SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark

Linxi Fan[*] **Yuke Zhu**[*] **Jiren Zhu** **Zihua Liu** **Orien Zeng**

Anchit Gupta **Joan Creus-Costa** **Silvio Savarese** **Li Fei-Fei**

Stanford Vision and Learning Lab (SVL)
Department of Computer Science, Stanford University
surreal.stanford.edu

Abstract: Reproducibility has been a significant challenge in deep reinforcement learning and robotics research. Open-source frameworks and standardized benchmarks can serve an integral role in rigorous evaluation and reproducible research. We introduce SURREAL, an open-source scalable framework that supports state-of-the-art distributed reinforcement learning algorithms. We design a principled distributed learning formulation that accommodates both on-policy and off-policy learning. We demonstrate that SURREAL algorithms outperform existing open-source implementations in both agent performance and learning efficiency. We also introduce SURREAL Robotics Suite, an accessible set of benchmarking tasks in physical simulation for reproducible robot manipulation research. We provide extensive evaluations of SURREAL algorithms and establish strong baseline results.

Keywords: Robot Manipulation, Reinforcement Learning, Distributed Learning Systems, Reproducible Research

1 Introduction

Reinforcement learning (RL) has been an established framework in robotics to learn controllers via trial and error [1]. Classic reinforcement learning literature in robotics has largely relied on handcrafted features and shallow models [2, 3]. The recent success of deep neural networks in learning representations [4] has incentivized researchers to use them as powerful function approximators to tackle more complex control problems, giving rise to *deep reinforcement learning* [5, 6]. Prior work has explored the potentials of using deep RL for robot manipulation [7, 8]. Recently we have witnessed an increasing number of successful demonstrations of deep RL in simulated environments [9, 10, 11] and on real hardware [12, 13].

However, the challenges of reproducibility and replicability in deep RL have impaired research progress [14]. Today's deep RL research has not been as accessible as it should be. Reproducing and validating published results is rarely straightforward, as it can be affected by numerous factors such as hyperparameter choices, initializations, and environment stochasticity, especially in absence of open-source code releases. Furthermore, owing to the data-hungry nature of deep RL algorithms, we have observed a rising number of state-of-the-art results being achieved by highly sophisticated distributed learning systems [15, 16, 17]. The heavy engineering factor in cutting-edge deep RL research has increased the barrier of entry even more for new researchers and small labs. Meanwhile, benchmarking and reproducibility of robotics research have also been a long-standing challenge in the robotics community [18, 19]. Attempts to improve this situation include annual robotic competitions [20, 21] and standardized object sets [22]. However, designing robotic benchmarks that can be both standardized and widely accessible remains to be an open problem.

Many fields of AI, e.g., computer vision, have made significant progress powered by open-source software tools [23, 24, 25] and standardized benchmarks [26]. To sustain and facilitate the research of

[*]These two authors contributed equally. Email: {jimfan,yukez}@cs.stanford.edu

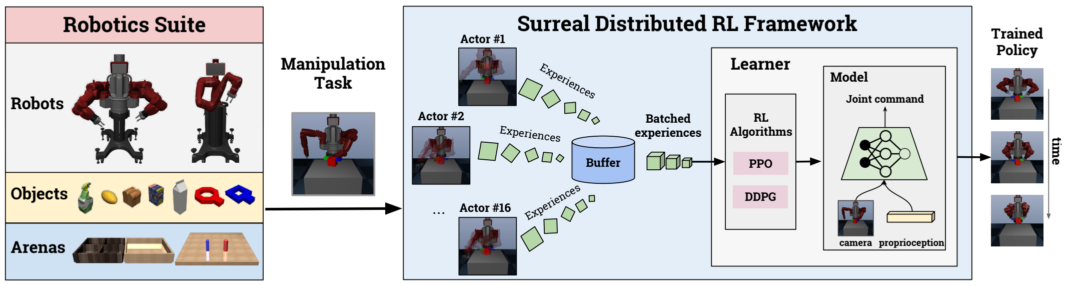

Figure 1: SURREAL is an open-source framework that facilitates reproducible deep reinforcement learning (RL) research for robot manipulation. We implement scalable reinforcement learning methods that can learn from parallel copies of physical simulation. We also develop Robotics Suite as an accessible benchmark for evaluating the RL agents' performances.

deep RL in robotics, we envision that it is vital to provide a flexible framework for rapid development of new algorithms and a standardized robotics benchmark for rigorous evaluation.

In this paper, we introduce the open-source framework SURREAL (**S**calable **R**obotic **RE**inforcement-learning **AL**gorithms). Standard approaches to accelerating deep RL training focus on parallelizing the gradient computation [27, 28]. SURREAL decomposes a distributed RL algorithm into four components: generation of experience (*actors*), storage of experience (*buffer*), updating parameters from experience (*learner*), and storage of parameters (*parameter server*). This decoupling of data generation and learning eliminates the need of global synchronization and improves scalability. SURREAL offers an umbrella support to distributed variants of both on-policy and off-policy RL algorithms. To enable scalable learning, we develop a four-layer computing infrastructure on which RL experiments can be easily orchestrated and managed. The system can be deployed effortlessly on commercial cloud providers or personal computers. Thanks to the layered design, our system can be fully replicated from scratch, which also contributes to the reproducibility of our experiments.

Furthermore, we introduce SURREAL Robotics Suite, a diverse set of physics engine-based robotic manipulation tasks, as an accessible benchmark for evaluating RL algorithms. Simulated systems have been traditionally used as a debugging tool in robotics to perform *mental rehearsal* prior to real-world execution [1]. A series of successful attempts have been made in utilizing simulated data for learning robot controllers [9, 10, 11, 29, 30]. We expect the simulation-reality gap to be further narrowed with more advanced simulation design and policy transfer techniques. We hope that this standardized benchmark, along with the open-source SURREAL codebase, will accelerate future research in closing the reality gap.

To this end, we develop well performing and distributed variants of PPO [31] and DDPG [6], called SURREAL-PPO and SURREAL-DDPG. We examine them in six of the Robotics Suite tasks with single-arm and bimanual robots. We report performance in various setups, including training on physical states or raw pixels, RL from scratch or aided by VR-based human demonstrations. We also quantify the scalability of distributed RL framework compared to popular open-source RL implementations [32, 33] in OpenAI Gym environments [34], the *de facto* standard continuous RL benchmark. The experiments show that SURREAL algorithms are able to achieve strong results and high scalability with increased numbers of parallel actors.

2 Related Work

Deep Reinforcement Learning in Robotics Deep RL methods have been applied to mobile robot navigation [35, 36] and robot arm manipulation [7, 8, 12, 13, 37]. Both model-based and model-free RL approaches have been studied. Model-based methods [7, 8, 38] often enjoy sample efficiency of learning, but pose significant challenges of generalization due to their strong model assumptions. Model-free methods [10, 12, 13, 37] are more flexible, but usually require large quantity of data. In this work, we build a distributed learning framework to offer a unified support of two families of model-free continuous RL methods: value-based methods based on deterministic policy gradients [6] and trust-region methods [39, 40, 31]. Our focus on developing model-free RL methods and building simulated robotic benchmarks is encouraged by a series of recent progress on simulation-to-reality policy transfer techniques [9, 10, 11, 29, 30].

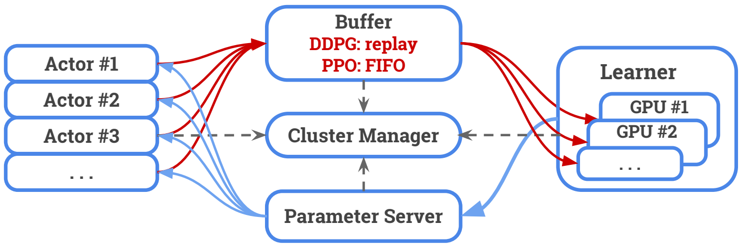

Figure 2: The SURREAL distributed components consists of *actors*, *buffer*, *learner*, and *parameter server*. The red arrows denote experience data flow and the blue arrows denote neural network parameter flow. All components report learning and system statistics to the *cluster manager*.

Distributed Deep RL Frameworks As the learning community tackles problems of growing sizes and wider varieties, distributed learning systems have played an integral role in scaling up today's learning algorithms to unprecedented scales [15, 41]. The high sample complexity and exploration challenge in deep RL have accentuated the advantages of distributed RL frameworks. Prior approaches have relied on asynchronous SGD-style learning (e.g., A3C [27], Gorila [42], ADPG-R [28]), batched data collection for high GPU throughput (e.g., batched A2C [43], GA3C [44], BatchPPO [32]), and more recently, multiple CPU actors for experience generation and single GPU learner for model update (e.g., Ape-X [16] and IMPALA [17]). SURREAL differs from asynchronous gradient methods like A3C, because the latter shares gradients between decentralized learners instead of sending experience, which is less desirable because gradients become outdated more rapidly than experience data. To date, Ape-X and IMPALA have reported state-of-the-art results with off-policy RL in several benchmarks. SURREAL resembles these two methods, which also separate experience generation from centralized learning. The major difference is that SURREAL provides a unified approach towards both on-policy trust region algorithms [31, 40] and off-policy value-based algorithms [6, 45].

3 SURREAL Distributed Reinforcement Learning Framework

SURREAL's goal is to provide highly scalable implementations of distributed RL algorithms for continuous control. We develop distributed variants of the on-policy PPO [31] and off-policy DPG [6] algorithms, and unify them under a single algorithmic framework. We further develop a distributed computing infrastructure that can be easily replicated and deployed. Here we start with a brief review of the basics of the PPO and DPG algorithms.

Proximal Policy Optimization Policy gradient algorithms are a robust family of continuous control techniques that directly maximize the expected sum of rewards. The vanilla policy gradient estimator is given by $\nabla_\theta J_{\mathrm{PG}} = \mathbb{E}_{\tau_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right]$, where τ_θ are the trajectories induced by the stochastic policy π_θ and A_t is the advantage function. Trust Region Policy Optimization (TRPO) [39] reduces the variance of policy gradients by enforcing a hard constraint on the Kullback-Leibler (KL) divergence of the old and new policies. More recently, Proximal Policy Optimization (PPO) [31] has been proposed as a first-order approximation to TRPO that adaptively regulates the strength of the KL regularization. PPO can be easily integrated with recurrent neural networks (RNN) in a distributed setting [40], and has been shown robust towards hyperparameters [31].

Deterministic Policy Gradient Value-based methods learn to maximize state-action value function $Q_\pi(s, a) = \mathbb{E}\left[R_t|s, a\right]$, i.e., the expected value of total returns over successive steps, via the Bellman update. Deterministic Policy Gradient (DPG) [45] is an off-policy actor-critic method to learn a policy (actor) that maximizes the estimated expected return computed by a Q-value function (critic): $\nabla_\theta J_{\mathrm{DPG}} = \mathbb{E}_{\tau_\theta}\left[\nabla_a Q_\pi(s, a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)\right]$. Deep DPG (DDPG) [6] represents both the actor and the critic by neural networks. We use the experience replay technique [5] to store past experiences and sample them in batches to train the policy with DPG gradient updates.

3.1 SURREAL Distributed RL Design

The distributed RL formulation in SURREAL consists of four major components illustrated in Fig. 2: actors, buffer, learner, and parameter server. Our key idea is to separate experience generation from learning. Parallel *actors* generate massive amount of experiences in the form of state-action transition tuples (s_t, a_t, s_{t+1}, r_t), while a centralized learner performs model updates. Each actor explores independently, which allows them to diversify the collectively-encountered state spaces.

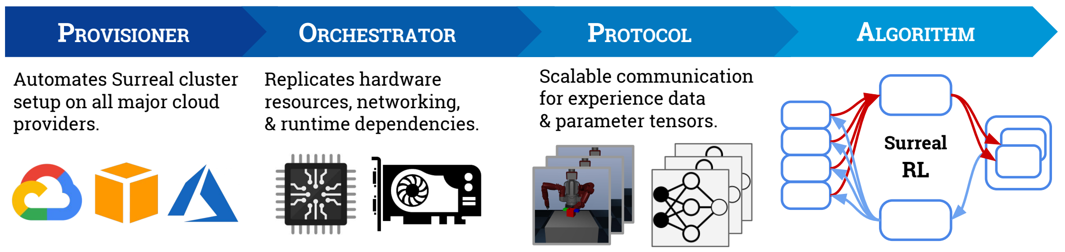
PROVISIONER
ORCHESTRATOR
PROTOCOL
ALGORITHM
Automates Surreal cluster setup on all major cloud providers.
Replicates hardware resources, networking, & runtime dependencies.
Scalable communication for experience data & parameter tensors.
Surreal RL

Figure 3: SURREAL reproducible and scalable learning infrastructure. The four layers from left to right are increasingly abstracted away from the hardware.

This alleviates the exploration challenge in long-horizon robotic manipulation tasks. The centralized learning eliminates global locking and reduces implementation complexity. Based on these design principles, we develop distributed versions of PPO and DDPG algorithms, which we will refer to as SURREAL-PPO and SURREAL-DDPG (pseudocode in Appendix).

On-policy and off-policy deep RL methods employ two different mechanisms of consuming experiences for learning. We introduce a centralized *buffer* structure to support both. In the on-policy case, the buffer is a FIFO queue that holds experience tuples in a sequential ordering and discards experiences right after the model updates. In the off-policy case, the buffer becomes a fixed-size replay memory [5] that uniformly samples batches of data upon request to allow experience reusing. The buffer can be sharded on multiple nodes to increase networking capacity.

The *learner* continuously pulls batches of experiences from the buffer and performs algorithm-specific parameter updates. Because learning is centralized, it can take advantage of multi-GPU parallelism. Periodically, the learner posts the latest parameters to the *parameter server*, which then broadcasts to all actors to update their behavior policies.

Due to our design's asynchronous nature and inevitable network latency, the actors' behavior policies that generate the experience trajectories can lag behind the learner's policy by several updates at the time of gradient computation. This would cause harmful off-policyness for on-policy methods. IMPALA [17] addresses this discrepancy by using a correction technique called V-trace. We propose a simpler alternative. SURREAL-PPO learner keeps a target network that is broadcasted to all actors at a lower frequency. This ensures that a much larger portion of the experience trajectories are on-policy, except for those generated within the policy lag (i.e. system delay between parameter server broadcasting target network parameters, to actors actually updating the behavior policy to the target network). Empirically, we find that the target network mechanism balances the trade-off between learning speed and algorithmic stability, which is crucial for agent performance.

3.2 SURREAL Heterogeneous Computing Infrastructure

Distributed RL, unlike data parallelism commonly used in supervised learning, requires complex communication patterns between heterogeneous components as seen in Fig. 2. This has increased the burden of engineering in distributed RL research. Our goal is to open source a set of well-engineered computing infrastructure that makes the runtime setup effortless to upper-level algorithm designers, with *reproducibility* and *scalability* as our guiding principles.

We design a four-layer distributed learning pipeline shown above, which decouples the RL algorithms from the underlying infrastructure (Fig. 3). SURREAL pipeline starts with the *provisioner* that guarantees the reproducibility of our cluster setup across Google Cloud, AWS, and Azure. The next layer, *orchestrator*, uses a well-established cloud API (Kubernetes) to allocate CPU/GPU resources and replicate the networking topology of our experiments. We use docker images to ensure that the runtime environment and dependencies can be exactly reproduced. Further down the pipeline, the *protocol* implements efficient communication directives. Some components can be sharded and load-balanced across multiple nodes to boost performance even further. We implement our *algorithms* in PyTorch [25] with benefits of fast prototyping and dynamic computation graphs.

Among open-source distributed RL libraries, TensorFlow-Agent [32], OpenAI Baselines [33], and GA3C [44] provide very limited support for multi-node training, while SURREAL can easily scale to hundreds of CPUs and GPUs. Ray [46] is one of the systems that natively feature multi-node training. It has preliminary support for cloud, but only applies to AWS and does not automate cluster setup in a systematic manner as we do.

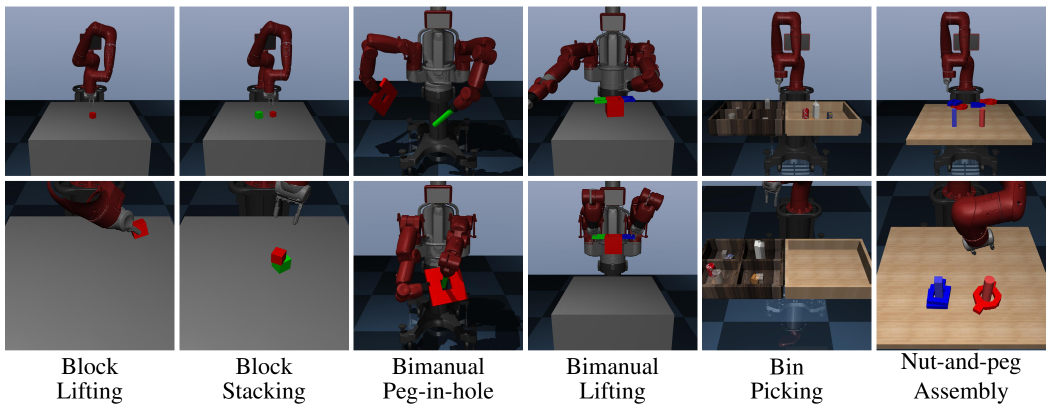

Figure 4: Six robot benchmarking environments. The first row shows initial configurations and the second row shows the states of task completion. When trained on raw pixel inputs, the agents take the RGB observations from the same cameras as illustrated in the second row.

4 Robotics Suite: Simulated Robot Manipulation Benchmark

We aim to build a standardized and widely accessible benchmark with high-quality physical simulation, motivated by a series of recent work on leveraging simulated data for robot learning [9, 10, 29]. We develop the Robotics Suite in the MuJoCo physics engine [47], which simulates fast with multi-joint contact dynamics. It has been a favorable choice adopted by existing continuous control benchmarks [34, 48]. We provide OpenAI gym-style interfaces [34] in Python with detailed API documentations, along with tutorials on how to import new robots and create new environments and new tasks. We highlight four primary features in our suite: 1) *procedural generation* (Fig. 1): we provide a modularized API to programmtically generate combinations of robot models, arenas, and parameterized 3D objects, enabling us to train policies with better robustness and generalization; 2) *control modes*: we support joint velocity controllers and position controllers to command the robots; 3) *multi-modal sensors*: we support heterogeneous types of sensory signals, including low-level physical states, RGB cameras, depth maps, and proprioception; and 4) *teleoperation*: we support using 3D motion devices, such as VR controllers, to teleoperate the robots and collect human demonstrations. Our current release of the benchmark consists of six manipulation tasks as illustrated in Fig. 4. We plan to keep expanding the benchmark with additional tasks, new robot models, and more advanced physics and graphics engines.

Block Lifting: A cube is placed on the tabletop. The Sawyer robot is rewarded for lifting the cube with a parallel-jaw gripper. We randomize the size and the placement of the cube.

Block Stacking: A red cube and a green cube are placed on the tabletop. The Sawyer robot is rewarded for lifting the red cube with a parallel-jaw gripper and stack it on top of the green cube.

Bimanual Peg-in-hole: The Baxter robot holds a board with a squared hole in the center in its right hand, and a long stick in the left hand. The goal is to move both arms to insert the peg into the hole.

Bimanual Lifting: A pot with two handles is placed on the tabletop. The Baxter robot is rewarded for lifting the pot above the table by a threshold while not tilting the pot over 30 degrees. Thus the robot has to coordinate its two hands to grasp the handles and balance the pot.

Bin Picking: The Sawyer robot tackles a pick-and-place task, where the goal is to pick four objects from each category in a bin and to place them into their corresponding containers.

Nut-and-peg Assembly: Two colored pegs are mounted to the tabletop. The Sawyer robot needs to declutter the nuts lying on top of each other and assembles them onto their corresponding pegs.

Our rationale of designing these tasks is to offer single-arm and bimanual manipulation tasks of large diversity and varying complexity. The complexity of a task is measured by the estimated number of steps an optimal agent can solve it and from the mean and variance of the time required by an experienced human operator using teleoperation routines that interface with a virtual reality controller. These mean episode durations from the successful human demonstrations — taken to be a proxy for task difficulty — are shown in Fig. 5. We infer that the Nut-and-peg Assembly and Bin Picking

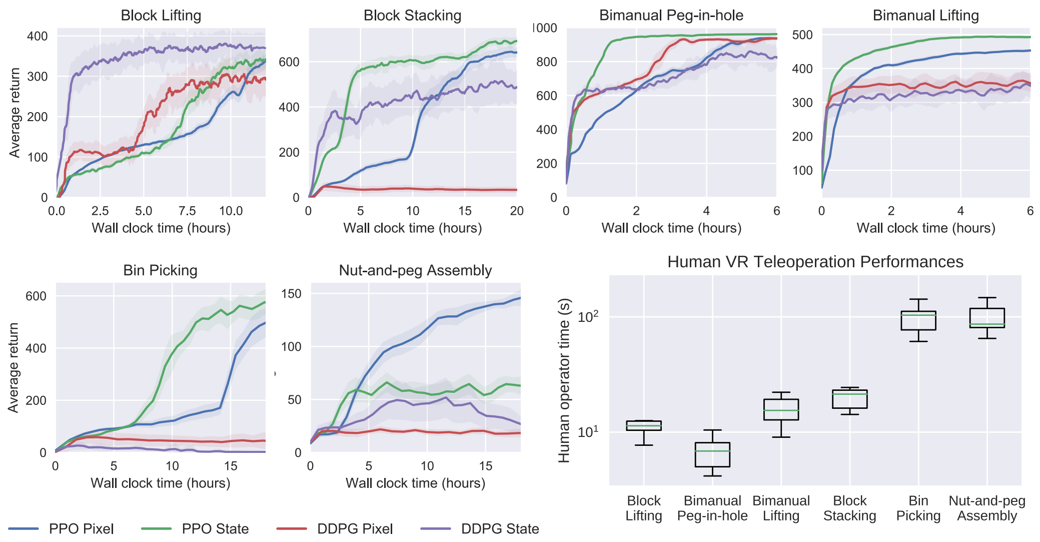

Figure 5: We report training curves of our SURREAL-DDPG and SURREAL-PPO on six SURREAL Robotics Suites tasks. The training curve represents the mean with standard deviation as the translucent band. We train the agents on both ground-truth physical states and raw pixel observations. We measure the complexity of the tasks as the median completion time by an experienced human operator using VR controllers to teleoperate the robots using APIs in Robotics Suite.

tasks are the hardest, whereas the Block Lifting and Block Stacking tasks are relatively easier. These human demonstrations were also used by our RL algorithms to accelerate exploration, as explained in Sec. 5.1. In the next section, we provide quantitative evaluations of SURREAL algorithms on these benchmarking tasks.

5 Experiments

We evaluate our SURREAL distributed RL algorithms in all six benchmarking tasks introduced in Sec. 4. For each task, we train RL agents with SURREAL-PPO and SURREAL-DDPG algorithms on two settings: ground-truth physical states and raw pixel observations. In the former case, we use low-dimensional object features (object positions, rotations, etc.) and proprioceptive features (robot joint positions and velocities) as input. In the latter case, we use RGB camera observations and proprioceptive features, which are usually available on real robots. We provide detailed experiment specifications, such as network architectures, hyperparameters, training configurations, in the Appendix. Qualitative results can be viewed at `http://surreal.stanford.edu`.

5.1 Performances: Robotics Suite

Fig. 5 shows learning curves of our SURREAL-DDPG and SURREAL-PPO implementations on six Robotics Suite environments. We notice that SURREAL-PPO is able to converge to policies with lower standard deviations than SURREAL-DDPG. This is because SURREAL-PPO adjusts its exploration noise standard deviation throughout training, whereas SURREAL-DDPG uses fixed exploration noise. This is also reflected by the fact that the mean reward of SURREAL-PPO varies more smoothly than that of SURREAL-DDPG.

Block Lifting and Bimanual Peg-in-hole are the easiest tasks that consist of a single stage. Both algorithms can solve the tasks with policies trained on both input modalities. We notice that training time for Block Lifting is longer than Bimanual Peg-in-hole even though these two tasks are similar in complexity measured by median human completion time illustrated in Fig. 5. We hypothesize that this is caused by random initialization of object size in addition to position and orientation. Thus, convergence of training indicates that our algorithms can find robust solutions capable of adapting to environment variations.

The tasks of Block Stacking (grasping, lifting, and stacking) and Bimanual Lifting (grasping handles and lifting the pot) have longer horizons. Our algorithms can solve subtasks and achieve intermediate

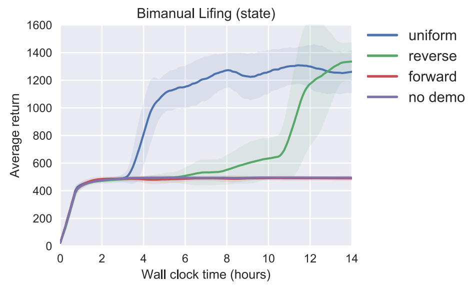

Figure 6: PPO agent trained on state for the Bimanual Lifting task with different types of demonstration curricula.

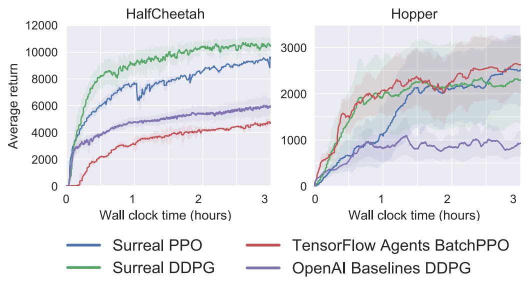

Figure 7: Training curves with 16 actors: our SURREAL algorithms and baselines on OpenAI Gym HalfCheetah and Hopper environments.

rewards: in the Bimanual Lifting task, actors learn to place the grippers on the handles but not to lift. We hypothesize that we need better exploration strategies to solve these tasks. Following [10], we build a curriculum from human demonstration to assist exploration. With some probability α, we initialize episodes with states taken along successful trajectories from the demonstrations. We experiment with three different curricula: in the "uniform" case the state is chosen uniformly at random from the entire demonstration dataset; "forward" samples states from the beginning of demonstration trajectories with a slowly growing window; "reverse" samples from the end of trajectories instead. Fig. 6 shows training results for Bimanual Lifting (states) with these curricula, using 64 actors. We see that with proper strategies, SURREAL-PPO is able to complete the full task, which is previously only partially solved.

Bin Picking and Nut-and-peg Assembly are intrinsically more difficult because they have long horizons and multiple stages. As indicated in Fig. 5, they take the longest time to complete for humans. RL agents are able to perform a specific subtask but unable to proceed. In the Bin Picking task, for example, the PPO agents are able to successfully pick up, move, and drop at most one out of the four items. We believe solving these tasks is beyond the scope of our current algorithms.

5.2 Performances: OpenAI Gym

To put the performances of our distributed RL implementations in context, we run SURREAL on *de facto* continuous RL benchmark environments used in previous work [14, 49]. We compare with OpenAI Baselines DDPG [33] and TensorFlow Agent BatchPPO [32], which are among the popular open-source distributed reference implementations. Fig. 7a compares the learning curves of our DDPG and PPO implementations against the baselines on wall-clock time. All algorithms are trained with 16 actors with the same hardware allocation. Except for OpenAI-DDPG that does not support GPU out of the box, all other experiments use a single Nvidia P100 GPU for the learner.

Our algorithms outperform all baselines on `HalfCheetah` by a large margin and perform on a comparable level as BatchPPO on `Hopper`. We hypothesize that the differences in both algorithmic and system design contribute to the performance gap. BatchPPO distribute the experience generation by producing a batch of actions synchronously, which would be bottlenecked by the slowest simulation. OpenAI-DDPG collects gradients in a synchronous fashion, which is not as desirable as communicating experiences asynchronously (see Sec. 3.1).

5.3 Scalability

Fig. 8 shows our system characteristics on Robotics Suite (training on pixels) and Gym environments (training on states). The total actor throughput is number of environment frames collected by all actors per second. Our Buffer is sharded and load-balanced to reduce network congestion, which yields an almost linear speedup with respect to the number of actors. The scalability of our algorithms becomes more remarkable in our tasks, where complex dynamics and graphical rendering makes simulation slower than OpenAI Gym by an order of magnitude .

We evaluate the scalability of our methods with respect to varying numbers of actors. Fig. 9 shows episode rewards on Gym `HalfCheetah` and `Hopper` environments. Both our methods and the

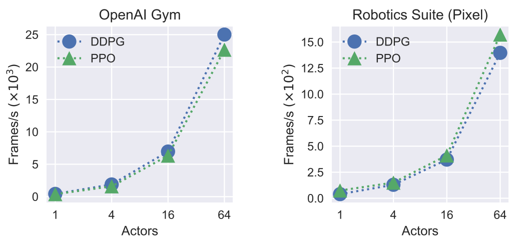

Figure 8: Total actor throughput: environment FPS from all actors combined scales linearly with the number of actors in both algorithms.

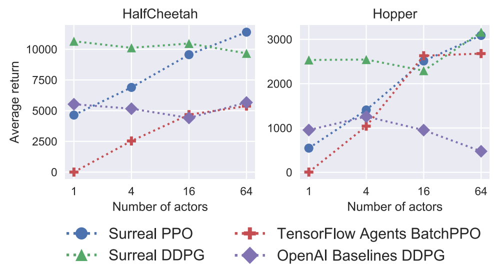

Figure 9: Scalability with respect to the number of actors: our methods v.s. baselines on gym HalfCheetah and Hopper environments.

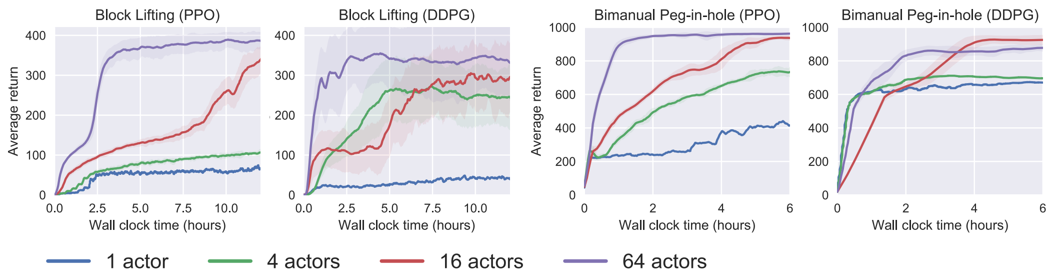

Figure 10: Learning curves for SURREAL-PPO and SURREAL-DDPG trained on the block lifting and bimanual peg-in-hole tasks with raw pixel inputs using different number of actors, ranging from 1, 4, 16, 64. Both algorithms learn sizeably faster with more actor experience throughput.

baselines are trained for 3 hours using 1, 4, 16 and 64 actors with the same hardware resource allocation. Our methods generally obtain better performance with growing number of actors, and score higher than the baselines across all actor settings. SURREAL-PPO is on-policy and uses every experience once; its learner speed is bound by total actor throughput until the learner machine capacity is saturated. Thus increasing the number of actors speeds up the learner, as demonstrated by the monotonically increasing curves in Fig. 9. In contrast, the DDPG learner samples from a replay buffer and reuses experiences; its speed is not directly correlated with the total actor throughput. If a small number of actors is enough to explore the environment extensively, the learner can still learn at the maximal possible speed. This is the case for Gym environments where simulation is fast. SURREAL-DDPG achieves the maximum score with as few as 1 or 4 actors.

In comparison, on Robotics tasks with slower simulation, a large number of actors is needed for both SURREAL-PPO and SURREAL-DDPG to learn; they show better performance when trained with more actors. Fig. 10 compares the learning curves of our DDPG and PPO on Block Lifting and Bimanual Peg-in-hole with pixel inputs. Training with 64 actors results in faster learning and better final performance. We hypothesize that, given fewer actors (especially 1 and 4 actors), both DDPG and PPO suffer from inadequate exploration, which causes them to learn much slower or become trapped at sub-optimal policies.

6 Conclusion

We address the challenge of reproducibility and benchmarking in deep reinforcement learning and robot manipulation research. We introduce SURREAL, a highly-scalable distributed framework that supports on-policy and off-policy deep RL algorithms. We develop a novel system infrastructure to enable reproducibility and extensibility. To rigorously evaluate the performance of the RL algorithms, we introduce Robotics Suite, which contains a set of manipulation tasks with varying levels of complexity. We illustrate that our distributed RL algorithms outperform widely used RL libraries on standard benchmarks, and perform well in manipulation tasks in our new benchmark. In the future, we plan to expand SURREAL with new algorithms and enrich the benchmark with new tasks. We hope SURREAL will become a valuable resource for a broad range of manipulation related research, as a training resource, a standardized benchmark, and a framework for rapid algorithm development.

Acknowledgments

We would like to thank many members of the Stanford People, AI & Robots group and the anonymous reviewers for their constructive feedback. This work is funded by ONR - MURI (1175361-6-TDFEK), Toyota Research Institute (S-2015-09-Garg), Weichai America Corp. (1208338-1-GWMZY), Tecent (1198791-1-GWMVE). We would also like to acknowledge Google Cloud for providing the computational support for the development and experimentation of this project.

References

[1] J. Kober and J. Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*. 2012.

[2] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE, 2006.

[3] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*. Atlanta, 2010.

[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[8] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*, 2016.

[9] A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[10] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *RSS*, 2018.

[11] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv preprint arXiv:1710.06537*, Oct. 2017.

[12] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine. Path integral guided policy search. In *ICRA*, 2017.

[13] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016.

[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[16] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[17] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[18] A. P. del Pobil, R. Madhavan, and E. Messina. Benchmarks in robotics research. In *Workshop IROS*, 2006.

[19] R. Madhavan, R. Lakaemper, and T. Kalmár-Nagy. Benchmarking and standardization of intelligent robotic systems. In *International Conference on Advanced Robotics*, pages 1–7, 2009.

[20] C. G. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, et al. What happened at the darpa robotics challenge, and why. 2016.

[21] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, et al. Analysis and observations from the first amazon picking challenge. *IEEE Trans. on Automation Science and Engineering*, 2016.

[22] B. Calli, A. Singh, A. Walsman, S. Srinivasa, et al. The YCB object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics*, 2015.

[23] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, et al. Caffe: Convolutional architecture for fast feature embedding. In *22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[24] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. 2017.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[28] I. Popov, N. Heess, T. P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, et al. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[29] S. James, A. J. Davison, and E. Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[32] D. Hafner, J. Davidson, and V. Vanhoucke. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878*, 2017.

[33] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[35] Y. Zhu, R. Mottaghi, E. Kolve, et al. Target-driven visual navigation in indoor scenes using deep rl. In *International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[36] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[37] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. *ArXiv e-prints*, 2017.

[38] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[40] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[41] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[42] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[43] A. V. Clemente, H. N. Castejón, and A. Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

[44] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement learning thorugh asynchronous advantage actor-critic on a GPU. In *ICLR*, 2017.

[45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[46] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. 2017.

[47] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[48] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[49] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *ICML*, 2016.

[50] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

A Experiment Details

A.1 SURREAL-PPO

When training with pixel inputs, an $84 \times 84 \times 3$ RGB image is fed into a convolutional neural network (CNN) feature extractor. The extractor consists of an 8×8 convolution with 16 filters and stride 4, followed by a 4×4 convolution with 32 filters and stride 2, with ReLU activations. The convolution outputs are flattened and passed into a linear layer of size 256, which is concatenated to the proprioceptive inputs. The concatenated feature is fed into a 1-layer long short-term memory (LSTM) network with cell size 100. The LSTM output is fed into two separate feedforward networks for actor and critic. Both feedforward networks have hidden layers of size 300 and 200. The actor network outputs an action mean and log of standard deviation for each action dimension, whereas the critic network outputs a scalar. The actions are sampled with action mean and standard deviation kept by the actor network before feeding back to the environment. The critic and actor network are updated through backpropagation through time with fixed-size LSTM rollouts.

To compute the loss, we first compute advantage for each timestep using generalized advantage estimate (GAE) [50]. In our implementation, we find the adaptive KL variant of PPO [31] to be more stable. In the distributed setting, when the learner publishes its parameters, agents may still be doing rollouts using an outdated set of parameters. This communication latency makes the algorithm not strictly on-policy, leading to divergent behaviors. In SURREAL-PPO learner, a set of target network parameters is kept along side of a set of updated model parameters. The target model parameters are broadcasted to the actors at a lower frequency to stabilize learning. We use the number of model update steps before broadcasting to measure the target network update frequency: more model update steps correspond to a lower frequency. The frequency used in our experiments is 64 steps; divergent behaviors would occur with fewer than 16 steps. One way to interpret the reference target model is that it maintains a fixed center of the trust region. The learner is encouraged to optimize policy within the trust region since the adaptive KL-penalty variant of PPO penalizes excess KL-divergence. We also find that normalizing the low-dimensional states with a running estimate of mean and variance (z-filtering) helps stability and convergence for SURREAL-PPO.

The learner iteration speed for SURREAL-PPO can vary for different tasks and hyperparameters. Particularly, LSTM rollout length L and Return horizon H have the most impact. Thus each datapoint in SURREAL-PPO corresponds to an entire sub-trajectory worth of training data as opposed to one environment step in DDPG. At the update time, truncated backpropagation through time is performed at each state within the sub-trajectory rollout. SURREAL-PPO also updates each experience received from actors multiple times to make the best use of on-policy nature of the algorithm. Thus, we find that the learner iteration throughput can vary based on hyperparameters used. Incidentally, increasing LSTM rollout and update epochs per experience can drastically impact learner throughput. Detailed hyperparameters are provided in Table 1. Notations of these hyperparameters are provided in parentheses. Algorithm 3 will refer to the same notations. We note that with the listed hyperparameter choice, SURREAL-PPO achieves 350 iterations per second on Gym benchmark environments and 70 iterations per second on SURREAL robotics benchmark in full capacity.

Table 1: Selected SURREAL-PPO hyperparameters

Parameter	Value
z-filtering on states	Yes
n-step reward/return horizon (H)	5
discount factor (γ)	0.99
target KL-divergence (KL_{target})	0.02
actor/critic update epoch (E)	5
learning Rate	1e-4
target network hard update interval	4096
initial log-sigma	-1.0

A.2 SURREAL-DDPG

When training with pixels, we stack the most recent 3 camera observations as input to the DDPG model. The stacked images are then fed through an 8×8 convolution with 16 filters and stride 4, followed by a 4×4 convolution with 32 filters and stride 2, with ReLU activations. The convolution filter parameters are shared between the actor and critic networks. For the purpose of gradient updates, the convolution parameters are updated only by gradient descent on the critic loss. The convolution outputs are flattened and passed into a linear layer of size 200, which is concatenated to the proprioceptive feature.

This combined image and proprioceptive feature is used in both the actor and critic networks. In the actor network, this feature is passed through two hidden layers of size 300 and 200 before being converted to the action output, with a layer normalization step after each hidden layer. In the critic network, this feature is passed through two hidden layers of size 400 and 300, with a layer normalization after each hidden layer. The action is concatenated to the first hidden layer after the layer normalization has been performed. Each actor explores using Ornstein-Uhlenbeck noise. Each actor is assigned an exploration noise σ parameter that remains constant over the course of training, where σ is scaled linearly between a minimum of 0 and a maximum of 1.0 across the actors.

For the Robotics Suite experiments, we find them to be more sensitive to layer normalization than the Gym environments. Bimanual Peg-in-hole experiments are run with layer normalization, 6-step rewards, no weight decay, and a maximum σ value of 1.0. Block lifting experiments use no layer normalization, 3-step rewards, weight decay of 0.0001, and a maximum σ value of 2.0. Table 2 provides a select list of hyperparameter choices mentioned above.

Table 2: Select SURREAL-DDPG hyperparameters

Parameter	Value
image dimension	$84 \times 84 \times 3$
frame stacking	3
batch size	512
clip actor network gradient norm	1.0
clip critic network gradient norm	No
actor (θ_{actor}) learning rate	1e-4
critic (θ_{critic}) learning rate	1e-4
target network (θ_{target}) hard update interval (U)	500
learner parameter publish interval (S)	3 seconds

A.3 Hardware Specifications

We deploy our distributed system onto a Kubernetes cluster based on Google Cloud. For Gym environments, each actor uses a *n1-standard-2* machine with 2 CPUs. For Robot Suite environments, every 8 actors share a *n1-standard-8* machine with 8 CPUs and an Nvidia Tesla K80 GPU. The GPU speeds up both physical simulation and neural network forward passes of the actors. The buffer and learner are deployed on a single machine with 16 CPUs and an Nvidia Tesla P100 GPU.

B Algorithm Pseudocode

Detailed Pseudocode for Actor and Learner are provided in Algorithm 1 and Algorithm 2, which provides a unified abstraction for on-policy and off-policy learning. Detailed Pseudocode for SURREAL-PPO and SURREAL-DDPG are provided in Algorithm 3 and Algorithm 4.

In SURREAL-PPO, L denotes LSTM rollout lengths. H denotes return horizon. λ denotes constant parameter for GAE calculation. ξ_{kl} denotes the adaptive KL-penalty constant. β_{low}, β_{high}, α denotes boundary range and scaling constant for ξ_{kl} respectively. KL_{target} denotes the target KL-divergence.

In SURREAL-DDPG, θ_{critic}, θ_{actor}, and θ_{target} denote the critic, actor, and target network parameters, and π_θ denotes to a policy given actor parameters θ, and Q_θ denotes a value estimation function given critic parameters θ. τ denotes a collection of observations states, actions, and rewards. Target parameters are updated every U gradient updates, and parameters are published to the parameter server every S seconds.

Algorithm 1 Actor

```
1:  procedure ACTOR                    ▷ Run actor in environment simulator and send experiences to Buffer.
2:      $\theta_0 \leftarrow$ PARAMETERSERVER.PARAMETERS( )          ▷ subscribe to parameter server updates.
3:      $s_0 \leftarrow$ ENVIRONMENT.INITIALIZE( )                   ▷ Get initial state from environment.
4:      for $t = 1$ to $T$ do                                      ▷ loop until max episodes
5:          $a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1})$         ▷ Select an action using the current policy.
6:          $(r_t, \gamma_t, s_t) \leftarrow$ ENVIRONMENT.STEP($a_{t-1}$)  ▷ Apply the action in the environment.
7:          LOCALBUFFER.APPEND($s_{t-1}, a_{t-1}, r_t, \gamma_t$)   ▷ Buffer experience to send in bulk.
8:          PERIODICALLY(BUFFER.UPLOAD(LocalBuffer))                ▷ Send experience to Buffer.
9:          AFTERUPLOAD(CLEAR(LocalBuffer))
10:         PERIODICALLY($\theta_t \leftarrow$ PARAMETERSERVER.PARAMETERS())  ▷ Obtain latest parameters.
11:     end for
12: end procedure
```

Algorithm 2 Learner

```
1: procedure LEARNER($T$)              ▷ Update network using batches sampled from memory.
2:     $\theta_0 \leftarrow$ INITIALIZENETWORK( )
3:     for $t = 1$ to $T$ do                                       ▷ Update the parameters $T$ times.
4:         $id, \tau \leftarrow$ BUFFER.SAMPLE( )                  ▷ Retrieve a batch of experience from Buffer.
5:         $\theta_{t+1} \leftarrow \langle$UPDATEPARAMETERS$\rangle(\tau; \theta_t)$  ▷ See Algorithm 3 for PPO and Algorithm 4 for DDPG.
6:         PERIODICALLY(PUBLISHPARAMETERS($\theta_{t+1}$))          ▷ Publish parameters to Parameter Server.
7:     end for
8: end procedure
```

C Benchmark Environment Details

All six environments are simulated at 10Hz control rate. The robots are controlled via joint velocity. There are three types of observations: proprioceptive features, object features, and camera observations. Proprioceptive features contain $\cos$ and $\sin$ of robot joint positions, robot joint velocities and current configuration of the gripper. Object features contain environment-specific values that describe the states and relationships of objects of interest. Camera observations are 84×84 RGB images. When trained on states, the agent receives proprioceptive and object features. When trained on pixels, the agent receives proprioceptive features and camera observations.

Block Lifting: A cube is placed on the table. The Saywer robot is rewarded for lifting the cube with a parallel-jaw gripper. Each episode lasts for 200 timesteps. Each step receives reward at most 2.25 (at most 450 for an entire episode). The cube is randomized in size, initial position and initial orientation. Object features contain position and orientation of the cube, absolute position of the gripper and position difference between the gripper and the cube. The agent gets reward $r = r_1 + r_2 + r_3$. $r_1 \in [0, 1]$ is 1 when the gripper's grip site is at the cube's position. It decreases as the grip site is further away. $r_2 = 0.25$ if both fingers of the gripper are touching the cube and $r_2 = 0$ otherwise. $r_3 = 1$ if the cube's center is above a certain value such that the robot must be lifting the cube. It is zero otherwise.

Block Stacking: A red cube and a green cube are placed on the tabletop. The Saywer robot is rewarded for lifting the red cube with a parallel-jaw gripper and stacking it on top of the green cube. Each episode lasts for 500 timesteps. Each step receives reward at most 2 (at most 1000 for an entire episode). The cubes are randomized in initial positions and initial orientations. Object features contain position and orientation of both cubes, absolute position of the gripper, position difference between the gripper and both cubes, position difference between the two cubes and position and orientation of the gripper. The agent gets reward $r \in [0, 0.25]$ for positioning the grip site at the red cube's position. If both fingers are touching the red cube, an additional 0.25 is rewarded. $r \in [1, 1.5]$ when the red cube is lifted above the table, where $r = 1.5$ if the red cube is lifted above the green cube. $r = 2$ if the red cube is on the green cube.

Algorithm 3 SURREAL-PPO Learner

```
1: procedure PPO-LEARNER.UPDATEPARAMETERS($\tau$, $\theta_t$)
2:     if $\theta_{target}$ not initialized then
3:         Initialize $\theta_{target}$ with random parameters        ▷ Target model to fix trust region center
4:     end if
5:     Compute TD error for every $t \in \{1, 2, ..., L-1\}$, $\delta_t^V = r_t + \gamma V_{\theta_{target}}(\tau_{t+1}) - V_{\theta_{target}}(\tau_t)$
6:     Estimate advantage for every $t \in \{1, 2, ..., L-H\}$, $\hat{A}_t = \sum_{l=0}^{H-1} (\gamma\lambda)^l \delta_{t+l}^V$
7:     Compute Return for $t \in \{1, 2, ..., L-H\}$ $R_t = \sum_{l=0}^{H-1} \gamma^l r_{t+l}$
8:     for $j \in \{1, \cdots, E\}$ do
9:         $J_{PPO}(\theta_t) = \sum_{t=1}^{L-H} \frac{\pi_{\theta_t}(a_t|\tau_t)}{\pi_{target}(a_t|\tau_t)} \hat{A}_t - \xi_{kl}\mathrm{KL}[\pi_{target}|\pi_{\theta_t}] - \xi_{max} max(\mathrm{KL}[\pi_{target}|\pi_{\theta_t}] - 2KL_{target}, 0)^2$
10:        Update $\theta_t$ by a gradient method w.r.t. $J_{PPO}(\theta_t)$ to $\hat{\theta}_t$
11:    end for
12:    for $j \in \{1, \cdots, E\}$ do
13:        $L_{BL}(\hat{\theta}_t) = -\sum_{t=1}^{L-H} (R_t - V_{\hat{\theta}_t}(\tau_t))^2$
14:        Update $\hat{\theta}_t$ by a gradient method w.r.t. $L_{BL}(\hat{\theta}_t)$
15:    end for
16:    if Updated parameters with $N$ data points without release then
17:        PPO-LEARNER.RELEASEPARAMETERS($\theta_{t+1}$)  ▷ Controls communication with parameter server
18:        Clear update counter                          ▷ Clears number of data points processed
19:    end if
20: end procedure
21:
22: procedure PPO-LEARNER.RELEASEPARAMETERS($\theta_{t+1}$)
23:    $\theta_{ref} \leftarrow \theta_{t+1}$                                ▷ Update trust region center
24:    PPO-LEARNER.PUBLISHPARAMETER($\theta_{t+1}$)            ▷ Send model to parameter server
25:    if $\mathrm{KL}[\pi_{ref}|\pi_{\theta_{t+1}}] > \beta_{high}\mathrm{KL}_{target}$ then
26:        $\xi_{kl} \leftarrow \alpha\xi_{kl}$
27:    else if $\mathrm{KL}[\pi_{ref}|\pi_{\theta_{t+1}}] < \beta_{low}\mathrm{KL}_{target}$ then
28:        $\xi_{kl} \leftarrow \xi_{kl}/\alpha$
29:    end if
30: end procedure
```

Bimanual Peg-in-hole: The Baxter robot holds a board with a squared hole in the center in its right hand, and a long stick in the left hand. The goal is to move both arms to insert the peg into the hole. Each episode lasts for 200 timesteps. Each step receives reward at most 5 (at most 1000 for an entire episode). Initial position of robot joints are randomized. Object features contain the position and orientation of the hole, the position and orientation of the peg and the relative position and orientation of the peg in the frame of the hole. The agent gets reward $r = r_1 + r_2 + r_3$. $r_1 \in [0, 3]$ gets larger for positioning the center of the peg near the center of the hole. $r_2 \in [0, 1]$ is the cos of the desired direction of the peg and its actual direction to reward putting the peg in the right direction. $r_3 = 1$ when the position and orientation of the peg are within toleration and $r_3 = 0$ otherwise.

Bimanual Lifting: A pot with two handles is placed on the tabletop. The Baxter robot is rewarded for lifting the pot above the table by a threshold while not tilting the pot over 30 degrees. Thus the robot has to coordinate its two hands to grasp the handles and balance the pot. Each episode lasts for 500 timesteps. Each step receives reward at most 3 (at most 1500 for an entire episode). The original position and orientation of the pot are randomized. Object features contain the position of the cube and both of its handles, the positions of two grippers and the position offset from each gripper to its target handle. Reward $r = r_1 + c \times r_2$. $r_1 = r_l + r_r$ with r_l, resp. r_r, $\in [0, 0.5]$ being larger as the left, resp. right, gripper is closer to the handle and is equal to 0.5 when the gripper touches the handle. $c = 1$ if the angle between z-direction of the pot is within 30 degrees of the direction of up and $c = 0$ otherwise. $r_2 \in [0, 2]$ is proportional to the height of the pot's center of mass.

Bin Picking: The Sawyer robot tackles a pick-and-place task, where the goal is to pick four objects from each category in a bin and to place them into their corresponding containers. Each episode lasts for 2000 timesteps. Each step receives reward at most 4 (at most 8000 for an entire episode). The original positions and orientations of the objects are randomized. Object features contain the position and orientation of the gripper, the positions and orientations of the objects and relative positions and

Algorithm 4 SURREAL-DDPG Learner

```
1: procedure DDPG-LEARNER.TRAINPOLICY
2:     Initialize $\theta_{critic}, \theta_{actor}$ with random parameters
3:     Initialize $\theta_{critic\_target} = \theta_{critic}, \theta_{actor\_target} = \theta_{actor}$
4:     for $i \in \{1, \cdots, N\}$ do
5:         Collect a batch of observations $\tau = \{\tau_t, a_t, r_t, \tau_{t+n}\}$    ▷ Uniformly sample from replay memory
6:         $\theta \leftarrow$ DDPG-Learner.ParameterUpdate($\tau, \theta$)
7:         if Updated parameter $U$ times without target update then
8:             Set $\theta_{target} \leftarrow \theta$
9:         end if
10:        if $S$ seconds have passed without parameter publish then
11:            DDPG-Learner.PublishParameters($\theta$)    ▷ Send model to parameter server
12:        end if
13:    end for
14: end procedure
15:
16: procedure DDPG-LEARNER.PARAMETERUPDATE($\tau, \theta$)
17:     if Terminal($\tau_{t+n}$) then
18:         Set $y = \Sigma_{k=0}^{n-1}\gamma^k r_{t+k}$
19:     else
20:         Set $y = (\Sigma_{k=0}^{n-1}\gamma^k r_{t+k}) + \gamma^n Q_{target}(\tau_{t+n}, \pi_{target}(\tau_{t+n}))$
21:     end if
22:     Compute $L_{critic} = (Q_\theta(\tau_t, a_t) - y)^2$
23:     Update $\theta_{critic}$ by gradient descent w.r.t. $L_{critic}$    ▷ Update critic linear and convolution parameters
24:
25:     Compute $L_{actor} = -Q_\theta(\tau_t, \pi_\theta(\tau_t))$    ▷ Update actor linear layer parameters
26:     Update $\theta_{actor}$ by gradient descent w.r.t. $L_{actor}$
27: end procedure
```

orientations of of the objects in the gripper frame. Reward 1 is given to every object successfully placed into the bin. An additional reward $\max(r_1, r_2, r_3, r_4)$ is given to 1) $r_1 \leq 0.1$: placing the gripper near an unplaced object, 2) $r_2 \leq 0.35$: touching an unplaced object, 3) $r_3 \leq 0.5$: lifting an unplaced object or 4) $r_4 \leq 0.7$: hovering an unplaced object over the desired bin.

Nut-and-peg Assembly: Two colored pegs are mounted to the tabletop. The Sawyer robot needs to declutter the nuts lying on top of each other and assemble them onto their corresponding pegs. Each episode lasts for 2000 timesteps. Each step receives reward at most 4 (at most 8000 for an entire episode). The initial positions and orientations of the nuts are randomized. Object features contain the position and orientation of the gripper, positions and orientations of the nuts and their positions and orientations with respect to the gripper. Reward 1 is given to every object successfully placed into the bin. An additional reward $\max(r_1, r_2, r_3, r_4)$ is given to 1) $r_1 \leq 0.1$: placing the gripper near an unplaced nut, 2) $r_2 \leq 0.35$: touching an unplaced nut, 3) $r_3 \leq 0.5$: lifting an unplaced nut or 4) $r_4 \leq 0.7$: hovering an unplaced nut over the desired hole.

C.1 Human Benchmarking

In order to evaluate the difficulty of the tasks, a human operator repeatedly solved the tasks using a custom teleoperation rig. HTC Vive Virtual Reality (VR) controllers were used to get six degrees of freedom to control the end-effector position and orientation on each arm. We used one VR controller for the Sawyer robot and two for the Baxter robot, with target joint positions computed using inverse kinematics. This setup was also used to collect demonstrations for reverse curriculum training. For each environment, we collected 10 to 20 successful demonstrations and recorded the reward trajectories and time required. These are shown in Fig. 11. From the data we can infer that nut-and-peg assembly and bin picking are the hardest, with both a higher mean and more variance across successful runs.

D Details for Learning from Demonstrations

We provide details for the demonstration curricula used to facilitate exploration of the learning agents, as discussed in Sec. 5.1. To reiterate, uniform curriculum samples start states uniformly at random

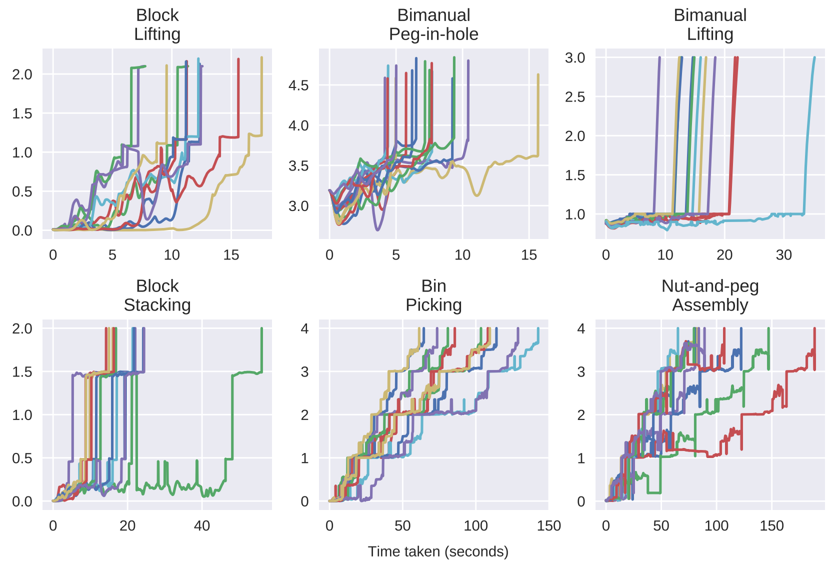

Figure 11: Sample immediate reward trajectories for successful task completions, collected using robot teleoperation facilities.

from demonstrations. Forward curriculum samples start states from a window of states starting from the first state of demonstration trajectories. Periodically, we increase the width of the window to incorporate more states from later stages of demonstrations for sampling. Similarly, reverse curriculum also maintains a growing window of the same length, but instead, the sampling window always ends at the last state of the demonstration trajectories and grows to incorporate states from earlier timesteps. Empirically it is important to include previously sampled sections of demonstration data, namely earlier states for forward curriculum and later states for reverse curriculum, to prevent catastrophic forgetting.

For each of the three curricula experimented, we mix in training episodes with natural start states to ensure that we do not simply overfit trained policy on the dataset of demonstrations. We note that there are three different hyperparameters at play: mixing ratio of sampling from curriculum (α), frequency of sampling window width update, number of new states to include in sampling window. Sec. 5.1 indicates that such simple engineering can produce tremendous benefits to tasks such as Bimanual Lifting. It remains an interesting problem for better designed curriculum to produce more better effects. Table 3 provides tested parameters in curricula.

Table 3: Demonstration curriculum hyperparameters

Parameter	Uniform	Forward/Reverse
mixing ratio (α)	0.5	0.5
window update frequency	None	100 episodes sampled
new states incorporated	None	50 states