ASYMMETRIC SELF-PLAY FOR AUTOMATIC GOAL DISCOVERY IN ROBOTIC MANIPULATION

OpenAI
Matthias Plappert*, Raul Sampedro*, Tao Xu*, Ilge Akkaya*, Vineet Kosaraju*, Peter Welinder*, Ruben D'Sa*, Arthur Petron*, Henrique Ponde de Oliveira Pinto*, Alex Paino*, Hyeonwoo Noh*, Lilian Weng*, Qiming Yuan*, Casey Chu*, Wojciech Zaremba*

ABSTRACT

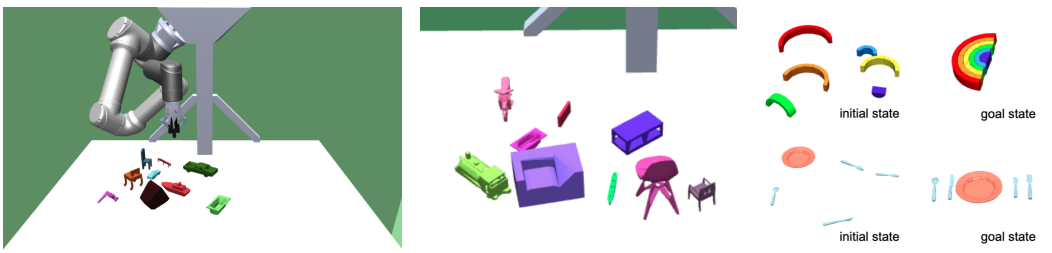
We train a single, goal-conditioned policy that can solve many robotic manipulation tasks, including tasks with previously unseen goals and objects. We rely on asymmetric self-play for goal discovery, where two agents, Alice and Bob, play a game. Alice is asked to propose challenging goals and Bob aims to solve them. We show that this method can discover highly diverse and complex goals without any human priors. Bob can be trained with only sparse rewards, because the interaction between Alice and Bob results in a natural curriculum and Bob can learn from Alice's trajectory when relabeled as a goal-conditioned demonstration. Finally, our method scales, resulting in a single policy that can generalize to many unseen tasks such as setting a table, stacking blocks, and solving simple puzzles. Videos of a learned policy is available at `https://robotics-self-play.github.io`.

1 INTRODUCTION

We are motivated to train a *single* goal-conditioned policy (Kaelbling, 1993) that can solve *any* robotic manipulation task that a human may request in a given environment. In this work, we make progress towards this goal by solving a robotic manipulation problem in a table-top setting where the robot's task is to change the initial configuration of a variable number of objects on a table to match a given goal configuration. This problem is simple in its formulation but likely to challenge a wide variety of cognitive abilities of a robot as objects become diverse and goals become complex.

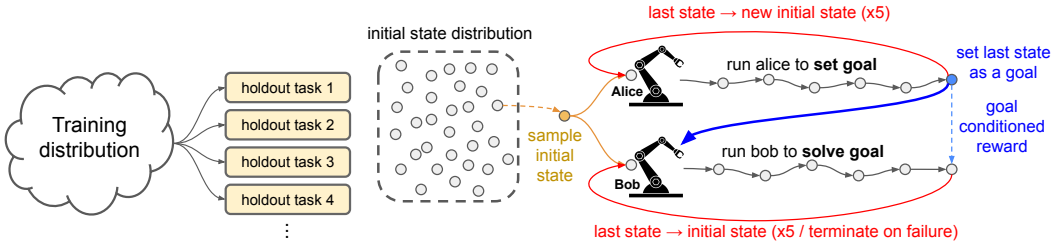
Motivated by the recent success of deep reinforcement learning for robotics (Levine et al., 2016; Gu et al., 2017; Hwangbo et al., 2019; OpenAI et al., 2019a), we tackle this problem using deep reinforcement learning on a very large training distribution. An open question in this approach is how we can build a training distribution rich enough to achieve generalization to many unseen manipulation tasks. This involves defining both an environment's initial state distribution and a goal distribution. The initial state distribution determines how we sample a set of objects and their configuration at the beginning of an episode, and the goal distribution defines how we sample target states given an initial state. In this work, we focus on a scalable way to define a rich goal distribution.

*Authors are listed at random and a detailed contribution section is at the end. Please cite as **OpenAI et al.**.



(a) Table-top setting with a robot arm (b) Example initial state for training (c) Example holdout tasks

Figure 1: (a) We train a policy that controls a robot arm operating in a table-top setting. (b) Randomly placed ShapeNet (Chang et al., 2015) objects constitute an initial state distribution for training. (c) We use multiple manually designed holdout tasks to evaluate the learned policy.



(a) Overall framework (b) Training distribution based on asymmetric self-play

Figure 2: (a) We train a goal-conditioned policy on a single training distribution and evaluate its performance on many unseen holdout tasks. (b) To construct a training distribution, we sample an initial state from a predefined distribution, and run a goal setting policy (Alice) to generate a goal. In one episode, Alice is asked to generate 5 goals and Bob solves them in sequence until it fails.

The research community has started to explore automated ways of defining goal distributions. For example, previous works have explored learning a generative model of goal distributions (Florensa et al., 2018; Nair et al., 2018b; Racaniere et al., 2020) and collecting teleoperated robot trajectories to identify goals (Lynch et al., 2020; Gupta et al., 2020). In this paper, we extend an alternative approach called asymmetric self-play (Sukhbaatar et al., 2018b;a) for automated goal generation. Asymmetric self-play trains two RL agents named Alice and Bob. Alice learns to propose goals that Bob is likely to fail at, and Bob, a goal-conditioned policy, learns to solve the proposed goals. Alice proposes a goal by manipulating objects and Bob has to solve the goal starting from the same initial state as Alice's. By embodying these two agents into the same robotic hardware, this setup ensures that all proposed goals are provided with at least one solution: Alice's trajectory.

There are two main reasons why we consider asymmetric self-play to be a promising goal generation and learning method. First, any proposed goal is *achievable*, meaning that there exists at least one solution trajectory that Bob can follow to achieve the goal. Because of this property, we can exploit Alice's trajectory to provide additional learning signal to Bob via behavioral cloning. This additional learning signal alleviates the overhead of heuristically designing a curriculum or reward shaping for learning. Second, this approach does not require labor intensive data collection.

In this paper, we show that asymmetric self-play can be used to train a goal-conditioned policy for complex object manipulation tasks, and the learned policy can zero-shot generalize to many manually designed holdout tasks, which consist of either previously unseen goals, previously unseen objects, or both. To the best of our knowledge, this is the first work that presents zero-shot generalization to many previously unseen tasks by training purely with asymmetric self-play.[1]

2 Problem Formulation

Our training environment for robotic manipulation consists of a robot arm with a gripper attached and a wide range of objects placed on a table surface (Figure 1a,1b). The goal-conditioned policy learns to control the robot to rearrange randomly placed objects (the initial state) into a specified goal configuration (Figure 1c). We aim to train a policy on a single training distribution and to evaluate its performance over a suite of holdout tasks which are independently designed and not explicitly present during training (Figure 2a). In this work, we construct the training distribution via *asymmetric self-play* (Figure 2b) to achieve generalization to many unseen holdout tasks (Figure 1c).

Mathematical formulation Formally, we model the interaction between an environment and a goal-conditioned policy as a goal-augmented Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{G} \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ denotes the transition probability, $\mathcal{G} \subseteq \mathcal{S}$ specifies the goal space and $\mathcal{R} : \mathcal{S} \times \mathcal{G} \mapsto \mathbb{R}$ is a goal-specific reward function. A goal-augmented trajectory sequence is $\{(s_0, g, a_0, r_0), \ldots, (s_t, g, a_t, r_t)\}$, where the goal is provided to the policy as part of the observation at every step. We say a goal is achieved if s_t is sufficiently close to g (Appendix A.2). With a slightly overloaded notation, we define the *goal distribution* $\mathcal{G}(g|s_0)$ as the probability of a goal state $g \in \mathcal{G}$ conditioned on an initial state $s_0 \in \mathcal{S}$.

[1] Asymmetric self-play is proposed in Sukhbaatar et al. (2018b;a), but to supplement training while the majority of training is conducted on target tasks. Zero-shot generalization to unseen tasks was not evaluated.

Training goal distribution A naive design of the goal distribution $\mathcal{G}(g|s_0)$ is to randomly place objects uniformly on the table, but it is unlikely to generate interesting goals, such as an object picked up and held above the table surface by a robot gripper. Another possible approach, collecting tasks and goals manually, is expensive and hard to scale. We instead sidestep these issues and automatically generate goals via training based on asymmetric self-play (Sukhbaatar et al., 2018b;a). Asymmetric self-play involves using a policy named Alice $\pi_A(a|s)$ to set goals and a goal-conditioned policy Bob $\pi_B(a|s, g)$ to solve goals proposed by Alice, as illustrated in Figure 2b. We run π_A to generate a trajectory $\tau_A = \{(s_0, a_0, r_0), \ldots, (s_T, a_T, r_T)\}$ and the last state is labelled as a goal g for π_B to solve. The goal distribution $\mathcal{G}(s_T = g|s_0)$ is fully determined by π_A and we train Bob only on this goal distribution. We therefore say *zero-shot generalization* when Bob generalizes to a holdout task which is not explicitly encoded into the training distribution.

Evaluation on holdout tasks To assess zero-shot generalization of $\pi_B(a|s, g)$ from our training setup, we hand-designed a suite of holdout tasks with goals that are never directly incorporated into the training distribution. Some holdout tasks also feature previously unseen objects. The holdout tasks are designed to either test whether a specific skill has been learned, such as the ability to pick up objects (Figure 3), or represent a semantically interesting task, such as setting a table (Figure 1c). Appendix B.6 describes the list of holdout tasks that we use in our experiments. Note that none of the holdout tasks are used for training $\pi_B(a|s, g)$.

3 ASYMMETRIC SELF-PLAY

To train Alice policy $\pi_A(a|s)$ and Bob policy $\pi_B(a|s, g)$, we run the following multi-goal game within one episode, as illustrated in Figure 2b:

1. An initial state s_0 is sampled from an initial state distribution. Alice and Bob are instantiated into their own copies of the environment. Alice and Bob alternate turns as follows.
2. **Alice's turn.** Alice interacts with its environment for a fixed number of T steps and may rearrange the objects. The state at the end of Alice's turn s_T will be used as a goal g for Bob. If the proposed goal is invalid (e.g. if Alice has not moved any objects, or if an object has fallen off the table), the episode terminates.
3. **Bob's turn.** Bob receives reward if it successfully achieves the goal g in its environment. Bob's turn ends when it succeeds at achieving the goal or reaches a timeout. If Bob's turn ends in a failure, its remaining turns are skipped and treated as failures, while we let Alice to keep generating goals.
4. Alice receives reward if Bob fails to solve the goal that Alice proposed. Steps 2–3 are repeated until 5 goals are set by Alice or Alice proposes an invalid goal, and then the episode terminates.

The competition created by this game encourages Alice to propose goals that are increasingly challenging to Bob, while Bob is forced to solve increasingly complex goals. The multi-goal setup was chosen to allow Bob to take advantage of environmental information discovered earlier in the episode to solve its remaining goals, which OpenAI et al. (2019a) found to be important for transfer to physical systems. Note however that in this work we focus on solving goals in simulation only. To improve stability and avoid forgetting, we have Alice and Bob play against past versions of their respective opponent in 20% of games. More details about the game structure and pseudocode for training with asymmetric self-play are available in Appendix A.

3.1 REWARD STRUCTURE

For Bob, we assign *sparse* goal-conditioned rewards. We measure the positional and rotational distance between an object and its goal state as the Euclidean distance and the Euler angle rotational distance, respectively. Whenever both distance metrics are below a small error (the *success threshold*), this object is deemed to be placed close enough to the goal state and Bob receives 1 reward immediately. But if this object is moved away from the goal state that it has arrived at in past steps, Bob obtains -1 reward such that the sum of per-object reward is at most 1 during a given turn. When all of the objects are in their goal state, Bob receives 5 additional reward and its turn is over.

For Alice, we assign a reward after Bob has attempted to solve the goal: 5 reward if Bob failed at solving the goal, and 0 if Bob succeeded. We shape Alice's reward slightly by adding 1 reward if it has set a valid goal, defined to be when no object has fallen off the table and any object has been moved more than the success threshold. An additional penalty of -3 reward is introduced when Alice sets a goal with objects outside of the placement area, defined to be a fixed 3D volume within the view of the robot's camera. More details are discussed in Appendix A.2.

3.2 Alice Behavioral Cloning (ABC)

One of the main benefits of using asymmetric self-play is that the generated goals come with at least one solution to achieve it: *Alice's trajectory*. Similarly to Sukhbaatar et al. (2018a), we exploit this property by training Bob with Behavioral Cloning (BC) from Alice's trajectory, in addition to the reinforcement learning (RL) objective. We call this learning mechanism *Alice Behavioral Cloning* (ABC). We propose several improvements over the original formulation in Sukhbaatar et al. (2018a).

Demonstration trajectory filtering Compared to BC from expert demonstrations, using Alice's trajectory needs extra care. Alice's trajectory is likely to be suboptimal for solving the goal, as Alice might arrive at the final state merely by accident. Therefore, we only consider trajectories with goals that Bob failed to solve as demonstrations, to avoid distracting Bob with suboptimal examples. Whenever Bob fails, we relabel Alice's trajectory τ_A to be a goal-augmented version $\tau_{\text{BC}} = \{(s_0, s_T, a_0, r_0), \ldots, (s_T, s_T, a_T, r_T)\}$ as a demonstration for BC, where s_T is the goal.

PPO-style BC loss clipping The objective for training Bob is $\mathcal{L} = \mathcal{L}_{\text{RL}} + \beta\mathcal{L}_{\text{abc}}$, where $\mathcal{L}_{\text{RL}}$ is an RL objective and $\mathcal{L}_{\text{abc}}$ is the ABC loss. β is a hyperparameter controlling the relative importance of the BC loss. We set $\beta = 0.5$ throughout the whole experiment. A naive BC loss is to minimize the negative log-likelihood of demonstrated actions, $-\mathbb{E}_{(s_t,g_t,a_t)\in\mathcal{D}_{\text{BC}}}\big[\log \pi_B(a_t|s_t, g_t; \theta)\big]$ where $\mathcal{D}_{\text{BC}}$ is a mini-batch of demonstration data and π_B is parameterized by θ. We found that overly-aggressive policy changes triggered by BC sometimes led to learning instabilities. To prevent the policy from changing too drastically, we introduce PPO-style loss clipping (Schulman et al., 2017) on the BC loss by setting the advantage $\hat{A} = 1$ in the clipped surrogate objective:

$$\mathcal{L}_{\text{abc}} = -\mathbb{E}_{(s_t,g_t,a_t)\in\mathcal{D}_{\text{BC}}}\left[\text{clip}\Big(\frac{\pi_B(a_t|s_t, g_t; \theta)}{\pi_B(a_t|s_t, g_t; \theta_{\text{old}})}, 1-\epsilon, 1+\epsilon\Big)\right]$$

where $\pi_B(a_t|s_t, g_t; \theta)$ is Bob's likelihood on a demonstration based on the parameters that we are optimizing, and $\pi_B(a_t|s_t, g_t; \theta_{\text{old}})$ is the likelihood based on Bob's behavior policy (at the time of demonstration collection) evaluated on a demonstration. This behavior policy is identical to the policy that we use to collect RL trajectories. By setting $\hat{A} = 1$, this objective optimizes the naive BC loss, but clips the loss whenever $\frac{\pi_B(a_t|s_t,g_t;\theta)}{\pi_B(a_t|s_t,g_t;\theta_{\text{old}})}$ is bigger than $1+\epsilon$, to prevent the policy from changing too much. ϵ is a clipping threshold and we use $\epsilon = 0.2$ in all the experiments.

4 Related Work

Training distribution for RL In the context of multi-task RL (Beattie et al., 2016; Hausman et al., 2018; Yu et al., 2020), multi-goal RL (Kaelbling, 1993; Andrychowicz et al., 2017), and meta RL (Wang et al., 2016; Duan et al., 2016), previous works manually designed a distribution of tasks or goals to see better generalization of a policy to a new task or goal. Domain randomization (Sadeghi & Levine, 2017b; Tobin et al., 2017; OpenAI et al., 2020) manually defines a distribution of simulated environments, but in service of generalizing to the same task in the real world.

There are approaches to grow the training distribution automatically (Srivastava et al., 2013). Self-play (Tesauro, 1995; Silver et al., 2016; 2017; Bansal et al., 2018; OpenAI et al., 2019b; Vinyals et al., 2019) constructs an ever-growing training distribution where multiple agents learn by competing with each other, so that the resulting agent shows strong performance on a single game. OpenAI et al. (2019a) automatically grew a distribution of domain randomization parameters to accomplish better generalization in the task of solving a Rubik's cube on the physical robot. Wang et al. (2019;

2020) studied an automated way to keep discovering challenging 2D terrains and locomotion policies that can solve them in a 2D bipedal walking environment.

We employ asymmetric self-play to construct a training distribution for learning a goal-conditioned policy and to achieve generalization to unseen tasks. Florensa et al. (2018); Nair et al. (2018b); Racaniere et al. (2020) had the same motivation as ours, but trained a generative model instead of a goal setting policy. Thus, the difficulties of training a generative model were inherited by these methods: difficulty of modeling a high dimensional space and generation of unrealistic samples. Lynch et al. (2020); Gupta et al. (2020) used teleoperation to collect arbitrary robot trajectories, and defined a goal distribution from the states in the collected trajectories. This approach likely requires a large number of robot trajectories for each environment configuration (e.g. various types of objects on a table), and randomization of objects was not studied in this context.

Asymmetric self-play Asymmetric self-play was proposed by Sukhbaatar et al. (2018b) as a way to supplement RL training. Sukhbaatar et al. (2018b) mixed asymmetric self-play training with standard RL training on the target task and measured the performance on the target task. Sukhbaatar et al. (2018a) used asymmetric self-play to pre-train a hierarchical policy and evaluated the policy after fine-tuning it on a target task. Liu et al. (2019) adopted self-play to encourage efficient learning with sparse reward in the context of an exploration competition between a pair of agents. As far as we know, no previous work has trained a goal-conditioned policy *purely* based on asymmetric self-play and evaluated generalization to unseen holdout tasks.

Curriculum learning Many previous works showed the difficulty of RL and proposed an automated curriculum (Andrychowicz et al., 2017; Florensa et al., 2017; Salimans & Chen, 2018; Matiisen et al., 2019; Zhang et al., 2020) or auxiliary exploration objectives (Oudeyer et al., 2007; Baranes & Oudeyer, 2013; Pathak et al., 2017; Burda et al., 2019; Ecoffet et al., 2019; 2020) to learn *predefined tasks*. When training goal-conditioned policies, relabeling or reversing trajectories (Andrychowicz et al., 2017; Florensa et al., 2017; Salimans & Chen, 2018) or imitating successful demonstrations (Oh et al., 2018; Ecoffet et al., 2019; 2020) naturally reduces the task complexity. Our work shares a similarity in that asymmetric self-play alleviates the difficulty of learning a goal-conditioned policy via an intrinsic curriculum and imitation from the goal setter's trajectory, but our work does not assume any predefined task or goal distribution.

Hierarchical reinforcement learning (HRL) Some HRL methods jointly trained a goal setting policy (high-level or manager policy) and a goal solving policy (low-level or worker policy) (Vezhnevets et al., 2017; Levy et al., 2019; Nachum et al., 2018). However, the motivation for learning a goal setting policy in HRL is not to challenge the goal solving policy, but to cooperate to tackle a task that can be decomposed into a sequence of sub-goals. Hence, this goal setting policy is trained to optimize task reward for the target task, unlike asymmetric self-play where the goal setter is rewarded upon the other agent's failure.

Robot learning for object manipulation. It has been reported that training a policy for multi-object manipulation is very challenging with *sparse* rewards (Riedmiller et al., 2018; Vecerik et al., 2018). One example is block stacking, which has been studied for a long time in robotics as it involves complex contact reasoning and long horizon motion planning (Deisenroth et al., 2011). Learning block stacking often requires a hand-designed curriculum (Li et al., 2019), meticulous reward shaping (Popov et al., 2017), fine-tuning (Rusu et al., 2017), or human demonstrations (Nair et al., 2018a; Duan et al., 2017). In this work, we use block stacking as one of the holdout tasks to test zero-shot generalization, but without training on it.

5 EXPERIMENTS

In this section, we first show that asymmetric self-play generates an effective training curriculum that enables generalization to unseen hold-out tasks. Then, the experiment is scaled up to train in an environment containing multiple random complex objects and evaluate it with a set of holdout tasks containing unseen objects and unseen goal configurations. Finally, we demonstrate how critical ABC is for Bob to make progress in a set of ablation studies.

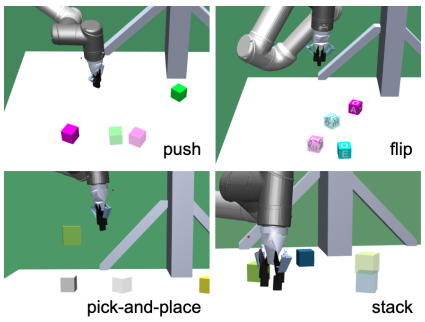


Figure 3: Holdout tasks in the environment using 1 or 2 blocks. The transparent blocks denote the desired goal state, while opaque blocks are the current state. (a) `push`: The blocks must be moved to their goal locations and orientations. There is no differentiation between the six block faces. (b) `flip`: Each side of the block is labelled with a unique letter. The blocks must be moved to make every face correctly positioned as what the goal specifies. (c) `pick-and-place`: One goal block is in the air. (d) `stack`: Two blocks must be stacked in the right order at the right location.

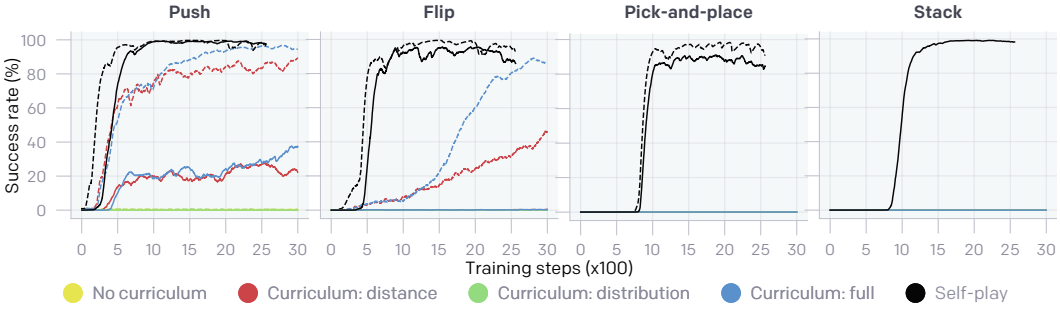


Figure 4: Generalization to unseen holdout tasks for blocks. Baselines are trained over a mixture of all holdout tasks. The solid lines represent 2-blocks, while the dashed lines are for 1-block. The x-axis denotes the number of training steps via asymmetric self-play. The y-axis is the zero-shot generalization performance of Bob policy at corresponding training checkpoints. Note that success rate curves of completely failed baselines are occluded by others.

5.1 Experimental setup

We implement the training environment[2] described in Sec. 2 with randomly placed ShapeNet objects (Chang et al., 2015) as an initial state distribution. In addition, we set up another simpler environment using one or two blocks of fixed size, used for small-scale comparisons and ablation studies. Figure 3 visualizes four holdout tasks for this environment. Each task is designed to evaluate whether the robot has acquired certain manipulation skills: pushing, flipping, picking up and stacking blocks. Experiments in Sec. 5.2, 5.3 and 5.5 focus on blocks and experimental results based on ShapeNet objects are present on Sec. 5.4. More details on our training setups are in Appendix B.

We implement Alice and Bob as two independent policies of the same network architecture with memory (Appendix B.4), except that Alice has no observation on goal state. The policies take state observations ("state policy") for experiments with blocks (Sec. 5.2, 5.3, and 5.5), and take both vision and state observations ("hybrid policy") for experiments with ShapeNet objects (Sec. 5.4). Both policies are trained with Proximal Policy Optimization (PPO) (Schulman et al., 2017).

5.2 Generalization to unseen goals without manual curricula

One way to train a single policy to acquire all the skills in Figure 3 is to train a goal-conditioned policy directly over a mixture of these tasks. However, training directly over these tasks without a curriculum turns out to be very challenging, as the policy completely fails to make any progress.[3] In contrast, Bob is able to solve all these holdout tasks quickly when learning via asymmetric self-play, without explicitly encoding any prior knowledge of the holdout tasks into the training distribution.

To gauge the effect of an intrinsic curriculum introduced by self-play, we carefully designed a set of non-self-play baselines using explicit curricula controlled by Automatic Domain Randomization (OpenAI et al., 2019a). All baselines are trained over a mixture of block holdout tasks as the

[2]Our training and evaluation environments are publicly available at `https://github.com/openai/robogym`

[3]The tasks was easier when we ignored object rotation as part of the goal, and used a smaller table.

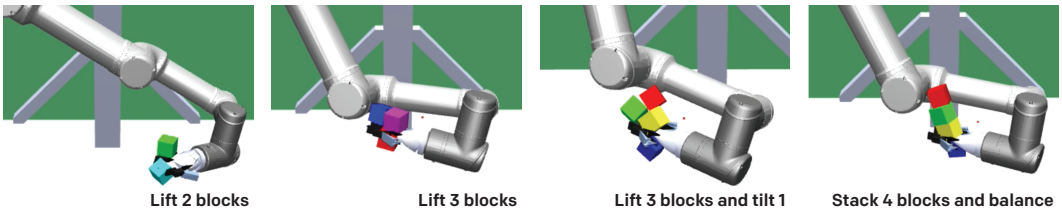


Figure 5: Goals discovered by asymmetric self-play. Alice discovers many goals that are not covered by our manually designed holdout tasks on blocks.

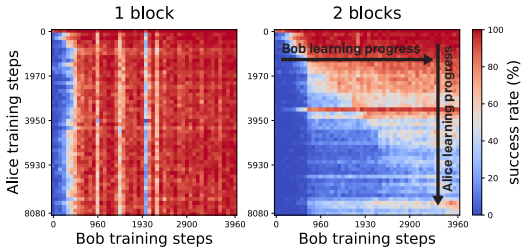


Figure 6: The empirical payoff matrix between Alice and Bob. Average success rate over multiple self-play episode is visualized. Alice with more training steps generates more challenging goals that Bob cannot solve yet. Bob with more training steps can achieve more goals against the same Alice.

goal distribution. We measure the effectiveness of a training setup by tracking the success rate for each holdout task, as shown in Figure 4. The `no curriculum` baseline fails drastically. The `curriculum:distance` baseline expands the distance between the initial and goal states gradually as training progresses, but only learns to push and flip a single block. The `curriculum:distribution` baseline, which slowly increases the proportion of pick-and-place and stacking goals in the training distribution, fails to acquire any skill. The `curriculum:full` baseline incorporates all hand-designed curricula yet still cannot learn how to pick up or stack blocks. We have spent a decent amount of time iterating and improving these baselines but found it especially difficult to develop a scheme good enough to compete with asymmetric self-play. See Appendix C.1 for more details of our baselines.

5.3 Discovery of novel goals and solutions

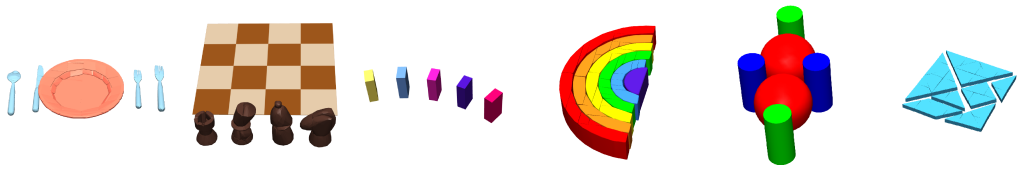
Asymmetric self-play discovers novel goals and solutions that are not covered by our holdout tasks. As illustrated in Figure 5, Alice can lift multiple blocks at the same time, build a tower and then keep it balanced using an arm joint. Although it is a tricky strategy for Bob to learn on its own, with ABC, Bob eventually acquires the skills for solving such complex tasks proposed by Alice. Videos are available at `https://robotics-self-play.github.io`.

Figure 6 summarizes Alice and Bob's learning progress against each other. For every pair of Alice and Bob, we ran multiple self-play episodes and measured the success rate. We observe an interesting trend with 2 blocks. As training proceeds, Alice tends to generate more challenging goals, where Bob shows lower success rate. With past sampling, Bob continues to make progress against versions of Alices from earlier optimization steps. This visualization suggests a desired dynamic of asymmetric self-play that could potentially lead to unbounded complexity: Alice continuously generates goals to challenge Bob, and Bob keeps making progress on learning to solve new goals.

5.4 Generalization to unseen objects and goals

The experiments above show strong evidence that efficient curricula and novel goals can autonomously emerge in asymmetric self-play. To further challenge our approach, we scale it up to work with many more complex objects using more computational resources for training. We train a hybrid policy in an environment containing up to 10 random ShapeNet (Chang et al., 2015) objects. During training, we randomize the number of objects and the object sizes via Automatic Domain Randomization (OpenAI et al., 2019a). The hybrid policy uses vision observations to extract information about object geometry and size. We evaluate the Bob policy on a more diverse set of manipulation tasks, including semantically interesting ones. Many tasks contain unseen objects and complex goals, as illustrated in Figure 7.

The learned Bob policy achieves decent zero-shot generalization performance for many tasks. Success rates are reported in Figure 8. Several tasks are still challenging. For example, `ball-capture`



(a) Table setting (b) Mini chess (c) Domino (d) Rainbow (e) Ball-capture (f) Tangram

Figure 7: Example holdout tasks involving unseen objects and complex goal states. The goal states are illustrated here, and the initial states have randomly placed objects.

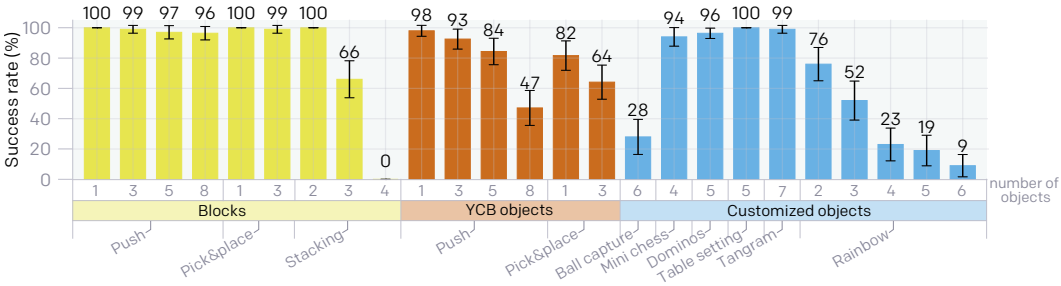


Figure 8: Success rates of a single goal-conditioned policy solving a variety of holdout tasks, averaged over 100 trials. The error bars indicate the 99% confidence intervals. Yellow, orange and blue bars correspond to success rates of manipulation tasks with blocks, YCB[4]objects and other uniquely built objects, respectively. Videos are available at `https://robotics-self-play.github.io`.

requires delicate handling of rolling objects and lifting skills. The `rainbow` tasks call for an understanding of concave shapes. Understanding the ordering of placement actions is crucial for stacking more than 3 blocks in the desired order. The Bob policy learns such an ordering to some degree, but fails to fully generalize to an arbitrary number of stacked blocks.

5.5 Ablation Studies

We present a series of ablation studies designed for measuring the importance of each component in our asymmetric self-play framework, including Alice behavioral cloning (ABC), BC loss clipping, demonstration filtering, and the multi-goal game setup. We disable a single ingredient in each ablation run and compare with the complete self-play baseline in Figure 9.

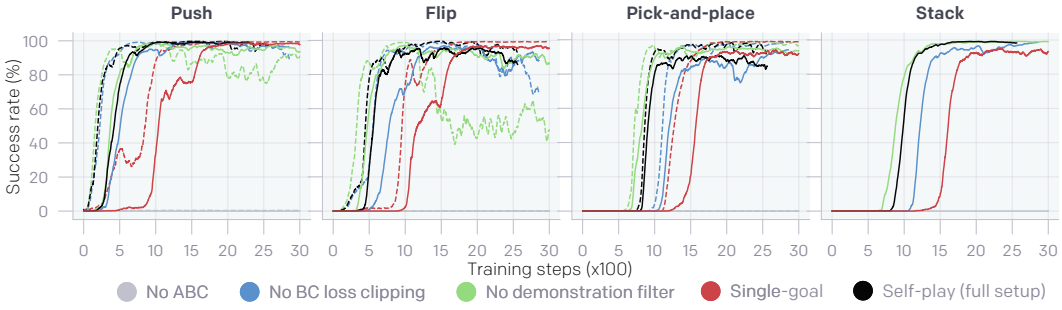


Figure 9: The ablation studies compare four ablation runs each with one component disabled with the full baseline. Solid lines are for 2-blocks, dashed lines are for 1-block. The x-axis denotes the number of training steps via asymmetric self-play. The y-axis is the zero-shot generalization performance of Bob policy at corresponding training steps.

The `no ABC` baseline shows that Bob *completely fails* to solve any holdout task without ABC, indicating that ABC is a critical mechanism in asymmetric self-play. The `no BC loss clipping` baseline shows slightly slower learning on `pick-and-place` and `stack`, as well as some instabilities in the

[4] `https://www.ycbbenchmarks.com/object-models/`

middle of training. Clipping in the BC loss is expected to help alleviate this instability by controlling the rate of policy change per optimizer iteration. The `no demonstration filter` baseline shows noticeable instability on `flip`, suggesting the importance of excluding suboptimal demonstrations from behavioral cloning. Finally, the `single-goal` baseline uses a single goal instead of 5 goals per episode during training. The evaluation tasks are also updated to require a single success per episode. Generalization of this baseline to holdout tasks turns out to be much slower and less stable. It signifies some advantages of using multiple goals per episode, perhaps due to the policy memory internalizing environmental information during multiple trials of goal solving.

The results of the ablation studies suggest that ABC with proper configuration and multi-goal gameplay are critical components of asymmetric self-play, alleviating the importance of manual curricula and facilitating efficient learning.

6 CONCLUSION

One limitation of our asymmetric self-play approach is that it depends on a resettable simulation environment as Bob needs to start from the same initial state as Alice's. Therefore asymmetric self-play training has to happen in a simulator which can be easily updated to a desired state. In order to run the goal-solving policy on physical robots, we plan to adopt sim-to-real techniques in future work. Sim-to-real has been shown to achieve great performance on many robotic tasks in the real world (Sadeghi & Levine, 2017a; Tobin et al., 2017; James et al., 2019; OpenAI et al., 2020). One potential approach is to pre-train two agents via asymmetric self-play in simulation, and then fine-tune the Bob policy with domain randomization or data collected on physical robots.

In conclusion, we studied asymmetric self-play as a framework for defining a single training distribution to learn many arbitrary object manipulation tasks. Even without any prior knowledge about the target tasks, asymmetric self-play is able to train a strong goal-conditioned policy that can generalize to many unseen holdout tasks. We found that asymmetric self-play not only generates a wide range of interesting goals but also alleviates the necessity of designing manual curricula for learning such goals. We provided evidence that using the goal setting trajectory as a demonstration for training a goal solving policy is essential to enable efficient learning. We further scaled up our approach to work with various complex objects using more computation, and achieved zero-shot generalization to a collection of challenging manipulation tasks involving unseen objects and unseen goals.

AUTHOR CONTRIBUTIONS

This manuscript is the result of the work of the entire OpenAI Robotics team. We list the contributions of every team member here grouped by topic and in alphabetical order.

- Hyeonwoo Noh and Lilian Weng designed and implemented the asymmmetric self-play algorithm for robotics manipulation tasks.
- Vineet Kosaraju, Hyeonwoo Noh, Alex Paino, Matthias Plappert, Lilian Weng and Qiming Yuan developed the simulation environments for RL training.
- Casey Chu, Vineet Kosaraju, Alex Paino, Henrique Ponde de Oliveira Pinto, Qiming Yuan and Tao Xu worked on the vision stack configuration.
- Ilge Akkaya, Ruben D'Sa, Arthur Petron and Raul Sampedro developed the robot controller and simulation.
- Qiming Yuan developed toolings for building holdout environments.
- Ilge Akkaya, Alex Paino, Arthur Petron, Henrique Ponde de Oliveira Pinto, Lilian Weng, Tao Xu and Qiming Yuan built a collection of the holdout environments.
- Henrique Ponde de Oliveira Pinto optimized the training infrastructure and developed monitoring metrics.
- Casey Chu, Hyeonwoo Noh and Lilian Weng drafted the manuscript.
- All authors refined and revised the manuscript.
- Ilge Akkaya, Alex Paino, Matthias Plappert, Peter Welinder, and Lilian Weng led aspects of this project and set research directions.
- Ilge Akkaya, Matthias Plappert and Wojciech Zaremba managed the team.

ACKNOWLEDGMENTS

We would like to thank Joel Lehman, Ingmar Kanitscheider and Harri Edwards for providing thoughtful feedback for the algorithm and earlier versions of the manuscript. We also would like to thank Bowen Baker and Lei Zhang for helpful discussion on the idea. Finally we are grateful for everyone at OpenAI, especially the Supercomputing team, for their help and support.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, 2017.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems VII*, pp. 57–64, 2011.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return then explore. *arXiv preprint arXiv:2004.12919*, 2020.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning*, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, 2018.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE international conference on robotics and automation (ICRA)*, 2017.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, 2020.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. *arXiv preprint arXiv:1912.11032*, 2019.

Hao Liu, Alexander Trott, Richard Socher, and Caiming Xiong. Competitive experience replay. In *International Conference on Learning Representations*, 2019.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pp. 1113–1132, 2020.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018a.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, 2018b.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *arXiv preprint arXiv:1806.05635*, 2018.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019a.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019b.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

Sebastien Racaniere, Andrew Lampinen, Adam Santoro, David Reichert, Vlad Firoiu, and Timothy Lillicrap. Automated curriculum generation through setter-solver interactions. In *International Conference on Learning Representations*, 2020.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing - solving sparse reward tasks from scratch, 2018.

Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, 2017.

Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017a. URL `http://www.roboticsproceedings.org/rss13/p34.html`.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems*, 2017b.

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Rupesh Kumar Srivastava, Bas R Steunebrink, and Jürgen Schmidhuber. First experiments with powerplay. *Neural Networks*, 41:130–136, 2013.

Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv preprint arXiv:1811.09083*, 2018a.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018b.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3): 58–68, 1995.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, 2018.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. *arXiv preprint arXiv:2003.08536*, 2020.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2020.

Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. Automatic curriculum learning through value disagreement. *arXiv preprint arXiv:2006.09641*, 2020.

A Asymmetric Self-play Game Setup

A.1 Goal validation

Some of the goals set via asymmetric self-play may not be useful or interesting enough to be included in the training distribution. For example, if Alice fails to touch any objects, Bob can declare a success without any action. A goal outside the table might be tricky to solve. We label such goals as *invalid* and make sure that Alice has generated a *valid* goal before starting Bob's turn.

Even when Alice generates a valid goal, we can still penalize certain undesired goals. Specifically, if the visual perception is limited to a restricted area on the table due to the camera setting, we can penalize goals containing objects outside that range of view.

For goal validation, we check in the following order:

1. We check whether any object has moved. If not, the goal is considered *invalid* and the episode resets.
2. We check whether all the objects are on the table. If not, the goal is considered *invalid* and the episode resets.
3. We check whether a valid goal has objects outside the placement area, defined to be a 3D space that the robot end effector can reach and the robot camera can see. If any object is outside the area, the goal is deemed valid but obtains a *out-of-zone* penalty reward and the episode continues to switch to Bob's turn.

A.2 Reward structure

Table 1 shows the reward structure for a single turn for goal setting and solving. The reward for Alice is based on whether it successfully generates a valid goal, and whether the generated goal is solved by Bob. Alice obtains 1 point for a valid goal, and obtains an additional 5 point game reward if Bob fails to achieve it. Additionally, a goal out of placement area triggers a -3 penalty. Rewarding Alice based only on Bob's success or failure is simpler than the original reward from Sukhbaatar et al. (2018b), but we didn't notice any degradation from this simplification (Appendix C.2).

Since Bob is a goal-conditioned policy, we provide *sparse* goal-conditioned rewards. Whenever one object is placed at its desired position and orientation, Bob obtains 1 point per-object reward. Bob obtains -1 reward such that the sum of per-object reward is at most 1 during a given turn. When all the objects are in the goal state, Bob obtains a 5 point success reward and its turn terminates. If Bob reaches a maximum number of allowed steps before achieving the goal, it is deemed a failure with 0 point reward.

When checking whether a goal has been achieved, we compare the position and orientation of each object with its goal position and orientation. For position, we compute the Euclidean distance between object centers. For rotation, we represent the orientation of an object by three Euler angles on three dimensions, roll, pitch, and yaw, respectively, and we compute the minimum angle needed to rotate the object into the goal orientation. If the distance and angle for all objects are less than a small error (0.04 meters and 0.2 radians respectively), we consider the goal achieved.

Table 1: Reward structure for a single goal.

Alice	Bob	Alice reward	Bob reward
Invalid goal	-	0	-
Out-of-zone goal	Failure	1 - 3 + 5	0 + per-object reward
Out-of-zone goal	Success	1 - 3 + 0	5 + per-object reward
Valid goal	Failure	1 + 5	0 + per-object reward
Valid goal	Success	1 + 0	5 + per-object reward

A.3 Multi-goal game structure

The overall flowchart of asymmetric self-play with a multi-goal structure is illustrated in Figure 10.

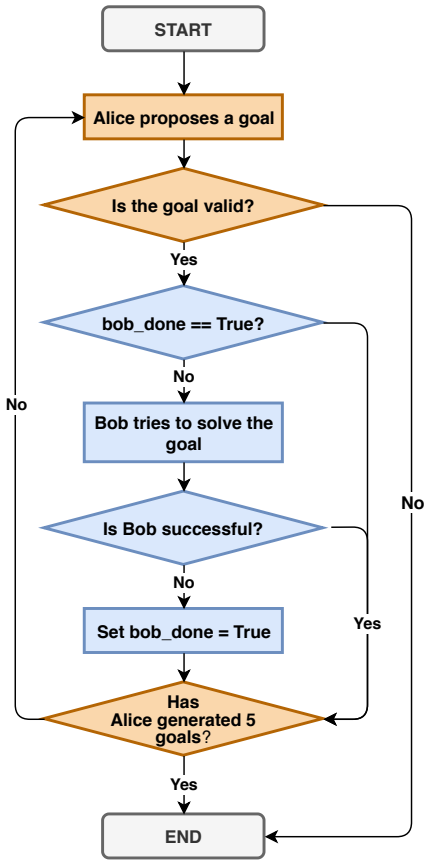


Figure 10: The flow chart of asymmetric self-play with a multi-goal game structure. The steps in orange belong to Alice while the blue ones belong to Bob.

We expect a multi-goal game structure to induce a more complicated goal distribution, as goals can be built on top of each other. For example, in order to stack 3 blocks, you might need to stack 2 blocks first as a subgoal at the first step. A multi-goal structure also encourages Bob to internalize environmental information during multiple trials of goal solving. Many aspects of the environment, such as the simulator's physical dynamics and properties of objects, stay constant within one episode, so Bob can systematically investigate these constant properties and exploit them to adapt its goal solving strategy accordingly. Similar behavior with multi-goal setting was observed by OpenAI et al. (2019a).

In our experiments, when we report the success rate from multi-goal episodes, we run many episodes with a maximum of 5 goals each and compute the success rate as

$$\texttt{success\_rate} = \frac{\texttt{total\_successes}}{\texttt{total\_goals}}.$$

Note that because each episode terminates after one failure, at the end of one episode, we would have either $\texttt{total\_goals} = \texttt{total\_successes}$ if Bob succeeded at every goal, or $\texttt{total\_goals} = \texttt{total\_successes} + 1$ if Bob fails in the middle.

A.4 Training algorithm

Algorithm 1 and 2 describe pseudocode for the training algorithm using asymmetric self-play. Both policies are optimized via Proximal Policy Optimization (PPO) (Schulman et al., 2017). Additionally, Bob optimizes the Alice behavioral cloning (ABC) loss using Alice's demonstrations collected during the interplay between two agents. In the algorithm, $\mathcal{L}_{\text{RL}}$ denotes a loss function for PPO and $\mathcal{L}_{\text{ABC}}$ denotes a loss function for ABC. A trajectory τ contains a list of (state, action, reward) tuples,

Algorithm 1 Asymmetric self-play

Require: $\theta_{\mathrm{A}}, \theta_{\mathrm{B}}$ ▷ Initial parameters for Alice and Bob
Require: η ▷ RL learning rate
Require: β ▷ weight of BC loss
  for training steps $= 1, 2, ...$ **do**
    $\theta_{\mathrm{A}}^{\mathrm{old}} \leftarrow \theta_{\mathrm{A}}, \theta_{\mathrm{B}}^{\mathrm{old}} \leftarrow \theta_{\mathrm{B}}$ ▷ initialize behavior policy parameters
    for each rollout worker **do** ▷ parallel data collection
      $\mathcal{D}_{\mathrm{A}}, \mathcal{D}_{\mathrm{B}}, \mathcal{D}_{\mathrm{BC}} \leftarrow$ **CollectRolloutData**$(\theta_{\mathrm{A}}^{\mathrm{old}}, \theta_{\mathrm{B}}^{\mathrm{old}})$ ▷ replay buffers for Alice, Bob and ABC
    end for
    $\theta_{\mathrm{A}} \leftarrow \theta_{\mathrm{A}} - \eta \nabla_{\theta_{\mathrm{A}}} \mathcal{L}_{\mathrm{RL}}$ ▷ optimize PPO loss with data popped from $\mathcal{D}_{\mathrm{A}}$
    $\theta_{\mathrm{B}} \leftarrow \theta_{\mathrm{B}} - \eta \nabla_{\theta_{\mathrm{B}}} \left[\mathcal{L}_{\mathrm{RL}} + \beta \mathcal{L}_{\mathrm{ABC}}\right]$ ▷ optimize RL loss with $\mathcal{D}_{\mathrm{B}}$ and ABC loss with $\mathcal{D}_{\mathrm{BC}}$
  end for

Algorithm 2 CollectRolloutData

Require: $\theta_{\mathrm{A}}^{\mathrm{old}}, \theta_{\mathrm{B}}^{\mathrm{old}}$ ▷ behavior policy parameters for Alice and Bob
Require: $\pi_{\mathrm{A}}(a|s; \theta_{\mathrm{A}}^{\mathrm{old}}), \pi_{\mathrm{B}}(a|s, g; \theta_{\mathrm{B}}^{\mathrm{old}})$ ▷ policies for Alice and Bob
Require: ξ ▷ whether Bob succeeded to achieve a goal
  $\mathcal{D}_{\mathrm{A}} \leftarrow \varnothing, \mathcal{D}_{\mathrm{B}} \leftarrow \varnothing, \mathcal{D}_{\mathrm{BC}} \leftarrow \varnothing$ ▷ Initialize empty replay buffers.
  $\xi \leftarrow$True ▷ initialize to True (success)
  for number of goals = 1, ..., 5 **do**
    $\tau_{\mathrm{A}}, g \leftarrow$ **GenerateAliceTrajectory**$(\pi_A, \theta_{\mathrm{A}}^{\mathrm{old}})$ ▷ generates a trajectory τ_{A} and a goal g
    if goal g is invalid **then**
      break
    end if
    if ξ is True **then**
      $\tau_{\mathrm{B}}, \xi \leftarrow$ **GenerateBobTrajectory**$(\pi_{\mathrm{B}}, \theta_{\mathrm{B}}^{\mathrm{old}}, g)$ ▷ generate a trajectory τ_{B} and update ξ
      $\mathcal{D}_{\mathrm{B}} \leftarrow \mathcal{D}_{\mathrm{B}} \cup \{\tau_{\mathrm{B}}\}$ ▷ update replay buffer for Bob
    end if
    $r_{\mathrm{A}} \leftarrow$ **ComputeAliceReward**(ξ, g)
    $\tau_{\mathrm{A}}[-1][2] \leftarrow r_{\mathrm{A}}$ ▷ overwrite the last reward in trajectory τ_{A} with r_{A}
    $\mathcal{D}_{\mathrm{A}} \leftarrow \mathcal{D}_{\mathrm{A}} \cup \{\tau_{\mathrm{A}}\}$ ▷ update replay buffer for Alice
    if ξ is False **then**
      $\tau_{\mathrm{BC}} \leftarrow$ **RelabelDemonstration**$(\tau_{\mathrm{A}}, g, \pi_{\mathrm{B}}, \theta_{\mathrm{B}}^{\mathrm{old}})$ ▷ relabeled to be goal-augmented
      $\mathcal{D}_{\mathrm{BC}} \leftarrow \mathcal{D}_{\mathrm{BC}} \cup \{\tau_{\mathrm{BC}}\}$ ▷ update replay buffer for ABC
    end if
  end for
  return $\mathcal{D}_{\mathrm{A}}, \mathcal{D}_{\mathrm{B}}, \mathcal{D}_{\mathrm{BC}}$

$\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots\}$. A goal-augmented trajectory τ_{BC} contains a list of (state, goal, action, reward) tuples, $\tau_{\mathrm{BC}} = \{(s_0, g, a_0, r_0), (s_1, g, a_1, r_1), \dots\}$.

B Training Setup

B.1 Simulation Setup

We utilize the MuJoCo physics engine (Todorov et al., 2012) to simulate our robot environment and render vision observations and goals. We model a UR16e robotic arm equipped with a RobotIQ 2F-85 parallel gripper end effector. The robot arm is controlled via its tool center point (TCP) pose that is actuated via MuJoCo constraints. Additionally, we use a PID controller to actuate the parallel gripper using position control.

B.2 Action Space

We define a 6-dimensional action space consisting of 3D relative gripper position, 2D relative gripper rotation, and a 1D desired relative gripper finger position output that is applied symmetrically

(a) Front camera (b) Wrist camera (c) Front camera (goal)

Figure 11: Example vision observations from our camera setup. (a) observation from a camera mounted in front of the table (the front camera). (b) observation from the mobile camera mounted on the gripper wrist. (c) goal observation from the front camera.

to the two gripper pinchers. The two rotational degrees of freedom correspond to yaw and pitch axes (wrist rotation and wrist tilt) respectively, with respect to the gripper base. We use a discretized action space with 11 bins per dimension and learn a multi-categorical distribution.

B.3 Observation Space

We feed observations of robot arm position, gripper position, object state, and goal state into the policy. The object state observation contains each object's position, rotation, velocity, rotational velocity, the distance between the object and the gripper, as well as whether this object has contacts with the gripper. The goal state observation includes each object's desired position and rotation, as well as the relative distance between the current object state and the desired state.

In the hybrid policy for the ShapeNet training environment, we additionally feed three camera images into the policy: an image of the current state captured by a fixed camera in front of the table, an image of the current state from a camera mounted on the gripper wrist, and an image of the goal state from the fixed camera. Figure 11 illustrates the example observations from our camera setup. Both Alice and Bob take robot and object state observations as inputs, but Alice does not take goal state inputs since it is not goal-conditioned.

B.4 Model Architecture

We use independent policy and value networks in the PPO policy. Both have the same observation inputs and network architecture, as illustrated in Figure 13. The permutation invariant embedding module concatenates all the observations per object, learns an embedding vector per object and then does max pooling in the object dimension. The vision module uses the same model architecture as in IMPALA (Espeholt et al., 2018). For all experiments, we use completely separate parameters for the policy and the value network except the vision module, which is shared between them.

B.5 Hyperparameters

Hyperparameters used in our PPO policy and asymmetric self-play setup are listed in Table 2 and Table 3.

The maximum goal solving steps for Bob reported in Table 3 is the number of steps allowed per object within one episode. If Bob has spent all these time steps but still cannot solve the goal, it deems a failure and the episode terminates for Bob.

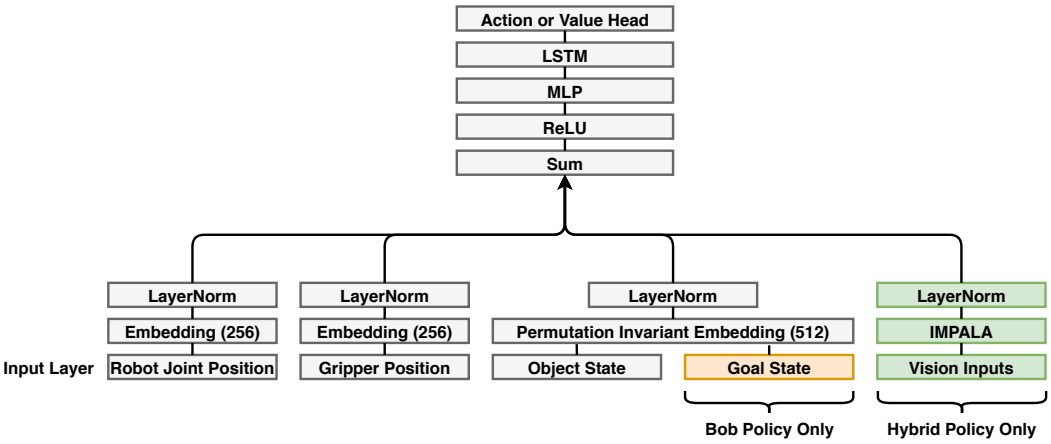


Figure 12: Network architecture of value/policy network.

Table 2: Hyperparameters used for PPO.

Hyperparameter	**Value**
discount factor γ	0.998
Generalized Advantage Estimation (GAE) λ	0.95
entropy regularization coefficient	0.01
PPO clipping parameter ϵ_{ppo}	0.2
ABC clipping parameter ϵ	0.2
optimizer	Adam (Kingma & Ba, 2014)
learning rate η	3×10^{-4}
sample reuse (experience replay)	3
value loss weight	1.0
ABC loss weight	0.5

Table 3: Hyperparameters used for hardware configuration, batch size and self-play episode length.

Hyperparameter	**1-2 Block manipulation** (state)	**ShapeNet object rearrangement** (hybrid)
GPUs per policy	1	32×8
rollout worker CPUs	64×29	576×29
batch size	4096	$55 \times 32 \times 8$
Alice's goal setting steps T	100	250
Bob's maximum goal solving steps	200	600

B.6 Holdout Tasks

Here are a list of tasks for evaluating zero-shot generalization capability of the hybrid policy. Some of them are visualized in Figure 8. Note that none of the objects here appear in the training data.

- Table setting: arrange a table setting consisting of a plate, spoon, knife, salad fork, and main course fork.
- Mini chess: place four chess pieces next to a chess board.
- Rainbow (2-6 pieces): build a rainbow out of colored, wooden pieces by placing the half-circle shapes together so they resemble a rainbow. We have 5 variations of the rainbow tasks by taking different numbers of pieces which are indexed from the outer circle to inner one.

- Ball-capture: capture two, red field-hockey balls by placing four (two blue and two green) cylinders at so their rotational axes intersect rays from the spheres at roughly 90, 240, and 300 degree angles about the same axial (Z) direction.
- Tangram Puzzle: move blue pieces to form the standard, seven piece tangram square solution.
- Domino: stand up 5 wooden domino pieces in a curved layout.
- Block push (1–8 objects): push blocks to match position and orientation of goal configurations. All goal objects are on the surface of the table.
- Block pick-and-place (1–3 objects): push blocks, and lift up one block in the air. One goal object is in the air and all other goal objects are on the table surface.
- Block stacking (2–4 objects): stack blocks to form a tower in a specific location and specific rotation.
- YCB object push (1–8 objects): push YCB objects[5] to match position and orientation of goal configurations. All goal objects are on the surface of the table.
- YCB pick-and-place (1–3 objects): push YCB objects and lift up one YCB object in the air. One goal object is in the air and all other goal objects are on the table surface.

By default, each holdout task presents 5 goals per episode and terminates episodes upon a failure. Exceptions are Table setting, Mini chess, Rainbow (2-6 pieces), Ball-capture, and Tangram, which only present a single goal per episode because only a single fixed goal configuration is available for each holdout task.

C Non Self-play Baselines

C.1 Baselines for curriculum

We compared asymmetric self-play with several baselines incorporating hand-designed curricula in Sec. 5.2.

All the baselines are trained on a mixture of push, flip, pick-and-place, and stacking tasks as the goal distribution. The initial state of objects is generated by randomly placing objects within the placement area of the table without overlaps. The number of objects is sampled from $\{1, 2\}$ with equal probability.

Factorized Automatic Domain Randomization (FADR) (OpenAI et al., 2019a) is applied to grow curriculum parameters described below. Precisely, for each parameter we track a list of performance scores when the parameter is configured at current maximum and other parameters are randomly sampled. The value of this parameter will be increased if the tracked score rises above a threshold.

1. The `no curriculum` baseline trains the goal-conditioned policy directly on a fixed goal distribution and environment parameters. Precisely there are 50% goals for push and flip, 35% for pick-and-place, and 15% for stacking.
2. The `curriculum:distance` baseline uses a hand-designed curriculum over (1) `goal_distance_ratio`: the Euclidean distance between the initial position of the objects and their goal positions, and (2) `goal_rotation_weight`: the weight applied on the object rotation distance for goal state matching. At the beginning of the episode, given an initial position x_0 and a goal position x_g, we artificially make the goal easier according to `goal_distance_ratio` by resetting the goal position to $x'_g = x_0 + (x_g - x_0) \times$ `goal_distance_ratio`. A small ratio reduces the distance and thus makes the task less difficult. The parameter `goal_rotation_weight` controls how much we care about a good match between object rotation. Given the current object rotation r_t and a desired rotation r_g, we consider them as a valid match if $|r_t - r_0| \times$ `goal_rotation_weight` $< r_{\text{threshold}}$, where $r_{\text{threshold}} = 0.2$ (radians) is the success threshold for rotational matching. In other words, a small goal rotation weight creates a less strict success threshold. Both parameters range between $[0, 1]$ and gradually increase from 0 to 1 as training progresses.

[5] `https://www.ycbbenchmarks.com/object-models/`

3. The `curriculum:distribution` controls the proportion of pick-and-place and stacking goals via two ADR parameters, `pickup_proba` and `stack_proba`. When sampling new goals, with probability `pickup_proba`, a random object is moved up to the air and with probability `stack_proba`, we consider a small 2-block tower as the goal. Both parameters range between $[0, 0.5]$ and gradually increase from 0 to 0.5 as training progresses.
4. The `curriculum:full` baseline adopts all the ADR parameters described so far, `goal_distance_ratio`, `goal_rotation_weight`, `pickup_proba` and `stack_proba`. When setting up a pick-and-place goal, the height above the table surface is also interpolated according to `goal_distance_ratio`.

C.2 Comparison with timestep-based reward

Contrary to timestep-based reward originally proposed by Sukhbaatar et al. (2018b), we reward Alice simply based on the success or failure of Bob's goal solving attempt. We compare our simplified reward structure with timestep-based reward for Alice as described in Sukhbaatar et al. (2018b) with time reward scale factor 0.01 ($\gamma = 0.01$ based on the notation from Sukhbaatar et al. (2018b)). The performance is very similar. Note that this result is not a direct comparison to Sukhbaatar et al. (2018b), but an ablation study of two different reward functions based on our best asymmetric self-play configuration.

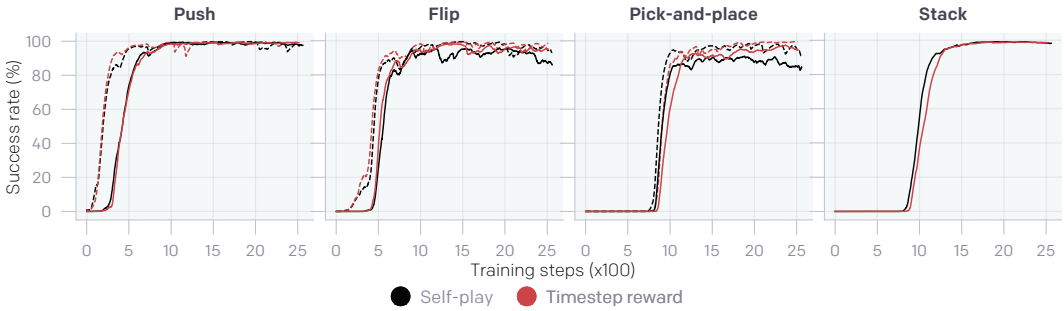


Figure 13: The comparison of our asymmetric self-play reward with the timestep-based reward.