (4) empirically correlates with the scale of the data used.

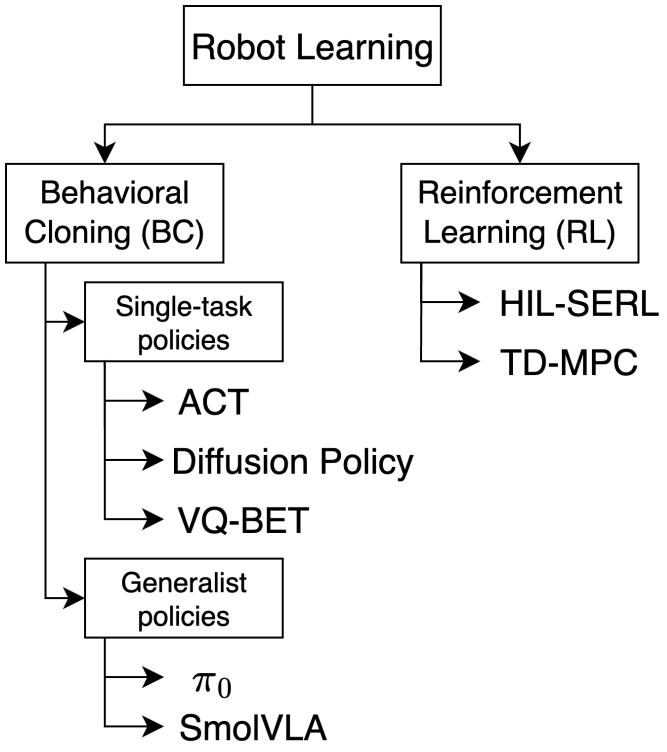


Figure 10 | Overview of the robot learning methods implemented in `lerobot`.

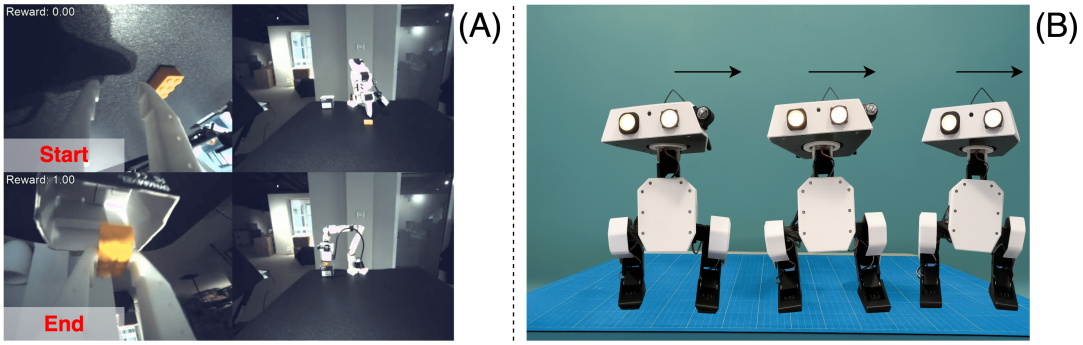


Figure 11 | Examples of two different robotics tasks performed using RL. In the manipulation task (A) an agent learns to reach for a yellow plastic block in its environment, and to put it inside of a box. In the locomotion task (B) an agent learns to move its center of mass sideways without falling.

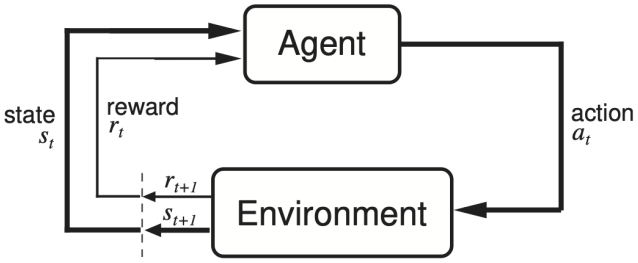


Figure 12 | Agent-Environment interaction diagram (image credits to Sutton and Barto (2018)).

2025), Human-in-the-loop Sample-efficient RL (HIL-SERL) (Luo et al., 2024) and TD-MPC (Hansen et al., 2022).

3.1 RL for Robotics

Applications of RL to robotics have been long studied, to the point the relationship between these two disciplines has been compared to that between physics and matematics (Kober et al.). Indeed, due to their interactive and sequential nature, many robotics problems can be directly mapped to RL problems. Figure 11 depicts two of such cases. Reaching for an object to move somewhere else in the scene is an indeed sequential problem where at each cycle the controller needs to adjust the position of the robotic arm based on their current configuration and the (possibly varying) position of the object. Figure 11 also shows an example of a locomotion problem, where sequentiality is inherent in the problem formulation. While sliding to the side, the controller has to constantly keeps adjusting to the robot's propioperception to avoid failure (falling).

3.1.1 A (Concise) Introduction to RL

The RL framework (Sutton and Barto, 2018), which we briefly introduce here, has often been used to model robotics problems (Kober et al.). RL is a subfield within ML fundamentally concerned with the development of autonomous systems (*agents*) learning how to *continuously behave* in an evolving environment, developing (ideally, well-performing) control strategies (*policies*). Crucially for robotics, RL agents can improve via trial-and-error only, thus entirely bypassing the need to develop explicit models of the problem dynamics, and rather exploiting interaction data only. In RL, this feedback loop (Figure 12) between actions and outcomes is established through the agent sensing a scalar quantity (*reward*).

Formally, interactions between an agent and its environment are typically modeled via a Markov Decision Process (MDP) (Bellman, 1957). Representing robotics problems via MDPs offers several advantages, including (1) incorporating uncertainty through MDP's inherently stochastic formulation and (2) providing a theoretically sound framework for learning *without* an explicit dynamic model. While accommodating also a continuous time formulation, MDPs are typically considered in discrete time in RL, thus assuming interactions to atomically take place over the course of discrete *timestep* $t = 0, 1, 2, 3, \dots, T$. MDPs allowing for an unbounded number of interactions ($T \to +\infty$) are typically termed *infinite-horizon*, and opposed to *finite-horizon* MDPs in which T cannot grow unbounded. Unless

diversely specified, we will only be referring to discrete-time finite-horizon (*episodic*) MDPs here.

Formally, a lenght-T Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{D}, r, \gamma, \rho, T \rangle$, where:

- $\mathcal{S}$ is the *state space*; $s_t \in \mathcal{S}$ denotes the (possibly non-directly observable) environment state at time t. In robotics, states often comprise robot configuration and velocities $(q_t, \dot{q}_t)$, and can accomodate sensor readings such as camera or audio streams.
- $\mathcal{A}$ is the *action space*; $a_t \in \mathcal{A}$ may represent joint torques, joint velocities, or even end-effector commands. In general, actions correspond to commands intervenings on the configuration of the robot.
- $\mathcal{D}$ represents the (possibly non-deterministic) environment dynamics, with $\mathcal{D} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ corresponding to $\mathcal{D}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t)$. For instance, for a planar manipulator dynamics could be considered deterministic when the environment is fully described (Figure 6a), and stochastic when unmodeled disturbances depending on non-observable parameters intervene (Figure 7).
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the *reward function*, weighing the transition (s_t, a_t, s_{t+1}) in the context of the achievement of an arbitrary goal. For instance, a simple reward function for quickly moving the along the x axis in 3D-space (Figure 11) could be based on the absolute position of the robot along the x axis (p_x), present negative penalties for falling over (measured from p_z) and a introduce bonuses $\dot{p}_x$ for speed, $r(s_t, a_t, s_{t+1}) \equiv r(s_t) = p_{x_t} \cdot \dot{p}_{x_t} - \frac{1}{p_{z_t}}$.

Lastly, $\gamma \in [0, 1]$ represent the discount factor regulating preference for immediate versus long-term reward (with an effective horizon equal to $\frac{1}{1-\gamma}$), and ρ is the distribution, defined over $\mathcal{S}$, the MDP's *initial* state is sampled from, $s_0 \sim \rho$.

A length-T *trajectory* is the (random) sequence

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T), \tag{3}$$

with per-step rewards defined as $r_t = r(s_t, a_t, s_{t+1})$ for ease of notation.Interestingly, assuming both the environment dynamics and conditional distribution over actions given states—the *policy*—to be *Markovian*:

$$\mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots s_0, a_0) = \mathbb{P}(s_{t+1}|s_t, a_t) \tag{4}$$

$$\mathbb{P}(a_t|s_t, a_{t-1}, s_{t-1}, s_0, a_0) = \mathbb{P}(a_t|s_t) \tag{5}$$

The probability of observing a given trajectory τ factorizes into

$$\mathbb{P}(\tau) = \mathbb{P}(s_0) \prod_{t=0}^{T-1} \mathbb{P}(s_{t+1}|s_t, a_t)\ \mathbb{P}(a_t|s_t). \tag{6}$$

Policies $\mathbb{P}(a_t|s_t)$ are typically indicated as $\pi(a_t|s_t)$, and often parametrized via θ, yielding $\pi_\theta(a_t|s_t)$. Policies are trained optimizing the (discounted) *return* associated to a given τ, i.e. the (random) sum of measured rewards over trajectory:

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t.$$

In that, agents seek to learn control strategies (*policies*, π_θ) maximizing the expected return $\mathbb{E}_{\tau \sim \pi_\theta} G(\tau)$. For a given dynamics $\mathcal{D}$—i.e., for a given problem—taking the expectation over the (possibly random) trajectories resulting from acting according to a certain policy provides a direct, goal-conditioned ordering in the space of all the possible policies Π, yielding the (maximization) target $J : \Pi \mapsto \mathbb{R}$

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\theta;\mathcal{D}}}[G(\tau)], \tag{7}$$

$$\mathbb{P}_{\theta;\mathcal{D}}(\tau) = \rho \prod_{t=0}^{T-1} \mathcal{D}(s_t, a_t, s_{t+1})\ \pi_\theta(a_t|s_t). \tag{8}$$

Because in the RL framework the agent is assumed to only be able to observe the environment dynamics and not to intervene on them, 7 varies exclusively with the policy followed. In turn, MDPs naturally provide a framework to optimize over the space of the possible behaviors an agent might enact ($\pi \in \Pi$), searching for the *optimal policy* $\pi^* = \arg\max_\theta J(\pi_\theta)$, where θ is the parametrization adopted by the policy set $\Pi : \pi_\theta \in \Pi,\ \forall \theta$. Other than providing

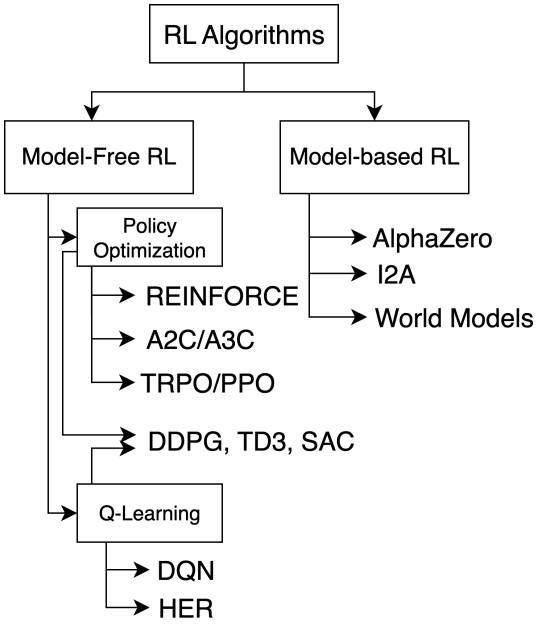


Figure 13 | Popular RL algorithms. See Achiam (2018) for a complete list of citations.

a target for policy search, $G(\tau)$ can also be used as a target to discriminate between states and state-action pairs. Given any state $s \in \mathcal{S}$—e.g., a given configuration of the robot—the *state-value* function

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}\big[G(\tau)\big|s_0 = s\big]$$

can be used to discriminate between desirable and undesirable state in terms of long-term (discounted) reward maximization, under a given policy π. Similarily, the *state-action* value function also conditions the cumulative discounted reward on selecting action a when in s, and thereafter act according to π:

$$Q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}\big[G(\tau)\big|s_0 = s, a_0 = a\big]$$

Crucially, value functions are interrelated:

$$Q_\pi(s_t,a_t) = \mathbb{E}_{s_{t+1}\sim\mathbb{P}(\bullet|s_t,a_t)}[r_t + \gamma V_\pi(s_{t+1})] \tag{9}$$

$$V_\pi(s_t) = \mathbb{E}_{a_t\sim\pi(\bullet|s_t)}[Q_\pi(s_t,a_t)] \tag{10}$$

Inducing an ordering over states and state-action pairs under π, value functions are central to most RL algorithms. A variety of methods have been developed in RL as standalone attemps to find (approximate) solutions to the problem of maximizing cumulative reward (Figure 13).

Popular approaches to continuous state and action space—such as those studied within robotics—include Schulman et al. (2017a,b); Haarnoja et al. (2018). Across manipulation (Akkaya et al., 2019) and locomotion (Lee et al., 2020) problems, RL proved extremely effective in providing a platform to (1) adopt a unified, streamlined perception-to-action pipeline, (2) natively integrate propioperception with multi-modal high-dimensional sensor streams (3) disregard a description of the environment dynamics, by focusing on observed interaction data rather than modeling, and (4) anchor policies in the experience collected and stored in datasets. For a more complete survey of applications of RL to robotics, we refer the reader to Kober et al.; Tang et al. (2024).

3.1.2 Real-world RL for Robotics

Streamlined end-to-end control pipelines, data-driven feature extraction and a disregard for explicit modeling in favor of interaction data are all features of RL for robotics. However, particularly in the context of real-world robotics, RL still suffers from limitations concerning machine safety and learning efficiency.

First, especially early in training, actions are typically explorative, and thus erractic. On physical systems, untrained policies may command high velocities, self-collisiding configurations, or torques exceeding joint limits, leading to wear and potential hardware damage. Mitigating these risks requires external safeguards (e.g., watchdogs, safety

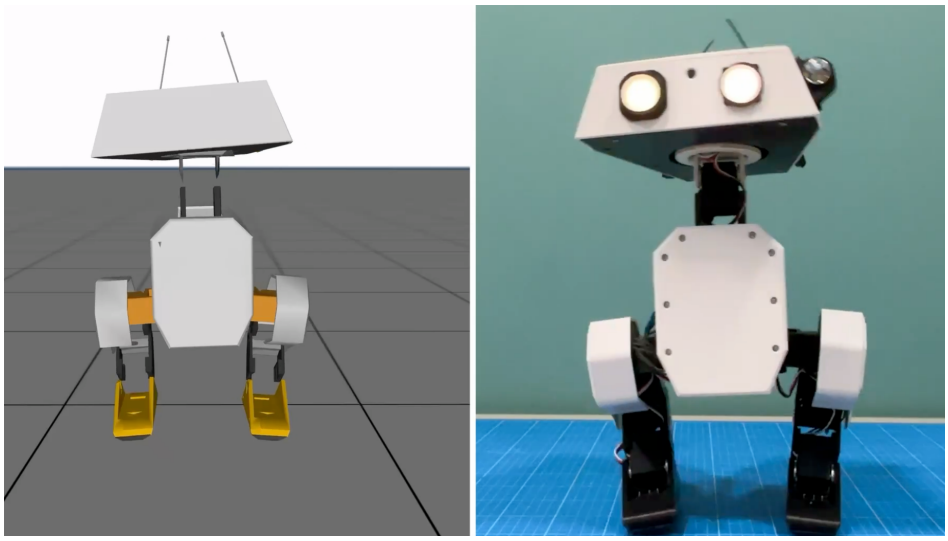


Figure 14 | Simulated (left) vs. real-world (right) OpenDuck. Discrepancies in the simulation dynamics (*reality gap*) pose risks to policy transfer.

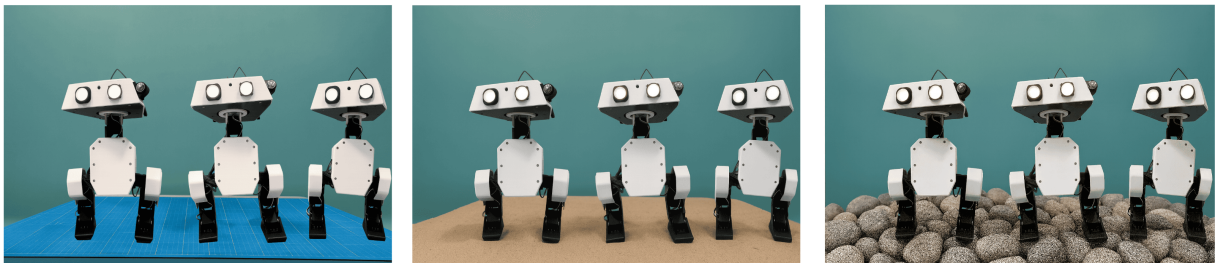


Figure 15 | The same locomotion task can be carried out in different (simulated) domains (exemplified by the difference in terrains) at training time, resulting to increased robustness over diverse environment dynamics.

monitors, emergency stops), often incuring in a high degree of human supervision. Further, in the typical episodic setting considered in most robotics problems, experimentation is substantially slowed down by the need to manually reset the environment over the course of training, a time-consuming and brittle process.

Second, learning with a limited number of samples remains problematic in RL, limiting the applicability of RL in real-world robotics due to consequently prohibitive timescales of training. Even strong algorithms such as SAC (Haarnoja et al., 2018) typically require a large numbers of transitions $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{N}$. On hardware, generating these data is time-consuming and can even be prohibitive.

Training RL policies in simulation (Tobin et al., 2017) addresses both issues: it eliminates physical risk and dramatically increases throughput. Yet, simulators require significant modeling effort, and rely on assumptions (simplified physical modeling, instantaneous actuation, static environmental conditions, etc.) limiting transferring policies learned in simulation due the discrepancy between real and simulated environments (*reality gap*, Figure 14). *Domain randomization* (DR) is a popular technique to overcome the reality gap, consisting in randomizing parameters of the simulated environment during training, to induce robustness to specific disturbances. In turn, DR is employed to increase the diversity of scenarios over the course of training, improving on the chances sim-to-real transfer (Akkaya et al., 2019; Antonova et al., 2017; Ji et al., 2023). In practice, DR is performed further parametrizing the *simulator*'s dynamics $\mathcal{D} \equiv \mathcal{D}_\xi$ with a *dynamics* (random) vector ξ drawn an arbitrary distribution, $\xi \sim \Xi$. Over the course of training—typically at each episode's reset—a new ξ is drawn, and used to specify the environment's dynamics for that episode. For instance, one could decide to randomize the friction coefficient of the surface in a locomotion task (Figure 15), or the center of mass of an object for a manipulation task.

While effective in transfering policies across the reality gap in real-world robotics (Tobin et al., 2017; Akkaya et al., 2019; Ji et al., 2023; Tiboni et al., 2024), DR often requires extensive manual engineering. First, identifying which parameters to randomize—i.e., the *support* $\text{supp}(\Xi)$ of Ξ—is an inherently task specific process. When locomoting over different terrains, choosing to randomize the friction coefficient is a reasonable choice, yet not completely resolutive as

other factors (lightning conditions, external temperature, joints' fatigue, etc.) may prove just as important, making selecting these parameters yet another source of brittlness.

Selecting the dynamics distribution Ξ is also non-trivial. On the one hand, distributions with low entropy might risk to cause failure at transfer time, due to the limited robustness induced over the course of training. On the other hand, excessive randomization may cause over-regularization and hinder performance. Consequently, the research community investigated approaches to automatically select the randomization distribution Ξ, using signals from the training process or tuning it to reproduce observed real-world trajectories. Akkaya et al. (2019) use a parametric uniform distribution $\mathcal{U}(a, b)$ as Ξ, widening the bounds as training progresses and the agent's performance improves (AutoDR). While effective, AutoDR requires significant tuning—the bounds are widened by a fixed, pre-specified amount Δ—and may disregard data when performance *does not* improve after a distribution update (Tiboni et al., 2024). Tiboni et al. (2024) propose a similar method to AutoDR (DORAEMON) to evolve Ξ based on training signal, but with the key difference of explicitly maximizing the entropy of parametric Beta distributions, inherently more flexible than uniform distributions. DORAEMON proves particularly effective at dynamically increasing the entropy levels of the training distribution by employing a max-entropy objective, under performance constraints formulation. Other approaches to automatic DR consist in specifically tuning Ξ to align as much as possible the simulation and real-world domains. For instance, Chebotar et al. (2019) interleave in-simulation policy training with repeated real-world policy rollouts used to adjust Ξ based on real-world data, while Tiboni et al. (2023) leverage a single, pre-collected set of real-world trajectories and tune Ξ under a simple likelihood objective.

While DR has shown promise, it does not address the main limitation that, even under the assumption that an ideal distribution Ξ to sample from was indeed available, many robotics problems cannot be simulated with high-enough fidelity under practical computational constraints in the first place. Simulating contact-rich manipulation of possibly deformable or soft materials—i.e., *folding a piece of clothing*—can be costly and even time-intensive, limiting the benefits of in-simulation training.

A perhaps more foundamental limitation of RL for robotics is the general unavailability of complicated tasks' *dense* reward function, the design of which is essentially based on human expertise and trial-and-error. In practice, *sparse* reward functions can be used to conclude whether one specific goal has been attained—*has this t-shirt been correctly folded?*—but unfortunately incur in more challenging learning. As a result, despite notable successes, deploying RL directly on real-world robots at scale remains challenging.

3.1.3 RL in `lerobot`: sample-efficient, data-driven, and real-world

To make the most of (1) the growing number of openly available datasets and (2) relatively inexpensive robots like the SO-100, RL could (1) be anchored in already-collected trajectories—limiting erratic and dangerous exploration—and (2) train in the real-world directly—bypassing the aforementioned issues with low-fidelity simulations. In such a context, sample-efficient learning is also paramount, as training on the real-world is inherently time-bottlenecked.

Off-policy algorithms like Soft Actor-Critic (SAC) (Haarnoja et al., 2018) tend to be more sample efficient then their on-policy counterpart (Schulman et al., 2017b), due to the presence a *replay buffer* used over the course of the training. Other than allowing to re-use transitions (s_t, a_t, r_t, s_{t+1}) over the course of training, the replay buffer can also accomodate for the injection of previously-collected data in the training process (Ball et al., 2023). Using expert demonstrations to guide learning together with learned rewards, RL training can effectively be carried out in the real-world (Luo et al., 2025). Interestingly, when completed with in-training human interventions, real-world RL agents have been shown to learn policies with near-perfect success rates on challenging manipulation tasks in 1-2 hours (Luo et al., 2024).

Sample-efficient RL In an MDP, the optimal policy π^* can be derived from its associated Q-function, Q_{π^*}, and in particular the optimal action(s) $\mu(s_t)$ can be selected maximizing the optimal Q-function over the action space,

$$\mu(s_t) = \max_{a_t \in \mathcal{A}} Q_{\pi^*}(s_t, a_t).$$

Interestingly, the Q^*-function satisfies a recursive relationship (*Bellman equation*) based on a very natural intuition[1]:

> [...] If the optimal value $Q^*(s_{t+1}, a_{t+1})$ of the [state] s_{t+1} was known for all possible actions a_{t+1}, then the optimal strategy is to select the action a_{t+1} maximizing the expected value of $r_t + \gamma Q^*(s_{t+1}, a_{t+1})$

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet | s_t, a_t)} \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^*(s_{t+1}, a_{t+1}) \middle| s_t, a_t \right]$$

[1] Quote from Mnih et al. (2013). The notation used has slightly been adapted for consistency with the rest of this tutorial.

In turn, the optimal Q-function is guaranteed to be self-consistent by definition. *Value-iteration* methods exploit this relationship (and/or its state-value counterpart, $V^*(s_t)$) by iteratively updating an initial estimate of Q^*, Q_k using the Bellman equation as update rule (*Q-learning*):

$$Q_{i+1}(s_t, a_t) \leftarrow \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_i(s_{t+1}, a_{t+1}) \middle| s_t, a_t \right], \quad i = 0, 1, 2, \dots, K$$

Then, one can derive the (ideally, near-optimal) policy by explicitly maximizing over the action space the final (ideally, near-optimal) estimate $Q_K \approx Q^*$ at each timestep. In fact, under certain assumptions on the MDP considered, $Q_K \to Q^*$ as $K \to \infty$.

Effective in its early applications to small-scale discrete problems and theoretically sound, vanilla Q-learning was found complicated to scale to large $\mathcal{S} \times \mathcal{A}$ problems, in which the storing of $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ alone might result prohibitive. Also, vanilla Q-learning is not directly usable for *continuous*, unstructured state-action space MPDs, such as those considered in robotics. In their seminal work on *Deep Q-Learning* (DQN), Mnih et al. (2013) propose learning Q-values using deep convolutional neural networks, thereby accomodating for large and even unstructured *state* spaces. DQN parametrizes the Q-function using a neural network with parameters θ, updating the parameters by sequentially minimizing the expected squared temporal-difference error (TD-error, δ_i):

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim \chi(\bullet)} \big[\underbrace{(y_i - Q_{\theta_i}(s_t, a_t))}_{\delta_i}{}^2 \big], \tag{11}$$

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \big[r_t + \gamma \max_{a_t \in \mathcal{A}} Q_{\theta_{i-1}}(s_{t+1}, a_{t+1}) \big], \tag{12}$$

Where χ represents a behavior distribution over state-action pairs. Crucially, χ can in principle be different from the policy being followed, effectively allowing to reuse prior data stored in a *replay buffer* in the form of (s_t, a_t, r_t, s_{t+1}) transitions, used to form the TD-target y_i, TD-error δ_i and loss function 11 via Monte-Carlo (MC) estimates.

While effective in handling large, unstructured state spaces for discrete action-space problems, DQN application's to continous control problems proved challenging. Indeed, in the case of high-capacity function approximators such as neural networks, solving $\max_{a_t \in \mathcal{A}} Q_\theta(s_t, a_t)$ at each timestep is simply unfeasible due to the (1) continous nature of the action space ($\mathcal{A} \subset \mathbb{R}^n$ for some n) and (2) impossibility to express the find a cheap (ideally, closed-form) solution to Q_θ. Silver et al. (2014) tackle this fundamental challenge by using a *deterministic* function of the state s_t as policy, $\mu_\phi(s_t) = a_t$, parametrized by ϕ. Thus, policies can be iteratively refined updating ϕ along the direction:

$$d_\phi = \mathbb{E}_{s_t \sim \mathbb{P}(\bullet)} \big[\nabla_\phi Q(s_t, a_t)|_{a_t = \mu_\phi(s_t)} \big] = \mathbb{E}_{s_t \sim \mathbb{P}(\bullet)} \big[\nabla_{a_t} Q(s_t, a_t)|_{a_t = \mu_\phi(s_t)} \cdot \nabla_\phi \mu(s_t) \big] \tag{13}$$

Provably, 13 is the *deterministic policy gradient* (DPG) of the policy μ_ϕ (Silver et al., 2014), so that updates $\phi_{k+1} \leftarrow \phi_k + \alpha d_\phi$ are guaranteed to increase the (deterministic) cumulative discounted reward, $J(\mu_\phi)$. Lillicrap et al. (2019) extended DPG to the case of (1) high-dimensional unstructured observations and (2) continuous action spaces, introducing Deep Deterministic Policy Gradient (DDPG), an important algorithm RL and its applications to robotics. DDPG adopts a modified TD-target compared to the one defined in 12, by maintaining a policy network used to select actions, yielding

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \big[r_t + \gamma Q_{\theta_{i-1}}(s_{t+1}, \mu_\phi(s_{t+1})) \big]. \tag{14}$$

Similarily to DQN, DDPG also employs the same replay buffer mechanism, to reuse past transitions over training for increased sample efficiency and estimate the loss function via MC-estimates.

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is a derivation of DDPG in the max-entropy (MaxEnt) RL framework, in which RL agents are tasked with maximizing the discounted cumulative reward, while acting as randomly as possible. MaxEnt RL (Haarnoja et al., 2017) has proven particularly robust thanks to the development of diverse behaviors, incentivized by its entropy-regularization formulation. In that, MaxEnt revisits the RL objective $J(\pi)$ to specifically account for the policy entropy,

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \chi} [r_t + \alpha \mathcal{H}(\pi(\bullet|s_t))] \tag{15}$$

This modified objective results in the *soft* TD-target:

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \big[r_t + \gamma \big(Q_{\theta_{i-1}}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\phi(a_{t+1}|s_{t+1}) \big) \big], \quad a_{t+1} \sim \pi_\phi(\bullet|s_t) \tag{16}$$

Similarily to DDPG, SAC also maintains an explicit policy, trained under the same MaxEnt framework for the maximization of 15, and updated using:

$$\pi_{k+1} \leftarrow \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\bullet|s_t) \middle\| \frac{\exp(Q_{\pi_k}(s_t, \bullet))}{Z_{\pi_k}(s_t)}\right) \tag{17}$$

The update rule provided in 17 optimizes the policy while projecting it on a set Π of tractable distributions (e.g., Gaussians, Haarnoja et al. (2017)).

Sample-efficient, data-driven RL Importantly, sampling (s_t, a_t, r_t, s_{t+1}) from the replay buffer D conveniently allows to approximate the previously introduced expectations for TD-target and TD-error through Monte-Carlo (MC) estimates. The replay buffer D also proves extremely useful in maintaining a history of previous transitions and using it for training, improving on sample efficiency. Furthermore, it also naturally provides an entry point to inject offline trajectories recorded, for instance, by a human demonstrator, into the training process.

Reinforcement Learning with Prior Data (RLPD) (Ball et al., 2023) is an Offline-to-Online RL algorithm leveraging prior data to effectively accelerate the training of a SAC agent. Unlike previous works on Offline-to-Online RL, RLPD avoids any pre-training and instead uses the available offline data D_{offline} to improve online-learning from scratch. During each training step, transitions from both the offline and online replay buffers are sampled in equal proportion, and used in the underlying SAC routine.

Sample-efficient, data-driven, real-world RL Despite the possibility to leverage offline data for learning, the effectiveness of real-world RL training is still limited by the need to define a task-specific, hard-to-define reward function. Further, even assuming to have access to a well-defined reward function, typical robotics pipelines rely mostly on propioperceptive inputs augmented by camera streams of the environment. As such, even well-defined rewards would need to be derived from processed representations of unstructured observations, introducing brittleness. In their technical report, Luo et al. (2025) empirically address the needs (1) to define a reward function and (2) to use it on image observations, by introducing a series of tools to allow for streamlined training of *reward classifiers* c, as well as jointly learn forward-backward controllers to speed up real-world RL. Reward classifiers are particularly useful in treating complex tasks—e.g., folding a t-shirt—for which a precise reward formulation is arbitrarily complex to obtain, or that do require significant shaping and are more easily learned directly from demonstrations of success (e^+) or failure (e^-) states, $s \in \mathcal{S}$, with a natural choice for the state-conditioned reward function being $r\mathcal{S} \mapsto \mathbb{R}$ being $r(s) = \log c(e^+ \; verts)$. Further, Luo et al. (2025) demonstrate the benefits of learning *forward* (executing the task from initial state to completion) and *backward* (resetting the environment to the initial state from completion) controllers, parametrized by separate policies.

Lastly, in order to improve on the robustness of their approach to different goals while maintaing practical scalability, Luo et al. (2025) introduced a modified state and action space, expressing proprioperceptive configurations q and actions $\dot{q}$ in the frame of end-effector pose at $t = 0$. Randomizing the initial pose of the end-effector (s_0),Luo et al. (2025) achieved a similar result to that of having to manually randomize the environment at every timestep, but with the benefit of maintaining the environment in the same condition across multiple training episodes, achieving higher scalability of their method thanks to the increased practicality of their approach.

Building on off-policy deep Q-learning with replay buffers, entropy regularization for better exploration and performance, expert demonstrations to guide learning, and a series of tools and recommendations for real-world training using reward classifiers (Figure 16), Luo et al. (2024) introduce human interactions during training, learning near-optimal policies in challenging real-world manipulation tasks in 1-2 hours.

Human in the Loop Sample Efficient Robot reinforcement Learning (HIL-SERL) (Luo et al., 2024) augments offline-to-online RL with targeted human corrections during training, and employs prior data to (1) train a reward classifier and (2) bootstrap RL training on expert trajectories. While demonstrations provide the initial dataset seeding learning and constraining early exploration, interactive corrections allow a human supervisor to intervene on failure modes and supply targeted interventions to aid the learning process. Crucially, human interventions are stored in both the offline and online replay buffers, differently from the autonomous transitions generated at training time and stored in the online buffer only. Consequently, given an intervention timestep $k \in (0, T)$, length-K human intervention data $\{s_k^{\text{human}}, a_k^{\text{human}}, r_k^{\text{human}}, s_{k+1}^{\text{human}}, \}_{k=1}^{K}$ is more likely to be sampled for off-policy learning than the data generated online during training, providing stronger supervision to the agent while still allowing for autonomous learning. Empirically, HIL-SERL attains near-perfect success rates on diverse manipulation tasks within 1-2 hours of training (Luo et al., 2024), underscoring how offline datasets with online RL can markedly improve stability and data efficiency, and ultimately even allow real-world RL-training.

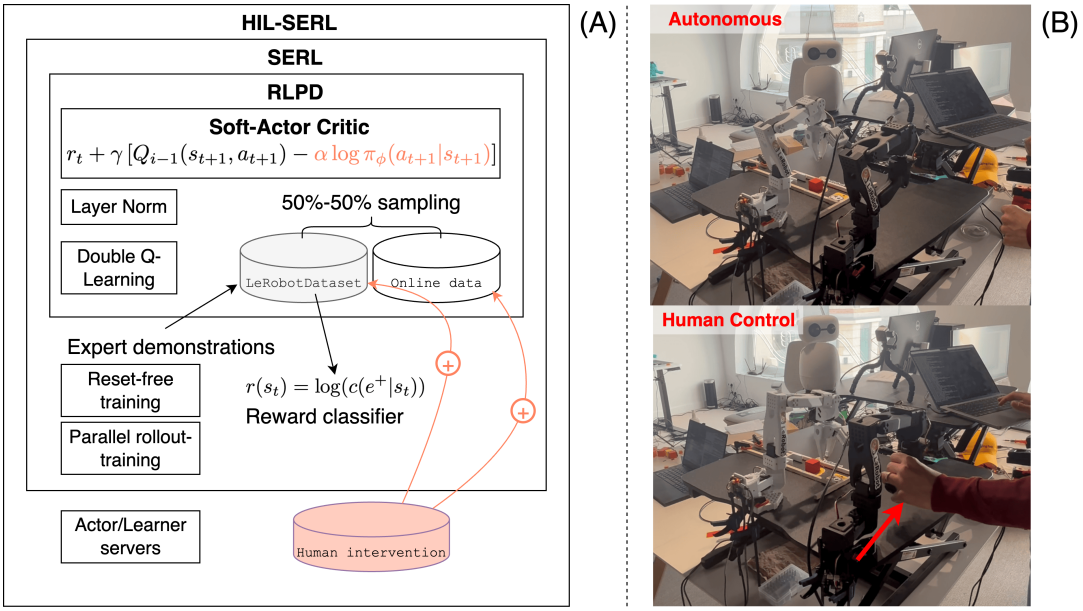


Figure 16 | (A) HIL-SERL allows for real-world training of high performance RL agents by building on top advancements presented by of SAC, RLPD and SERL. (B) Example of human intervention during a HIL-SERL training process on a SO-100.

3.1.4 Code Example: Real-world RL

TODO(fracapuano): work out rl training example

3.1.5 Limitations of RL in Real-World Robotics: Simulators and Reward Design

Despite the advancements in real-world RL training, solving robotics training RL agents in the real world still suffers from the following limitations:

- In those instances where real-world training experience is prohibitively expensive to gather (Degrave et al., 2022; Bellemare et al., 2020), in-simulation training is often the only option. However, high-fidelity simulators for real-world problems can be difficult to build and maintain, especially for contact-rich manipulation and tasks involving deformable or soft materials.
- Reward design poses an additional source of brittleness. Dense shaping terms are often required to guide exploration in long-horizon problems, but poorly tuned terms can lead to specification gaming or local optima. Sparse rewards avoid shaping but exacerbate credit assignment and slow down learning. In practice, complex behaviors require efforts shaping rewards: a britlle and error prone process.

Advances in Behavioral Cloning (BC) from corpora of human demonstrations address both of these concerns. By learning in a supervised fashion to reproduce expert demonstrations, BC methods prove competitive while bypassing the need for simulated environments and hard-to-define reward functions.

4 Robot (Imitation) Learning

The best material model for a cat is another, or preferably the same cat

Norbert Wiener

TL;DR

Behavioral Cloning provides a natural platform to learn from real-world interactions without the need to design any reward function, and generative models prove more effective than point-wise policies at dealing with

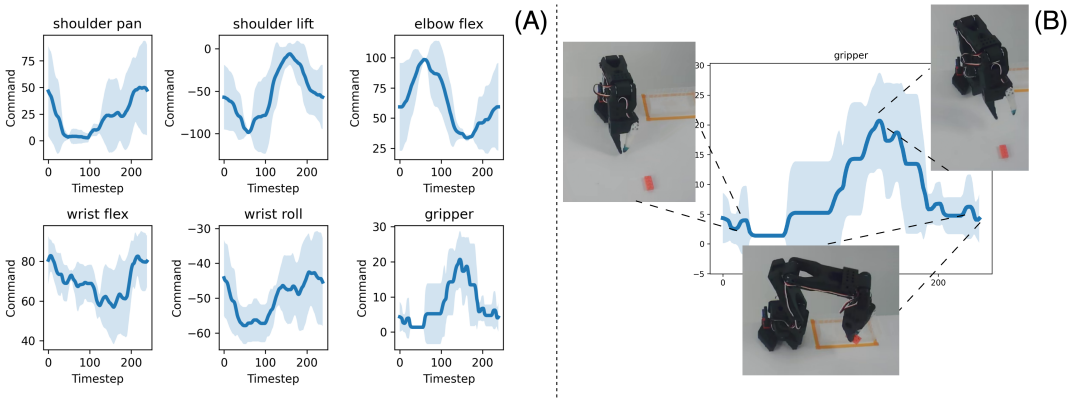


Figure 17 | (A) Average (with standard deviation) evolution of the actuation levels over the first 5 recorded episodes in `lerobot/svla_so101_pickplace`. Proprioperceptive state provide invaluable to determine the robot's state during an episode. (B) Camera frames are also recorded alongside measurements on the robot's state, capturing information about the robot's interaction with its environment.

multimodal demonstration datasets.

Learning from human demonstrations provides a pragmatic alternative to the reinforcement-learning pipeline discussed in Section 3. Indeed, in real-world robotics online exploration is typically costly and potentially unsafe, and designing (dense) reward signals is a brittle and task-specific process. In general, success detection itself may often require bespoke instrumentation, while episodic training demands reliable resets—all factors complicating training RL algorithms on hardware at scale. Behavioral Cloning (BC) sidesteps these constraints by casting control an imitation learning problem, leveraging previously collected expert demonstrations. Most notably, by learning to imitate autonomous systems naturally adhere to the objectives, preferences, and success criteria implicitly encoded in the data, which obviates reduces early-stage exploratory failures and obviates hand-crafted reward shaping altogether.

Formally, let $\mathcal{D} = \{\tau^{(i)}\}_{i=1}^{N}$ be a set of expert trajectories, with $\tau^{(i)} = \{(o_t^{(i)}, a_t^{(i)})\}_{t=0}^{T_i}$ representing the i-th trajectory in $\mathcal{D}$, $o_t \in \mathcal{O}$ denoting observations (e.g., images and proprioception altogether), and $a_t \in \mathcal{A}$ the expert actions. Typically, observations $o \in \mathcal{O}$ consist of both image and proprioperceptive information, while actions $a \in \mathcal{A}$ represent control specifications for the robot to execute, e.g. a joint configuration. Note that differently from Section 3, in the imitation learning context $\mathcal{D}$ denotes an offline dataset collecting N length-T_i reward-free (expert) human trajectories $\tau^{(i)}$, and *not* the environment dynamics. Similarily, in this section $\tau^{(i)}$ represent a length-T_i trajectory of observation-action pairs, which crucially *omits entirely any reward* information. Figure 17 graphically shows trajectories in terms of the average evolution of the actuation on the 6 joints over a group of teleoperated episodes for the SO-100 manipulator. Notice how proprioperceptive states are captured jointly with camera frames over the course of the recorded episodes, providing a unified high-frame rate collection of teleoperation data. Figure 18 shows (o_t, a_t)-pairs for the same dataset, with the actions performed by the human expert illustrated just alongside the corresponding observation. In principle, (expert) trajectories $\tau^{(i)}$ can have different lengths since demonstrations might exhibit multi-modal strategies to attain the same goal, resulting in possibly multiple, different behaviors.

Behavioral Cloning (BC) (Pomerleau, 1988) aims at synthetizing synthetic behaviors by learning the mapping from observations to actions, and in its most natural formulation can be effectively tackled as a *supevised* learning problem, consisting of learning the (deterministic) mapping $f : \mathcal{O} \mapsto \mathcal{A},\ a_t = f(o_t)$ by solving

$$\min_f \mathbb{E}_{(o_t,a_t)\sim p(\bullet)} \mathcal{L}(a_t, f(o_t)), \tag{18}$$

for a given risk function $\mathcal{L} : \mathcal{A} \times \mathcal{A} \mapsto \mathbb{R},\ \mathcal{L}(a, a')$.

Typically, the expert's joint observation-action distribution $p : \mathcal{O} \times \mathcal{A} \mapsto [0, 1]$ such that $(o, a) \sim p(\bullet)$ is assumed to be unknown, in keeping with a classic Supervised Learning (SL) framework[2]. However, differently from standard SL's assumptions, the samples collected in $\mathcal{D}$, correspoding to observations of the underlying p are *not* i.i.d., as expert demonstrations are collected *sequentially* in trajectories. In practice, this aspect can be partially mitigated by considering pairs in a non-sequential order—*shuffling* the samples in $\mathcal{D}$—so that the expected risk under p can be

[2]Throughout, we will adopt the terminology and notation for SL introduced in Shalev-Shwartz and Ben-David (2014)

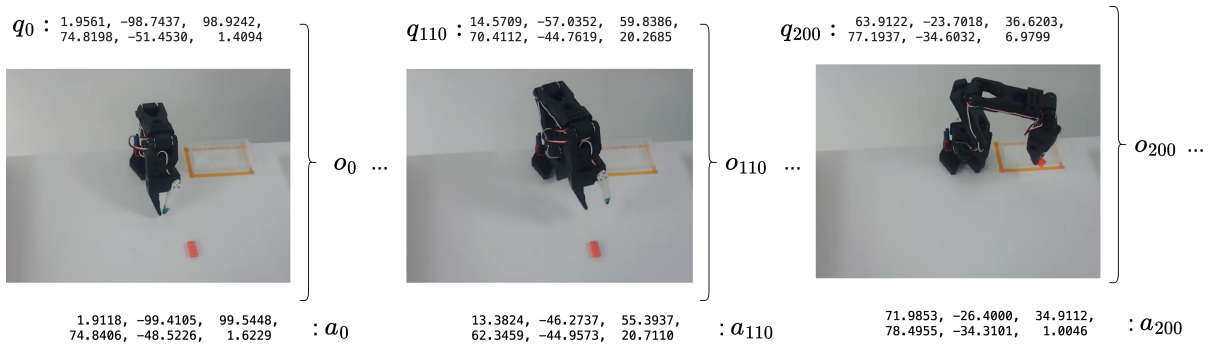


Figure 18 | Sample observations and action pairs over the course of a given trajectory recorded in `lerobot/svla_so101_pickplace`. Observations, comprising of both proprioperceptive and visual information, are recorded alongside the configuration of a second, leader robot controlled by a human expert, providing complete information for regressing actions given observation.

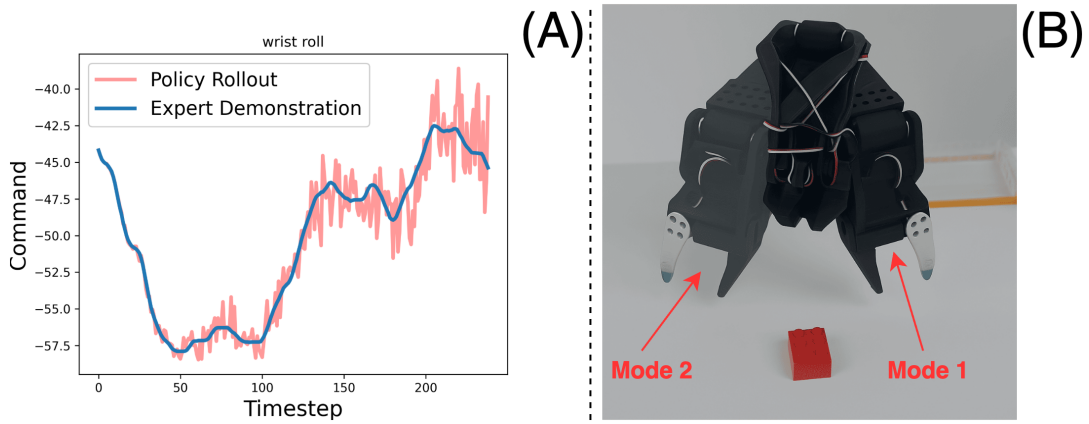


Figure 19 | Point-wise policies suffer from limitations due to (A) covariate shifts and poor approximation of (B) multimodal demonstrations. (A) Initially small errors may drive the policy out of distribution, incuring in a vicious circle ultimately resulting in failure. (B) Both modes of reaching for a target object in a scene, either left or right-first, are equally as good and thus equally as likely to be present in a dataset of human demonstrations, ultimately resulting in multimodal demonstrations.

approximated using MC estimates, although estimates may in general be less accurate. Another strategy to mitigate the impact of regressing over non-i.i.d. samples relies on the possibility of interleaving BC and data collection (Ross et al., 2011), aggregating multiple datasets iteratively. However, because we only consider the case where a single offline dataset $\mathcal{D}$ of (expert) trajectories is already available, dataset aggregation falls out of scope.

Despite the inherent challenges of learning on non-i.i.d. data, the BC formulation affords several operational advantages in robotics. First, training happens offline and typically uses expert human demonstration data, hereby severily limiting exploration risks by preventing the robot from performing dangerous actions altogether. Second, reward design is entirely unnecessary in BC, as demonstrations already reflect human intent and task completion. This also mitigates the risk of misalignment and specification gaming (*reward hacking*), otherwise inherent in purely reward-based RL (Heess et al., 2017). Third, because expert trajectories encode terminal conditions, success detection and resets are implicit in the dataset. Finally, BC scales naturally with growing corpora of demonstrations collected across tasks, embodiments, and environments. However, BC can in principle only learn behaviors that are, at most, as good as the one exhibited by the demonstrator, and thus critically provides no mitigation for the suboptimal decision making that might be enaced by humans. Still, while problematic in sequential-decision making problems for which expert demonstrations are not generally available—data migth be expensive to collect, or human performance may be inherently suboptimal—many robotics applications benefit from relative cheap pipelines to acquire high-quality trajectories generated by humans, thus justifying BC approaches.

While conceptually elegant, point-estimate policies $f : \mathcal{O} \mapsto \mathcal{A}$ learned by solving 18 have been observed to suffer from (1) compounding errors (Ross et al., 2011) and (2) poor fit to multimodal distributions (Florence et al., 2022; Ke et al., 2020). Figure 19 illustrates these two key issues related to learning *explicit policies* (Florence et al., 2022). Besides sequentiality in $\mathcal{D}$, compounding errors due to *covariate shift* may also prove catastrophic, as even small

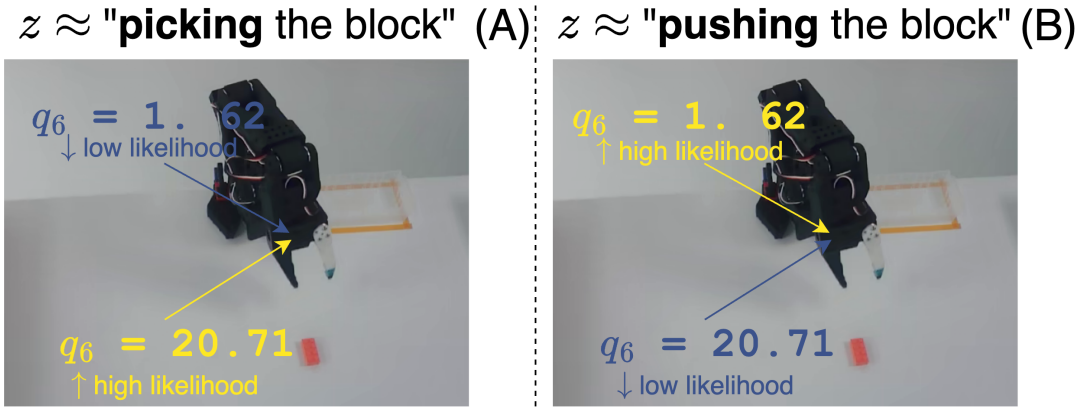


Figure 20 | Intuitively, latent variable in a single latent model may contain information regarding the task being performed, which directly results in the likelihood of the same observation-action pair being different for two different tasks. When (A) picking a block the likelihood of a wide gripper's opening should be higher than narrower one, while it should be the opposite when (B) pushing the block.

ϵ-prediction errors $0 < \|\mu(o_t) - a_t\| \leq \epsilon$ can quickly drive the policy into out-of-distribution states, incuring in less confident generations and thus errors compounding (Figure 19, left).Moreover, point-estimate policies typically fail to learn *multimodal* targets, which are very common in human demonstrations solving robotics problems, since multiple trajectories can be equally as good towards the accomplishment of a goal (e.g., symmetric grasps, Figure 19, right). In particular, unimodal regressors tend to average across modes, yielding indecisive or even unsafe commands (Florence et al., 2022). To address poor multimodal fitting, Florence et al. (2022) propose learning the generative model $p(o, a)$ underlying the samples in $\mathcal{D}$, rather than an explicitly learning a prediction function $f(o) = a$.

4.1 A (Concise) Introduction to Generative Models

Generative Models (GMs) aim to learn the stochastic process underlying the very generation of the data collected, and typically do so by fitting a probability distribution that approximates the unknown *data distribution*, p. In the case of BC, this unknown data distribution p represents the expert's joint distribution over (o, a)-pairs. Thus, given a finite set of N pairs $\mathcal{D} = \{(o, a)_i\}_{i=0}^{N}$ used as an imitation learning target (and thus assumed to be i.i.d.), GM seeks to learn a *parametric* distribution $p_\theta(o, a)$ such that (1) new samples $(o, a) \sim p_\theta(\bullet)$ resemble those stored in $\mathcal{D}$, and (2) high likelihood is assigned to the observed regions of the unobservable p. Likelihood-based learning provides a principled training objective to achieve both objectives, and it is thus extensively used in GM (Prince, 2023).

Variational Auto-Encoders A common inductive bias used in GM posits samples (o, a) are influenced from an unobservable latent variable $z \in Z$, resulting in

$$p(o, a) = \int_{\text{supp}(Z)} p(o, a|z)p(z) \tag{19}$$

Intuitively, in the case of observation-action pairs (o, a) for a robotics application, z could be some high level representation of the underlying task being performed by the human demonstrator. In such case, treating $p(o, a)$ as a marginalization over $\text{supp}(Z)$ of the complete joint distribution $p(o, a, z)$ natively captures the effect different tasks have on the likelihood of observation-action pairs. Figure 20 graphically illustrates this concept in the case of a (A) picking and (B) pushing task, for which, nearing the target object, the likelihood of actions resulting in opening the gripper—the higher q_6, the wider the gripper's opening—should intuitively be (A) high or (B) low, depending on the task performed. While the latent space Z typically has a much richer structure than the set of all actual tasks performed, 19 still provides a solid framework to learn joint distribution conditioned on unobservable yet relevant factors. Figure 21 represents this framework of latent-variable for a robotics application: the true, z-conditioned generative process on assigns *likelihood* $p((o, a)|z)$ to the single (o, a)-pair. Using Bayes' theorem, one can reconstruct the *posterior* distribution on $\text{supp}(Z)$, $q_\theta(z|o, a)$ from the likelihood $p_\theta(o, a|z)$, *prior* $p_\theta(z)$ and *evidence* $p_\theta(o, a)$. VAEs approximate the latent variable model presented in 19) using an *approximate posterior* $q_\phi(z|o, a)$ while regressing parameters for a parametric likelihood, $p_\theta(o, a|z)$ (Figure 21).

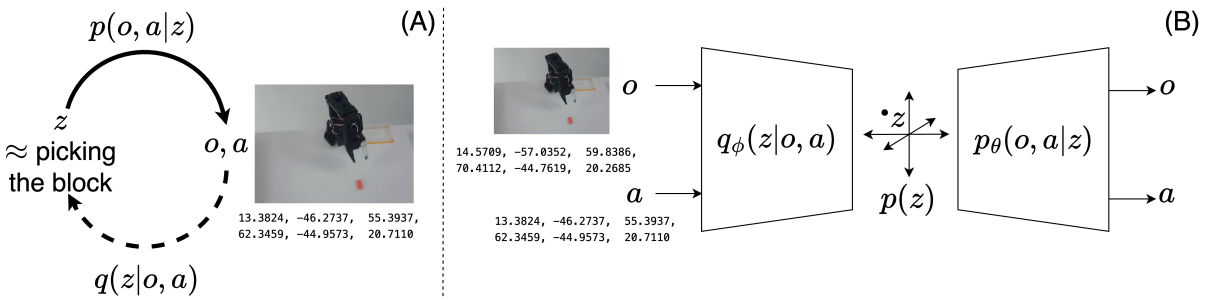


Figure 21 | (A) The latent variable model in a robotics application regulates influence between observed (o, a) variables and an unobservable latent variable. (B) VAEs approximate exact latent variable models by means of variational inference.

Given a dataset $\mathcal{D}$ consisting of N i.i.d. observation-action pairs, the log-likelihood of all datapoints under θ (in Bayesian terms, the *evidence* $p_\theta(\mathcal{D})$) can thus be written as:

$$\log p_\theta(\mathcal{D}) = \log \sum_{i=0}^{N} p_\theta((o, a)_i) \tag{20}$$

$$= \log \sum_{i=0}^{N} \int_{\text{supp}(Z)} p_\theta((o, a)_i|z)p(z) \tag{21}$$

$$= \log \sum_{i=0}^{N} \int_{\text{supp}(Z)} \frac{q_\theta(z|(o, a)_i)}{q_\theta(z|(o, a)_i)} \cdot p_\theta((o, a)_i|z)p(z) \tag{22}$$

$$= \log \sum_{i=0}^{N} \mathbb{E}_{z\sim p_\theta(\bullet|(o,a)_i)} \left[\frac{p(z)}{q_\theta(z|(o, a)_i)} \cdot p_\theta((o, a)_i|z) \right], \tag{23}$$

where we used 19 in 20, multiplied by $1 = \frac{q_\theta(z|(o,a)_i)}{q_\theta(z|(o,a)_i)}$ in 21, and used the definition of expected value in 23.

In the special case where one assumes distributions to be tractable, $p_\theta(\mathcal{D})$ is typically tractable too, and $\max_\theta \log p_\theta(\mathcal{D})$ provides a natural target for (point-wise) infering the unknown parameters θ of the generative model. Unfortunately, 23 is rarely tractable when the distribution p is modeled with approximators such as neural networks, especially for high-dimensional, unstructured data.

In their seminal work on Variational Auto-Encoders (VAEs), Kingma and Welling (2022) present two major contributions to learn complex latent-variable GMs on unstructured data, proposing (1) a tractable, variational lower-bound to 23 as an optimization target to jointly learn likelihood and posterior and (2) high-capacity function approximators to model the likelihood $p_\theta(o, a|z)$ and (approximate) posterior distribution $q_\phi(z|o, a) \approx q_\theta(z|o, a)$.

In particular, the lower bound on 23 (Evidence LOwer Bound, *ELBO*) can be derived from 23 applying Jensen's inequality—$\log \mathbb{E}[\bullet] \geq \mathbb{E}[\log(\bullet)]$—yielding:

$$\log p_\theta(\mathcal{D}) \geq \sum_{i=0}^{N} \left(\mathbb{E}_{z\sim p_\theta(\cdot|(o,a)_i)} \left[\log p_\theta((o, a)_i|z) \right] + \mathbb{E}_{z\sim p_\theta(\cdot|(o,a)_i)} \left[\log \left(\frac{p(z)}{q_\theta(z|(o, a)_i)} \right) \right] \right) \tag{24}$$

$$= \sum_{i=0}^{N} \left(\mathbb{E}_{z\sim p_\theta(\cdot|(o,a)_i)} \left[\log p_\theta((o, a)_i|z) \right] - \mathrm{D}_{\mathrm{KL}} \left[q_\theta(z|(o, a)_i) \| p(z) \right] \right) \tag{25}$$

The true, generally intractable posterior $p_\theta(z|o, a)$ prevents computing both the expectation and KL divergence terms in 25, and therefore Kingma and Welling (2022) propose deriving the ELBO using an *approximate* posterior $q_\phi(z|o, a)$, resulting in the final, tractable ELBO objective,

$$\text{ELBO}_{\mathcal{D}}(\theta, \phi) = \sum_{i=0}^{N} \left(\mathbb{E}_{z\sim q_\phi(\cdot|(o,a)_i)} \left[\log p_\theta((o, a)_i|z) \right] - \mathrm{D}_{\mathrm{KL}} \left[q_\phi(z|(o, a)_i) \| p(z) \right] \right) \tag{26}$$

From Jensen's inequality, maximizing ELBO results in maximizing the log-likelihood of the data too, thus providing a natural, tractable optimization target. Indeed, expectations can be estimated using MC estimates from the

learned distributions in 26, while the KL-divergence term can typically be computed in closed-form (1) modeling q_ϕ as a Gaussian $q_\phi(z|o,a) = \mathcal{N}\big(\mu_\phi(o,a), \Sigma_\phi(o,a)\big)$ and (2) imposing a standard Gaussian prior on the latent space, $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

An intuitive explanation of the learning dynamics of VAEs can be given considering the equivalent case of *minimizing the negative ELBO*, which admits a particularly interpretable factorization

$$\min_{\theta,\phi} -\text{ELBO}_{(o,a)\sim\mathcal{D}}(\theta,\phi) = \min_{\theta,\phi} \mathbf{L}^{\text{rec}}(\theta) + \mathbf{L}^{\text{reg}}(\phi) \tag{27}$$

$$\mathbf{L}^{\text{rec}}(\theta) = \mathbb{E}_{z\sim q_\phi(\cdot|o,a)}\big[\log p_\theta(o,a|z)\big] \tag{28}$$

$$\mathbf{L}^{\text{reg}}(\phi) = \text{D}_{\text{KL}}\big[q_\phi(z|o,a) \| p(z)\big] \tag{29}$$

For any given (o,a) pair, the expected value term of 28 is typically computed via MC estimates, resulting in

$$-\mathbb{E}_{z\sim q_\phi(\bullet|o,a)}\big[\log p_\theta(o,a|z)\big] = \mathbf{L}^{\text{rec}} \approx -\frac{1}{n}\sum_{i=0}^{n} \log p_\theta(o,a|z_i).$$

Assuming $p_\theta(o,a|z)$ is parametrized as an isotropic Gaussian distribution with mean $\mu_\theta(z) \in \mathbb{R}^d$ and variance σ^2, the log-likelihood thus simplifies to:

$$\log p(o,a|z_i) = -\frac{1}{2\sigma^2}\big\|(o,a) - \mu_\theta(z_i)\big\|_2^2 - \frac{d}{2}\log(2\pi\sigma^2) \implies \mathbf{L}^{\text{rec}} \approx \frac{1}{n}\sum_{i=0}^{n}\big\|(o,a) - \mu_\theta(z_i)\big\|_2^2$$

Indeed, it is very common in practice to approximate from the learned likelihood $p_\theta(o,a|z)$ as a parametric distribution (e.g. Gaussians) parametrized by some learned vector of coefficients derived from $\mu_\theta(z),\ z \sim p(\bullet)$. In all such cases, learning a VAE corresponds to optimally *reconstructing* the examples in $\mathcal{D}$ by minimizing the L2-error—a very common *supervised learning* objective for regression targets—while regularizing the information compression into the latent, as under the common modeling choice $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ 29 regularizes the posterior limiting the expressivity of $q_\phi(z|o,a)$.

Diffusion Models VAEs approximate probability distributions via a *single* latent variable model, assuming the underlying unknown distribution can be factored according to 19, and solve the variational inference problem of jointly learning the likelihood p_θ and (approximate) posterior q_ϕ for such model. In that, the unknown data distribution $p(o,a)$ is effectively approximated via $\int_Z p(z)p_\theta(o,a|z)$, and the underlying generative process reproduced by (1) sampling a latent variable and (2) learning to decode it into a (ideally) high-likelihood sample under the (unknown) $p(o,a)$. Diffusion Models (DMs) (Ho et al., 2020) are another class of GMs which treat the similar problem of approximating an underlying unknown data distribution—*variational inference*—by *partially* extending VAEs to the case where *multiple* latent variables influence each other and the generative process underlying o, a itself. In particular, DMs posit the generative process can be decomposed to a series of piece-wise (Markovian) interactions between (latent) variables (Figure 22), resulting in

$$p(\underbrace{o,a}_{=z_0}) = \int_{\text{supp}(Z_0)}\int_{\text{supp}(Z_1)}\dots\int_{\text{supp}(Z_T)} p(z_0, z_1, \dots z_T) \tag{30}$$

$$p(z_0, z_1, \dots z_T) = p(z_T)\prod_{t=0}^{T} p(z_{t-1}|z_t), \tag{31}$$

where we explicitly showed the marginalization over the multiple latents in 30, and used the law of conditional probability and Markov property in 31.

Similarily to VAEs, providing an exact interpretation for the latent variables is typically not possible. Still, one fairly reasonable application-driven intuition is that, by providing a model of the hierarchical, decoupled interaction of latent variables, Hierarchical Markov Latent Variable (HMLV) models attempt to capture the different resolutions at which different conditioning factors intervene, so that in a robotics application for instance, one could naturally distinguish between early-stage trajectory planning ($t \to T$) and fine-grained adjustments ($t \to 0$). In that, HMLV models thus provide a framework to perform variational inference via multiple, sequential sampling steps from different higher

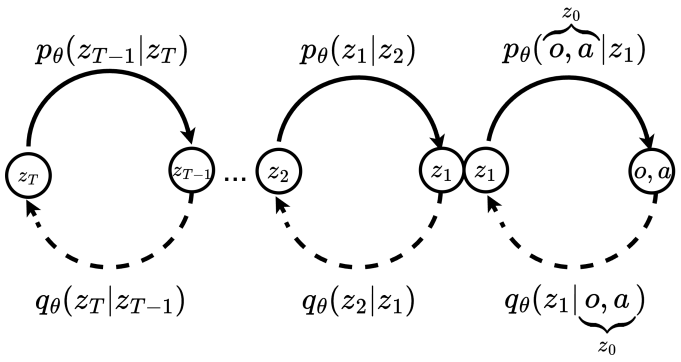


Figure 22 | HMLV models posit the data generation process is influenced by a stack of Markov-dependent latent variables, with samples from the posterior distribution being progressively higher up in the hierarchy.

level distributions instead of approximating the generative process with a single-latent variable model. DMs are a particular instantiation of HMLV models for which the posterior $q(z_t|z_{t-1}) = \mathcal{N}(z_t\sqrt{1-\beta_t}, \beta_t\mathbf{I})$ for a given $\beta_t \in \mathbb{R}^+$, thereby iteratively reducing the signal-to-noise ratio as β_t increases along the latents hierarchy.

Just like VAEs, DMs attemp to learn to reproduce an underlying data distribution $p(o,a)$ given a collection of i.i.d. samples approximating the model posited to have generated the data in the first place (30). Similarily to VAEs, DMs approximate the process of sampling from the unknown $p(o,a)$ (1) sampling from an easy-to-sample distribution (e.g., Gaussian) and (2) learning to reconstruct high-likelihood samples under the unknown distribution. However, in stark contrast with VAEs, the easy-to-sample distribution contains *no mutual information* regarding the data distribution $p(o,a)$. Crucially, as no information from the sample (o,a) (denoted as $z_0 \equiv (o,a)$ for the sake of notation) is assumed to be propagated throughout the chain of latents, the posterior $q(z_t|z_{t-1})$ assumes a relatively amicable structure in DMs, reducing complexity. The *true* likelihood $p(z_{t-1}|z_t)$ is instead typically approximated using the parametrization $p_\theta(z_{t-1}|z_t)$. In that, the information contained in the unknwon data distribution is *reconstructed* via a process in which samples from a fixed distribution are turned into (ideally) high-likelihood samples under $p(o,a)$—a process referred to as *denoising*.

Under such model, we can express the log-likelihood of an arbitrary sample as[3]

$$\begin{aligned}\log p_\theta(\underbrace{o,a}_{=z_0}) =& \mathbb{E}_{z_1\sim q(\bullet|z_0)} \log p_\theta(z_0|z_1) - \\ & \mathbb{E}_{z_{T-1}\sim q(\bullet|z_0)}\big[\mathrm{D}_{\mathrm{KL}}(q(z_T|z_{T-1})\|p(z_T))\big] - \\ & \sum_{t=1}^{T-1} \mathbb{E}_{(z_{t-1},z_{t+1})\sim q(\bullet|z_0)}\big[\mathrm{D}_{\mathrm{KL}}(q(z_t|z_{t-1})\|p_\theta(z_t|z_{t-1}))\big],\end{aligned} \tag{32}$$

providing an optimization target in the form of $\max_\theta \log p_\theta(\mathcal{D})$.

In their seminal work on using DMs for variational inference, Ho et al. (2020) introduce major contributions regarding solving $\min_\theta -\log p_\theta(o,a)$. In particular, Ho et al. (2020) exclusively adopt a fixed *Gaussian* posterior in the form of $q(z_t|z_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}z_{t-1}, \beta_t\mathbf{I})$. The choice of adopting Gaussians has profound implications on the generative process modeled. Indeed, under the (mild) assumption that the variance is sufficiently small $\beta_t \leq \eta, \eta \in \mathbb{R}^+$, Sohl-Dickstein et al. (2015) proved that the likelihood $p(z_{t-1}|z_t)$ is Gaussian as well, which allows for the particularly convenient parametrization of the approximate likelihood $p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t,t), \Sigma_\theta(x_t,t))$, $t \in [1,T]$, as well as for closed-form tractability of the KL-divergence terms in 32. Further, the posterior's structure also enables an analytical description for the distribution of the t-th latent variable, $q(z_t|z_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}z_0, (1-\bar{\alpha}_t)\mathbf{I})$, with $\alpha_t = 1-\beta_t$, $\bar{\alpha}_t = \prod_{k=1}^t \alpha_k$, which conveniently prevents iterative posterior sampling.

Finally, adopting Gaussian posteriors permits a particularly pleasing interpretation of the dynamics of training DMs (Permenter and Yuan, 2024). By using Gaussian posteriors, the hierarchical latent variables effectively lose increasingly more information circa the original (unknown) distribution's sample, z_0, increasingly distributing according to a standard Gaussian and thus containing no information at all (Figure 23). Figure 23 illustrates this procedure on a simplified, bidimensional observation-action distribution, where we considered $o = q_2$ and $a = q_2^h$, with q_2 representing the robot's *elbow flex* actuation and q_2^h the human teleoperator's robot elbow flex.

[3] $o, a = z_0$ for the sake of notation. Steps omitted for brevity. See Section A in Ho et al. (2020) for a complete derivation.

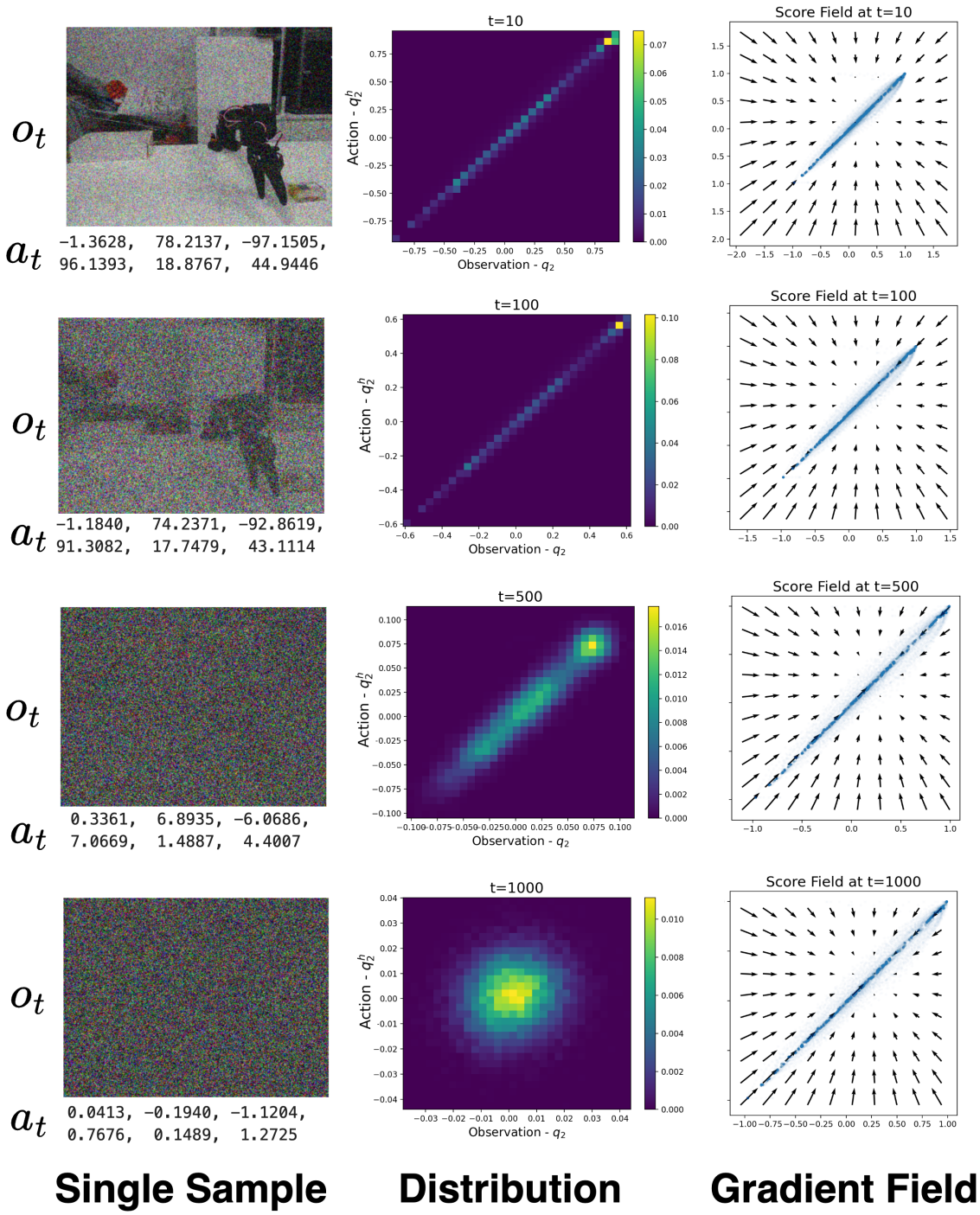


Figure 23 | DMs iteratively corrupt samples (left) from an unknown distribution into a quasi-standard Gaussian (center), learning the displacement field (right) that permits to reconstruct samples from the unknown target distribution by iteratively denoising samples of a tractable, easy-to-sample distribution.

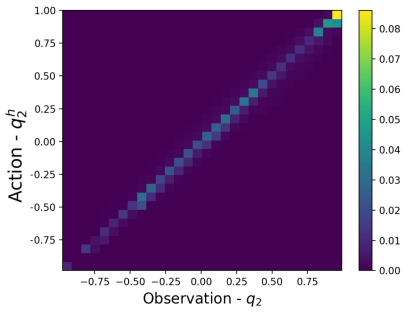


Figure 24 | A joint action-observation distribution, in the simplified case where the observation is the elbow-flex actuation in a SO-100, and the action is the recorded position for the same joint in the teleoperator arm. The motion recorded being teleoperated, the points distribute along a the diagonal.

Because the recorded behavior is teleoperated, measurements mostly distribute along the line $a = o + \eta, \eta \sim N(0, 1)$, with η-variability accouting for minor control inconsistencies (Figure 24). Using Gaussian posteriors—i.e., adding Gaussian noise—effectively simulates a *Brownian motion* for the elements in the distribution's support (in Figure 23, $\mathcal{O} \times \mathcal{A}$), whereby information *diffuses away* from the samples, and comparing the diffused samples to the original data points one can derive an estimate of the total displacement induced by diffusion. Under the only assumption that the likelihood of the diffused samples is low under the original unknown data distribution, then one can effectively approximate the unkwown distribution by learning to *reverse* such displacement. This key intuition allows to write a simplified training objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,z_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}z_0 + \epsilon\sqrt{1-\bar{\alpha}_t}, t)\|^2\right], \quad t \sim \mathcal{U}(\{1,\dots,T\}), \quad z_0 \sim \mathcal{D}, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{33}$$

In this simplified (minimization) objective, the optimization process differs from 32 in that, rather than maxizing p_θ directly, the parameters θ of the pairwise likelihood $p_\theta(z_{t-1}|z_t)$ are adjusted to *predict the total displacement* ϵ for a randomly long ($t \sim \mathcal{U}(\{1,\dots,T\}$)) diffusion process starting from a sample of the target distribution.

By learning the total displacement from a generally, uninformative corrupted sample obtained diffusing information and a sample from an unknown distribution—significant ($\|\epsilon\| > 0$) whenever input and target distribution are sufficiently different— Ho et al. (2020) show that one can approximate the underlying distribution reversing the displacement, *denoising* samples. Interestingly, under the hypothesis real-world data belongs to a single higher dimensional manifold (Manifold Hypothesis), Permenter and Yuan (2024) show that diffusion learns the gradient of a distance function from any off-point manifold (such as perturbed, uniformative samples), and the data manifold itself. Following this gradient—i.e., denoising a sample from an uninformative distribution—corresponds to projecting back into the manifold, yielding a procedure to sample from unknown distributions by means of Euclidean projection. Indeed, under the assumption that $p_\theta(z_{t-1}|z_t)$ is Gaussian, then sampling $z_{t-1} \sim p_\theta(\bullet|z_t)$ corresponds to computing

$$z_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(z_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(z_t, t)\right) + \sigma_t\epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{34}$$

thus showing that the lower-level latent variables in a DM can be obtained by iteratively removing noise from the one-step higher order variable, using the noise regressor $\epsilon_\theta(z_t, t)$ learned minimizing 33.

4.2 Action Chunking with Transformers

While GMs prove useful in learning complex, high-dimensional multi-modal distributions, they do not natively address the compouding errors problem characteristic of online, sequential predictions. In Action Chunking with Transformers (ACT), Zhao et al. (2023) present an application of VAEs to the problem of learning purely from offline trajectories, introduce a simple, yet effective method to mitigate error compounding, learning high-fidelity autonomous behaviors. Drawing inspiration from how humans plan to enact atomically sequences of the kind $a_{t:t+k}$ instead of single actions a_t, Zhao et al. (2023) propose learning a GM on a dataset of input demonstrations by modeling *action chunks*. Besides contributions to learning high-performance autonomous behaviors, Zhao et al. (2023) also introduce hardware contributions in the form of a low-cost bimanual robot setup (ALOHA) capable of performing fine-grained manipulation tasks, such as opening a lid, slotting a battery in its allotment or even prepare tape for application.

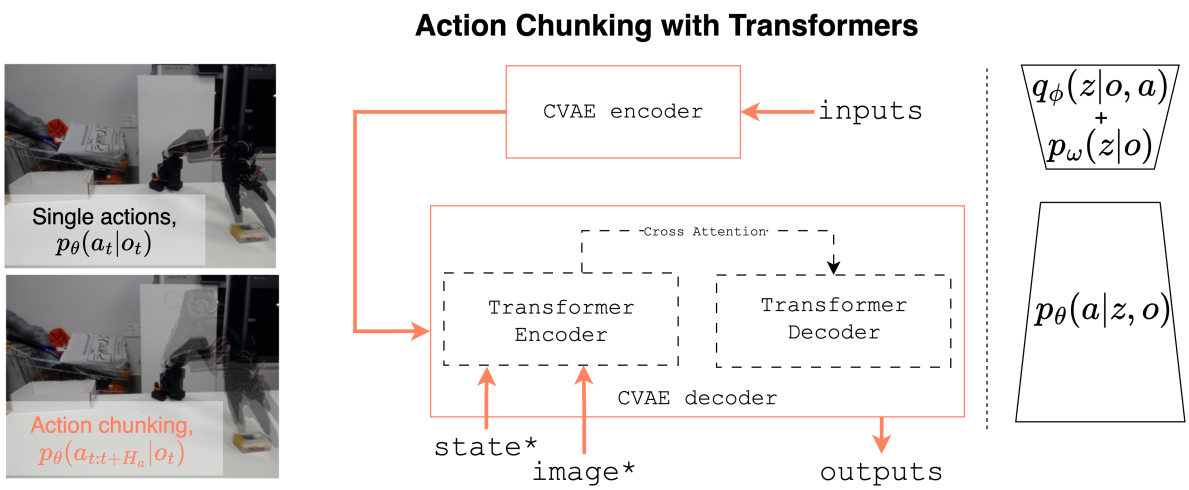


Figure 25 | Action Chunking with Transformer (ACT), as in Zhao et al. (2023). ACT introduces an action chunking paradigm to cope with high-dimensional multi-modal demonstration data, and a transformer-based CVAE architecture.

On the robot learning side of their contributions, Zhao et al. (2023) adopt transformers as the architectural backbone to learn a *Conditional* VAE (Sohn et al., 2015). Conditional VAEs are a variation of the more standard VAE formulation introducing a conditioning variable on sampling from the latent prior, allowing the modeling of *one-to-many* relationships between latent and data samples. Further, in stark contrast with previous work (Florence et al., 2022; Janner et al., 2022), Zhao et al. (2023) do not learn a full joint $p_\theta(o, a)$ on observation and actions. While the *policy* distribution $p_\theta(a|o)$ can in principle be entirely described from its joint $p_\theta(o, a)$, it is often the case that the conditional distribution is intractable when using function approximators, as $p_\theta(a|o) = \frac{p_\theta(o,a)}{\int_{\mathcal{A}} p_\theta(o,a)}$ and the integral in the denominator is typically intractable. Instead of modeling the full joint using a vanilla VAE, Zhao et al. (2023) propose learning a *conditional* VAE (Sohn et al., 2015) modeling the policy distribution directly $p(a|o)$.

In practice, when learning from demonstrations adopting CVAEs results in a slight modification to the VAE objective in 26, which is adapted to

$$\text{ELBO}_{\mathcal{D}}(\theta, \phi, \omega) = \sum_{i=0}^{N} \left(\mathbb{E}_{z \sim q_\phi(\cdot|o_i,a_i)} \left[\log p_\theta(a_i|z, o_i) \right] - \text{D}_{\text{KL}} \left[q_\phi(z|o_i, a_i) \| p_\omega(z|o_i) \right] \right) \tag{35}$$

Notice how in 35 we are now also learning a new set of parameters ω for the prior distribution in the latent space. Effectively, this enables conditioning latent-space sampling (and thus reconstruction) during training, and potentially inference, providing useful when learning inherently conditional distributions like policies. Further, ACT is trained as a β-CVAE (Higgins et al., 2017), using a weight of the KL regularization term in 35 as an hyperparameter regulating the information condensed in the latent space, where higher β results in a less expressive latent space.

In their work, Zhao et al. (2023) ablated using a GM to learn from human demonstrations compared to a simpler, supervised objective, $\mathcal{L}_1(a, a') = \|a - a'\|_1$. Interestingly, they found the performance of these two approaches to be comparable when learning from *scripted* demonstrations. That is, when learning from data collected rolling out a predetermined set of commands $[q_0^c, q_1^c, \dots]$, GM did *not* prove competitive compared to standard supervised learning. However, when learning from human demonstrations—i.e., from data collected executing commands coming from a human controller $[q_0^h, q_1^h, \dots]$—they found performance (success rate on a downstream task) to be severily (-33.3%) hindered from adopting a standard supervised learning objective compared to a richer, potentially more complex to learn variational objective, in keeping with the multimodal nature of human demonstrations data and findings presented in Florence et al. (2022). The authors also ablate the action chunking paradigm, reporting significant performance gains for performing action chunking (1% vs. 44% success rate). To avoid acting openloop, Zhao et al. (2023) design an inference process consisting in performing inference at every timestep t and then aggregate overlapping chunks using chunks' exponential moving average.

In ACT (Figure 25), inference for a given observation $o \in \mathcal{O}$ could be performed by (1) computing a prior $p_\omega(z|o)$ for the latent and (2) decoding an action chunk from a sampled latent $z \sim p_\omega(\bullet|o)$, similarily to how standard VAEs generate samples, with the exception that vanilla VAEs typically pose $p(z|o) \equiv p(z) \sim N(\mathbf{0}, \mathbf{I})$ and thus skip (1).

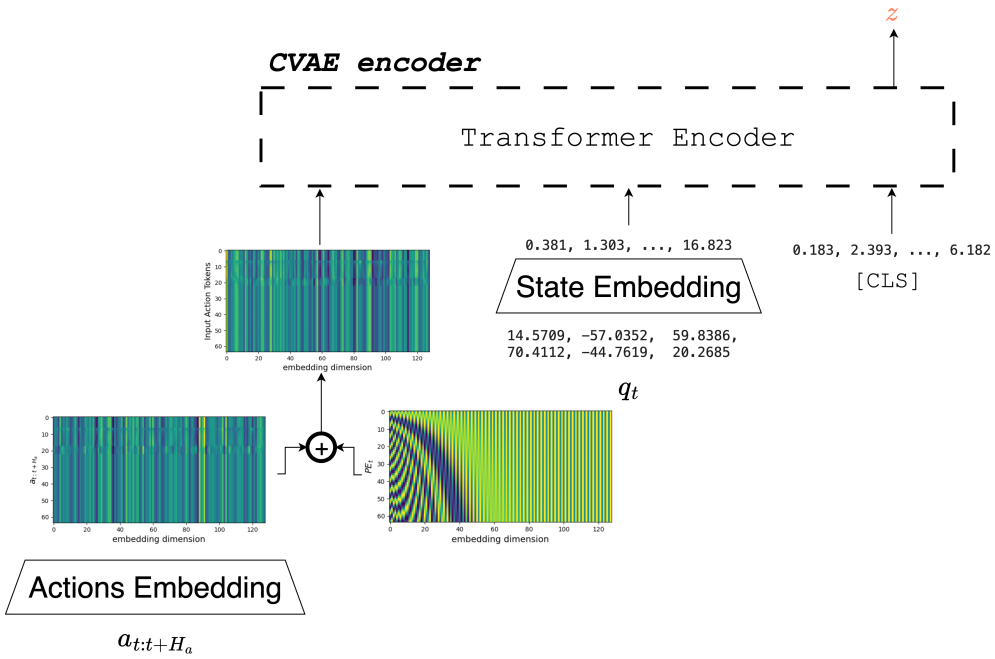


Figure 26 | The CVAE encoder used in ACT. Input action chunks are first embedded and aggregated with positional embeddings, before being processed alongside embedded proprioperceptive information, and a learned `[CLS]` token used to aggregate input level information, and predict the style variable z. The encoder is entirely disregarded at inference time.

However, the authors claim using a deterministic procedure to derive z may benefit policy evaluation, and thus avoid sampling from the conditional prior at all. At test time, instead, they simply use $z = \mathbf{0}$, as the conditional prior on z used in training is set to be the unit Gaussian. At test time, conditioning on the observation o is instead achieved through explicitly feeding proprioperceptive and visual observations to the decoder, $p_\theta(a|z, o)$, while during training z is indeed sampled from the approximate posterior distribution $p_\phi(z|o, a)$, which, however, disregards image observations and exclusively uses proprioperceptive states to form o for efficiency reasons (as the posterior q_ϕ is completely disregarded at test time).

4.2.1 Code Example: Learning ACT

using act example

4.3 Diffusion Policy

DMs proved very effective in approximating complex highly dimensional distributions, such as distributions over images (Ho et al., 2020) or videos (Polyak et al., 2025), thanks to their inherent capability to deal with multimodal data and training stability. In Diffusion Policy (DP), Chi et al. (2024) present an application of DMs the field of robot learning, leveraging diffusion to model human expert demonstrations in a variety of simulated and real-world tasks. Similarily to Action Chunking with Transformer (Zhao et al., 2023), Chi et al. (2024) (1) adopt a modified *observation-conditioned target distribution* instead of the full joint $p(o, a)$ and (2) predict multiple actions into the future instead of a single action. Besides the intractability of the observations' marginal $p_\theta(o)$ given $p_\theta(o, a)$, DP's rationale for modeling the data distribution via $p_\theta(a|o)$ stems from the rather test-time compute intensive nature of diffusion, whereby generating actions *alongside* observations is likely to result in higher complexity and thus a likely larger number of denoising operations, which would prove ultimately pointless considering robotics applications rely on the capability to generate controls rather than reproducing observations.

In practice, conditioning on observation data is achieved conditioning the added noise regressor ϵ_θ introduced in 33 on a stack of T_o observations, resulting in the *conditional* simplified diffusion objective

$$\mathcal{L}(\theta) = \mathbb{E}_{t, a_{t:t+H_a}, \epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} a_{t:t+T_a} + \epsilon\sqrt{1-\bar{\alpha}_t}, t, o_{t-T_o:t})\|^2\right], \tag{36}$$
$$t \sim \mathcal{U}(\{1, \dots, T\}), \quad a_{t:t+T_a}, o_{t-T_o:t} \sim \mathcal{D}, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Notice how in 36 the noise regressor is conditioned both on the latent variable rank t *and* on a stack of previous observations $o_{t-T_o:t}$. Chi et al. (2024) claim the combination of (1) conditioning on a horizon of previous observations

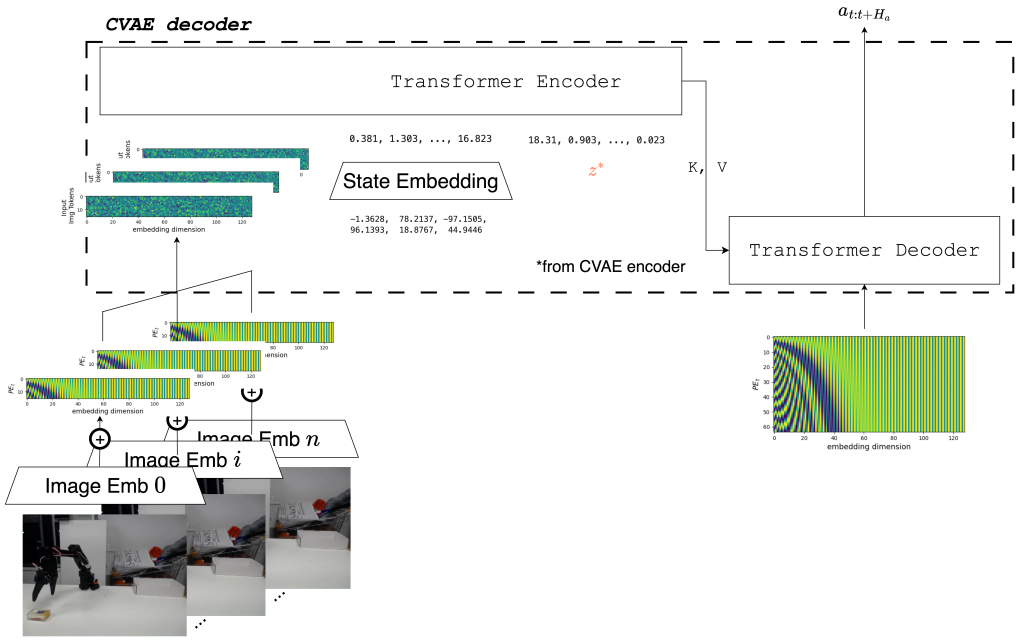


Figure 27 | The CVAE decoder used in ACT, comprising of a full encoder-decoder Transformer architecture. Camera observations from all n camera views are first embedded using pre-trained visual encoders, and then concatenated to the corresponding positional embeddings. Then, alongside embeddings for the proprioperceptive information available and the style variable z retrieved from the CVAE encoder, the Transformer encoder shares the matrices K, Q with the Transformer decoder, trained to decode fixed position embeddings into action valid chunks.

and (2) predicting multiple actions into the future allows DP to *commit to specific modes* in the data at inference time, which proves essential for good performance and avoiding undecisiveness.

Figure 28 shows the convolution-based version of the architecture proposed by Chi et al. (2024), illustrating inference on a single sample from $\mathcal{D}$ for simplicity. An arbitrarily noisy chunk of H_a actions $\tilde{a}_{t:t+H_a}$ is mapped to a learned high-dimensional space. Similarily, both image observations and poses are embedded before being aggregated to the action embeddings. Then, a U-Net (Ronneberger et al., 2015) is trained to regress the noise added into $\tilde{a}_{t:t+H_a}$, using observation conditioning information at every layer and seeking to optimize 36. At inference time, the noise predictor is used to predict the quantity of noise at every $t \in [T, \dots, 0]$ and iteratively subtract it from $\tilde{a}_{t:t+T_a}$, reversing the diffusion process simulated in training conditioned on $o_{t-T_o:t}$ to predict $a_{t:t+T_a}$.

Training using 50-150 demos (15-60 minutes of teleoperation data) DP achieves strong performance on a variety of simulated and real-world tasks, including dexterous and deformable manipulation tasks such as sauce pouring and mat unrolling. Notably, the authors ablated the relevance of using RGB camera streams as input to their policy, and observed how high frame-rate visual observations can be used to attain performance (measured as success rate) comparable to that of state-based policies, typically trained in simulation with priviledged information not directly available in real-world deployments. As high-frame rate RGB inputs naturally accomodate for dynamic, fast changing environments, Chi et al. (2024)'s conclusion offers significant evidence for learning streamlined control policies directly from pixels. In their work, Chi et al. (2024) also ablate the performance of DP against their baseline against the size of the dataset collected, showing that DP outperforms the considered baseline for every benchmark size considered. Further, to accelerate inference, Chi et al. (2024) employ Denoising Diffusion Implicit Models (Song et al., 2022), a variant of Denoising Diffusion Probabilistic Models (Ho et al., 2020) (DDPM) adopting a strictly deterministic denoising paradigm (differently from DDPM's natively stochastic one) inducing the same final distribution's as DDPM's, and yet resulting in 10 times less denoising steps at inference time (Chi et al., 2024). Across a range of simulated and real-world tasks, Chi et al. (2024) find DPs particularly performant when implementing a transformer-based network as ϵ_θ, although the authors note the increased sensitivity of transformer networks to hyperparameters and thus explicitly recommend starting out with a simpler, convolution-based architecture for diffusion (Figure 28), which are however reported to be biased towards learning low-frequency components (Tancik et al., 2020) and thus may prove more challenging to train with non-smooth action sequences.

4.3.1 Code Example: Learning Diffusion Policies

using diffusion policy example

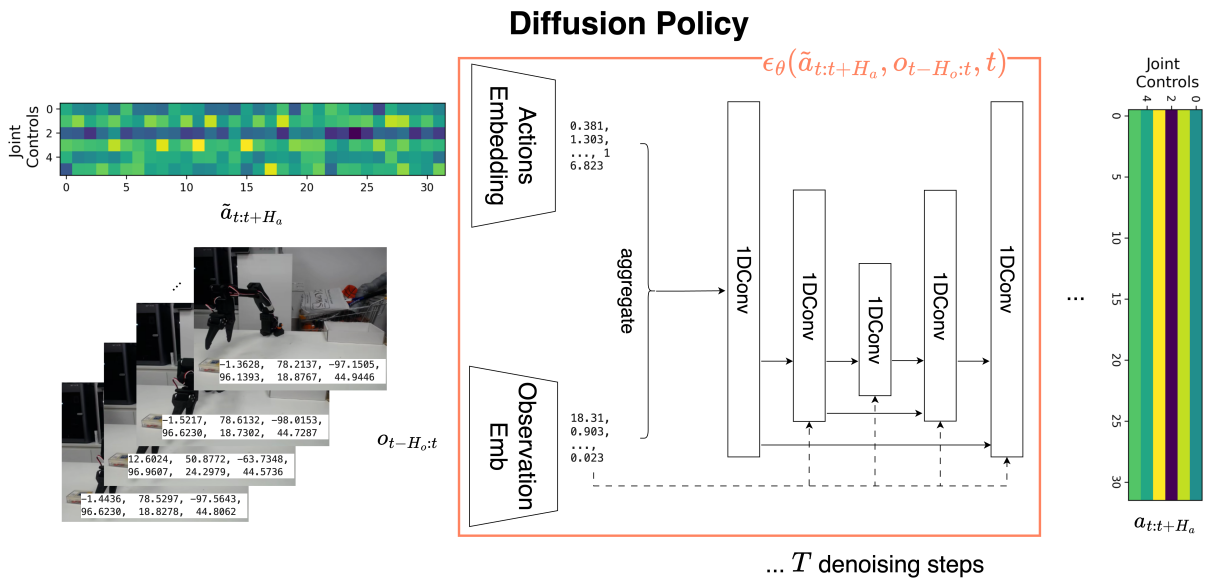


Figure 28 | The Diffusion Policy archicture, as in Chi et al. (2024). A stack of H_o previous observations is used as external conditioning to denoise a group of H_a actions. Conditioning is used at every layer of a U-Net block, and in practice allows to obtain fully-formed action chunks with as little as $T = 10$ denoising steps.

4.4 Optimized Inference by Decoupling Action Prediction and Execution

Modern visuomotor policies output *action chunks*–sequences $\pi(o_t) = \mathbf{A}_t$ with $\mathbf{A}_t = \left(a_t, a_{t+1}, \dots, a_{t+H_a}\right)$ being a sequence of $H_a \gg 1$ low-level commands enqueued in an action queue, originating from an environment observation, o_t. Predicting series of actions instead of single commands proved essential in learning complex, multi-modal behavior (Zhao et al., 2023; Chi et al., 2024).

Typically, the robot executes the entire action chunk $\mathbf{A}_t$, before a new observation o_{t+H_a} is passed to the policy π to predict the next chunk. This results in open-loop inference in between observations captured every H_a timesteps. Zhao et al. (2023) adopts a different strategy whereby the robot controller interleaves chunk prediction $\mathbf{A}_t \leftarrow \pi(o_t)$ and chunk consumption $a_t \leftarrow \textsc{PopFront}(\mathbf{A}_t)$, computing a new chunk of actions at every timestep t and aggregating the predicted chunks on overlapping sections. While adaptive—every observation at every timestep o_t is processed—such approaches rely on running inference continuously, which can be prohibitive in resource-constrained scenarios, such as edge deployments.

A less resource-intensive approach is to entirely exhaust the chunk $\mathbf{A}$ before predicting a new chunk of actions, a strategy we refer to as *synchronous* (sync) inference. Sync inference efficiently allocates computation every H_a timesteps, resulting in a reduced average computational burden at control time. In contrast, it inherently hinders the responsiveness of robot systems, introducing blind lags due to the robot being *idle* while computing $\mathbf{A}$.

We directly assess the lack of adaptiveness of robot systems due to acting open-loop, and the presence of lags at runtime by decoupling action chunk prediction $\mathbf{A}$ from action execution $a_t \leftarrow \textsc{PopFront}(\mathbf{A}_t)$, developing an *asynchronous* (async) inference stack (1), whereby a RobotClient sends an observation o_t to a PolicyServer, receiving an action chunk $\mathbf{A}_t$ once inference is complete (29). In this, we avoid execution lags by triggering chunk prediction while the control loop is still consuming a previously available queue, aggregating it with the newly incoming queue whenever available. In turn, async-inference tightens the loop between action prediction and action execution, by increasing the frequency at which observations are processed for chunk prediction. Crucially, decoupling action prediction from action execution also directly allows to allocate more computational resources on a remote policy server sending actions to the robot client over networks, something which may prove very effective in resource-constrained scenarios such as low-power robots.

Implementation details *Async* inference (1) tightens the control loop by capturing observations more often, directly eliminates idle gaps at runtime, and (2) directly allows to run inference on more powerful computational resources than the ones typically available onboard autonomous robotic platforms.

Algorithmically, we attain (1) on the RobotClient-side by consuming actions from a readily available queue until a

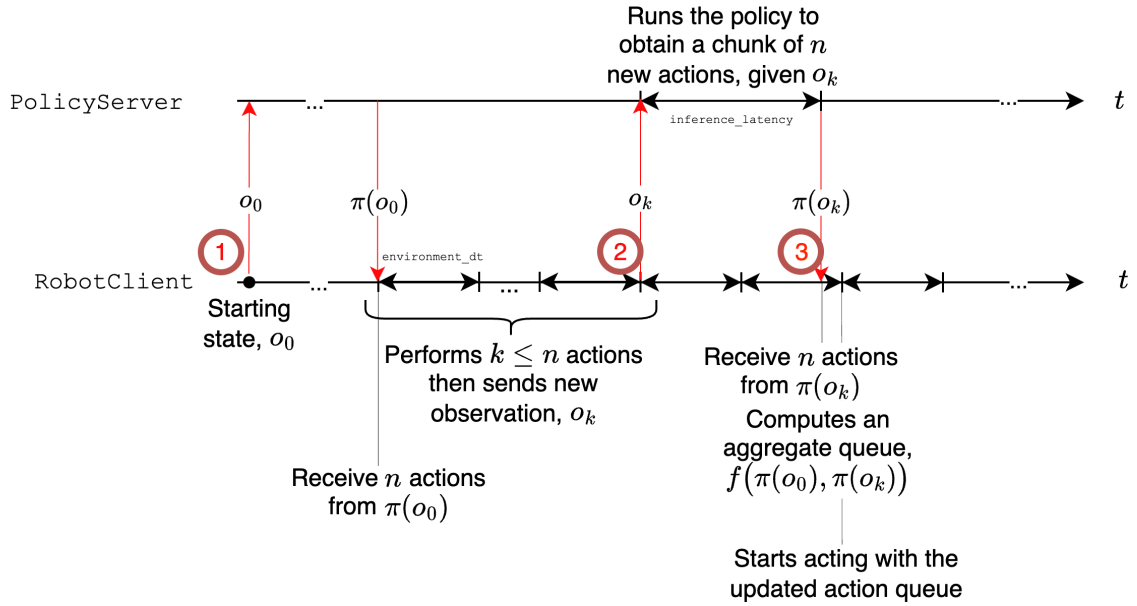


Figure 29 | **Asynchronous inference**. Illustration of the asynchronous inference stack. Note that the policy can be run on a remote server, possibly with GPUs.

Algorithm 1 Asynchronous inference control-loop

Input: horizon T, chunk size H_a, threshold $g \in [0, 1]$
Init: capture o_0; send o_0 to PolicyServer; receive $\mathbf{A}_0 \leftarrow \pi(o_0)$
for t **to** H_a **do**
  $a_t \leftarrow \text{PopFront}(\mathbf{A}_t)$
  $\text{Execute}(a_t)$ ▷ execute action at step t
  if $\frac{|\mathbf{A}_t|}{H_a} < g$ **then** ▷ queue below threshold
    capture new observation, o_{t+1}
    if $\text{NeedsProcessing}\ (o_{t+1})$ **then** ▷ similarity filter, or triggers direct processing
      `async_handle` $\leftarrow \text{AsyncInfer}(o_{t+1})$ ▷ Trigger new chunk prediction (non blocking)
      $\tilde{\mathbf{A}}_{t+1} \leftarrow \pi(o_{t+1})$ ▷ New queue is predicted with the policy
      $\mathbf{A}_{t+1} \leftarrow f(\mathbf{A}_t, \tilde{\mathbf{A}}_{t+1})$ ▷ aggregate overlaps (if any)
    end if
  end if
  if NotCompleted(`async_handle`) **then**
    $\mathbf{A}_{t+1} \leftarrow \mathbf{A}_t$ ▷ No update on queue (inference is not over just yet)
  end if
end for

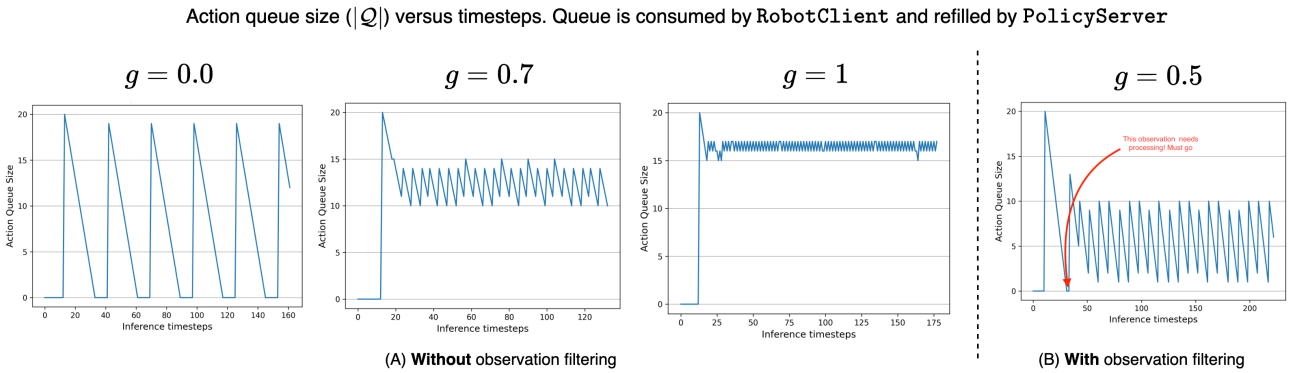


Figure 30 | Action queue size evolution at runtime for various levels of g when (A) not filtering out observation based on joint-space similarity and (B) filtering out near-duplicates observation, measuring their similarity in joint-space.

threshold condition on the number of remaining actions in the queue ($|\mathbf{A}_t|/H_a < g$) is met. When this condition is triggered, a new observation of the environment is captured and sent to the (possibly remote) PolicyServer. To avoid redundant server calls and erratic behavior at runtime observations are compared in joint-space, and near-duplicates are dropped. Two observations are considered near-duplicates if their distance in joint-space is under a predetermined threshold, $\epsilon \in \mathbb{R}_+$. Importantly, when the queue available to robot client eventually becomes empty, the most recent observation is processed regardless of similarity.

Interestingly, the behavior of async inference can be studied analytically. First, let ℓ be a random variable modeling the time needed to receive an action chunk $\mathbf{A}$ after sending an observation o, i.e. the sum of (1) the time to send across the observation o between the RobotClient and PolicyServer, $t_{C\to S}$ (2) the inference latency on the PolicyServer, ℓ_S and (3) the time to send $\mathbf{A}$ between the PolicyServer and RobotClient, $t_{S\to C}$. Assuming independence, $\mathbb{E}[\ell] = \mathbb{E}[t_{C\to S}] + \mathbb{E}[\ell_S] + \mathbb{E}[t_{S\to C}]$ which can be further simplified to $\mathbb{E}[\ell] \simeq \mathbb{E}[\ell_S]$, assuming communication time is (1) equal in both directions and (2) negligible with respect to the inference latency. Second, let Δt be the environment's control cycle. With a real-world frame-rate of 30 frames per second, $\Delta t = 33$ms. Consequently, exhausted queues at runtime–i.e. being idle awaiting for a new chunk–are avoided for $g \geq \frac{\mathbb{E}[\ell_S]/\Delta t}{H_a}$. In this, the queue threshold g plays a major role relatively to the availability of actions to the RobotClient.

30 illustrates how the size of the action chunk $|\mathbf{A}_t|$ evolves over time for three representative values of g, detailing the following key scenarios:

- **Sequential limit** ($g = 0$). The client drains the entire chunk before forwarding a new observation to the server. During the round-trip latency needed to compute the next chunk, the queue is empty, leaving the robot *incapable of acting*. This reproduces the behavior of a fully sequential deployment and results in an average of $\mathbb{E}[\ell_S]$ idle seconds.
- **Asynchronous inference** ($g \in (0, 1)$). Allowing the client to consume $1 - g$ of its available queue $\mathbf{A}_{t-1}$ before triggering inference for a new action queue $\mathbf{A}_t$, amortizing computation while keeping the queue from emptying. The overlap between successive chunks provides a buffer against modeling errors without the full cost of the $g = 1$ regime. The updated queue $\mathbf{A}_t$ is obtained aggregating queues on the overlapping timesteps between $\mathbf{A}_{t-1}$ and the incoming $\tilde{\mathbf{A}}_t$.
- **Compute-intensive limit** ($g = 1$). As an extreme case, and in keeping with Zhao et al. (2023), an observation is sent at *every* timestep. The queue is therefore almost always filled, with only a minor saw-tooth due to $\Delta t/\mathbb{E}[\ell_s] < 1$. While maximally reactive, this setting incurs one forward pass per control tick and can prove prohibitively expensive on limited hardware. Importantly, because the client is consuming actions while the server computes the next chunk, the available queue never gets filled again.

30 emphasizes the trade-off governed by g: small values place result in idle periods, whereas $g \approx 1$ assumes a highly accurate model and pays a significant compute price. In practice, choosing $g \in (0, 1)$ allows to strike a balance between reactivity against resource budgets. If not for the aforementioned similarity filter, the RobotClient would send observations for processing every $(1 - g)H_a \cdot \Delta t$ seconds, receiving a new chunk of actions every $(1 - g)H_a \cdot \Delta t + \mathbb{E}[\ell_S]$, on average. The presence of the observation similarity filter dilates this processing time, and serves the scope of avoiding the robot stalling due to the queue being constantly integrated with an incoming, nearly identical, action

chunk. In particular, 30 results in a queue which is filled with incoming actions *unless* near-duplicate observations are filtered out from the processing pipeline. For clarity, the red arrow in 30 highlights a timestep where the observation similarity mechanism is bypassed, forcing a (nearly identical) observation to be processed as the queue results empty.

4.4.1 Code Example: Using Async Inference

async inference example

References

DROID: A Large-Scale In-the-Wild Robot Manipulation Dataset. https://droid-dataset.github.io/.

Open X-Embodiment: Robotic Learning Datasets and RT-X Models. https://robotics-transformer-x.github.io/.

Joshua Achiam. Spinning up in deep reinforcement learning. 2018.

AgiBot-World-Contributors, Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xuan Hu, Xu Huang, Shu Jiang, Yuxin Jiang, Cheng Jing, Hongyang Li, Jialu Li, Chiming Liu, Yi Liu, Yuxiang Lu, Jianlan Luo, Ping Luo, Yao Mu, Yuehan Niu, Yixuan Pan, Jiangmiao Pang, Yu Qiao, Guanghui Ren, Cheng Ruan, Jiaqi Shan, Yongjian Shen, Chengshi Shi, Mingkang Shi, Modi Shi, Chonghao Sima, Jianheng Song, Huijie Wang, Wenhao Wang, Dafeng Wei, Chengen Xie, Guo Xu, Junchi Yan, Cunbiao Yang, Lei Yang, Shukai Yang, Maoqing Yao, Jia Zeng, Chi Zhang, Qinglin Zhang, Bin Zhao, Chengyue Zhao, Jiaqi Zhao, and Jianchao Zhu. AgiBot World Colosseo: A Large-scale Manipulation Platform for Scalable and Intelligent Embodied Systems, August 2025.

Pulkit Agrawal. Computational Sensorimotor Learning.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand, October 2019.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: A Visual Language Model for Few-Shot Learning, November 2022.

Jorge Aldaco, Travis Armstrong, Robert Baruch, Jeff Bingham, Sanky Chan, Debidatta Dwibedi, Chelsea Finn, Pete Florence, Spencer Goodrich, Wayne Gramlich, Alexander Herzog, Jonathan Hoech, Thinh Nguyen, Ian Storz, Baruch Tabanpour, Jonathan Tompson, Ayzaan Wahid, Ted Wahrburg, Sichun Xu, Sergey Yaroshenko, and Tony Z Zhao. ALOHA 2: An Enhanced Low-Cost Hardware for Bimanual Teleoperation.

Mohammad Alizadeh and Zheng H. Zhu. A comprehensive survey of space robotic manipulators for on-orbit servicing. *Frontiers in Robotics and AI*, 11, October 2024. ISSN 2296-9144. doi: 10.3389/frobt.2024.1470950.

Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement Learning for Pivoting Task, March 2017.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient Online Reinforcement Learning with Offline Data, May 2023.

Kostas E. Bekris, Joe Doerr, Patrick Meng, and Sumanth Tangirala. The State of Robot Motion Generation, October 2024.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836): 77–82, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2939-8.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 0095-9057.

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, October 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion, March 2024.

Jonathan H. Connell and Sridhar Mahadevan, editors. *Robot Learning*. Springer US, Boston, MA, 1993. ISBN 978-1-4613-6396-5 978-1-4615-3184-5. doi: 10.1007/978-1-4615-3184-5.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602 (7897):414–419, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04301-9.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A. Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit Behavioral Cloning. In *Proceedings of the 5th Conference on Robot Learning*, pages 158–168. PMLR, January 2022.

Jun Fujita, Daisuke Soda, Chotaro Murata, and Hiroyuki Tsuhari. Development of Robots for Nuclear Power Plants and Their Application to New Fields. 57(4), 2020.

Robert J. Griffin, Georg Wiedebach, Sylvain Bertrand, Alexander Leonessa, and Jerry Pratt. Walking Stabilization Using Step Timing and Location Adjustment on the Humanoid Robot, Atlas. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 667–673, September 2017. doi: 10.1109/IROS.2017.8202223.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies, July 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, August 2018.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal Difference Learning for Model Predictive Control, July 2022.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments, July 2017.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with Diffusion for Flexible Behavior Synthesis, December 2022.

Yandong Ji, Gabriel B. Margolis, and Pulkit Agrawal. DribbleBot: Dynamic Legged Manipulation in the Wild, April 2023.

Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots, and Siddhartha Srinivasa. Grasping with Chopsticks: Combating Covariate Shift in Model-free Imitation Learning for Fine Manipulation, November 2020.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022.

Rob Knight, Pepijn Kooijmans, Thomas Wolf, Simon Alibert, Michel Aractingi, Dana Aubakirova, Adil Zouitine, Russi Martino, Steven Palma, Caroline Pascal, and Remi Cadene. Standard Open SO-100 & SO-101 Arms.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning Quadrupedal Locomotion over Challenging Terrain. *Science Robotics*, 5(47):eabc5986, October 2020. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior Generation with Latent Actions, June 2024.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and Dexterous Robotic Manipulation via Human-in-the-Loop Reinforcement Learning, October 2024.

Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning, March 2025.

Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 1 edition, May 2017. ISBN 978-1-316-66123-9 978-1-107-15630-2 978-1-316-60984-2. doi: 10.1017/9781316661239.

Gabriel B. Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid Locomotion via Reinforcement Learning, May 2022.

John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks, September 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013.

Frank Permenter and Chenyang Yuan. Interpreting and Improving Diffusion Models from an Optimization Perspective, June 2024.

Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, David Yan, Dhruv Choudhary, Dingkang Wang, Geet Sethi, Guan Pang, Haoyu Ma, Ishan Misra, Ji Hou, Jialiang Wang, Kiran Jagadeesh, Kunpeng Li, Luxin Zhang, Mannat Singh, Mary Williamson, Matt Le, Matthew Yu, Mitesh Kumar Singh, Peizhao Zhang, Peter Vajda, Quentin Duval, Rohit Girdhar, Roshan Sumbaly, Sai Saketh Rambhatla, Sam Tsai, Samaneh Azadi, Samyak Datta, Sanyuan Chen, Sean Bell, Sharadh Ramaswamy, Shelly Sheynin, Siddharth Bhattacharya, Simran Motwani, Tao Xu, Tianhe Li, Tingbo Hou, Wei-Ning Hsu, Xi Yin, Xiaoliang Dai, Yaniv Taigman, Yaqiao Luo, Yen-Cheng Liu, Yi-Chiao Wu, Yue Zhao, Yuval Kirstain, Zecheng He, Zijian He, Albert Pumarola, Ali Thabet, Artsiom Sanakoyeu, Arun Mallya, Baishan Guo, Boris Araya, Breena Kerr, Carleigh Wood, Ce Liu, Cen Peng, Dimitry Vengertsev, Edgar Schonfeld, Elliot Blanchard, Felix Juefei-Xu, Fraylie Nord, Jeff Liang, John Hoffman, Jonas Kohler, Kaolin Fire, Karthik Sivakumar, Lawrence Chen, Licheng Yu, Luya Gao, Markos Georgopoulos, Rashel Moritz, Sara K. Sampson, Shikai Li, Simone Parmeggiani, Steve Fine, Tara Fowler, Vladan Petrovic, and Yuming Du. Movie Gen: A Cast of Media Foundation Models, February 2025.

Dean A. Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, March 2011.

Lindsay Sanneman, Christopher Fourie, and Julie A. Shah. The State of Industrial Robotics: Emerging Technologies, Challenges, and Key Research Directions, October 2020.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017b.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 1 edition, May 2014. ISBN 978-1-107-05713-5 978-1-107-29801-9. doi: 10.1017/CBO9781107298019.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics, June 2025.

Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer Handbooks. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32550-7 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, January 2014.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, November 2015.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models, October 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains, June 2020.

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes, September 2024.

Yang Tang, Chaoqiang Zhao, Jianrui Wang, Chongzhen Zhang, Qiyu Sun, Weixing Zheng, Wenli Du, Feng Qian, and Juergen Kurths. Perception and Navigation in Autonomous Systems in the Era of Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):9604–9624, December 2023. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2022.3167688.

Russ Tedrake. Robotic Manipulation. Perception, Planning and Control., a.

Russ Tedrake. Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation, b.

Gabriele Tiboni, Karol Arndt, and Ville Kyrki. DROPO: Sim-to-Real Transfer with Offline Domain Randomization, January 2023.

Gabriele Tiboni, Pascal Klink, Jan Peters, Tatiana Tommasi, Carlo D'Eramo, and Georgia Chalvatzaki. Domain Randomization via Entropy Maximization, March 2024.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, March 2017.

Chong Zhang, Wenli Xiao, Tairan He, and Guanya Shi. WoCoCo: Learning Whole-Body Humanoid Control with Sequential Contacts, November 2024.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, April 2023.