Appendix A
Experiments Setting Details

In this appendix, more details on the robot manipulation environments setup are provided.

- **Environments without Obstacle:** The robotic manipulation tasks without obstacle are set exactly the same as how [1] did.
- **Environments with Obstacle:** Based on the aforementioned tasks, a static brick-like obstacle is randomly placed in the robot workspace. Specifically, in pushing and sliding tasks, bricks are placed with uniformly random pose (denoted by $(x_{obs}, y_{obs}, z_{obs}, \alpha_{obs}, \beta_{obs}, \gamma_{obs})$) on the table in the 30×30cm square with the center under the initial position of the robot's end-effector, and for pick-and-place task a brick's x and y coordinates are sampled from the same square and its z coordinate is sampled uniformly from $[0, 25cm]$. The brick size $(length \times width \times height)$ in pushing and pick-and-place tasks is $16 \times 5 \times 8$cm while the one in sliding task is $8 \times 8 \times 12$cm. Video of training on these environments is available in [2] (https://www.youtube.com/watch?v=wJsKl8ZrqiQ).

Appendix B
Hypotheses and Experiments on Performance Drop

To systematically analyze the performance drop met in KER ablative study, hypotheses listed below are proposed and relative experiments are conducted to verify. For simplicity, we conducted experiments only in the pick-and-place task, and each plot of learning performance is averaged with 2 runs in this section.

- **Hypothesis 1: Overfitting towards Buffer Data**
  In deep learning, the network tends to overfit towards the training dataset after a long-run learning, leading to poorer generalization. Does it related to the performance drop here? To answer this question, we design an experiment with a new evaluation on the robot learning performance, in which goals are sampled from buffer (Fig. 1). If it were the case of overfitting towards buffer, there should be no drop in performance. However, the experimental results in Fig. 2 show that the drops are still occurred in around same epochs (80 epochs and 150 epochs for $n_{ker} = 16$ and $n_{ker} = 32$ respectively) as before. Therefore, the drop is not caused by overfitting towards buffer data.

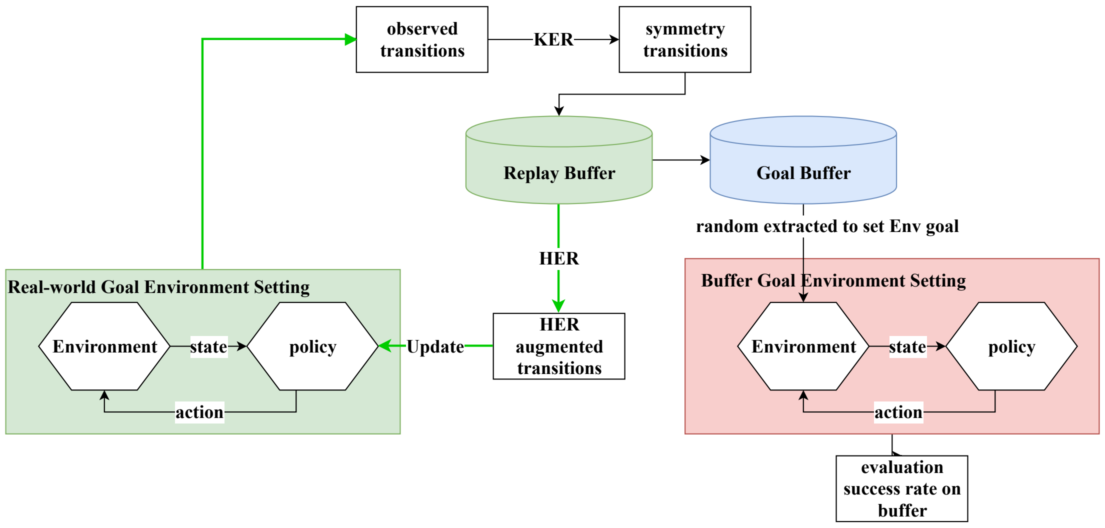

observed transitions
KER
symmetry transitions
Replay Buffer
Goal Buffer
random extracted to set Env goal
HER
Real-world Goal Environment Setting
Environment
state
policy
action
Update
HER augmented transitions
Buffer Goal Environment Setting
Environment
state
policy
action
evaluation success rate on buffer

Fig. 1: Proposed evaluation framework of the learning performance in terms of buffer data in testing phase.

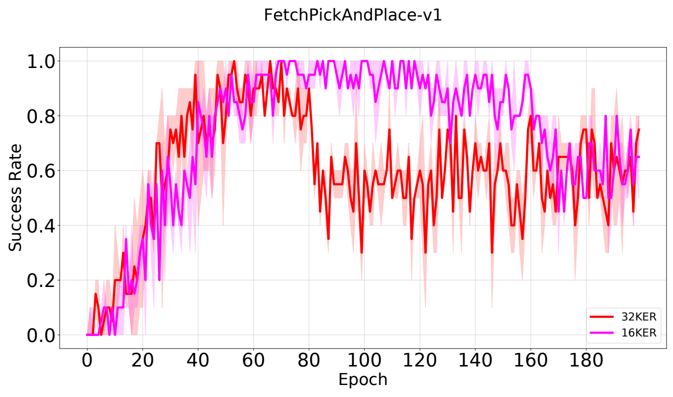


Fig. 2: Plots of the learning performance in terms of buffer data in testing phase

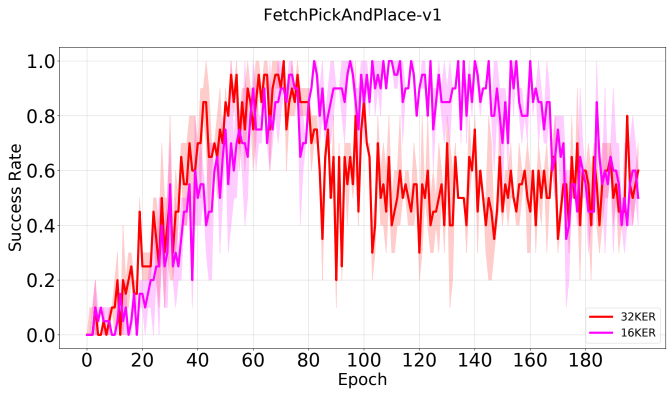


Fig. 3: Plots of the learning performance with stopping using HER in 20 epochs

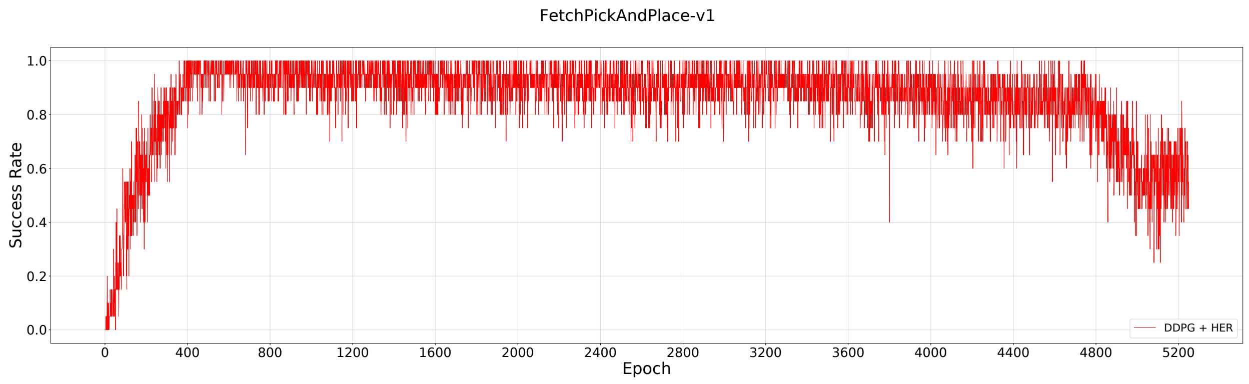


Fig. 4: Plots of the learning performance in terms of buffer data in testing phase.

- **Hypothesis 2: Data Augmentation or DDPG caused the Performance Drop**
  In those experiments met with performance drop, we used KER, HER and DDPG to train policies. Does any of these techniques cause the drop? First, we tested and observed that what happen when we do not use HER in around 20 epochs (since KER could not learn without her, we used HER until the robot starts to learn), with $n_{ker} = 16$ and $n_{ker} = 32$ through all the training. The experimental results is shown in Fig. 3, which is still met with the drop, proving that it is not HER caused the drop. Then we conducted long-run trains only with HER and DDPG, and the drops are also showed up around 4700 epochs (around 23,500,000 time steps) in Fig. 4. These experiments sufficiently proved that the drop is caused by the instability of DDPG, instead of KER nor HER (Since learning only with DDPG can never succeed, here we don't include it to help to prove this claim).

Appendix C
Deployment on the Real Robot

In this section, we show that a policy trained with ITER can be transferred to a real robot and also achieve satisfied performance.

To apply a policy control to the real Baxter robot, first we need to train the simulated one in MuJoCo[1]. The learning plots of ablative experiments on GER and KER are shown in Figs. 5 and 6, and the learned behaviors are shown in Fig. 7. These experiments show that our method still overwhelms the vanilla HER in the Baxter simulation.

[1]Our Baxter simulation code is available at https://github.com/huangjiancong1/gym_baxter.

Then we applied the well-trained policy to the real Baxter as a high-level controller through Robot Operation System (ROS). ROS provides communications between services (*e.g.* the policy) and clients (*e.g.* Baxter). We used Alvar markers[2] to detect the object position (Fig. 8) with a camera equipped in Baxter right arm end effector. To begin an episode, the observed state in real world was sent to the simulator for a virtual environment initialization, and then we keep the policy being used inside. At each timestep, the joints configuration of the simulated Baxter was sent to the real one controlled with position loop (Fig. **??**) The robot failed in picking up the object and colliding with it (Fig. 10)when there is an unexpected shift on the detected pose. Video is available in [2].

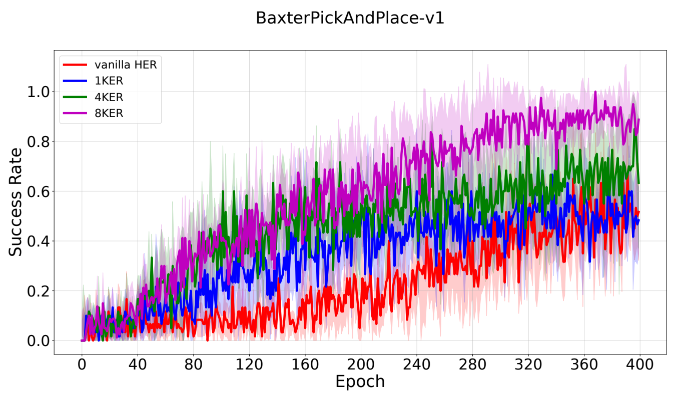


Fig. 5: Comparison of different n_{KER} for KER with a single GER on pick-and-place task with a simulated Baxter.

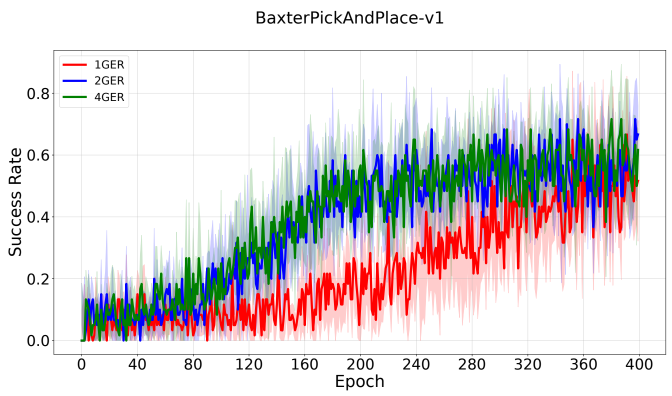


Fig. 6: Comparison of different n_{GER} for GER without KER on pick-and-place task with a simulated Baxter.

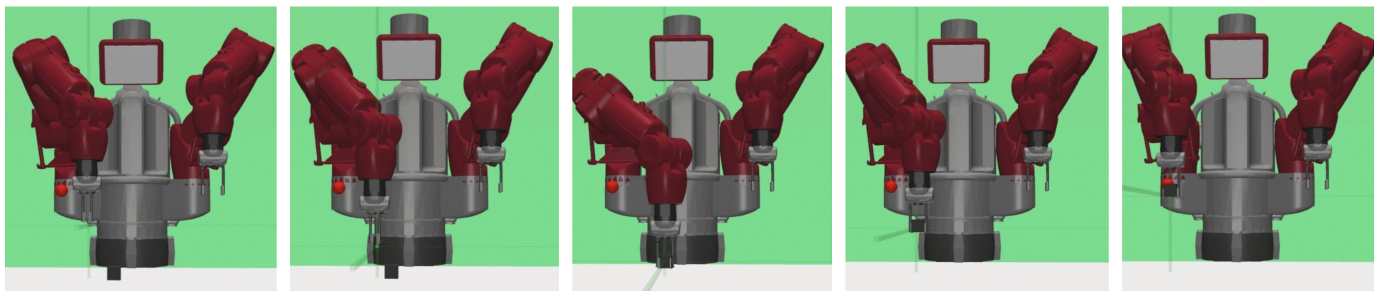
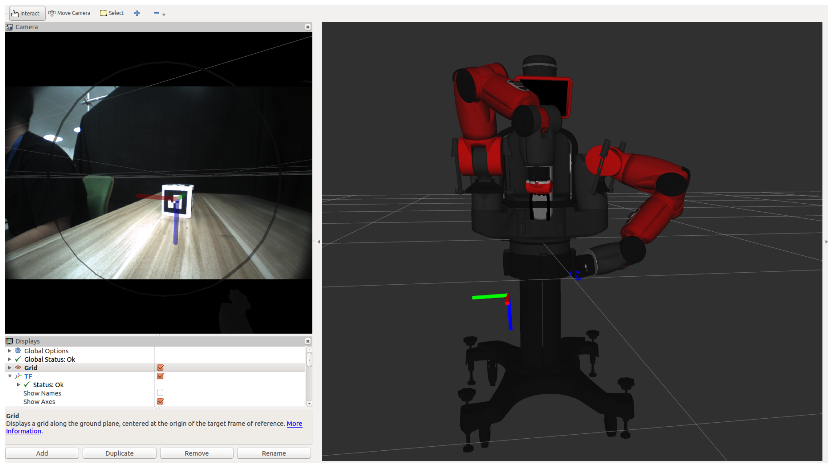


Fig. 7: Learned behaviors of simulated Baxter with ITER in pick-and-place task

Appendix D

For single-goals, our method shows 3 and 4 times speedup over HER in pushing and sliding tasks respectively (Fig. 11). Surprisingly, our method achieves a starting learning in the pick-and-place task while HER cannot resolve it within 200 epochs (all comparisons are made by measuring the number of epochs to get to convergence of success rates). The pick-and-place task in the single-goal setting is also difficult. Note that in the HER paper, HER does not learn anything without training tricks. Our approach, on the other hand, starts to learn after about 180 epochs.

[2]An open source AR tag tracking library http://wiki.ros.org/ar_track_alvar/.



Grid
Displays a grid along the ground plane, centered at the origin of the target frame of reference. More Information.

Fig. 8: Object pose detection with Alvar markers

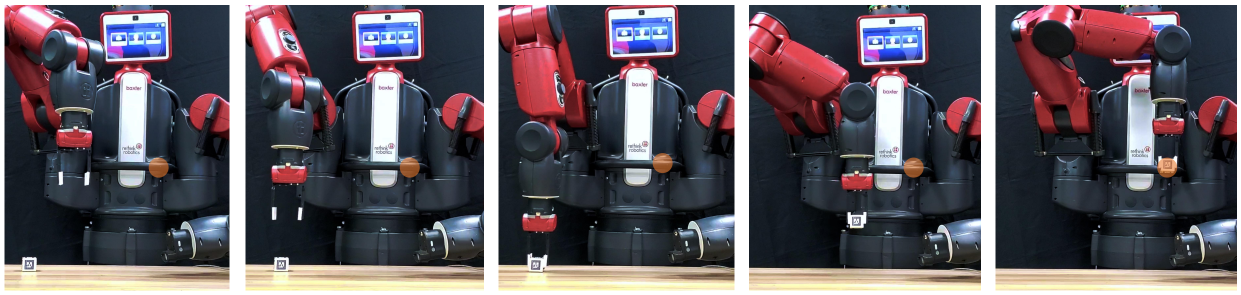


Fig. 9: Deployment on the real Baxter robot with a well-trained policy from simulation with ITER

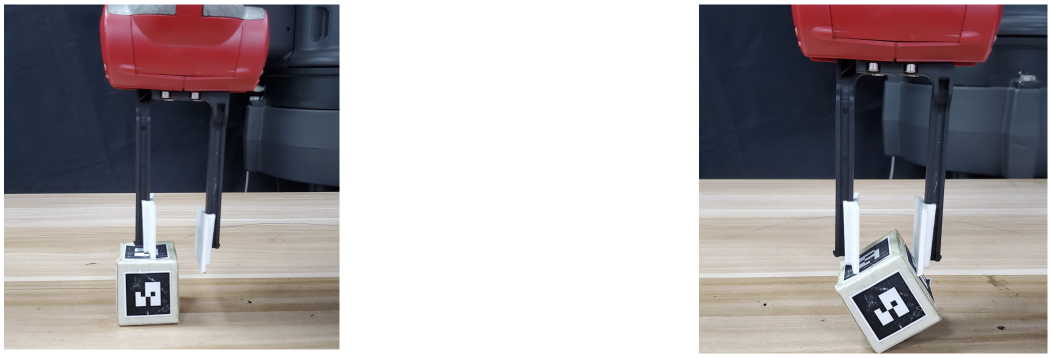


Fig. 10: Failures in the real-world pick-and-place task

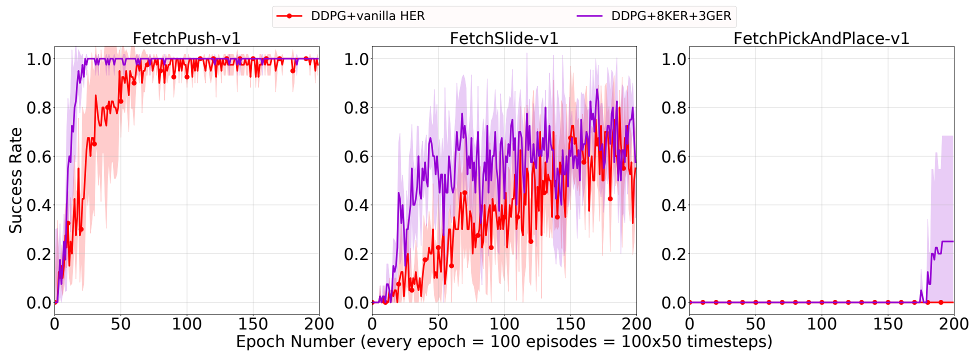


Fig. 11: Comparison of vanilla HER and ITER with 8 KER symmetries and 4 GER applications on single-goal tasks.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5049–5059.

[2] Y. Lin, J. Huang, M. Zimmer, Y. Guan, J. Rojas, and P. Weng, "Invariant transform experience replay: Data augmentation for deep reinforcement learning supplement," Tech. Rep., 2020. [Online]. Available: http://www.juanrojas.net/iter/