LEROBOT: AN OPEN-SOURCE LIBRARY FOR END-TO-END ROBOT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Robotics is undergoing a significant transformation powered by advances in high-level control techniques based on machine learning, giving rise to the field of robot learning. Recent progress in robot learning has been accelerated by the increasing availability of affordable teleoperation systems, large-scale openly available datasets, and scalable learning-based methods. However, development in the field of robot learning is often slowed by fragmented, closed-source tools designed to only address specific sub-components within the robotics stack. In this paper, we present `lerobot`, an open-source library that integrates across the entire robotics stack, from low-level middleware communication for motor controls to large-scale dataset collection, storage and streaming. The library is designed with a strong focus on real-world robotics, supporting accessible hardware platforms while remaining extensible to new embodiments. It also supports efficient implementations for various state-of-the-art robot learning algorithms from multiple prominent paradigms, as well as a generalized asynchronous inference stack. Unlike traditional pipelines which heavily rely on hand-crafted techniques, `lerobot` emphasizes scalable learning approaches that improve directly with more data and compute. Designed for accessibility, scalability, and openness, `lerobot` lowers the barrier to entry for researchers and practitioners to robotics while providing a platform for reproducible, state-of-the-art robot learning.

1 INTRODUCTION

Early successes in robotics relied on the precise description of robot-environment interactions, typically consisting in analytical descriptions of rigid-body kinematics, contact modeling, and planning under uncertainty (*explicit models*). While effective in controlled settings, deriving accurate models for diverse deployment scenarios is difficult and error-prone, often requiring substantial expert effort and thus has limited scalability. Recent advances in Machine Learning (ML) have catalyzed a shift toward tackling robotics problems with *implicit models*, typically *learned* rather than formulated.

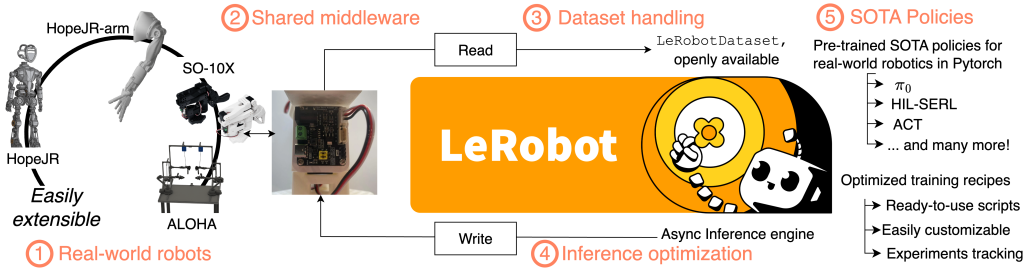


Figure 1: `lerobot` is an open-source library for end-to-end robot learning. It integrates across the entire robotics stack, from middleware motor interfaces to large-scale data collection and dataset streaming, supporting an optimized inference stack, scalable implementations of SOTA robot learning algorithms, and providing support for training custom models as well as reusing pre-trained ones.

A key advantage of learning-based methods (*implicit models*) is their scalability: performance empirically improves with larger datasets, and more compute. In turn, the shift from explicit to implicit models promises to address many of the challenges holding robotics back: rather than hand-tuning the different components of a typical robotics pipeline, robot learning algorithms learn monolithic control policies end-to-end, adapting to different input modalities and typically improve with increasing quantities of data, echoing broader trends in vision, language, and multimodal learning.

Despite this momentum, the robot learning ecosystem is fragmented as (1) middleware interfaces are often tailored to specific robots and difficult to adapt, and (2) datasets lack common formats and tooling, resulting in robot and task-specific contributions that are difficult to reproduce and use in practice. `lerobot` is an open-source library providing a unified, end-to-end stack for robot learning, and it is vertically integrated across the stack featuring:

- **Unified robot integration.** A consistent, Python-based middleware API for real-world motor control across diverse platforms, bridging typical ML frameworks and real-world robotics across a variety of robots, ranging from low-end manipulators to humanoid arms and hands.
- **Standardized datasets.** An efficient, multimodal format for recording, storing, and streaming high frame-rate sensory and image data via `LeRobotDataset`, a custom dataset format built for scale. With seamless integration into the open-source ecosystem, `LeRobotDataset` encourages openness and research reproducibility.
- **Optimized inference.** An optimized inference stack that decouples action planning from control execution both (1) physically and (2) logically, enabling policies to (1) run on separate machines with increased computational resources compared to those aboard robots, and (2) parallely to low-level control loops, for robust deployment and dynamic adaptability at runtime.
- **Efficient, reusable algorithms.** Clean, PyTorch-based implementations of state-of-the-art (SOTA) robot learning methods, optimized for (1) training custom models from scratch and (2) using openly-available pre-trained models.

Together, these components address fragmentation issues in the field, reducing the barrier to entry for robotics by providing vertical integration across the entire robotics stack, with a clear emphasis on accessibility and scalability, aiming at accelerating progress in robot learning.

2 Background

2.1 Explicit and Implicit Models

Autonomous motion leverages either *explicit* or *implicit* models (Bekris et al., 2024). Classical robotics historically uses *explicit models*, implemented as modular pipelines for perception, planning, and control (Siciliano & Khatib, 2016). This approach suffers from compounding errors, poor scalability to diverse deployment scenarios, and *undermodeling issues* due to simplified analytical models of physical interactions, limiting its effectiveness to unstructured, dynamic environments (e.g., a house versus a factory line). In contrast, robot learning relies on *implicit models* to develop monolithic, data-driven policies directly mapping observations to action. Robot learning also prioritizes interaction data over rigid assumptions, and replaces hand-engineered components with learned representations, offering a more robust and adaptable solution for unstructured environments.

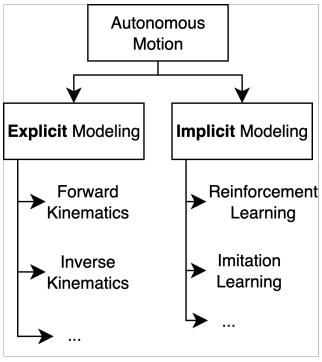


Figure 2: Some of the explicit and implicit models for autonomous motion.

The adaptability of these learned, implicit models stems directly from their scalability with data—a primary advantage over classical approaches. In this context, real-world robotics data is often collected in the form of expert demonstrations via *teleoperation*, a process where *"cognitive decisions are made by [a] human user, while the robot is responsible for their mechanical implementation"* (Siciliano & Khatib, 2016, Ch.43, §1). In recent years, teleoperation hardware has become increasingly affordable, making it more and more relevant for robot

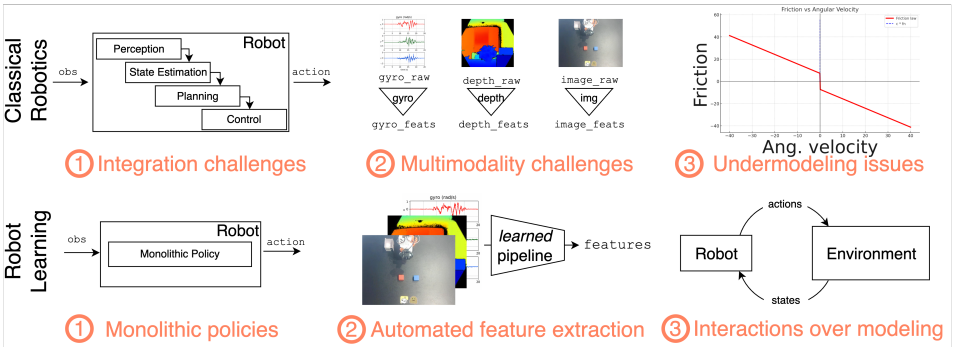


Figure 3: Classical robotics uses modular, model-based pipelines with hand-crafted features, while robot learning employs monolithic, data-driven policies that learn directly from interaction data.

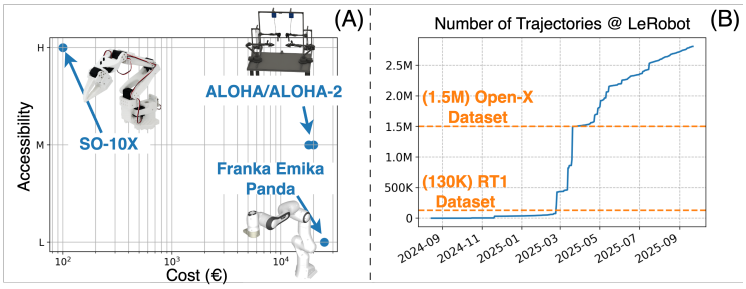


Figure 4: (A) Low-cost, open-source robots like SO-10X and ALOHA cost a fraction of proprietary industrial arms, using consumer-grade parts and 3D-printable designs. (B) Decentralized efforts to collect expert demonstrations in the form of trajectories surpassed centralized efforts for the collection of large amounts of real-world robotics data.

learning. Consumer-grade VR teleoperation headsets for robotics have been used to collect robot data both on real-world and simulated robots (Bjorck et al., 2025), and low-cost teleoperated robotic arms (Zhao et al., 2023; Aldaco et al., 2024; Knight et al., 2024) are increasingly empowering researchers and practitioners to collect real-world robotics data. In turn, this results in a multiplication of centralized (Brohan et al., 2023; Collaboration et al., 2025; Khazatsky et al., 2025) and de-centralized (Section 3.2) efforts to collect robot data. Figure 4 shows how fully accessible teleoperated platforms such as the SO-100, SO-101 (jointly referred to as SO-10X, (Knight et al., 2024)) and ALOHA-2 (Aldaco et al., 2024) can cost down to a fraction of closed-source, industrial-grade robots such as the Franka Emika Panda arm. Consequently, these low end robot platforms can be used to collect large amounts of data in a decentralized effort powered by the very accessibility—low-cost, open designs, 3D-printable parts—of these low-end robot platforms.

2.2 Robot Learning

Reinforcement Learning Reinforcement learning (RL) (Sutton & Barto, 2018) has been extensively applied to robotics (Kober et al., 2013), for the inherently sequential nature of control problems and Deep RL's capability to learn strategies for return maximization $\max_\pi J(\pi) = \max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$ directly from highly-dimensional, unstructured observations such as images (Mnih et al., 2013). Off-policy, entropy-regularized methods such as Soft Actor Critic (Haarnoja et al., 2018) can be adapted to exploit teleoperation data and safely train in the real-world, thereby sidestepping concerns related to operative safety and simulation-induced discrepancies. Reinforcement Learning with Prior Data (RLPD) (Ball et al., 2023) mixes offline and online buffers without pretraining to speed up convergence, and in conjuction with (1) learned reward classifiers overcoming the need to define brittle hand-crafted rewards (Luo et al., 2025) and (2) targeted human interventions during training, can yield near-perfect success rates in challenging manipulation tasks within 1-2 hours of real-world training (HIL-SERL, Luo et al. (2024)).

Imitation Learning Imitation Learning via Behavioral Cloning (BC) offers a pragmatic alternative to real-world RL by learning control directly from human demonstrations, eliminating the need for reward design and reducing exploration risk by learning to reproduce the behavior of an expert demonstrator (Pomerleau, 1988). Collected via teleoperation on increasingly affordable hardware, large corpora of robotics data also enable training at a scale across tasks and embodiments (Collaboration et al., 2025; Khazatsky et al., 2025). BC relies on learning *generative* models of the the joint (or conditional) distribution over state-action pairs $p : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, $p(a, s)$ (or $p(a|s)$) to learn from data distributions exhibiting multiple modes, such as teleoperation data (Florence et al., 2022). Recent works in BC thus employ powerful generative models to learn the conditional distribution $p(a|s)$, learning from multimodal demonstrations and produce coherent action sequences: Zhao et al. (2023) leverages (conditional) Variational Auto-Encoders (Kingma & Welling, 2022; Sohn et al., 2015), Chi et al. (2024) relies on Diffusion Models (Ho et al., 2020) whereas Black et al. (2024); Shukor et al. (2025) both rely on Flow Matching (Lipman et al., 2023). Inspired by successes in developing foundation models for vision (Dosovitskiy et al., 2020) and language (OpenAI, 2024), BC is also increasingly being used in efforts aiming to develop *robot foundation models* (Jang et al., 2022; Brohan et al., 2023; Black et al., 2024), scaling up both data and compute used to learn visuomotor policies suitable for real-world deployment across tasks and even robot embodiments.

Robot learning algorithms are often implemented as standalone components and their integration with the rest of the robotics stack remains challenging.

2.3 Practical Challenges for Robot Learning Research

Despite scientific advances, the robot learning ecosystem remains fragmented, impeding reproducibility and raising the barrier to entry for research.

- **Disaggregated Middleware:** While middleware abstractions are available, it is common to encounter middleware components tailored to specific platforms in practice. This heterogeneity often forces teams to develop bespoke adaptations, siloing efforts.
- **Datasets and Formats:** Large-scale datasets are typically shared in a different formats. Data is often released in varied formats like TensorFlow Datasets, ROS bags, or bespoke JSON layouts. The absence of a universal, modality-rich schema prevents the seamless aggregation of disparate datasets into larger mixtures.
- **Learning Frameworks:** The deep learning literature has consistently demonstrated that minor implementation differences in algorithms, data handling, and evaluation pipelines can lead to significant variance in results (Henderson et al., 2018). In robotics, these issues are compounded by hardware variability, further hindering reproducibility.

This ecosystem-wide fragmentation imposes significant incidental complexity on researchers, diverting resources from core scientific inquiry to systems integration. `lerobot`addresses these limitations by providing an end-to-end, open, and scalable library designed to unify hardware interfacing, collecting and streaming data, and training and deploying advanced policies with minimal engineering overhead.

3 Features

`lerobot` is designed for accessibility, scalability, and reproducibility in robot learning. The library natively integrates (1) entirely open-source hardware platforms costing a fraction of closed-source devices, (2) a unified middleware shared across low-level robot interfaces (3) data collection, storage and streaming tools, (4) an optimized inference engine and (5) many ready-to-use implementations of SOTA methods in robot learning, useful to both train models from scratch and re-use openly available pre-trained models. `lerobot` is entirely open-source, and highly accessible due to its reliance on low-cost teleoperation kits, focus on empowering large-scale datasets via streaming, and simple interface to adopt models in fully reproducible pipelines.

	Robot Type	Cost (€)
SO-100/101	Manipulator (Bimanual)	$\sim$ 225 (550)
Koch-v1.1	Manipulator (Bimanual)	$\sim$ 670 (1346)
ALOHA	Bimanual Manipulator	$\sim$ 21k
HopeJR-Arm	Humanoid Arm and hand	$\sim$ 500
LeKiwi	Mobile Manipulator	$\sim$ 230

(a) Cost for all robot platforms supported by `lerobot`and with an openly-available Bill Of Materials (BOM).

Robot	**# Downloads**	**# Datasets**	**# Episodes**
Panda	1'878'395	588	926'776
xArm	1'107'329	74	450'329
WidowX	832'177	100	214'117
KUKA	662'550	3	419784
SO-101	319'586	3'965	58'299
SO-100	278'697	5'161	78'510
Koch-v1.1	43'561	849	20'959

(b) All Top-4 robots for number of downloads, datasets and episodes openly shared, listed in decreasing order by the total number of downloads.

3.1 Accessible Real-world Robots

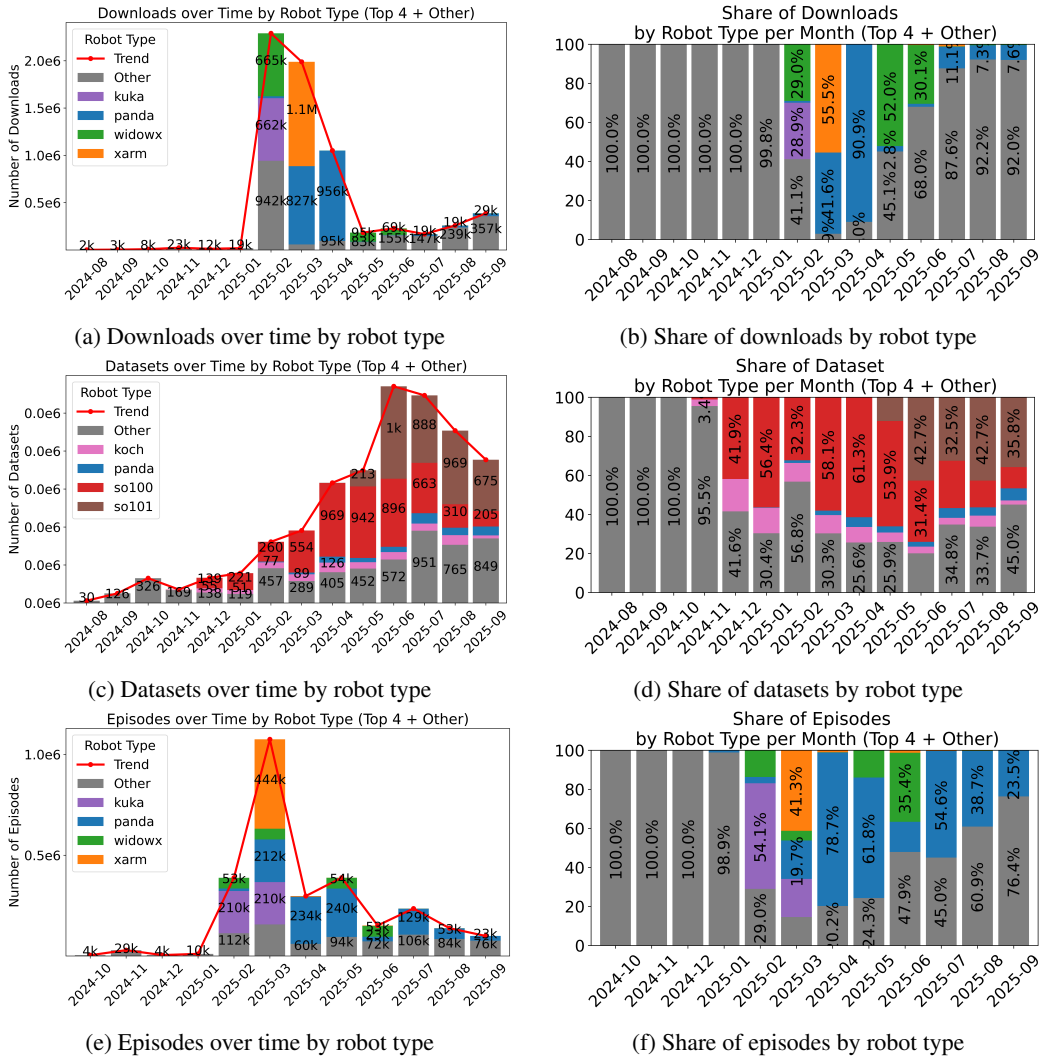
`lerobot` currently supports multiple real-world robot platforms for both static and mobile manipulation. The library fully integrates the SO-100 and SO-101 arms (Knight et al., 2024), both in a single and bimanual setup. The library also supports the Koch-v1.1 (Moss, 2025) and ALOHA-2 (Aldaco et al., 2024) manipulators, the Hope-JR humanoid arm (TheRobotStudio, 2025), the Stretch-3 (Hello Robot, 2025) and LeKiwi (SIGRobotics-UIUC, 2025) mobile manipulation platforms, and lastly the Reachy-2 humanoid (Mick et al., 2019). `lerobot` is designed to interface multiple open devices with a shared middleware that can be used to (1) read the configuration on a *leader* robot and write it on *follower* robot for teleoperation and (2) directly control the *follower* with a learned policy.

Table 1a shows the cost for all the robot platforms currently supported by `lerobot` with an openly-available Bill of Materials (BOM), reported for completeness in Appendix A. `lerobot` can support multiple robot platforms thanks to a shared middleware embedded in higher-level abstractions for the different robots supported, and engineered to interface directly with the low-level SDKs of major low-cost actuator producers (FeeTech and Dynamixel). Crucially, the middleware is designed to be easily extensible and highly composable. We refer to Appendix B for an example of teleoperation using `lerobot`.

3.2 Datasets

To address the fragmented nature of data in robotics research, we introduce `LeRobotDataset`, `lerobot`'s unified multimodal dataset schema. This standardized format is engineered to provide convenient and standardized access to robotics data spanning diverse modalities, including high-frequency sensorimotor readings, multiple camera feeds, and teleoperation status signals. The schema is designed to be self-contained, accommodating general metadata such as textual descriptions of the demonstrated tasks for filtering and language-conditioned policies, specifics of the robot embodiment considered, and relevant experimental parameters such as frames-per-second (FPS) of data capture and the types sensors used. As of September 2025, 16K+ datasets from 2.2K+ individual contributors are openly shared via the `LeRobotDataset` format, featuring robots directly integrated in the library such as the SO-10X arm and unsupported robots (Franka Emika Panda, xArm, R1Pro), ported to the `LeRobotDataset` format by the open-source community. We argue the support of different robot configuration underscores the flexibility of our dataset format, and that the coexistance of both large-scale academic benchmarks and small-scale data collection efforts exemplifies the breadth of use-cases that our dataset format can accomodate.

Open datasets are available for downloads, and Figure 5a shows the evolution of the number of downloads over time, with a breakdown of the share of downloads per robot type (Table 1b) and per robot type over time (Figure 5b, see Appendix C for further details). Despite `lerobot` only supporting a limited number of robots (grouped under the *Other* tag in Figure 5a and Figure 5e), datasets collected for other platforms such as the Franka Emika Panda and xArm lead in the number of downloads and size of the datasets collected (Figure 5e). We argue this follows from these platforms being often featured in research-oriented centralized data collection efforts (**?**Khazatsky et al., 2025). Conversely, platforms such as the SO-10X are increasingly featured in small-scale decentralized community efforts powered by the accessibility of (1) the hardware platforms used and (2) `LeRobotDataset` format, with 50%+ of the datasets contributed being collected directly on the SO-10X platforms (Figure 5d).



(a) Downloads over time by robot type

(b) Share of downloads by robot type

(c) Datasets over time by robot type

(d) Share of datasets by robot type

(e) Episodes over time by robot type

(f) Share of episodes by robot type

Figure 5: Numbers and trends of downloads, datasets, and episodes by robot type over time. The number of episodes in each dataset has been explicitly tracked starting in October 2024 only. For completeness, we report the top-5 robots grouped in *Other*, for each of the metrics considered, in Table 4.

A primary design principle of the dataset format is scalability. The dataset architecture is optimized to handle large-scale repositories potentially containing millions of expert trajectories. This unified interface for multi-modal, sequential data is designed for seamless integration with the PyTorch ecosystem, further promoting standardized and repeatable research workflows. This design is complemented by a native streaming capability designed to enhance accessibility: users can process remotely-hosted large-scale datasets without the prerequisite of downloading the entire corpus, thereby lowering barriers to entry for the broader community and improving on the accessibility of robot learning research. See Appendix C for more details on streaming.

```python
from lerobot.datasets.lerobot_dataset import LeRobotDataset
from lerobot.datasets.streaming_dataset import StreamingLeRobotDataset

repo_id = "lerobot/svla_so101_pickplace"
# Downloads the whole dataset and loads it in memory
# (allowing for random access)
dataset = LeRobotDataset(repo_id)

```

	# Params	Peak Memory CPU	MPS	RTX 4090	A100
ACT	52M	817.4MB	462MB	211.24 MB	211.24 MB
Diffusion Policy	263M	1.22GB	224MB	1.12 GB	1.12 GB
π_0	3.5B	4.13GB	97MB	13.32 GB	13.32 GB
SmolVLA	450M	1.69GB	555MB	1.75 GB	1.75 GB

Table 2: Peak memory consumption of policy models currently supported by `lerobot`. All models are run in full precision (fp32). Diffusion and Flow Models are run with 10 denoising steps at inference. All models maintain their original outputs shapes.

```
# Streams frames on the fly without downloading
# (access frames sequentially, .next())
dataset = StreamingLeRobotDataset(repo_id)
```

3.3 MODELS

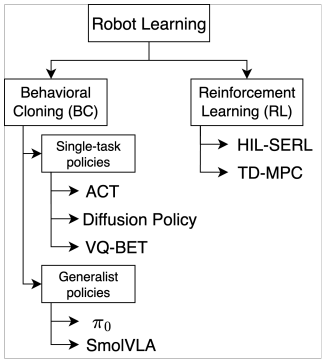


Figure 6: The different robot learning algorithms currently supported by `lerobot`.

`lerobot` supports reference implementation for multiple SOTA robot learning algorithms, providing useful baselines for experimentation and accessible models across RL, such as HIL-SERL (Luo et al., 2024) and TD-MPC (Hansen et al., 2022) and BC, both for single-task ACT (Zhao et al., 2023), Diffusion Policy (Chi et al., 2024) and VQ-BET (Lee et al., 2024), and multi-task models such as π_0 (Black et al., 2024) and SmolVLA (Shukor et al., 2025) (Figure 6).

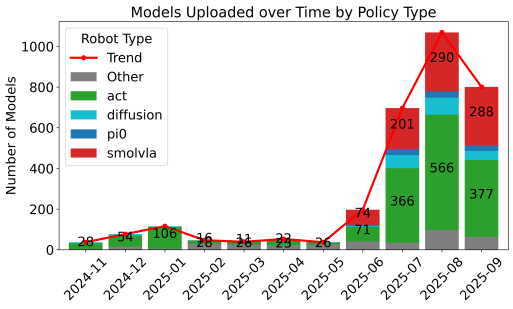
`lerobot` offers support for custom models too, grouped together under the *Other* tag in Figure 7. All the control policies implemented in `lerobot` are written in pure Pytorch (Paszke et al., 2019), and integrated with the library to allow (1) training models from scratch on datasets collected via real-world demonstrations, and (2) inference using openly available pre-trained models. The library is designed to for high accessibility, providing a composable set of recipes which can be used to train a model from scratch in less than 100 lines-of-code (LOC), and serve models in less than 40 LOC (Appendix D).

In its effort to foster accessibility, `lerobot` supports multiple models with different computational constraints, ranging from lightweight single-task models to larger, multi-task models. ACT (Zhao et al., 2023) is a particularly popular model dominating the number of uploads (Figure 7a), consistenly ranking as one of the most popular policies trained (Figure 7b) and used (Figure 7d). We ascribe the popularity of ACT to (1) its small size and fast inference speed and (2) straightforward application to limited amount of real-world demonstrations, allowing users to obtain well-performing policies with as little as 50 real-world trajectories. As a single-task model, however, ACT necessitates retraining whenever changes in the experimental conditions occur. SmolVLA (Shukor et al., 2025) is a powerful, small-scale Vision-Language-Action model which allows to control real-world robots via language conditioning, resulting in an overall wider applicability to practical scenarios.

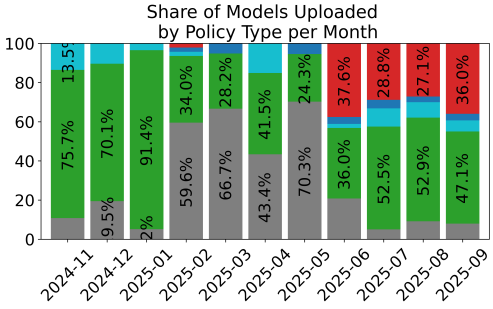
Table 2 and Table 3 report the peak memory footprint and the average inference latency, measured over 100 test samples, for the most widely used policies supported by `lerobot`. Evaluations were conducted on four platforms: (1) a MacBook Pro M1 (2021, 16GB, CPU only), (2) the same MacBook Pro with the MPS backend, (3) an NVIDIA RTX 4090, and (4) an NVIDIA A100. All models were executed in full `fp32` precision at runtime, with inference timed-out after 5 seconds. Overall, peak memory footprints largely align with theoretical estimates obtained from the combination of model parameter count and numerical precision. The main exceptions are the CPU and MPS backends, where unified memory and frequent offloading to swap introduce variability, obscuring direct performance comparisons and increasing latency. Latency measurements are averaged across all non—timed-out trials, with both mean and standard deviation reported in Table 3. Smaller, task-specific models such as ACT exhibit high efficiency on accelerated backends like MPS and achieve inference rates of $\sim$100-200Hz on high-end GPUs such as the RTX 4090 and A100. Crucially,

	# Params	Avg Inference Latency (ms) CPU	MPS	RTX 4090	A100
ACT	52M	182.313 ± 40.82	42.667 ± 10.085	5.013 ± 0.061	13.77 ± 0.445
Diffusion Policy	263M	(100%)	3453.838 ± 39.271	369.788 ± 0.193	613.893 ± 10.173
π_0	3.5B	(100%)	(100%)	209.381 ± 2.762	568.978 ± 2.937
SmolVLA	450M	2028.461 ± 302.59 (2%)	721.826 ± 57.748	99.244 ± 1.195	278.833 ± 1.886

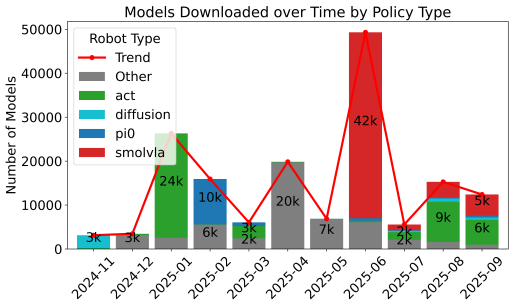
Table 3: Average and standard deviation inference latency over 100 forward passes for policy models currently supported by lerobot. Diffusion and Flow Models are run with 10 denoising steps at inference time. (x%) indicates the percentage of samples that timed-out before the 5000ms hard stop (0% omitted).



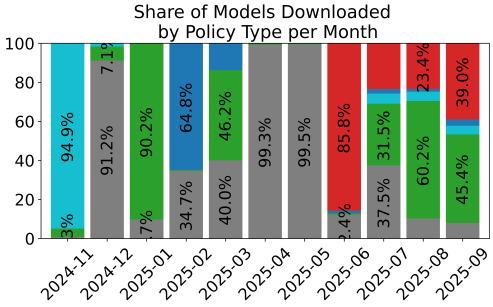
(a) Models uploaded over time by policy type.



(b) Share of the models uploaded over time by policy type.



(c) Models downloaded over time by policy type.



(d) Share of models downloaded over time by policy type.

Figure 7: Numbers and trends of uploads and downloads of robot learning models by policy type over time. TD-MPC (Hansen et al., 2022), HIL-SERL (Luo et al., 2024) and VQ-BET (Lee et al., 2024) are absent from all visualizations as they are not typically uploaded by users.

larger models such as π_0 require substantially longer per each forward passes on average on all platforms, and even fail to complete inference within the 5s limit on lower-tier devices, underscoring the challenges in deploying robotics foundation models in practice.

3.4 Inference

`lerobot` defines a custom inference stack which is designed to decouple action prediction (*inference*) from action execution (*control*), at both the physical and logical level (Figure 8). This optimized stack is designed for modern robot learning policies, increasingly predicting sequences of actions (*action chunks*, $a_{t:t+H-1}$, (Zhao et al., 2023)) rather than single controls. All the BC policies supported by `lerobot` predict action chunks.

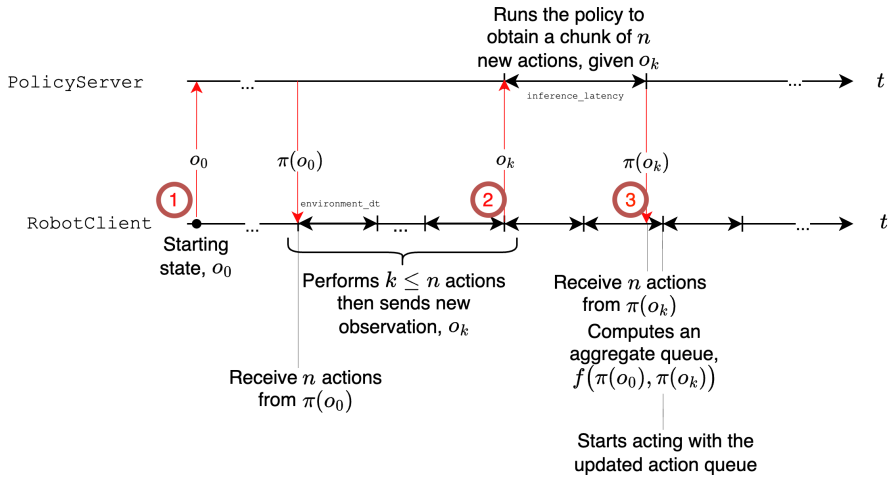


Figure 8: Overview of the generalized inference schema supported by `lerobot`, whereby a remote server can be used to host compute-expensive policies for inference, while the robot client receives a stream of the actions chunks to enact. The schema provides scalability and flexibility through the possibility to fully customize the function f used to aggregate overlapping chunks.

Physical decoupling allows inference to run on a remote machine connected over the network to the robot's low-level controller. This design enables the use of higher-end computational resources than those typically available aboard a robot for inference, while control is maintained at the desired control frequency stepping through the multiple actions received. Further, *logical* decoupling implements inference via an *asynchronous* producer-consumer scheme: the inference process predicts *action sequences* with a look-ahead horizon H *in parallel* with environment control, which consumes actions at a fixed control rate. Overlapping predictions are merged via a generalized aggregation function f, which users can easily specify for their own use cases, ensuring a non-empty action queue and preventing idleness of the robot by overlaying action prediction and action execution. We refer to Appendix E for more details on the performance of decoupled inference.

4 Conclusions

In this work we introduced `lerobot`, a unified, open-source stack for end-to-end robot learning that bridges low-level control, large-scale data tooling, and scalable learning algorithms. We showed how accessible teleoperation of multiple real-world robot through a shared middleware can be used to collect real-world data across a variety of robot platforms. Further, we illustrated how standardized datasets can be exploited to collect and reuse data at scale, powering advancements in robot learning thanks to the thousands of datasets collected, resulting in hundreds of thousands of episodic data, and hundreds of models openly contributed by the robot learning community.

Limitations We identify several limitations remaining in our contribution. First, robots coverage is currently far from exhaustive, as we support a practical but incomplete set of arms, grippers, sensors, and controllers. Over the course of 2025, `lerobot` went from supporting 3 manipulation setups (Koch-v1.1, SO-100, ALOHA) to the 8 regular, humanoid and mobile manipulators currently supported, and we highlight that keeping a similar rate of progress is paramount to properly serve the robot learning community. Second, the coverage in terms of robot learning algorithms is also non-exhaustive. We provide strong, reproducible implementations across key paradigms, while extending the library with additional algorithms remains future work. Third, achieving strong practical inference performance still requires low-level optimization (quantization, graph compilation, etc) that are currently disregarded by the library. We view these limitations as concrete, tractable avenues for community contributions and future development, and in the very spirit of open-source, invite the broader robot learning community to address them. However, despite these limitations, our work takes a significant step toward an end-to-end stack for robot learning, providing a useful tool for researchers and practioners in the field.

REFERENCES

Jorge Aldaco, Travis Armstrong, Robert Baruch, Jeff Bingham, Sanky Chan, Kenneth Draper, Debidatta Dwibedi, Chelsea Finn, Pete Florence, Spencer Goodrich, et al. Aloha 2: An enhanced low-cost hardware for bimanual teleoperation. *arXiv preprint arXiv:2405.02292*, 2024.

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.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots, March 2025.

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.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale, August 2023.

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.

Open X.-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I. Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana

Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Muhammad Zubair Irshad, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J. Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R. Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vikash Kumar, Vincent Vanhoucke, Vitor Guizilini, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic Learning Datasets and RT-X Models, May 2025.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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*, pp. 158–168. PMLR, January 2022.

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.

Hello Robot. Stretch 3®: A fully integrated mobile manipulator. `https://hello-robot.com/stretch-3-product`, 2025. URL `https://hello-robot.com/stretch-3-product`. Last accessed: 22 September 2025.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

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

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning, February 2022.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree

Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R. Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J. Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset, April 2025.

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, 2024.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

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

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling, February 2023.

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.

Sébastien Mick, Mattieu Lapeyre, Pierre Rouanet, Christophe Halgand, Jenny Benois-Pineau, Florent Paclet, Daniel Cattaert, Pierre-Yves Oudeyer, and Aymar de Rugy. Reachy, a 3d-printed human-like robotic arm as a testbed for human-robot control strategies. *Frontiers in neurorobotics*, 13:65, 2019.

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

Jess Moss. koch-v1.1: A version 1.1 of the alexander koch low cost robot arm with some small changes, September 22 2025. URL `https://github.com/jess-moss/koch-v1-1`. GitHub repository, Apache-2.0 license.

OpenAI. GPT-4 Technical Report, March 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

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 (eds.). *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.

SIGRobotics-UIUC. LeKiwi: Low-Cost Mobile Manipulator. `https://github.com/SIGRobotics-UIUC/LeKiwi`, September 22 2025. URL `https://github.com/SIGRobotics-UIUC/LeKiwi`. GitHub repository, Apache-2.0 license.

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.

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.

TheRobotStudio. HOPEJr/Arm: Robotic Arm Module of HOPEJr. `https://github.com/TheRobotStudio/HOPEJr/tree/main/Arm`, September 22 2025. URL `https://github.com/TheRobotStudio/HOPEJr/tree/main/Arm`. GitHub repository, accessed: 22 September 2025.

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

A Openly-available robots

- **SO-10X** Guide from Knight et al. (2024): `https://github.com/TheRobotStudio/SO-ARM100`.
- **Koch-v1.1** Guide from Moss (2025): `https://github.com/jess-moss/koch-v1-1`
- **ALOHA** Guide from Zhao et al. (2023) here.
- **HopeJR-Arm** Guide from TheRobotStudio (2025): `https://github.com/TheRobotStudio/HOPEJr/blob/main/Arm/BOM.md`
- **LeKiwi** Guide from SIGRobotics-UIUC (2025): `https://github.com/SIGRobotics-UIUC/LeKiwi/blob/main/BOM.md`

B Real-world Robots API

```python
from lerobot.teleoperators.so100_leader.so100_leader import \
    SO100Leader
from lerobot.teleoperators.so100_follower.so100_follower import \
    SO100Follower

teleop = SO100Leader()
# (provided teleop matches) can also be Reachy-2, LeKiwi, etc.
robot = SO100Follower()

teleop.connect()
robot.connect()

action = teleop.get_action()
print(action)
# {'shoulder_pan.pos': 84.74,
#  'shoulder_lift.pos': 4.95,
#  'elbow_flex.pos': 70.6,
#  'wrist_flex.pos': -88.41,
#  'wrist_roll.pos': 57.89,
#  'gripper.pos': 4.13}

robot.send_action(action)  # moves robot according to `action`
```

Robot	# Datasets
unknown	2370
lekiwi	535
arx5	371
aloha	334
aiworker	202

(a) Top-5 robot platforms in the "Other" category for number of datasets.

Robot	# Downloads
unknown	711729
google_robot	438560
so101	319586
so100	278697
aloha	45219

(b) Top-5 robot platforms in the "Other" category for number of downloads.

Robot	# Episodes
google_robot	213852
unknown	170706
so100	78510
so101	58299
easo	45652

(c) Top-5 robot platforms in the "Other" category for number of episodes.

Table 4: Breakdown of the *Other* category by top-5 robot platforms across datasets, downloads, and episodes.

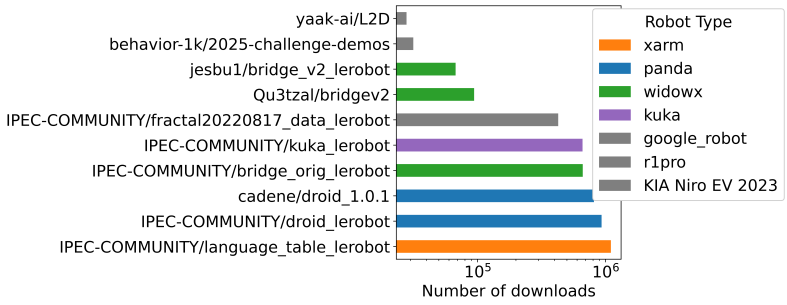


Figure 9: Openly-available datasets with the largest number of downloads using the `LeRobotDataset` format. The most downloaded datasets are academic benchmarks released by the research community (Collaboration et al., 2025; Khazatsky et al., 2025).

C Datasets

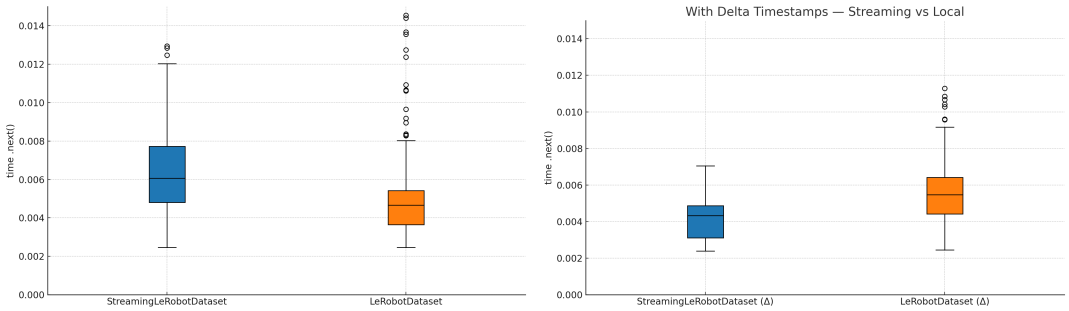
Table 4 further breaks down the *Other* category for the number of downloads, datasets and episodes, and it shows how faulty dataset that do not explicitly record the robot platform used (tagged as *unknown*) dominate in the *Other* category.

Figure 9 shows the most downloaded datasets by robot type. Crucially, the largest number of downloads is not achieved for a platform natively integrated in `lerobot`, further undescoring the adoption of the `LeRobotDataset` format in the robotics community.

C.1 Streaming Datasets

The development of `StreamingLeRobotDataset` addresses several fundamental challenges associated with the efficient utilization of large-scale robotic datasets in robot learning pipelines. Traditional approaches to dataset handling rely on pre-loading data into local memory, which becomes increasingly impractical as datasets grow to the million-episodes scale. `StreamingLeRobotDataset` supports a streaming paradigm, whereby *frames*—defined as individual items in a dataset—are fetched on-demand from remote storage rather than preloaded in their entirety. This architectural shift required addressing three core challenges: (1) efficient data access under strict memory constraints, (2) ensuring sufficient randomness during iteration to support robust learning, and (3) enabling multi-frame retrieval in a setting that is inherently sequential and non-indexable.

Efficient Streaming of Large-Scale Data. The `LeRobotDataset`format partitions robotic data into tabular records (`.parquet` files) and compressed videos (`.mp4` files), alongside lightweight metadata. Metadata files are downloaded fully due to their negligible size relative to the dataset, but all high-volume video and control streams are processed on demand. This is achieved through two principal design choices: (1) adoption of an `IterableDataset` interface, and (2) integration with `torchcodec` for on-the-fly video decoding. These components together enable data consumption through simple iterative calls, while maintaining memory usage bounded irrespective of dataset size.



(a) Timing performance of stepping through single frames of a `StreamingLeRobotDataset` compared to a pre-loaded `LeRobotDataset`.

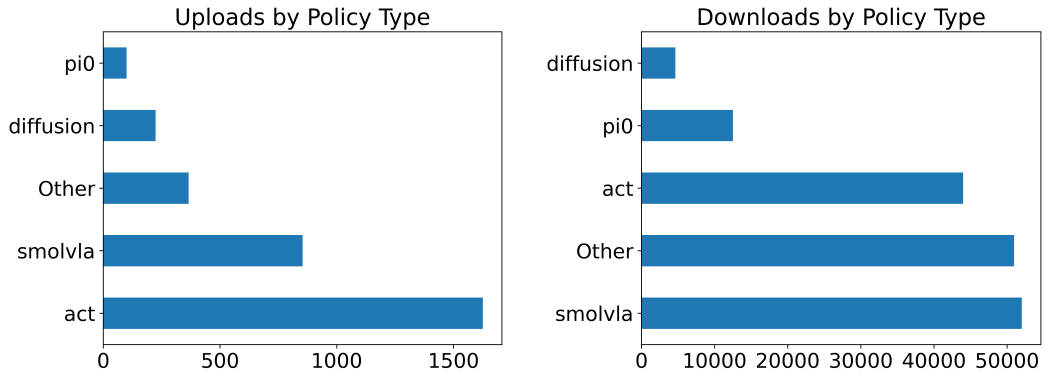
(b) Timing performance of stepping through a dataset retrieving multiple frames of a `StreamingLeRobotDataset` compared to a pre-loaded `LeRobotDataset`.

Figure 10: Timing performance of `StreamingLeRobotDataset` versus a regular `LeRobotDataset`.

Provided a good network connectivity, Figure 10 shows timing performance is comparable between the two formats in the steady-state regime (after initialization).

C.2 Example: Use a Dataset

```python
import torch
from lerobot.datasets.lerobot_dataset import LeRobotDataset

delta_timestamps = {
    # 0.2, and 0.1 seconds *before* each frame
    "observation.images.wrist_camera": [-0.2, -0.1, 0.0]
}

# Optionally, use StreamingLeRobotDataset to avoid downloading the dataset
dataset = LeRobotDataset(
    "lerobot/svla_so101_pickplace",
    delta_timestamps=delta_timestamps
)

# Get frame in the dataset by their index
sample = dataset[0]
print(sample)
# {
# 'observation.state': tensor([...]),
# 'action': tensor([...]),
# # extra dimension due to delta timesteps
# 'observation.images.wrist_camera': tensor([3, C, H, W])
# ...
# }

batch_size=16
# wrap the dataset in a DataLoader for training/inference purposes
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=batch_size
)

# Iterate over the DataLoader in a training loop
num_epochs = 1
device = "cuda" if torch.cuda.is_available() else "cpu"

for epoch in range(num_epochs):
    for batch in data_loader:
```



(a) Models uploaded by policy type. Policies not present have not been publicly uploaded.

(b) Models downloaded by policy type. Policies not present have not been publicly downloaded.

```
        # Move data to the appropriate device (e.g., GPU)
        observations = batch["observation.state"].to(device)
        actions = batch["action"].to(device)
        images = batch["observation.images.wrist_camera"].to(device)

        # Next, process the data for training or inference
        ...
```

C.3 Example: Use a Streaming Dataset

```
from lerobot.datasets.streaming_dataset import StreamingLeRobotDataset

# Streams frames on the fly without downloading the dataset
dataset = StreamingLeRobotDataset(
    "lerobot/svla_so101_pickplace",
    delta_timestamps=delta_timestamps
)
```

D Models

D.1 Example: Train a Model

```
import torch

from lerobot.configs.types import FeatureType
from lerobot.datasets.lerobot_dataset import (
    LeRobotDataset, LeRobotDatasetMetadata
)
from lerobot.datasets.utils import dataset_to_policy_features
from lerobot.policies.factory import make_pre_post_processors

# Users can use many plug-in policies from the library
from lerobot.policies.diffusion.configuration_diffusion import \
    DiffusionConfig
from lerobot.policies.diffusion.modeling_diffusion import DiffusionPolicy

output_directory = "outputs/train/example_pusht_diffusion"
device = torch.device("cuda")
training_steps = 5000
log_freq = 1

repo_id = "lerobot/pusht" # Replace with your dataset
dataset_metadata = LeRobotDatasetMetadata(repo_id)

features = dataset_to_policy_features(dataset_metadata.features)
```

```
output_features = {
    key: ft for key, ft in features.items()
    if ft.type is FeatureType.ACTION
}
input_features = {
    key: ft for key, ft in features.items()
    if key not in output_features
}

cfg = DiffusionConfig(
    input_features=input_features,
    output_features=output_features
)

policy = DiffusionPolicy(cfg)
policy.train()
policy.to(device)
preprocessor, postprocessor = make_pre_post_processors(
    cfg, dataset_stats=dataset_metadata.stats
)


delta_timestamps = {
    "observation.image": [-0.1, 0.0],
    "observation.state": [-0.1, 0.0],
    "action": [
        -0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
        0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4
    ],
}

dataset = LeRobotDataset(repo_id, delta_timestamps=delta_timestamps)

optimizer = torch.optim.Adam(policy.parameters(), lr=1e-4)
dataloader = torch.utils.data.DataLoader(
        dataset,
        num_workers=4,
        batch_size=64,
        shuffle=True,
        pin_memory=device.type != "cpu",
        drop_last=True,
    )

step = 0
done = False
while not done:
    for batch in dataloader:
        batch = preprocessor(batch)
        loss, _ = policy.forward(batch)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if step % log_freq == 0:
            print(f"step: {step} loss: {loss.item():.3f}")
        step += 1
        if step >= training_steps:
            done = True
            break

# Save a policy checkpoint.
policy.save_pretrained(output_directory)
preprocessor.save_pretrained(output_directory)
postprocessor.save_pretrained(output_directory)
```

Inference	Success Rate (%)			
	Pick-Place	Stacking	Sorting	Avg
Sync	75	90	70	78.3
Async	80	90	50	73.3

(a) Performance (success rates).

Inference	Time (s)		
	Total	Avg	Std
Sync	137.5	13.75	2.42
Async	97.0	9.70	2.95

(b) Task completion time.

Inference	# of Cubes		
	Total	Avg	Std
Sync	9	1.8	0.45
Async	19	3.8	1.3

(c) Performance in fixed time (60s per each episode).

Table 5: Comparison between regular (Sync) and optimized (Async) inference. We evaluate the SmolVLA implementation provided in `lerobot` on three real-world performed using the SO-100 arm, consisting of (1) pick and place cubes (2) stacking cubes on top of each other and (3) sorting cubes. `lerobot`'s decoupled inference schema achieves similar success rates (left) but results in significantly reduced cycle times (middle) and thus higher throughput (right), over the 10 test episodes (60s each) for the task considered.

D.2 Example: Use a Pre-trained Model

```python
from typing import Any
from lerobot.policies.smolvla.configuration_smolvla import \
    SmolVLAConfig
from lerobot.policies.smolvla.modeling_smolvla import SmolVLAPolicy
from lerobot.datasets.lerobot_dataset import \
    LeRobotDatasetMetadata

from lerobot.policies.factory import make_pre_post_processors
from lerobot.teleoperators.so100_follower.so100_follower import \
    SO100Follower

# Take a dataset on which SmolVLA was trained, for normalization
repo_id = "lerobot/svla_so101_pickplace"
dataset_metadata = LeRobotDatasetMetadata(repo_id)

cfg = SmolVLAConfig()
policy = SmolVLAPolicy(cfg)
preprocessor, postprocessor = make_pre_post_processors(
    cfg, dataset_stats=dataset_metadata.stats
)

robot = SO100Follower(...)
raw_obs: dict[str, Any] = robot.get_observation()

# Preprocess the observation for inference
policy_input = preprocessor(raw_obs)
# Select the action from the policy
policy_output = policy.select_action(policy_input)
# Postprocess the action for the robot
policy_action = postprocessor(policy_output)

robot.send_action(policy_action)
```

E Inference

Optimized inference accelerate cycle times across multiple tasks with comparable performance (Table 5), and provide a scalable path to higher model capacity without compromising on real-time control, provided access to a network. In particular, the speedup presented in Table 5 derives from *logical* decoupling—asynchronously computing the next chunk while the current one has not been exhausted yet—rather than physical decoupling, as both the server and client run on the same machine, though in principle the inference stack allows for communication between different machines.

E.1 Example: Host a Remote Server

```python
# Run this script to start the policy server on any machine
from lerobot.scripts.server.configs import PolicyServerConfig
from lerobot.scripts.server.policy_server import serve

config = PolicyServerConfig(
    host="localhost",
    port=8080,
)
serve(config)
```

E.2 Example: Stream Actions to a Robot

```python
# Run this script to start the robot client on the robot's computer
import threading
from lerobot.scripts.server.configs import RobotClientConfig
from lerobot.scripts.server.robot_client import RobotClient


camera_cfg = ...   # cameras used by the visuomotor policy
robot_cfg = ...  # a given robot supported by the library

# 3. Create client configuration
client_cfg = RobotClientConfig(
    robot=robot_cfg,
    # attach to the server running the policy
    server_address="localhost:8080",
    # use a higher-end device for inference
    policy_device="cuda:0",
    policy_type="pi0",
    pretrained_name_or_path="lerobot/pi0"
)

# 4. Create and start client
client = RobotClient(client_cfg)

task = ... # Specify the task using natural language

if client.start():
    # Start action receiver thread
    action_receiver_thread = threading.Thread(
        target=client.receive_actions, daemon=True
    )
    action_receiver_thread.start()

    try:
        # Run the control loop
        client.control_loop(task)
    except KeyboardInterrupt:
        client.stop()
        action_receiver_thread.join()
```