SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning

Anonymous Author(s)

Abstract— **Recent years have seen the development of many methods for robotic reinforcement learning (RL), some of which can even operate on complex image observations, train in the real world, and incorporate auxiliary data, such as demonstrations and prior experience. However, despite these advances, robotic RL remains hard to use. It is acknowledged among practitioners that the particular implementation details of these algorithms are often just as important (if not moreso) for performance as the choice of algorithm that is actually used. We posit that a significant challenge to widespread adoption of robotic RL, as well as further development of robotic RL methods, is the comparative inaccessibility of such methods. To address this challenge, we developed a carefully implemented library containing a sample efficient off-policy deep RL method, together with methods for computing rewards and resetting the environment, high-quality controllers for a few common robots, and a number of challenging example tasks. We provide this library as a resource for the community, describe its design choices, and present experimental results. Perhaps surprisingly, we find that our implementation can achieve very efficient learning, acquiring policies for PCB board assembly, cable routing, and object relocation in less than an hour of training per policy, matching or improving over state-of-the-art results reported for similar tasks in the literature. We hope that these promising results and our high-quality open-source implementation will provide a tool for the robotics community to study new developments in robotic RL. Our code and videos can be found at `https://serl-robot.github.io/`**

I. Introduction

Considerable progress on robotic reinforcement learning (RL) over the recent years has produced impressive results, with robots playing table tennis [1], manipulating objects from raw images [2, 3, 4], grasping diverse objects [5, 6], and performing a wide range of other skills. However, despite the significant progress on the underlying algorithms, RL remains challenging to use for real-world robotic learning problems, and practical adoption has been more limited. We argue that part of the reason for this is that the implementation of RL algorithms, particularly for real-world robotic systems, presents a very large design space, and it is the challenge of navigating this design space, rather than limitations of algorithms *per se*, that limit adoption. It is often acknowledged by practitioners in the field that details in the implementation of an RL algorithm might be as important (if not more important) as the particular choice of algorithm. Furthermore, real-world learning presents additional challenges with reward specification, implementation of environment resets, sample efficiency, compliant and safe control, and other difficulties that put even more stress on this issue. Thus, adoption and further research progress on real-world robotic RL may well be bottlenecked on *implementation* rather than novel algorithmic innovations.

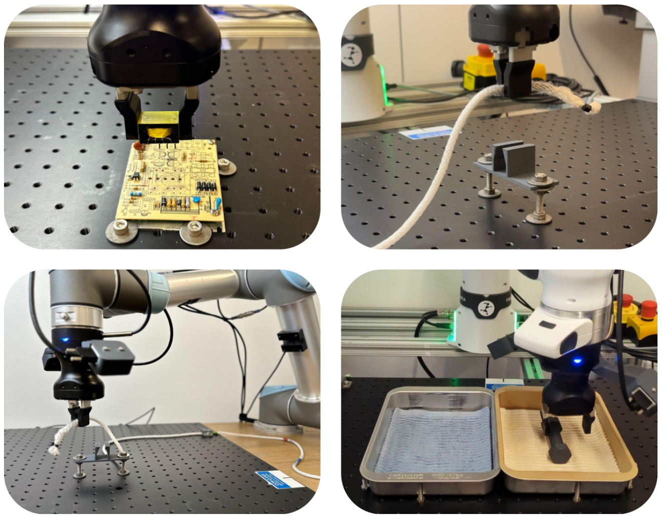


Fig. 1: Depiction of various tasks solved using SERL in the real world. These include PCB board insertion (top left), cable routing (Franka, top right, and UR5, bottom left), and object relocation (bottom right). SERL provides an out-of-the-box package for real-world reinforcement learning, with support for efficient learning, learned rewards, and automation of resets.

To address this challenge, our aim in this paper is to provide an open-source software framework, which we call **S**ample-**E**fficient **R**obotic reinforcement **L**earning (SERL), that aims to facilitate wider adoption of RL in real-world robotics. SERL consists of the following components: (1) a high-quality RL implementation that is geared towards real-world robotic learning and supports image observations and demonstrations; (2) implementations of several reward specification methods that are compatible with image observations, including classifiers and adversarial training; (3) support for learning "forward-backward" controllers that can automatically reset the task between trials [7]; (4) a software package that can in principle connect the aforementioned RL component to any robotic manipulator; and (5) an impedance controller design principle that is particularly effective for dealing with contact-rich manipulation tasks.

Our aim in this paper is not to propose novel algorithms or methodology, but rather to offer a resource for the community to provide roboticists with a well-designed foundation both for future research on robotic RL, and other methods that might employ robotic RL as a subroutine. However, in the process of evaluating our framework, we also make a scientifically interesting empirical observation: when implemented properly in a carefully engineered software package, current sample-efficient robotic RL methods can attain very high success rates with relatively modest training times. The tasks in our evaluation are illustrated in Fig. 1: precise insertion tasks involving dynamic contact, deformable object manipulation with complex dynamics, and object relocation where the robot must learn without manually designed resets. For each of these tasks, SERL is able to learn effectively within 15 - 60 min of training per policy (in terms of total wall-clock time), achieving near-perfect success rates, despite learning policies that operate on image observations. This result is significant because RL, particularly with deep networks and image inputs, is often considered to be highly inefficient. Our results challenge this assumption, suggesting careful implementations of existing techniques, combined with well-designed controllers and carefully selected components for reward specification and resets, can provide an overall system that is efficient enough for real-world use.

II. Related Work

While our framework combines existing RL methods into a complete robotic learning system, the particular combination of parts is carefully designed to provide for efficient and out-of-the-box reinforcement learning directly in the real world and, as shown in our experiments, achieves excellent results on a wide range of tasks. Here, we summarize both related prior methods and systems.

Algorithms for Real-World RL: Real-world robotic RL demands algorithms that are sample-efficient, can utilize onboard perception, and support easily specified rewards and resets. A number of algorithms have shown the ability to learn very efficiently directly in the real world [8, 9, 10, 11, 12, 13, 14], using variants of off-policy RL [15, 16], model-based RL [17, 18, 19, 20], and on-policy RL [21]. These advances have been paired with advances in inferring rewards from raw visual observation through success classifiers [22, 23], foundation-model-based rewards [24, 25, 26], and rewards from videos [27, 28]. Additionally, to enable autonomous training, there have been a number of algorithmic advances in reset-free learning [2, 29, 30, 31, 32] that enable autonomous training with minimal human interventions. While these algorithmic advances are important, the contribution we make in this work is to provide a framework and software package to enable sample efficient reinforcement learning in the real world with a ready-made choice of methods that can work well for a variety of tasks. In doing so, we hope to lower the barrier of entry for new researchers to build better algorithms and training methodologies for robot learning in the real world.

Software Packages for RL: There are a number of packages [33, 34, 35, 36] for RL, though to our knowledge none aim to directly address real-world robotic RL specifically. SERL builds on the recently proposed RLPD algorithm, which is an off-policy RL algorithm with high update-to-data ratio. SERL is not a library of RL algorithms for training agents in simulation, although it could be adapted to be so. Rather, SERL offers a full stack pipeline for robot control, going from low-level controllers to the interface for asynchronous and efficient training with an RL algorithm, to additional machinery for inferring rewards and training without resets. In doing so, SERL provides an off-the-shelf package to help non-experts start using RL to train their physical robots in the real world, unlike prior libraries that aim to provide implementations of many methods – that is, SERL offers a full "vertical" integration of components, whereas prior libraries focus on the "horizontal." SERL is also not an RL benchmark package such as [37, 38, 39]. SERL allows users to define their own tasks and success metrics directly in the real world, providing the software infrastructure for actually controlling and training robotic manipulators in these tasks.

Software Packages for Real-World RL: There have been several previous packages that have proposed infrastructure for real world RL: for dexterous manipulation [40], tabletop furniture assembly [41], legged locomotion [15], and peg insertion [42]. These packages are effective in narrow situations, either using privileged information or training set ups such as explicit tracking [42, 40] or pure proprioception [15], or limited to imitation learning. In SERL, we show a full stack system that can be used for a wide variety of robotic manipulation tasks without requiring privileging of the training setups as in prior work.

III. Preliminaries and Problem Statement

Robotic reinforcement learning tasks can be defined via an MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \rho, \mathcal{P}, r, \gamma\}$, where $\mathbf{s} \in \mathcal{S}$ is the state observation (e.g., an image in combination with the current end-effector position), $\mathbf{a} \in \mathcal{A}$ is the action (e.g., the desired end-effector pose), $\rho(\mathbf{s}_0)$ is a distribution over initial states, $\mathcal{P}$ is the unknown and potentially stochastic transition probabilities that depend on the system dynamics, and $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, which encodes the task. An optimal policy π is one that maximizes the cumulative expected value of the reward, i.e., $E[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$, where the expectation is taken with respect to the initial state distribution, transition probabilities, and policy π.

While the specification of the RL task is concise and simple, turning real-world robotic learning problems into RL problems requires care. First, the sample efficiency of the algorithm for learning π is paramount: when the learning must take place in the real world, every minute and hour of training comes at a cost. Sample efficiency can be improved by using effective off-policy RL algorithms [43, 44, 45], but it can also be accelerated by incorporating prior data and demonstrations [46, 47, 48], which is important to achieve the fastest training times.

Additionally, many of the challenges with robotic RL lie beyond just the core algorithm for optimizing π. For example, the reward function r might depend on image observations, and difficult for the user to specify manually. Additionally, for episodic tasks where the robot resets to an initial state $\mathbf{s}_0 \sim \rho(\mathbf{s}_0)$ between trials, actually resetting the robot (and its environment) into one of these initial states is a mechanical operation that must somehow be automated.

Furthermore, the controller layer, which interfaces the MDP actions $\mathbf{a}$ (e.g., end-effector poses) to the actual low-level robot controls, also requires great care, particularly for contact-rich tasks where the robot physically interacts with objects in the environment. Not only does this controller need to be accurate, but it must also be safe enough that the RL algorithm can explore with random actions during training.

SERL will aim to provide ready-made solutions to each of these challenges, with a high-quality implementation of a sample-efficient off-policy RL method that can incorporate prior data, several choices for reward function specification, a forward-backward algorithm for learning resets, and a controller suitable for learning contact-rich tasks without damaging either the robot or objects in the environment.

IV. Sample Efficient Robotic Reinforcement Learning in the Real-World

Our software package, which we call **S**ample-**E**fficient **R**obotic reinforcement **L**earning (SERL), aims to make robotic RL in the real world accessible by providing ready-made solutions to the problems detailed in the previous section. This involves providing efficient vision-based reinforcement learning algorithms *and* the infrastructure needed to support these learning algorithms for autonomous learning. We note that the purpose of such an endeavor is not to propose novel algorithms or tools, but rather to develop a software package that anyone can use easily for robotic learning, without complex setup procedures and painful integration across libraries.

The core reinforcement learning algorithm is derived from RLPD [47], which itself is a variant of soft actor-critic [44]: an off-policy Q-function actor-critic method that can readily incorporate prior data (either suboptimal data or demonstrations) into the replay buffer for efficient learning. The reward functions can be specified either with a binary classifier or VICE [22], which provides a method to update the classifier during RL training with additional negatives from the policy. The reward function can also be specified by hand in cases where the robot state is sufficient to evaluate success (e.g., as in our PCB board assembly task). The resets can be provided via a forward-backward architecture [29], where the algorithm simultaneously trains two policies: a forward policy that performs the task, and a backward policy that resets the environment back to the initial state. On the robot system side, we also provide a universal adapter for interfacing our method to arbitrary robots, as well as an impedance controller that is particularly well-suited for contact-rich manipulation tasks.

A. Core RL Algorithm: RLPD

There are several desiderata for reinforcement learning algorithm to be deployed in this setting: (1) it must be efficient and able to make multiple gradient updates per time step, (2) it must be able to incorporate prior data easily and then continue improving with further experience, (3) it must be simple to debug and build on for new users. To this end, we build on the recently proposed RLPD [47] algorithm, which has shown compelling results on sample-efficient robotic learning. RLPD is an off-policy actor-critic reinforcement learning algorithm that builds on the success of temporal difference algorithms such as soft-actor critic [44], but makes some key modifications to satisfy the desiderata above. RLPD makes three key changes: (i) high update-to-data ratio training (UTD), (ii) symmetric sampling between prior data and on-policy data, such that half of each batch comes from prior data and half from the online replay buffer, and (iii) layer-norm regularization during training. This method can train from scratch, or use prior data (e.g., demonstrations) to bootstrap learning. Each step of the algorithm updates the parameters of a parametric Q-function $Q_\phi(\mathbf{s}, \mathbf{a})$ and actor $\pi_\theta(\mathbf{a}|\mathbf{s})$ according to the gradient of their respective loss functions:

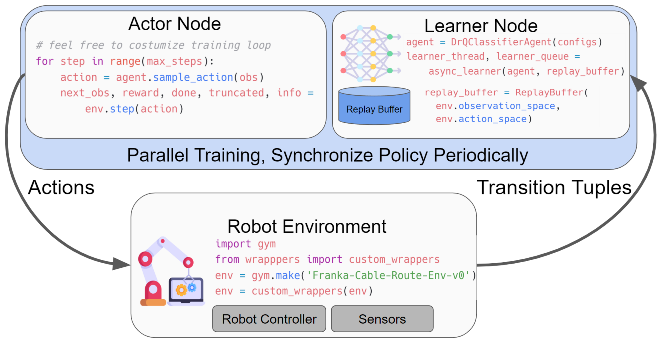
$$\mathcal{L}_Q(\phi) = E_{\mathbf{s},\mathbf{a},\mathbf{s}'}\Big[\big(Q_\phi(\mathbf{s},\mathbf{a}) - \big(r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{a}'\sim\pi_\theta}[Q_{\bar{\phi}}(\mathbf{s}',\mathbf{a}')]\big)\big)^2\Big]$$
$$\mathcal{L}_\pi(\theta) = -E_{\mathbf{s}}\left[E_{\mathbf{a}\sim\pi_\theta(\mathbf{a})}[Q_\phi(\mathbf{s},\mathbf{a})] + \alpha\mathcal{H}(\pi_\theta(\cdot|\mathbf{s})\right],$$

where $Q_{\bar{\phi}}$ is a *target network* [49], and the actor loss uses entropy regularization with an adaptively adjusted weight α [44]. Each update step uses a sample-based approximation of each expectation, with half of the samples drawn from the prior data (e.g., demonstrations), and half drawn from the *replay buffer* [49]. For efficient learning, multiple update steps are performed per time step in the environment, which is referred to as the update-to-date (UTD) ratio, and regularizing the critic with layer normalization allows for higher UTD ratios and thus more efficient training [47].

B. Reward Specification with Classifiers

Reward functions are difficult to specify by hand when learning with image observations, as the robot typically requires some sort of perception system just to determine if the task was performed successfully. While some tasks, such as the PCB board assembly task in Fig. 1, can accommodate hand-specified rewards based on the location of the end effector (under the assumption that the object is held rigidly in the gripper), most tasks require rewards to be deduced from images. In this case, the reward function can be defined by a binary classifier that takes in the state observation $\mathbf{s}$ and outputs the probability of a binary "event" e, corresponding to successful completion. The reward is then given by $r(\mathbf{s}) = \log p(e|\mathbf{s})$.

This classifier can be trained either using hand-specified positive and negative examples, or via an adversarial method called VICE [22]. The latter addresses a reward exploitation problem that can arise when learning with classifier based rewards, and removes the need for negative examples in the classifier training set: when the RL algorithm optimizes the reward $r(\mathbf{s}) = \log p(e|\mathbf{s})$, it can potentially discover "adversarial" states that fool the classifier $p(e|\mathbf{s})$ to erroneously output high probabilities. VICE addresses this issue by adding all states visited by the policy into the training set for the classifier with negative labels, and updating the classifier after each iteration. In this way, the RL process is analogous to a generative adversarial network (GAN) [50], with the policy acting as the generator and the reward classifier acting as the discriminator. Our framework thus supports all three types of rewards.



```
# feel free to costumize training loop
for step in range(max_steps):
    action = agent.sample_action(obs)
    next_obs, reward, done, truncated, info =
        env.step(action)
```

```
agent = DrQClassifierAgent(configs)
learner_thread, learner_queue =
    async_learner(agent, replay_buffer)

replay_buffer = ReplayBuffer(
    env.observation_space,
    env.action_space)
```

```
import gym
from wrapppers import custom_wrappers
env = gym.make('Franka-Cable-Route-Env-v0')
env = custom_wrappers(env)
```

Fig. 2: Software architecture and real-world robot training example codes. The best practice with SERL contains three components: Actor, Learner, and robot environments.

C. Reset-Free Training with Forward-Backward Controllers

When learning episodic tasks, the robot must reset the environment between task attempts. For example, when learning the object relocation task in Figure 1, each time the robot successfully moves the object to the target bin, it must then take it out and place it back into the initial bin. To remove the need for human effort in “resets”, SERL supports “reset-free” training by using forward and backward controllers [51, 52]. In this setup, two policies are trained simultaneously using two independent RL agents, each with their own policy, Q-function, and reward function (specified via the methods in the previous section). The *forward* agent learns to perform the task, and the *backward* agent learns to return to the initial state(s). While more complex reset-free training procedures can also be possible [52], we find that this simple recipe is sufficient for learning object manipulation tasks like the repositioning skill in Figure 1.

D. Software Components

Environment Adapters: SERL aims to be easily usable for many robot environments. Although we provide a set of Gym environment wrappers and robot environments for the Franka and UR robot arms as starter guides, users can also use their own existing environments or develop new environments as they see fit. Thus, the library does not impose additional constraints on the robot environment as long as it is Gym-like [53] as shown in Fig. 2. We welcome contributions from the community to extend the support for readily deployable environment wrappers for other robots and tasks.

Actor and Learner Nodes SERL includes options to train and act in parallel to decouple inferring actions and updating policies with a few lines of code as illustrated in Fig. 2. We found this to be beneficial in sample-efficient real-world learning problems with high UTD ratios. By separating actor and learner on two different threads, SERL not only preserves the control frequency at a fixed rate, which is crucial for tasks that require immediate feedback and reactions, such as deformable objects and contact-rich manipulations, but also reduces the total wall-clock time spend training in the real world.

E. Impedance Controller for Contact-Rich Tasks

Although our package should be compatible with any OEM robot controller as described in Sec. IV, we found

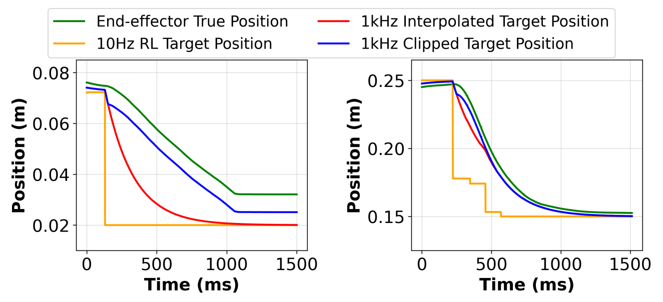


Fig. 3: Visualization of controller logs from the robot when commanded with different movements, for the z-axis of the end-effector. The orange line is the commanded target (the output of RL), red is the smoothed target sent to the real-time controller, blue is the clipped target, and green is the robot position after executing this controller. **Left:** The robot end-effector was commanded to move into contact with a hard surface and continue the movement despite the contact. The reference limiting mechanism clipped the target to avoid a hard collision. **Right:** The command is a fast free-space movement, which our reference limiting mechanism does *not* block, allowing fast motion to the target.

that the choice of controllers can heavily affect the final performance. This is more pronounced for contact-rich manipulation. For example, in the PCB insertion task in Fig. 1, an overly stiff controller might bend the fragile pins and make insertion difficult, while an overly compliant controller might struggle to move the object into position quickly.

A typical setup for robotic RL employs a two-layered control hierarchy, where an RL policy produces set-point actions at a much lower frequency than the downstream real-time controller. The RL controller can set targets for the low-level controller to cause physically undesirable consequences. To illustrate this, let’s consider the hierarchical controller structure presented in Fig. 4, where a high-level RL controller $\pi(\mathbf{a}|\mathbf{s})$ sends control targets at 10HZ for the low-level impedance controller to track at 1K HZ, so one timestep from RL will block 100 timesteps of the low-level controller to execute. A typical impedance control objective for this controller is

$$F = k_p \cdot e + k_d \cdot \dot{e} + F_{ff} + F_{cor},$$

where $e = p - p_{ref}$, p is the measured pose, and p_{ref} is the target pose computed by the upstream controller, F_{ff} is the feed-forward force, F_{cor} is the Coriolis force, this objective will then be converted into joint space torques by multiplying Jacobian transpose and offset by nullspace torques. It acts like a spring-damper system around the equilibrium set by p_{ref} with the stiffness coefficient being k_p and the damping coefficient being k_d. As described above, this system will yield large forces if p_{ref} is far away from the current pose, which can lead to a hard collision or damage when the arm is in contact with something. Therefore it’s crucial to constrain the interaction force generated by it. However, directly reducing gains will hurt the controller’s accuracy. Thus, we should bound e so that $|e| \leq \Delta$, and then the generated force from the spring-damper system will be bounded to $k_p \cdot |\Delta| + 2k_d \cdot |\Delta| \cdot f$, f is the control frequency.

One might wonder if we should directly clip the action output by the RL policy. This might seem reasonable, but

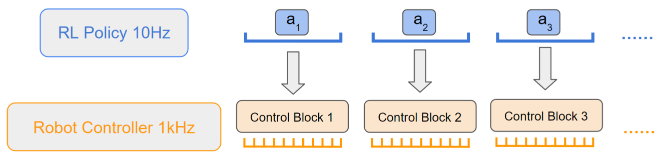


Fig. 4: A typical controller hierarchy for robotics RL. The output from the RL policy is tracked within a block of time by the downstream controller.

can be impractical in some scenarios: some objects such as the PCB board may require a very small interaction force, implying a very small Δ, usually on the order of micrometers; if the RL policy is only allowed to move at increments of micrometers, it would result in an extremely long learning process or very unstable training, because the episode would need enough time steps to allow the arm to move over long distances (e.g., approaching the insertion point). However, if we directly clip at the real-time layer, this will largely mitigate the issue without the need to constrain the RL policy to small actions. It will not block the free space movement of the RL policy as long as $M \cdot |\Delta| \geq |a|_{max}$, where M is the number of control time-steps inside a block, as in Fig. 4. This value is usually large (e.g., $M = 100$). At the same time, we strictly enforce the reference constraint at the real-time level whenever in contact. One can also wonder if it's possible to achieve the same result by using an external force/torque sensor. This may be undesirable for several reasons: (1) force/torque sensors can have significant noise, and obtaining the right hardware and calibration can be difficult; (2) even if we get such a threshold value, it's nontrivial to design robot motions to accommodate policy learning as well as obeying the force constraint. In practice, we found that clipping the reference in this way is simple but very effective, and is crucial to enable RL-based contact-rich manipulation tasks. We tested our controller on a Franka Panda robot and included the Franka Panda implementation with our package. However, this principle can be easily implemented on any torque-controlled robot. To verify the actual performance of the proposed controller, we report the actual tracking performance of moving the robot in free space and in contact with a table surface as in Fig. 3, where we can see the controller indeed clamps the reference whenever in contact, while permitting fast movement in free space.

V. Experiments

Our experimental evaluation aims to study how efficiently our system can learn a variety of robotic manipulation tasks, including contact-rich tasks, deformable object manipulation, and free-floating object manipulation. These experiments demonstrate the breadth of applicability and efficiency of SERL. We use two robots: a Franka Panda arm and a UR5 arm and use wrist cameras attached to their end-effectors to get close-in views. Further details can be found at `https://serl-robot.github.io/`. We use an ImageNet pre-trained EfficientNet [58] as a vision backbone for the policy network and connect it to a 2-layer MLP; observations to the policy include camera images, robot proprioceptive information such as end-effector pose, and twist. The policy outputs a 6D end-effector delta pose from the current pose, and gets tracked by the downstream controller. The tasks are illustrated in Fig. 5 and described below:

PCB insertion: Inserting connectors into a PCB board demands fine-grained, contact-rich manipulation with sub-millimeter precision. This task is ideal for real-world training, as simulating and transferring such contact-rich interactions can be challenging.

Cable routing: This task involves routing a deformable cable into a clip's tight-fitting slot. The challenge arises from managing objects with intricate dynamics and relying on visual perception. It is representative of typical manufacturing and maintenance scenarios where robots route cables or hoses, posing a unique challenge for robotic manipulation.

Object relocation: This task requires moving a free-floating object between bins. It represents common pick-and-place tasks during deployment. The intricacies of reward inference and reset-free training become especially pronounced in manipulation of such free-floating objects.

For each task, we initialize training from 20 teleoperated demonstrations using a SpaceMouse. To confirm that these demos alone are insufficient to solve the task, we include a behavioral cloning (BC) baseline. All training was done on a single Nvidia RTX 4090 GPU.

a) Results: We report the results in Table I, and show their execution in Fig. 5. Our RL policies achieve a nearly perfect success rate on two of these tasks within an hour of wall-clock time, which includes computation, resets, and intended stops. The object relocation task learns two policies (forward and backward), and total time amounts to less than an hour *per policy*. For the cable routing task and PCB insertion task, our policies outperform BC baselines by a large margin, suggesting the demos alone are insufficient. We also visualize the learned Q values heatmap over time during one trial of policy learning in Fig. 6: we fix the z coordinate, and observe the changes of Q values over time for different state-action pairs in that surface. We find that the Q values converge as the learning proceeds, resulting in deterministic maximizing actions.

b) Comparison to prior systems: While it's difficult to directly compare our results to those of prior systems due to numerous differences in the setup, lack of consistently open-sourced code, and other discrepancies, we provide a summary of training times and success rates reported for tasks that are most similar to our PCB board insertion task in Table I. We chose this task because similar insertion or assembly tasks have been studied in prior work, and such tasks often present challenges with precision, compliant control, and sample efficiency. Compared to these prior works, our experiments do not use shaped rewards, which might require extensive engineering, though we do utilize a small amount of demonstration data (which some prior works eschew). The results reported in these prior works generally have either lower success rates or longer training times, or both, suggesting our implementation of sample-efficient RL matches or exceeds the performance of state-of-the-art methods in the literature, at least on this type of task. The closest performance to ours in the work of Spector et al. [57]

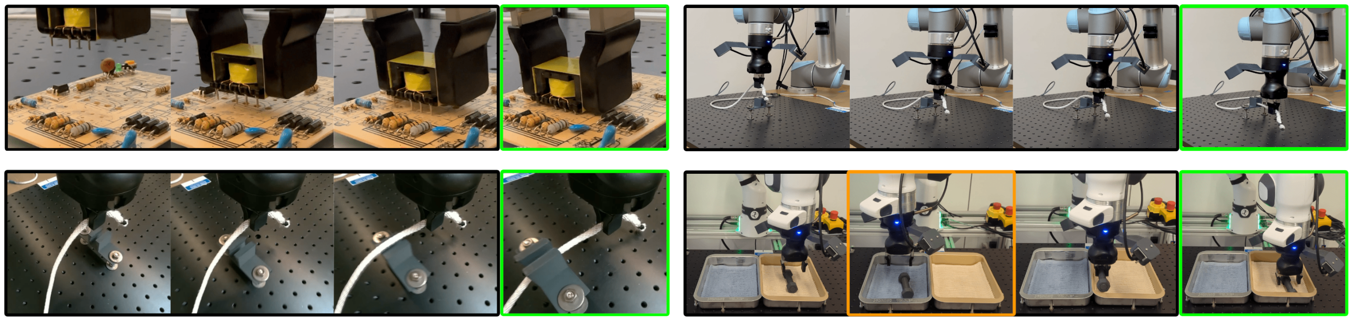


Fig. 5: Illustration of the robot performing each task with our method: PCB insertion (top left), cable routing with UR5 (top right), cable routing with Franka Panda (bottom left), and object relocation (bottom right). The green box indicates a state where the robot receives high reward for completing the task. In the object relocation task, the orange and green boxes represent the completion of the forward and backward tasks in reset-free RL training.

Package	Task	Training time	Success rate	Demos	Shaping?	Vision?	Open-sourced?
Levine et al. 2016 [4]	Peg insertion	3 hours	70%	0	Yes	Yes	Yes
Vecerik et al. 2018 [54]	Peg/clip insertion	1.5-2.5 hours	97% / 77%	30	No	Yes	No
Schoettler et al. 2019 [55]	Connector insertion	Not mentioned	52% $\sim$ 100%	0	Yes	Yes	No
Luo et al 2021 [56]	Connector insertion	1.5 hours	99.8%	25	No	Yes	No
Spector et al 2021 [57]	Connector insertion	40 mins	78.5% - 100%	0	Yes	Yes	No
Ours	PCB Insertion	36 mins	100%	20	No	Yes	Yes

TABLE I: Comparison to results reported on similar tasks in prior work. The overall success rates for our method are generally higher, and the training times are generally lower, as compared to prior results. Note also that the PCB board assembly task, shown in Figure 1, has very tight tolerances, likely significantly tighter than the coarser peg and connector insertion tasks studied in the prior works.

Task	RL Time	RL Results (ours)	BC Results (baseline)
PCB Insertion	36 min	100/100	64/100
Object relocation	120 min	84/100	70/100
UR5 Cable Routing	90 min	96/100	N/A
Franka Cable Routing	60 min	100/100	61/100

TABLE II: Comparison to results reported on similar tasks in prior work. The overall success rates for our method are generally higher, and the training times are generally lower, as compared to prior results. Note also that the PCB board assembly task, shown in Figure 1, has very tight tolerances, likely significantly tighter than the coarser peg and connector insertion tasks studied in the prior works.

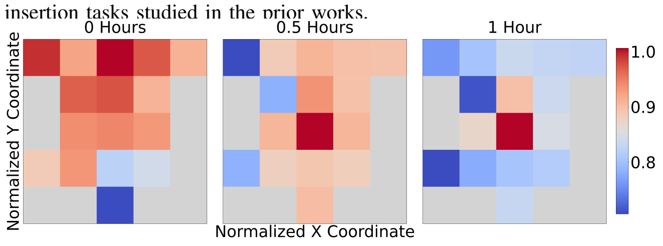


Fig. 6: A visualization of learned Q values during training, we group states nearby into a grid, sample actions using the current policy, and report the average Q values.

includes a number of design decisions and inductive biases that are specific to insertion, whereas our method is generic and makes minimal task-specific assumptions. Although the components of our system are all based on (recent) prior work, the state-of-the-art performance of this combination illustrates our main thesis: the details of how deep RL methods are implemented can make a big difference.

VI. Discussion

We described a system and software package for robotic reinforcement learning that is aimed at making real-world RL more accessible both to researchers and practitioners. Our software package provides a carefully designed combination of ingredients for sample-efficient RL, automating reward design, automating environment resets with forward-backward controllers, and a controller framework that is particularly well-suited for contact-rich manipulation tasks. Furthermore, our experimental evaluation of our framework demonstrates that it can learn a range of diverse manipulation tasks very efficiently, with under an hour of training per policy when provided with a small number of demonstrations. These results qualitatively compare well to state-of-the-art results in RL for manipulation in the literature, indicating that the particular choices in our framework are well-suited for obtaining very good real-world results even from image observations. Our framework does have a number of limitations. First, we do not aim to provide a comprehensive library of every possible RL method, some tasks and settings might be outside of our framework (e.g., non-manipulation tasks). Second, the full range of reward specifications and reset-free learning challenges still constitute an open problem in robotic RL research. Our classifier-based rewards and forward-backward controller might not be appropriate in every setting. Further research on these topics is needed to make robotic RL more broadly applicable but we do however hope that our software package will provide a reasonable "default" starting point for both researchers and practitioners wanting to experiment with real-world RL methods.

References

[1] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters. "Learning to Play Table Tennis From Scratch Using Muscular Robots". In: *IEEE Trans. Robotics* 38.6 (2022), pp. 3850–3860. URL: https://doi.org/10.1109/TRO.2022.3176207.

[2] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine. "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention". In: *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, 2021, pp. 6664–6671. URL: https://doi.org/10.1109/ICRA48506.2021.9561384.

[3] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. "MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *CoRR* abs/2104.08212 (2021). arXiv: 2104.08212. URL: https://arxiv.org/abs/2104.08212.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[5] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *Int. J. Robotics Res.* 37.4-5 (2018), pp. 421–436. URL: https://doi.org/10.1177/0278364917710318.

[6] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*. Ed. by N. M. Amato, S. S. Srinivasa, N. Ayanian, and S. Kuindersma. 2017. URL: http://www.roboticsproceedings.org/rss13/p58.html.

[7] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=S1vuO-bCW.

[8] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. "Reinforcement learning for robot soccer". In: *Autonomous Robots* 27 (2009), pp. 55–73.

[9] T. Westenbroek, F. Castaneda, A. Agrawal, S. Sastry, and K. Sreenath. "Lyapunov design for robust and efficient robotic reinforcement learning". In: *arXiv preprint arXiv:2208.06721* (2022).

[10] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani. "Data efficient reinforcement learning for legged robots". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1–10.

[11] A. Zhan, R. Zhao, L. Pinto, P. Abbeel, and M. Laskin. "A framework for efficient robotic manipulation". In: *Deep RL Workshop NeurIPS 2021*. 2021.

[12] Z. Hou, J. Fei, Y. Deng, and J. Xu. "Data-efficient hierarchical reinforcement learning for robotic assembly control applications". In: *IEEE Transactions on Industrial Electronics* 68.11 (2020), pp. 11565–11575.

[13] J. Tebbe, L. Krauch, Y. Gao, and A. Zell. "Sample-efficient reinforcement learning in robotic table tennis". In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, pp. 4171–4178.

[14] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. "Data-efficient deep reinforcement learning for dexterous manipulation". In: *arXiv preprint arXiv:1704.03073* (2017).

[15] I. Kostrikov, L. M. Smith, and S. Levine. "Demonstrating A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning". In: *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*. Ed. by K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu. 2023. URL: https://doi.org/10.15607/RSS.2023.XIX.056.

[16] Z. Hu, A. Rovinsky, J. Luo, V. Kumar, A. Gupta, and S. Levine. "REBOOT: Reuse Data for Bootstrapping Efficient Real-World Dexterous Manipulation". In: *CoRR* abs/2309.03322 (2023). arXiv: 2309.03322. URL: https://doi.org/10.48550/arXiv.2309.03322.

[17] T. Hester and P. Stone. "Texplore: real-time sample-efficient reinforcement learning for robots". In: *Machine learning* 90 (2013), pp. 385–429.

[18] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg. "DayDreamer: World Models for Physical Robot Learning". In: *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*. Ed. by K. Liu, D. Kulic, and J. Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2022, pp. 2226–2240. URL: https://proceedings.mlr.press/v205/wu23c.html.

[19] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. "Deep Dynamics Models for Learning Dexterous Manipulation". In: *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*. Ed. by L. P. Kaelbling, D. Kragic, and K. Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1101–1112. URL: http://proceedings.mlr.press/v100/nagabandi20a.html.

[20] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn. "Offline Reinforcement Learning from Images with Latent Space Models". In: *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland*. Ed. by A. Jadbabaie, J. Lygeros, G. J. Pappas, P. A. Parrilo, B. Recht, C. J. Tomlin, and M. N. Zeilinger. Vol. 144. Proceedings of Machine Learning Research. PMLR, 2021, pp. 1154–1168. URL: http://proceedings.mlr.press/v144/rafailov21a.html.

[21] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. "Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost". In: *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 3651–3657. URL: https://doi.org/10.1109/ICRA.2019.8794102.

[22] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. "Variational inverse control with events: A general framework for data-driven reward definition". In: *Advances in neural information processing systems* 31 (2018).

[23] K. Li, A. Gupta, A. Reddy, V. H. Pong, A. Zhou, J. Yu, and S. Levine. "MURAL: Meta-Learning Uncertainty-Aware Rewards for Outcome-Driven Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 6346–6356. URL: http://proceedings.mlr.press/v139/li21g.html.

[24] Y. Du, K. Konyushkova, M. Denil, A. Raju, J. Landon, F. Hill, N. de Freitas, and S. Cabi. "Vision-language models

as success detectors". In: *arXiv preprint arXiv:2303.07280* (2023).

[25] P. Mahmoudieh, D. Pathak, and T. Darrell. "Zero-Shot Reward Specification via Grounded Natural Language". In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Ed. by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 14743–14752. URL: https://proceedings.mlr.press/v162/mahmoudieh22a.html.

[26] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D. Huang, Y. Zhu, and A. Anandkumar. "MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge". In: *NeurIPS*. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/74a67268c5cc5910f64938cac4526a90-Abstract-Datasets%5C_and%5C_Benchmarks.html.

[27] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang. "VIP: Towards Universal Visual Reward and Representation via Value-Implicit Pre-Training". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=YJ7o2wetJ2.

[28] Y. J. Ma, V. Kumar, A. Zhang, O. Bastani, and D. Jayaraman. "LIV: Language-Image Representations and Rewards for Robotic Control". In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 23301–23320. URL: https://proceedings.mlr.press/v202/ma23b.html.

[29] A. Sharma, K. Xu, N. Sardana, A. Gupta, K. Hausman, S. Levine, and C. Finn. "Autonomous Reinforcement Learning: Benchmarking and Formalism". In: *arXiv preprint arXiv:2112.09605* (2021).

[30] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine. "The Ingredients of Real World Robotic Reinforcement Learning". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rJe2syrtvS.

[31] A. Xie, F. Tajwar, A. Sharma, and C. Finn. "When to Ask for Help: Proactive Interventions in Autonomous Reinforcement Learning". In: *NeurIPS*. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/6bf82cc56a5fa0287c438baa8be65a70-Abstract-Conference.html.

[32] A. Sharma, A. M. Ahmed, R. Ahmad, and C. Finn. "Self-Improving Robots: End-to-End Autonomous Visuomotor Reinforcement Learning". In: *CoRR* abs/2303.01488 (2023). arXiv: 2303.01488. URL: https://doi.org/10.48550/arXiv.2303.01488.

[33] T. Seno and M. Imai. "d3rlpy: An Offline Deep Reinforcement Learning Library". In: *Journal of Machine Learning Research* 23.315 (2022), pp. 1–20. URL: http://jmlr.org/papers/v23/22-0017.html.

[34] A. Nair and V. Pong. "rlkit". In: *Github* (). URL: https://github.com/rail-berkeley/rlkit.

[35] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[36] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. https://github.com/tensorflow/agents. [Online; accessed 25-June-2019]. 2018. URL: https://github.com/tensorflow/agents.

[37] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*. Ed. by L. P. Kaelbling, D. Kragic, and K. Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1094–1100. URL: http://proceedings.mlr.press/v100/yu20a.html.

[38] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. "RLBench: The Robot Learning Benchmark & Learning Environment". In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 3019–3026. URL: https://doi.org/10.1109/LRA.2020.2974707.

[39] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, P. P. Tehrani, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. *ORBIT: A Unified Simulation Framework for Interactive Robot Learning Environments*. 2023. eprint: arXiv:2301.04195.

[40] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar. "ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots". In: *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*. Ed. by L. P. Kaelbling, D. Kragic, and K. Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1300–1313. URL: http://proceedings.mlr.press/v100/ahn20a.html.

[41] M. Heo, Y. Lee, D. Lee, and J. J. Lim. "FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation". In: *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*. Ed. by K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu. 2023. URL: https://doi.org/10.15607/RSS.2023.XIX.041.

[42] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-End Training of Deep Visuomotor Policies". In: *J. Mach. Learn. Res.* 17 (2016), 39:1–39:40. URL: http://jmlr.org/papers/v17/15-522.html.

[43] V. R. Konda and J. N. Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. The MIT Press, 1999, pp. 1008–1014. URL: http://papers.nips.cc/paper/1786-actor-critic-algorithms.

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[45] S. Fujimoto, H. van Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR,

2018, pp. 1582–1591. URL: http://proceedings.mlr.press/v80/fujimoto18a.html.

[46] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018.

[47] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. “Efficient online reinforcement learning with offline data”. In: *arXiv preprint arXiv:2302.02948* (2023).

[48] A. Nair, M. Dalal, A. Gupta, and S. Levine. *Accelerating Online Reinforcement Learning with Offline Datasets*. 2020. arXiv: 2006.09359 [cs.LG].

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[50] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[51] W. Han, S. Levine, and P. Abbeel. “Learning compound multi-step controllers under unknown dynamics”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 6435–6442.

[52] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine. “Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6664–6671.

[53] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[54] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz. *A Practical Approach to Insertion with Variable Socket Position Using Deep Reinforcement Learning*. 2018. arXiv: 1810.01531 [cs.RO].

[55] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine. *Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards*. 2019. arXiv: 1906.05841 [cs.RO].

[56] J. Luo, O. Sushkov, R. Pevceviciute, W. Lian, C. Su, M. Vecerik, N. Ye, S. Schaal, and J. Scholz. “Robust Multi-Modal Policies for Industrial Assembly via Reinforcement Learning and Demonstrations: A Large-Scale Study”. In: *Proceedings of Robotics: Science and Systems*. Virtual, July 2021.

[57] O. Spector and D. D. Castro. *InsertionNet – A Scalable Solution for Insertion*. 2021. arXiv: 2104.14223 [cs.RO].

[58] M. Tan and Q. Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.