Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

USER: A Unified and Extensible System for Real-World Online Policy Learning in Embodied AI

Hongzhi Zang[1*], Shu'ang Yu[17*], Hao Lin[2*], Tianxing Zhou[35], Zefang Huang[45], Zhen Guo[2], Xin Xu[1], Jiakai Zhou[1], Yuze Sheng[1], Si Xu[2], Shizhe Zhang[1], Feng Gao[16], Wenhao Tang[1], Yufeng Yue[3], Quanlu Zhang[2], Xinlei Chen[1], Chao Yu[1#†], and Yu Wang[1†]

[1]Tsinghua University [2]Infinigence AI [3]Beijing Institute of Technology
[4]Zhejiang University [5]Zhongguancun Academy [6]Striding.AI [7]Shanghai AI Laboratory
[*]Equal Contribution [#]Project Leader
[†]Corresponding Authors: yuchao@sz.tsinghua.edu.cn, yu-wang@mail.tsinghua.edu.cn
Code: https://github.com/RLinf/RLinf.

***Abstract*—Online policy learning directly in the physical world is a promising yet challenging direction for embodied intelligence. Unlike simulation, real-world systems cannot be arbitrarily accelerated, cheaply reset, or massively replicated, suggesting that real-world policy learning is not merely an algorithmic problem, but inherently a systems problem. We present USER, a Unified and extensible SystEm for real-world online policy leaRning. On the systems side, USER introduces a hardware abstraction layer for unified robot management and an adaptive communication plane that enables efficient cloud-edge training. On the learning side, USER adopts a fully asynchronous training framework, designs a persistent and cache-aware replay buffer, and provides extensible abstractions for rewards, algorithms, and policies. Experiments in both simulation and the real world demonstrate that USER supports multi-robot coordination, heterogeneous manipulators, cloud-edge training with large models, and long-running asynchronous training. Together, these capabilities establish USER as a unified and extensible systems foundation for real-world online policy learning.**

I. Introduction

A common paradigm for online policy learning in embodied intelligence is to train policies in simulation and then deploy them in the real world [29, 27]. However, unavoidable gaps in dynamics, sensing, and interaction often cause policies to degrade significantly after transfer [14, 29]. This has motivated growing interest in learning policies directly in the physical world. Nevertheless, unlike simulation, real-world online learning cannot be arbitrarily accelerated, reset, or replicated. Robots operate in real time, platforms are heterogeneous, networks are unstable, and experiments are long-running and frequently interrupted [21]. These properties make real-world online policy learning not merely an algorithmic challenge, but fundamentally a systems problem that tightly couples physical execution, communication, and optimization.

Several system-level challenges limit scalable real-world learning. First, robots are often treated as external environments, which makes it hard to jointly schedule robots and accelerators in distributed deployments. Second, real-world online learning increasingly relies on cloud–edge infrastructures, particularly for large-scale Vision-Language-Action (VLA) models, with rollout at the edge and training in the cloud. These components span heterogeneous and isolated network domains, causing bandwidth asymmetry and cross-domain latency [10]. Moreover, the physical world cannot be accelerated, making data efficiency the main bottleneck. Synchronous pipelines commonly used in simulation therefore result in poor learning efficiency [5]. Third, embodied learning is moving toward data-driven training with high-dimensional visual streams and long horizons, which greatly increase data volume and experiment duration. However, existing buffers and pipelines remain memory-centric and short-lived, lacking support for persistence, recovery, and cross-phase data reuse [30].

These challenges call for rethinking real-world robot learning from a systems perspective. Practical physical-world learning should not rely solely on isolated algorithmic advances, but instead require unified abstractions and extensible infrastructure. In this paper, we present USER, a Unified and extensible SystEm for online Real-world policy learning.

At the system level, USER unifies physical robots, GPUs, and other accelerators under a common hardware abstraction layer, enabling automatic discovery, uniform management, and flexible scheduling of heterogeneous resources for diverse learning workloads. To support large-scale real-world learning across cloud–edge domains, USER further introduces an adaptive communication plane that provides tunneling-based cross-domain connectivity, distributed data channels for traffic localization and bandwidth reduction, and streaming multiprocessor (SM)-budgeted weight synchronization to regulate GPU communication overhead during training.

Beyond system architecture, USER organizes the real-world policy learning as a fully asynchronous learning framework, in which data generation, training, data transmission, and weight synchronization proceed independently rather than through tightly synchronized stages. To support long-horizon and data-intensive learning, USER introduces a persistent

TABLE I: Comparison between USER and other online learning systems.

	Real robot	Multiple robot	Heterogeneous robot	Large model	Open source
SERL [21]	✔	✘	✘	✘	✔
SOP [26]	✔	✔	✘	✔	✘
Qt-Opt [16]	✔	✔	✘	✘	✘
RLlib [19]	✘	✘	✘	✘	✔
TMRL [6]	✘	✘	✘	✘	✔
USER (ours)	✔	✔	✔	✔	✔

and cache-aware buffer that enables streaming trajectory ingestion, recovery, and reuse across training phases. Unlike traditional memory-centric pipelines, this design persistently stores data, enabling long-running experiments to recover from network interruptions and temporary training pauses. On top of this framework, USER provides extensible abstractions for policies, algorithms, and rewards. Rewards can be specified via rules, human feedback, or learned reward models, while policies range from conventional CNN policies to flow-based generative policies and large VLA models, and support both imitation and reinforcement learning algorithms. Together, USER enables researchers to explore diverse real-world learning paradigms without re-engineering the underlying system infrastructure.

Our contributions are summarized as follows:

- A unified and extensible systems framework for real-world online policy learning, featuring a hardware abstraction layer that treats robots as schedulable resources alongside accelerators, together with an adaptive communication plane for stable and efficient cloud–edge deployment across heterogeneous and isolated network domains.
- A fully asynchronous learning framework with persistent and cache-aware data management, as well as extensible abstractions for rewards, algorithms, and policies, enabling scalable online imitation and reinforcement learning with CNN, generative, and large VLA policies in real-world settings.
- Extensive simulation and real-world experiments demonstrating support for multi-robot coordination, heterogeneous embodiment, cloud-edge collaboration, and long-running asynchronous training under realistic deployment conditions.
- An open-source implementation of USER that provides a reusable systems foundation for real-world online policy learning.

II. Related Work

A. Robot Learning Systems

The rapid progress of embodied intelligence has been driven by high-parallel simulators [23, 24, 2] and large-scale learning frameworks [25, 18, 13, 31]. However, these systems are simulation-centric and rely on synchronous pipelines. In real-world learning, the physical environment cannot be accelerated, making data efficiency the bottleneck and synchronous

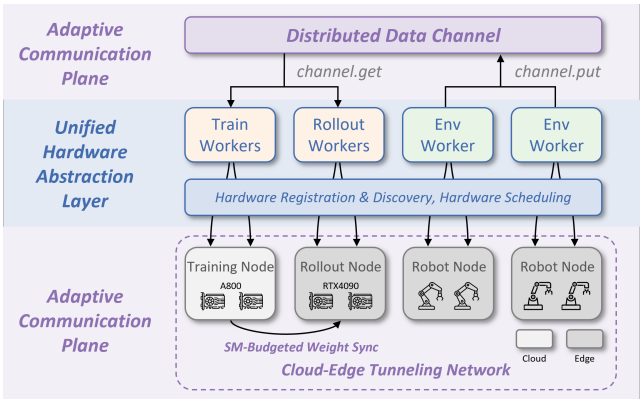


Fig. 1: The system architecture design of USER.

execution inefficient. To address this, USER adopts a fully asynchronous pipeline that lets robots execute continuously while learning runs in parallel. Beyond asynchrony, scaling real-world learning requires multi-robot parallelism with proper abstraction and communication. ROS2 [22] and Zenoh [9] offer connectivity but limited learning orchestration, while systems such as SERL [21] and Qt-Opt [16] mainly support single-robot or small-model settings. The most recent related work, SOP [26], supports large-model training across multiple robots but does not support heterogeneous fleets. Tab. I provides comparisons between USER and existing frameworks.

While many components have appeared in prior frameworks, USER uniquely provides *open-source* support for real-world training on heterogeneous robot fleets with large-model optimization. It natively supports geographically distributed, heterogeneous, cloud-edge training, thereby unlocking real-world RL for emerging robot foundation models.

B. Data Management for Long-Horizon Learning

High-performance, memory-centric replay buffers, exemplified by Reverb [8] and Flashbax [28], leverage volatile RAM to achieve extreme sampling throughput. However, this reliance on memory constrains their ability to support the management requirements of real-world policy learning systems characterized by long-horizon and massive visual data, particularly with the emergence of VLA models. While recent works such as LeRobot Dataset [7] have attempted to facilitate efficient streaming access to static datasets via metadata, the systematic handling of large-scale dynamic data during active policy learning remains unresolved. To address this gap, USER proposes a persistent and cache-aware buffer that maintains high-throughput sampling capabilities while supporting asynchronous disk persistence, enabling arbitrarily large datasets with efficient revisiting of historical data, as well as long-horizon experiments with robust crash recovery in real-world training pipelines.

III. System Architecture Design

Fig. 1 illustrates the system architecture of USER, featuring a unified hardware abstraction layer and an adaptive communication plane. Overall, USER abstracts each functional component in online learning as a *worker*, such as the env

worker for interaction, the rollout worker for inference, and the actor worker for updating model. The placement, scheduling, and communication discussed below all operate at the worker level.

A. Unified Hardware Abstraction Layer

```yaml
cluster:
  num_nodes: 3
  component_placement:
    actor:
      node_group: "4090"
      placement: 0
    env:
      node_group: franka
      placement: 0-1
    rollout:
      node_group: "4090"
      placement: 0:0-1

node_groups:
  - label: "4090"
    node_ranks: 0-0
  - label: franka
    node_ranks: 1-2
    hardware:
      type: Franka
      configs:
        - robot_ip: ROBOT_IP_FOR_RANK1
          node_rank: 1
          camera_serials: ["camera_serial_1", "camera_serial_2"]
        - robot_ip: ROBOT_IP_FOR_RANK2
          node_rank: 2
          camera_serials: ["camera_serial_1", "camera_serial_2"]
```

Fig. 2: Example robot and accelerator configurations. The system contains two environment nodes, corresponding to two robots, each equipped with two cameras. A single RTX 4090 is shared by both the rollout node and the training node (actor in the figure).

In pursuit of scalable real-world policy learning, the core idea of USER is to treat physical robots at the same resource level as accelerators (e.g., GPUs and TPUs). To this end, USER abstracts and manages all types of robots and accelerators uniformly as schedulable hardware units in a special *hardware abstraction layer* (HAL). The HAL provides unified interfaces for extending new types of hardware, automatically discovering available hardware resources, and scheduling hardware for different learning tasks.

Node and Hardware Abstraction. USER models a deployment as a cluster of nodes, each exporting a set of hardware units. A *hardware unit* is the scheduler's atomic allocatable entity—a single GPU device, or a single physical robot, optionally bundled with its required peripherals such as cameras and a spacemouse. Each unit is described by a lightweight, typed descriptor (hardware type and model) and configuration metadata such as robot network identity and sensor bindings. A *node* is inherently heterogeneous, potentially hosting multiple types of hardware units, like different types of robots and GPUs, to maximize flexibility in deployment. In USER, we typically deploy three types of nodes: *rollout nodes* equipped with GPUs for policy inference, *robot nodes* running on CPU-only machines for action execution at the edge, and *training nodes* with large-scale accelerators for centralized training. Nodes can be organized into *node groups* to capture this heterogeneity, where each group contains homogeneous hardware units, and a node may belong to multiple groups, depending on its hardware composition. In this manner, hardware scheduling policies can reason over homogeneous pools while still supporting a heterogeneous cluster. Fig. 2 shows an example hardware configuration.

Hardware Registration and Discovery. The HAL uses a pluggable *checker* interface to register and extend new hardware. Each hardware type supplies a *HAL checker* that defines

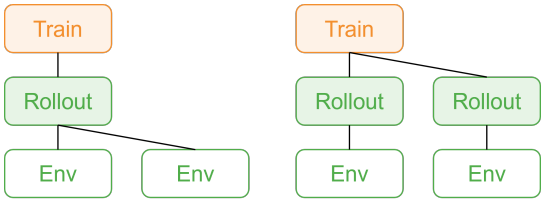


Fig. 3: USER supports flexible placement of train, rollout, and environment workers.

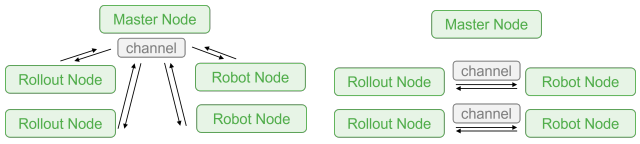
(i) its type identifier, (ii) how to discover the hardware on a node, and (iii) what metadata is attached to each instance. This modularity isolates device-specific logic (GPU vendor tooling, robot connectivity checks, sensor discovery) from the rest of the system. At cluster initialization, USER constructs a global hardware inventory by launching a lightweight *hardware probe* process on each node and invoking the registered HAL checkers to discover available hardware. This discovery can either be *automatic* (typically for PCIe- or USB-connected devices like GPUs, cameras and space mouse), or *configuration-driven* (typically for IP-bound robots) due to the need for explicit bindings and safety checks. For physical robots, discovery includes optional validation like network reachability, presence of required cameras, and basic health checks, ensuring that only usable robots enter the schedulable pool. The resulting inventory exposes for each node the available units and their ranks, forming the substrate for subsequent scheduling.

Hardware Scheduling. USER schedules robots and accelerators through a single rank-based placement interface—each component selects node groups and a set of resource ranks, where ranks resolve to accelerator or robot units within the node groups' nodes. The scheduler then deterministically maps process ranks to these resource ranks (evenly sharing units or assigning multiple units per process) and launches each process with an explicit binding to only its assigned hardware (e.g., constrained visible GPU devices or injected robot endpoint configuration). This uniform mechanism enables heterogeneous placements in one job, such as training on one GPU pool while binding different subsets of rollout processes to different types of robots.

This placement flexibility is particularly important in real-world deployments with complex network topologies. For example, in the two-env case shown in Fig. 3, both configurations are feasible: the left configuration reduces weight synchronization overhead, while the right configuration reduces communication overhead between the environment and rollout nodes. As a result, the optimal placement strategy depends on the actual bandwidth and communication characteristics between nodes. USER enables exploring and realizing efficient setups.

B. Adaptive Communication Plane

Real-world policy learning runs over cloud–edge distributed compute resources, especially for large-scale VLA models, with rollout and robot nodes at the edge and training nodes in the cloud. USER's *adaptive communication plane* addresses the resulting communication challenges across heterogeneous



(a) Centralized (b) Distributed

Fig. 4: Illustration of the centralized and distributed channels.

cloud–edge network domains. USER enables cross-domain connectivity via *tunneling-based cloud–edge networking* for data exchange across network boundaries. To handle uneven bandwidth between intra-domain and cross-domain communication, USER employs a *distributed data channel* to localize traffic and reduce unnecessary cross-domain transfer. In addition, USER introduces *streaming-multiprocessor-budgeted weight synchronization* to prevent NCCL-based communication from monopolizing GPU resources and degrading throughput.

Tunneling-based Cloud-Edge Networking. USER operates over cloud–edge compute resources across distinct administrative network domains (e.g., NATs, campus networks, factory VLANs) that are inherently isolated and do not support direct communication. To bridge this gap, USER builds its communication plane on a flattened TCP/IP substrate based on the UDP tunneling technology, implemented with EasyTier [1]. This design enables all robot, training, and rollout nodes to establish bidirectional connections through the tunnel. The control plane uses Ray [25] for cluster membership and worker placement, while the data plane relies on TCP rendezvous to bootstrap point-to-point communication groups; critically, USER binds all control/data traffic to the tunnel interface to make heterogeneous deployments robust to multi-homed hosts and to avoid accidentally routing traffic over slow or firewalled links.

Distributed Data Channel. In USER, data exchange among nodes is unified through a *channel* abstraction. A channel represents a dataflow conduit connecting system components and carries observations, actions, states, and intermediate results. Conventional centralized communication (see Fig. 4a), where data is first sent to a cloud node and then redistributed to edge nodes, is simple to implement but incurs substantial cross-domain traffic in cloud–edge deployments. Under multi-robot or high-frequency real-world interaction, such designs suffer from high latency and poor stability [5, 10]. To address this, USER provides a distributed data *channel* (see Fig. 4b): a named, first-in-first-out (FIFO) producer–consumer queue hosted by lightweight channel services and accessed via asynchronous put/get APIs. Channels are internally sharded across service instances based on data keys (e.g., robot IDs) to localize traffic within edge or cloud regions and reduce cross-domain transfer. They support multiple producers and consumers, enabling robots and rollout nodes to stream data without direct, synchronous coupling. This design minimizes cross-domain communication and balances load across channel services.

SM-Budgeted Weight Synchronization. In USER, when weight synchronization is performed via the NVIDIA Collective Communications Library (NCCL) and transferred directly between GPUs, it occupies GPU execution resources, because NCCL's collectives execute as CUDA kernels that consume streaming multiprocessors (SMs). USER makes this contention explicit and controllable through a tunable configuration that caps the maximum number of NCCL CTAs (cooperative thread arrays) used during weight transfer. By throttling NCCL's SM footprint, USER prevents background weight synchronization from monopolizing GPU execution resources and degrading rollout latency and throughput. This allows USER to sustain stable, low-latency rollouts under asynchronous weight updates.

IV. Learning Framework Design

Building on the system architecture design that virtualizes and connects heterogeneous robots and compute resources, the learning framework design of USER focuses on how data and computation are organized at runtime. Specifically, it defines a unified framework for real-world learning, including a fully asynchronous pipeline, a persistent and cache-aware buffer, and extensible abstractions for policies, algorithms, and rewards.

A. Fully Asynchronous Pipeline

In real-world policy learning, the primary bottleneck lies in data collection rather than computation [21]. Physical interaction cannot be accelerated, which makes it essential to keep robots executing continuously. Synchronous pipelines that tightly couple data generation and training are prone to cascading stalls: delays in training propagate to execution, forcing robots to pause and significantly reducing data efficiency.

As illustrated in Fig. 5 and Fig. 6, USER organizes real-world policy learning as a fully asynchronous pipeline. On the data generation side, multiple environment workers execute policies on physical robots through rollout workers, continuously streaming observations and actions without being blocked by optimization. In parallel, human operators can intervene via teleoperation to provide corrections or demonstrations, while a reward worker assigns supervision signals. All interaction data are asynchronously ingested into a persistent and cache-aware buffer (Sec. IV-B), which includes a trajectory replay buffer for autonomous rollouts and a demonstration buffer for pre-collected data and human interventions.

On the training side, learning workers asynchronously sample mini-batches from these buffers to update model parameters, supporting both reinforcement and imitation learning. The updated weights are periodically synchronized back to rollout workers, closing the loop while maintaining uninterrupted robot execution.

B. The Persistent and Cache-Aware Buffer

Real-world embodied learning involves long horizons, non-stationary policies, and asynchronous pipelines. Training spans

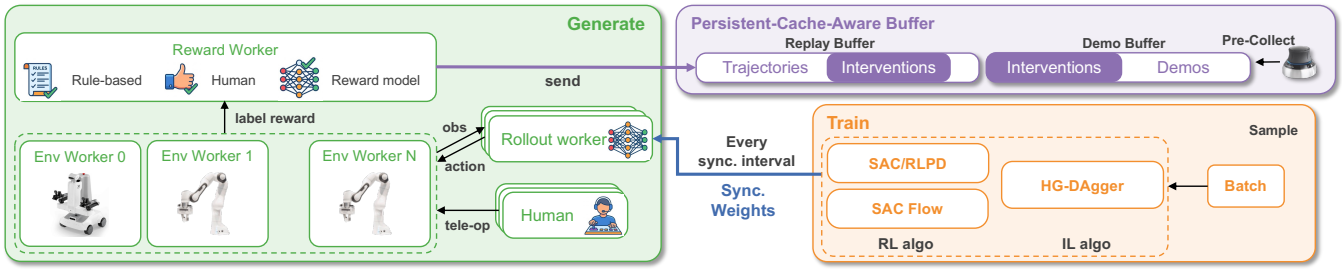


Fig. 5: **Overview of learning framework design:** a fully asynchronous real-world learning pipeline with a persistent, cache-aware buffer and extensible abstractions for policies, algorithms, and reward models.

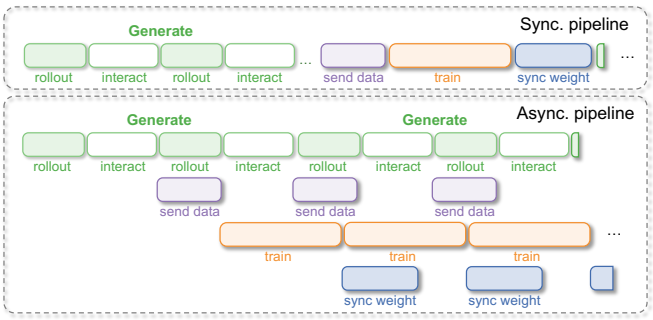


Fig. 6: **Fully Asynchronous pipeline.** USER decouples data generation, training, data transmission, and weight synchronization, significantly improving both data collection and training throughput.

multiple sessions and policy versions, where network failures, restarts, and human intervention are common. Data collected under different policies must often be reused for off-policy updates or recovery, making short-lived, in-memory buffers inadequate for real deployments.

USER adopts a persistent, index-based buffer (Fig. 7) that decouples storage from memory. Trajectories are asynchronously written to disk, while the buffer stores lightweight indices with metadata such as policy version, timestamps, and episode IDs, enabling temporal- and policy-aware sampling over long horizons. To balance efficiency and memory, USER adds a bounded in-memory cache with FIFO replacement. New samples enter the cache first; when full, old entries are evicted but remain indexed on disk. Evicted samples are transparently reloaded when requested, preserving high-throughput sampling with bounded memory.

Unlike in-memory buffers that keep only recent data [8, 30], the persistent and cache-aware design supports arbitrarily large

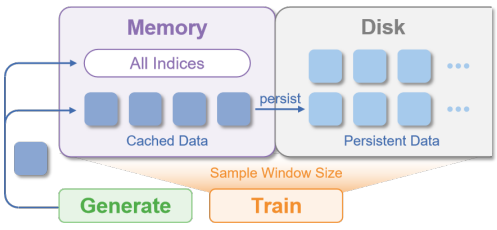


Fig. 7: **The Persistent and Cache-Aware Buffer.** Recent data is stored in memory while historical data is persisted to disk, effectively balancing access efficiency with storage capacity.

datasets and retains historical data across evolving policies. Persistence also improves robustness through crash recovery and pipeline decoupling, enabling reliable long-horizon real-world learning beyond volatile memory-centric designs.

C. Extensible Policies, Algorithms, and Rewards

USER is *agnostic* to model architectures and learning algorithms, exposing unified interfaces that enable heterogeneous policies, optimizers, and reward mechanisms to share a single execution and data pipeline.

At the policy level, USER supports both lightweight and large-scale models. Lightweight policies include CNN-based models, such as ResNet-style visual policies [12], and expressive flow-matching policies [20] that represent actions via continuous probability flows. At the larger scale, USER integrates VLA models (e.g., $\pi_0/\pi_{0.5}$ architectures [4, 15]) that reason over multimodal inputs and output continuous actions. Despite structural and computational differences, all policies are deployed through a unified rollout abstraction.

At the algorithm level, USER supports multiple learning paradigms for robotics. These include off-policy RL such as Soft Actor-Critic (SAC) [11], sample-efficient RL for flow policies (e.g., SAC-Flow [32]), human-in-the-loop methods like RL with pretrained data (RLPD) [3], and imitation-style updates for large models such as HG-DAgger [17]. Training workers interact with policies through standardized sampling and update APIs, making the optimization layer interchangeable.

Reward specification is also modular. USER supports (i) rule-based task rewards, (ii) human-provided labels, and (iii) learned reward models. Rewards can be attached during rollout or computed offline in post-processing, enabling flexible supervision under real-world constraints.

By unifying policy representations, algorithms, and reward sources, USER enables diverse real-world learning setups—from RL to imitation and human-in-the-loop learning—without re-engineering deployment, data handling, or execution pipelines.

V. Experiment

We evaluate USER on a suite of simulated and real-world embodied tasks to validate both its system architecture and learning framework design. Through five groups of experiments, we demonstrate how USER's core components support extensible and reliable real-world online learning:

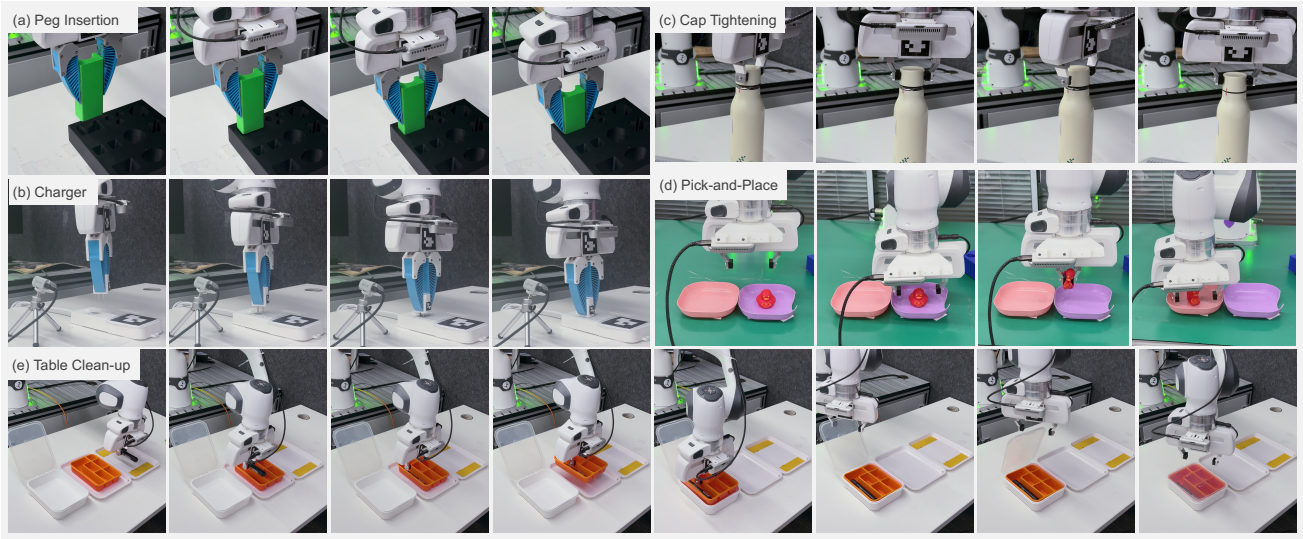
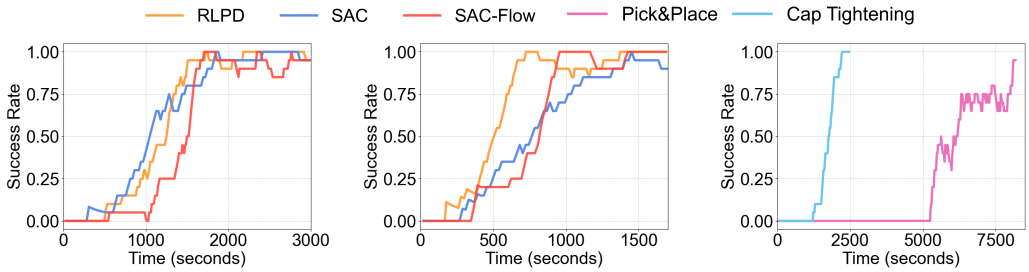


Fig. 8: **Visualization of 5 manipulation tasks.** (a) **Peg Insertion** inserts a peg into a target hole. (b) **Charger Plugging** requires contact-rich manipulation with sub-millimeter precision. (c) **Pick-and-Place** involves grasping and transporting a randomly initialized object (a rubber duck) to a target container. (d) **Cap Tightening** rotates and tightens a bottle cap to a specified torque or pose. (e) **Table Clean-up** clears cluttered objects from the tabletop into a designated box, then close the lid.



(a) Peg Insertion (b) Charger (c) Other Tasks with RLPD

Fig. 9: Training curves of reinforcement learning on multiple manipulation tasks. The x-axis shows wall-clock time, and the y-axis shows the success rate computed as a moving average with a window size of 20.

1) **Sec. V-A: Extensibility of the learning framework:** USER achieves high performance across multiple tasks while accommodating diverse policies, algorithms, and reward sources within a unified pipeline.
2) **Sec. V-B: Unified hardware abstraction:** USER enables training over multi-robot and heterogeneous robot fleets through its hardware abstractio layer.
3) **Sec. V-C: Adaptive communication plane:** USER supports cross-domain edge–cloud collaborative training via its adaptive communication design.
4) **Sec. V-D: Persistent and cache-aware buffer:** USER provides high-capacity and high-throughput storage for long-horizon data ingestion and reuse.
5) **Sec. V-E: Fully asynchronous pipeline:** USER improves learning efficiency and convergence through its fully asynchronous execution pipeline.

A. Main Results

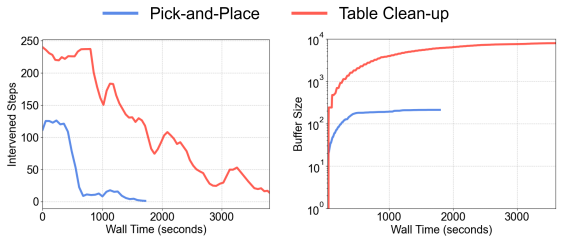
Experiment Setup We design a suite of five real-world manipulation tasks to evaluate the extensibility of USER's learning framework, as shown in Fig. 8: *Peg Insertion, Charger, Cap Tightening, Pick-and-Place, and Table Clean-up*. All tasks are conducted on a Franka robotic arm. Detailed task descriptions are provided in Appendix B.

All experimental settings are summarized in Tab. II. Experiments with small policies, including CNN and flow-based models, are executed on a local workstation equipped with an RTX 4090 (24GB) GPU, while large VLA policies such as the π_0 model are full-parameter trained and evaluated on a server with 4 NVIDIA A100 (80GB) GPUs. We select four representative algorithms spanning online reinforcement learning and imitation learning: SAC, RLPD, SAC-Flow and HG-DAgger. We select appropriate reward source for each task from rule-based rewards, human-provided rewards, and learned reward models. Rule-based rewards are used for fixed-position manipulation tasks and computed from the end-effector pose. Human rewards are binary, with operators assigning 1 for success and 0 otherwise. The reward model classifies success (1) or failure (0) from observations. Implementation details are in Appendix A.

Task Performance Fig. 9 summarizes RL results on multiple manipulation tasks. Figs. 9a and 9b report RLPD, SAC, and SAC-Flow on Peg-Insertion and Charger. Both achieve near-perfect success within 2000s with similar overall performance.

SAC performs worse on Charger, likely because higher precision and dense rewards induce suboptimal behaviors. We further evaluate RLPD on Pick and Place and Cap Tightening (Fig. 9c). Cap Tightening converges quickly, while Pick and Place involves richer object dynamics and needs longer training. Still, RLPD reaches success rates approaching 1.0 on all tasks. Unlike RL, imitation learning depends on human demonstrations and intervention. During HG-DAgger training of π_0, operators ensure success in each episode. As intervention steps approach zero, the policy becomes fully autonomous. π_0 is initialized with full-parameter supervised fine-tuning (SFT) and then trained online. Tab. III compares success before and after training, and Fig. 10 shows intervention steps and buffer growth. For short-horizon Pick-and-Place, HG-Dagger reaches a 96% success rate in $\sim$30 minutes using only $\sim$200 online samples.

We compare USER's performance against relative realworld learning systems, results are available in Appendix C.



(a) Intervened steps (b) Buffer size

Fig. 10: Training curves of imitation learning. Human operator intervenes to guarantee the success of every episode. And we report intervened steps per episode as a performance metric. A lower intervention steps directly reflects a higher policy performance.

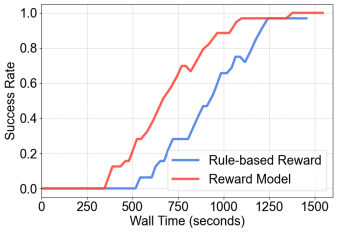


Fig. 11: USER supports reward model to enable automated annotation. In the peg insertion task, the trained reward model provides supervision comparable to human labels.

TABLE II: Experiment setting summary.

Task Name	Model	Algorithm	Reward Type	Demo (trajs)
Peg Insertion	CNN	SAC	rule-based, dense	/
		RLPD	rule-based, sparse	20
	Flow	SAC Flow		
Charger	CNN	SAC	rule-based, dense	/
		RLPD	rule-based, sparse	20
	Flow	SAC Flow	rule-based, sparse	
Cap Tightening	CNN	RLPD	human reward, sparse	20
Pick-and-Place	CNN	RLPD	human-reward, sparse	40
	π_0	HG-DAgger	/	40
Table Clean-up	π_0	HG-DAgger	/	40

TABLE III: USER enables significant performance gains after online training for foundation VLA models.

	Before online training	After online training
Pick-and-Place	39/60	**58/60**
Table Clean-up	9/20	**16/20**

Reward Model To train a reliable reward model, we formulate the reward model as a binary classifier with a pretrained ResNet18 [12] backbone. Human operators collect 20 successful trajectories, each remaining stationary for 20 timesteps after task completion to accumulate sufficient positive samples. The final dataset for the reward model comprises approximately 1,600 frames, with a success-to-failure ratio of roughly 1:3. Fig. 11 shows the results of a CNN policy trained with our reward model. Notably, our reward model achieves performance comparable to human rewards on the peg insertion task.

B. Advantages of a Unified Hardware Layer

With a unified hardware abstraction, USER enables online policy learning on real robots without platform-specific modification. We demonstrate that this design supports efficient multi-robot training and robust learning across heterogeneous embodiments.

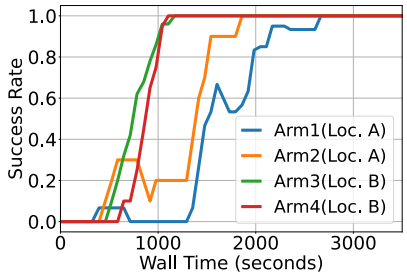


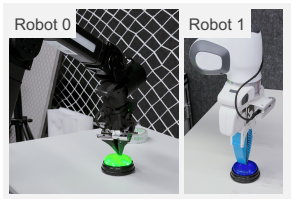
Fig. 12: Parallel training on four Franka robot arms. The unified hardware layer enables distributed data collection and learning within a single system framework.

Training with Multiple Robots We first evaluate USER in a multi-robot setting, across two sites (3km apart). Using the unified hardware abstraction, we concurrently train policies on four Franka robot arms executing peg-insertion task. As shown in Fig. 12, all robots achieves 100% success rate within $\sim$2600 seconds, matching the convergence speed of single-robot baselines. The result indicate that USER can effectively scale real-world training through parallel data collection, improving sample efficiency without degrading learning stability.

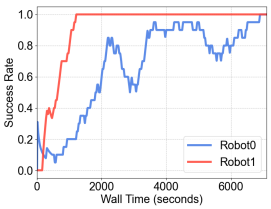
Training with Heterogeneous Robots Leveraging heterogeneous platforms enables policies to learn shared visual-semantic representations, thereby improving generalization across embodiments. We use a 7-DoF Franka arm and a 6-DoF low-cost ARX robotic arm to demonstrate the heterogeneous policy capacity. (Hardware setup details in Appendix D). We first unify the observation spaces of the two distinct robot arms as a single image, the end-effector pose, and the normalized gripper state. The action spaces are further unified as the end-effector pose and the normalized gripper action. As shown

TABLE IV: Communication performance of distributed channels under cross-domain and same-domain network settings. Enabling distributed channels reduces episode generation time by up to 3× in cross-domain deployments.

Domain	Distributed	Rollout (s/chunk)	Interact (s/chunk)	Send Obs (s/data)	Send Action (s/data)	Total Generation Time ↓ (s/episode)
cross	w/	0.002(±0.000)	0.106(±0.001)	**0.042**(±0.031)	**0.070**(±0.017)	**21.979**(±0.435)
	w/o	0.002(±0.000)	0.107(±0.001)	0.671(±0.012)	0.270(±0.009)	69.265(±1.905)
same	w/	0.006(±0.001)	0.106(±0.002)	**0.025**(±0.006)	**0.028**(±0.008)	**17.304**(±0.001)
	w/o	0.007(±0.001)	0.106(±0.002)	0.169(±0.018)	0.021(±0.008)	18.696(±0.710)



(a) Heterogeneous setting



(b) Results

Fig. 13: Training with two heterogeneous robots. A single unified policy trained on joint data converges successfully, demonstrating cross-embodiment learning.

in Fig. 13a, we train a unified CNN-based policy to control two distinct robot arms in a multi-colored button-pressing task. Using SAC with dense rewards, the policy achieves full convergence (Fig. 13b). Compared to single-robot baselines, heterogeneous training presents significant challenges due to variations in arm DoF, end-effector morphology, camera parameters, and target colors. Consequently, the training process requires approximately two hours to reach convergence.

C. Capability of the Communication Plane

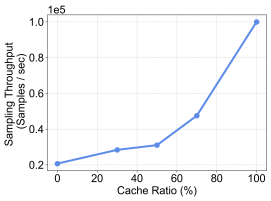
In this section, we demonstrate the design of USER's adaptive communication plane through both cross-domain and same-domain experiments.

Cross-Domain Communication We evaluate the effectiveness of the distributed data channel under a cross-domain deployment setting. The master node is located in City A and performs policy training, while two nodes in City B handle rollout and robot control, respectively. The two nodes in City B are connected via a high-bandwidth local network, whereas the connection between City A and City B spans thousands of kilometers and incurs high latency and limited bandwidth. This cross-domain deployment reflects a realistic cloud-edge setup, where communication between geographically distributed nodes is enabled via tunneled-based connections.

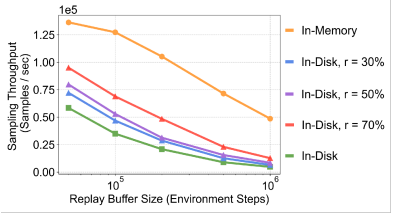
The first two rows of Tab. IV report the cross-domain results. Across both configurations, the rollout and environment interaction times remain similar, as these operations are executed locally without cross-domain communication. In contrast, enabling distributed channels substantially reduces the communication overhead for transmitting observations and actions between the two nodes in City B. As a result, the total time required to generate a single episode is reduced by approximately 3× compared to the centralized baseline.

Same-Domain Communication We further evaluate USER in a same-domain setting, where all three nodes are connected via high-speed local networks. The third and fourth rows of Tab. IV show the corresponding results. Even under this favorable condition, distributed channels consistently reduce communication overhead.

Notably, comparing the first and third rows of Tab. IV shows that the two nodes in City B under cross-domain deployment achieve communication efficiency comparable to the all-same-domain setup, indicating that USER effectively exploits local high-bandwidth links while avoiding unnecessary cross-domain data transfers.



(a) Throughput v.s. cache ratio

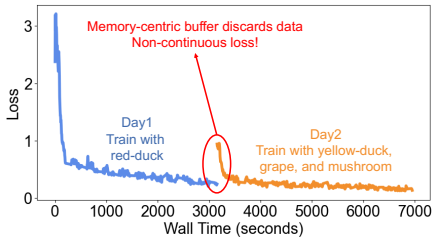


(b) Throughput v.s. buffer size

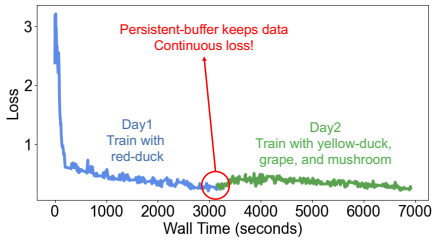
Fig. 14: Testing sampling throughput of our persistence and cache-aware buffer.

D. Persistent Training with Cache-aware Buffer

USER's buffer combines persistent storage with an in-memory cache to balance efficiency and capacity. Let s be the buffer size and c the cache size, with ratio $r = c/s$. We profile throughput under different cache ratios in Fig. 14a, showing that larger r improves throughput. We further evaluate throughput under varying buffer sizes (Fig. 14b). A pure in-memory buffer achieves the highest throughput but is memory-



(a) Without persistent buffer



(b) With persistent buffer

Fig. 15: DAgger losses during a two-day persistent training with and without persistent buffer

TABLE V: Profiling results of synchronous and asynchronous training pipelines. Generation and training periods denote the intervals between consecutive episode generation and training executions, respectively.

		Rollout (s/chunk)	Interact (s/chunk)	Send data	Train (s/update)	Sync weights	Generation Period ↓ (s/episode)	Training Period ↓ (s/update)
π_0 + HG-Dagger	Sync	0.214(±0.001)	1.093(±0.013)	0.606(±0.028)	6.128(±0.247)	0.816(±0.024)	45.068(±0.304)	45.011(±0.251)
	Async	0.213(±0.016)	1.091(±0.010)	7.903(±)	5.969(±0.196)	0.789(±0.052)	37.538(±0.363)	7.903(±0.234)
	Speed Up	/	/	/	/	/	1.20×	5.70×
CNN + SAC	Sync	0.006(±0.002)	0.108(±0.002)	0.144(±0.008)	0.108(±0.002)	0.162(±0.004)	20.291(±0.632)	0.643(±2.984)
	Async	0.004(±0.004)	0.107(±0.002)	0.004(±0.001)	0.123(±0.005)	0.174(±0.003)	13.108(±0.218)	0.135(±0.034)
	Speed Up	/	/	/	/	/	1.55×	4.61×

limited, while an in-disk buffer provides large capacity at less than half the throughput. Our design strikes a balance, achieving higher throughput than standard disk buffers while retaining large capacity.

This buffer enables continual learning and robust crash recovery. We design a multi-day, multi-task persistent learning task to show the advantage of our buffer. We first initialized the π_0 model using offline datasets for grasping the red duck, to obtain the initial success rate (the first column of Tab. VI). On the first day of training, the model interacts with the red duck and improvs its performance under human guidance, reaching nearly 100% success (the second column of Tab. VI). On the second day, the model interacted online with three other objects (a yellowduck, a mushroom, and a grape), with and without the persistent buffer. The third and fourth columns of Tab. VI report the success rates on different objects after the second-day training under these two settings. The results show that our persistent buffer effectively mitigates forgetting of the object trained on the first day, i.e., the red duck.

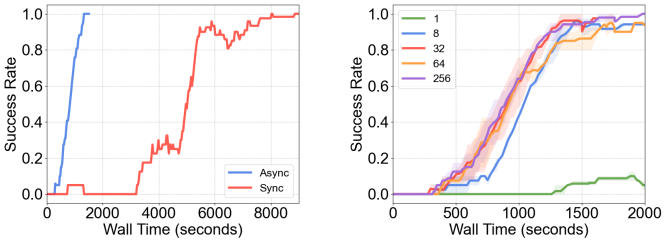
Fig. 15 shows the loss curves during continual training. The orange curve, without the persistent buffer, quickly falls below the green curve because the limited non-persistent buffer retains only recent data, making the model more prone to overfitting. Tab. VI further confirms that the model forgets how to grasp the red duck. In contrast, the persistent buffer maintains a larger amount of data. Although the green curve has a higher loss than the orange curve, the resulting model achieves better performance across all tasks (Tab. VI).

TABLE VI: Persistent buffer enables continual learning without forgetting. Table entries report success rates.

	Initial	After Day 1	After Day 2	
			w/o persistent	w/ persistent
Red duck	9/20	19/20	6/10	10/10
Yellow Duck	-	5/10	10/10	10/10
Grape	-	5/10	10/10	10/10
Mushroom	-	4/10	9/10	10/10

E. Asynchronous Design

To demonstrate the advantages of our asynchronous framework over traditional synchronous systems, we first profiled the latency of individual stages in both configurations, including rollout, interacting, sending data, training, and syncing weights. As shown in Tab. V, although per-stage latencies remain similar, the asynchronous system achieves significantly higher throughput via pipeline overlapping (Fig. 6). Specifically, for π_0 and CNN models, USER's asynchronous pipeline improves generation throughput by $1.20\times$ and $1.55\times$, and training throughput by $5.70\times$ and $4.61\times$, respectively. Fig. 16a further illustrates the disparity in convergence speeds on the peg insertion task, CNN policy, with RLPD. The asynchronous pipeline significantly accelerates training, reducing the convergence time from 8000+ seconds to $\sim$ 1500 seconds.



(a) Convergence speed (b) Policy weight sync. intervals

Fig. 16: Ablation study on asynchronous pipeline design choices for CNN Policy on the Peg-Insertion Task. (a) The async pipeline reduces convergence time from 8,000+ seconds to ∼1,500 seconds. (b) Weight synchronization intervals do affect convergence behavior, where interval = 1 is the most unstable setting.

Ablation for Policy Weights Synchronization Interval The weight synchronization interval is a critical design parameter in asynchronous systems. We conducted an ablation study on the peg insertion task using the RLPD algorithm with a CNN policy. As shown in Fig. 16b, small synchronization intervals (like 1 and 8) cause frequent in-episode weight updates. This induces policy non-stationarity, resulting in slower convergence or complete divergence. Conversely, larger synchronization intervals ensure a stable update.

VI. Conclusion

This paper presents USER, a unified and extensible system for real-world online policy learning. USER places robots at the same hardware resource level as accelerators and integrates adaptive communication, persistent cache-aware buffering, and a fully asynchronous training pipeline into a single learning infrastructure. By remaining agnostic to policy architectures and optimization methods, USER supports diverse paradigms—from imitation learning and reinforcement learning to human-in-the-loop learning—deployed on heterogeneous robots within a unified pipeline. Experiments on both simulated and real-world benchmarks show that USER enables efficient policy learning over heterogeneous multi-robot fleets, provides robust edge–cloud cross-domain communication, and maintains high-capacity, high-throughput buffering for long-horizon training.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No.62406159, 62325405), Zhongguancun Academy (Grant No. C20250301), Beijing National Research Center for Information Science, Technology (BNRist) and Beijing Innovation Center for Future Chips.

REFERENCES

[1] EasyTier Authors. Easytier. https://github.com/EasyTier/EasyTier, 2025.

[2] Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

[3] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[4] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[5] Michael Bloesch, Jan Humplik, Viorica Patraucean, Roland Hafner, Tuomas Haarnoja, Arunkumar Byravan, Noah Yamamoto Siegel, Saran Tunyasuvunakool, Federico Casarini, Nathan Batchelor, et al. Towards real robot learning in the wild: A case study in bipedal locomotion. In *Conference on Robot Learning*, pages 1502–1511. PMLR, 2022.

[6] Yann Bouteiller et al. tmrl: Reinforcement learning for real-time applications. https://github.com/trackmania-rl/tmrl, 2021.

[7] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024.

[8] Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. Reverb: a framework for experience replay. *arXiv preprint arXiv:2102.04736*, 2021.

[9] Angelo Corsaro, Luca Cominardi, Olivier Hecart, Gabriele Baldoni, Julien Enoch Pierre Avital, Julien Loudet, Carlos Guimares, Michael Ilyin, and Dmitrii Bannov. Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller. In *2023 26th Euromicro Conference on Digital System Design (DSD)*, pages 422–428. IEEE, 2023.

[10] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Matthew W Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[14] Peide Huang, Xilun Zhang, Ziang Cao, Shiqi Liu, Mengdi Xu, Wenhao Ding, Jonathan Francis, Bingqing Chen, and Ding Zhao. What went wrong? closing the sim-to-real gap via differentiable causal discovery. In *Conference on Robot Learning*, pages 734–760. PMLR, 2023.

[15] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $pi\_\{0.5\}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[16] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018.

[17] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

[18] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

[19] Eric Liang, Zhanghao Wu, Michael Luo, Sven Mika, Joseph E Gonzalez, and Ion Stoica. Rllib flow: Distributed reinforcement learning is a dataflow problem. *Advances in Neural Information Processing Systems*, 34: 5506–5517, 2021.

[20] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*.

[21] 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. In *2024 IEEE International Conference on Robotics and*

Automation (ICRA), pages 16961–16969. IEEE, 2024.

[22] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

[23] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[24] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6): 3740–3747, 2023.

[25] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

[26] Mingjie Pan, Siyuan Feng, Qinglin Zhang, Xinchen Li, Jianheng Song, Chendi Qu, Yi Wang, Chuankang Li, Ziyu Xiong, Zhi Chen, et al. Sop: A scalable online post-training system for vision-language-action models. *arXiv preprint arXiv:2601.03044*, 2026.

[27] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *Robotics: Science and Systems XIV*, 2018.

[28] Edan Toledo, Laurence Midgley, Donal Byrne, Callum Rhys Tilbury, Matthew Macfarlane, Cyprien Courtot, and Alexandre Laterre. Flashbax: Streamlining experience replay buffers for reinforcement learning with jax, 2023. URL https://github.com/instadeepai/flashbax/.

[29] Andrew Wagenmaker, Kevin Huang, Liyiming Ke, Kevin Jamieson, and Abhishek Gupta. Overcoming the sim-to-real gap: Leveraging simulation to learn to explore for real-world rl. *Advances in Neural Information Processing Systems*, 37:78715–78765, 2024.

[30] Hanjing Wang, Man-Kit Sit, Congjie He, Ying Wen, Weinan Zhang, Jun Wang, Yaodong Yang, and Luo Mai. Gear: a gpu-centric experience replay system for large reinforcement learning models. In *International Conference on Machine Learning*, pages 36380–36390. PMLR, 2023.

[31] Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, et al. Rlinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. *arXiv preprint arXiv:2509.15965*, 2025.

[32] Yixian Zhang, Shu'ang Yu, Tonghe Zhang, Mo Guang, Haojia Hui, Kaiwen Long, Yu Wang, Chao Yu, and Wenbo Ding. Sac flow: Sample-efficient reinforcement learning of flow-based policies via velocity-reparameterized sequential modeling. *arXiv preprint arXiv:2509.25756*, 2025.