Benchmarking Model-Free Reinforcement Learning Algorithms for Robotic Manipulation

Ngoc Quoc Huy Hoang, Yasaman Mohammadidargah and Dmitrii Dobriborsci

***Abstract*—Reinforcement Learning (RL) is increasingly transitioning from controlled simulation environments to real-world robotic applications. However, this shift presents a major challenge: designing effective reward functions and training procedures tailored to specific robotic tasks remains time-consuming and largely empirical. In this work-in-progress study, we investigate the impact of reward shaping and algorithm selection on robotic manipulation, focusing on two foundational skills: reaching and pushing. We benchmark five RL algorithms – PPO, SAC, SAC-HER, DDPG, and DDPG-HER using a UR10 robotic arm in the PyBullet simulator. We compare sparse and dense reward formulations and analyze their influence on training dynamics and preliminary success rates. Our results demonstrate the effectiveness of hindsight-based approaches in sparse-reward scenarios and highlight the importance of proper reward design. Ongoing work includes extending the benchmark to more complex tasks such as sliding and pick-and-move, exploring hierarchical learning methods, and validating sim-to-real transfer on a physical UR10 platform.**

***Index Terms*—reinforcement learning, robotics, hyperparameter optimization, benchmarking**

I. Introduction

Reinforcement Learning (RL) has become a foundational paradigm for training robots to autonomously acquire manipulation skills through interaction with their environment [1], [2]. With growing interest in the deployment of RL beyond academic settings, there is a need to better understand how to design reward functions and choose algorithms that are generalized to real-world robotic tasks.

Despite significant progress in RL for robotics, applying these methods to practical manipulation problems remains labor intensive and highly empirical. A key challenge lies in reward shaping: defining a reward function that leads to desired behaviors, avoids unintended incentives, and accelerates learning. Sparse rewards often cause slow convergence, while dense rewards require careful manual tuning and domain-specific task knowledge. Moreover, selecting the appropriate RL algorithm for a given task and reward structure is nontrivial, especially in environments where rewards do not align easily with incremental progress. These challenges are particularly critical in industrial robotics, where trial-and-error tuning is costly and may lead to hardware risks.

In this work-in-progress study, we investigate the interaction between reward design and algorithm selection in the context of robotic manipulation. We focus on two foundational tasks: **Reach**, where the end effector must move to a target position, and **Push**, where an object must be displaced to a specified goal. These tasks represent common primitives in robotic assembly and object handling pipelines.

We benchmark five widely used model-free RL algorithms: Proximal Policy Optimization (PPO) [3], Soft Actor-Critic (SAC) [4], SAC with Hindsight Experience Replay (SAC-HER), Deep Deterministic Policy Gradient (DDPG) [5], and DDPG with HER [6] under both sparse and dense reward regimes using a UR10 robotic arm simulated in PyBullet. Preliminary results show distinct advantages of hindsight-based methods in sparse-reward settings and highlight the importance of proper reward shaping for sample-efficient training.

The goals of this study are to:

- Benchmark and compare RL algorithms for manipulation tasks with sparse and dense rewards.
- Analyze the effect of reward design on training efficiency, stability, and final performance.
- Derive practical guidelines for robotics practitioners on algorithm and reward selection.

The current results are for the Reach and Push tasks. More complex skills such as **Slide** and **Pick-and-Move** which involve longer temporal horizons and more precise contact interactions are under development.

II. Methodology

This section presents the methodological setup used to evaluate reinforcement learning algorithms for robotic manipulation. We describe the tasks, the observation and action spaces, the reward functions used, the reinforcement learning algorithms under comparison, and the hyperparameter tuning strategies applied.

A. Environment and Tasks

All experiments were performed in the PyBullet physics simulator. The simulated robot is a 6-DOF UR10 manipulator equipped with a Robotiq 2F-85 parallel gripper. The simulation scene is illustrated in Fig.1. We study four canonical manipulation tasks:

- **Reach:** Move the end-effector to a 3D target point.
- **Push:** Push a cylindrical object to a goal position on the table.
- **Slide:** Slide a cylindrical object across the table to a target.
- **Pick:** Grasp and relocate an object to a specified position in 3D.

This work is supported by Deggendorf Institute of Technology, Technology Campus Cham

All the authors are with Deggendorf Institute of Technology, Technology Campus Cham, 93413 Cham, Germany `dmitrii.dobriborsci@th-deg.de`

Each task follows a goal-conditioned interface, where the agent receives both the current and desired states.

B. Observation and Action Spaces

The observation vector includes:

- End-effector position and velocity.
- Object position and orientation (if applicable).
- Goal position.

The action space is continuous and 4-dimensional:

- `[dx, dy, dz, grip_cmd]`, where the first three components denote Cartesian motion commands for the end-effector, and the fourth controls gripper open/close.

C. Reward Functions

Each manipulation skill was evaluated with both sparse and dense (shaped) reward functions, depending on task complexity and the need for continuous feedback. Below we outline the reward formulations used for each task.

a) Reach: The goal is for the end-effector to reach the object with an accuracy of 5 cm. Two reward types were tested:

Sparse:

$$R = \begin{cases} 0, & \text{if } \|\mathbf{p}_{ee} - \mathbf{p}_{goal}\| > 0.05 \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

Dense:

$$R = -\|\mathbf{p}_{ee} - \mathbf{p}_{goal}\| \tag{2}$$

where $\mathbf{p}_{ee}$ is the position of the end-effector and $\mathbf{p}_{goal}$ is the position of the target.

b) Push: In this task, the robot must push an object to a target location. The sparse reward is:

$$R = \begin{cases} 1, & \text{if } \|\mathbf{o} - \mathbf{g}\| < 0.05 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

For dense reward, the following formulation was used:

$$R = R^d_{\text{goal}} + R^d_{\text{long}} \tag{4}$$

where:

- $R^d_{\text{goal}} = -\|\mathbf{o} - \mathbf{g}\|$ — distance between object and goal,
- R^d_{long} — penalty for overshooting the target.

c) Slide: Sliding requires maintaining contact and pushing a cylindrical object toward the goal. The reward structure includes additional shaping:

$$R = R^d_{\text{goal}} + R^d_{\text{gripper}} + R^d_{\text{long}} \tag{5}$$

where:

- $R^d_{\text{goal}} = -\|\mathbf{o} - \mathbf{g}\|$,
- $R^d_{\text{gripper}} = -\|\mathbf{o} - \mathbf{ee}\|$,
- R^d_{long} — penalty for overshooting the goal.

d) Pick and Move. Proposition.: This is the most complex task. The goal is to grasp and lift the object to a specific height above the table.

$$R = R^d_{\text{goal}} + R^d_{\text{gripper}} + R^d_{\text{long}} + R^g_{\text{success}} \tag{6}$$

with:

- $R^d_{\text{goal}} = -\|\mathbf{o} - \mathbf{g}\|$,
- $R^d_{\text{gripper}} = -\|\mathbf{o} - \mathbf{ee}\|$,
- $R^d_{\text{long}} = -100$ — penalty if object flies past the goal,
- $R^g_{\text{success}} = +1000$ — reward for successful grasp and lift.

Due to the complexity of the Pick and Move task, we observed low learning stability and sample efficiency, motivating future work using hierarchical learning approaches.

D. Reinforcement Learning Algorithms

We selected a diverse set of widely used model-free RL algorithms to investigate their performance in a range of robotic manipulation tasks. Each method was chosen to represent a different family of approaches: on-policy vs. off-policy, stochastic vs. deterministic policies, and with or without Hindsight Experience Replay (HER). The primary evaluation criteria were sample efficiency, performance under sparse rewards, and training stability.

- **Proximal Policy Optimization (PPO)** [3]: PPO is a robust and widely adopted actor-critic algorithm on-policy that relies on clipped policy updates to ensure training stability. Although it is less sample-efficient compared to off-policy methods, PPO is relatively easy to tune and has been used successfully in many continuous control tasks.
- **Soft Actor-Critic (SAC)** [4]: SAC is an off-policy algorithm that maximizes a trade-off between expected return and policy entropy, encouraging exploration, and improving robustness. It uses stochastic policies and twin Q-functions, which help mitigate overestimation bias.
- **SAC with Hindsight Experience Replay (SAC-HER)**: By integrating SAC with HER, we improve learning efficiency in sparse-reward environments. HER relabels failed trajectories by substituting goals with outcomes the agent actually achieved, thus converting failed attempts into informative training signals.
- **Deep Deterministic Policy Gradient (DDPG)** [5]: DDPG is an off-policy algorithm that uses deterministic policies for efficient action learning in continuous domains. It incorporates experience replay and target networks, and is known to be sensitive to hyperparameters and exploration noise.
- **DDPG with Hindsight Experience Replay (DDPG-HER)** [6]: Similar to SAC-HER, the integration of HER with DDPG dramatically improves performance in environments with sparse binary rewards.

These algorithms were implemented using the Stable-Baselines3 library. All agents were trained on the same task definitions and environments to ensure fair comparisons. We

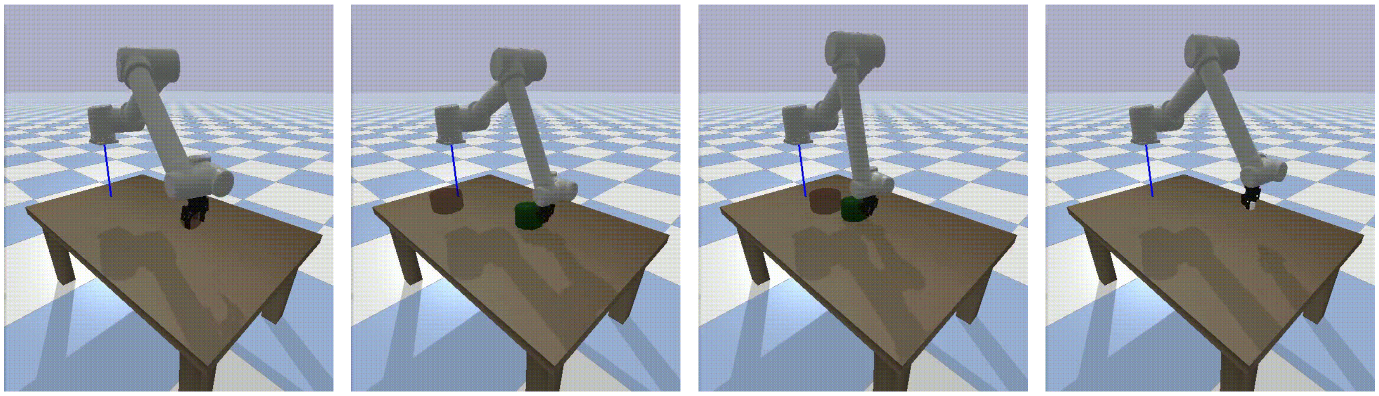

(a) Reach (b) Push (c) Slide (d) Pick and Move

Fig. 1: Simulation environments for each manipulation task: (a) Reach, (b) Push, (c) Slide, and (d) Pick and Place. Each scene is implemented using PyBullet and features a UR10 robotic arm with a Robotiq 2F-85 gripper.

also conducted ablations on reward types (sparse vs. dense) to study their interaction with algorithm performance.

E. Hyperparameters

All algorithms were trained for 1 million timesteps per task. The following settings were used unless otherwise stated:

- **Learning rate:** 0.001 (DDPG/SAC), 0.0003 (PPO)
- **Batch size:** 256
- **Buffer size:** 10^6 (for off-policy methods)
- **Gamma:** 0.98
- **Tau (Polyak update):** 0.005
- **Target update frequency:** every 2 episodes
- **Policy network:** Two hidden layers with 256 units each (ReLU)

Hyperparameters were tuned manually and via grid search for key cases, ensuring fair comparison across algorithms and reward settings.

F. Evaluation Metrics

To evaluate algorithm performance across reward structures and tasks, we report three key metrics:

- **Success Rate:** Percentage of evaluation episodes where the agent reaches the goal within a set threshold (e.g., 5 cm for Reach, 1 cm for Push). This reflects the reliability of task execution and is the primary indicator of task success.
- **Average Episode Reward:** Mean cumulative reward per episode. While sensitive to reward shaping and less correlated with success under sparse rewards, it offers insight into training progress and policy behavior.
- **Sample Efficiency (planned):** Defined as the number of training steps required to reach a target success rate (e.g., 90%). This is especially relevant in robotics, where data collection is expensive.
- **Learning Stability (planned):** Measures variance in performance across random seeds. Algorithms with low variance are more reliable for deployment.
- **Training Time (planned):** Tracks wall-clock time required per algorithm. Though not central in this study, it is important for assessing real-world feasibility.

Each policy was evaluated over 100 episodes with randomized initial and goal configurations. Evaluations were run in deterministic mode to ensure reproducibility. Metrics were logged and visualized using TensorBoard.

G. Hyperparameter Configuration

Hyperparameters were optimized using `Optuna` [7]. Below we list the configurations used in training:

TABLE I: PPO Hyperparameters

Parameter	**PPO (dense)**	**PPO (sparse)**
Batch size	8	256
Policy	PPO	PPO
Policy kwargs	{'activation_fn': ReLU, 'net_arch': {'pi': [32, 32], 'vf': [32, 32]}}	{'activation_fn': ReLU, 'net_arch': {'pi': [32, 32], 'vf': [32, 32]}}
γ	0.95	0.95
Learning rate	8.5e-5	3e-3
Device	auto	auto
n_steps	256	2048
vf_coef	0.3531	0.5
clip_range	0.4	0.2
GAE λ	0.99	0.99
n_epochs	10	10
Max grad norm	2	2
Ent coef	4.5e-8	0

H. Software Framework

All experiments were conducted in the PyBullet physics simulator using custom Python-based environments for each robotic skill. We developed task-specific simulation scenes for a UR10 6-DOF robotic arm equipped with a Robotiq 2F-85 parallel gripper. These environments were designed to reflect realistic physical constraints and sensing capabilities.

We used the **Stable-Baselines3** [8] library for all RL implementations. This framework provides a unified interface for state-of-the-art algorithms such as PPO, SAC, and DDPG,

TABLE II: Hyperparameters for DDPG, SAC, and SAC-HER

Parameter	**DDPG**	**SAC**	**SAC-HER**
Reward type	dense	dense	dense
Batch size	100	256	256
Buffer size	200k	500k	1M
Gradient steps	1	1	1
Policy	DDPG	SAC	SAC
Policy kwargs	{'net_arch': [400, 200], 'n_critics': 1}	{'log_std_init': -2, 'net_arch': [256, 256], 'use_sde': False}	{'net_arch': [256, 256], 'activation_fn': ReLU}
γ	0.98	0.95	0.99
Learning rate	1e-3	3e-4	3e-4
Learning starts	10k	50k	1k
Device	auto	auto	auto
Train freq	1	1	1
Ent coef	NaN	auto	auto
τ	0.005	0.005	0.005
Replay buffer kwargs	–	–	{'n_sampled_goal': 4, 'goal_selection_strategy': 'future'}
Replay buffer class	–	–	HerReplayBuffer

and supports extensions such as Hindsight Experience Replay (HER) through its `HerReplayBuffer` module.

All training was performed using vectorized environments to accelerate data collection and policy updates. Environments were run with fixed seeds and deterministic evaluation episodes for reliable benchmarking.

III. Preliminary Results

This section presents the initial benchmarking results for the **Reach** and **Push** skills, comparing five reinforcement learning algorithms (PPO, SAC, SAC-HER, DDPG, DDPG-HER) under both sparse and dense reward conditions. All experiments were run across five random seeds to ensure statistical reliability.

A. Reach Skill

For the **Reach** task, both sparse and dense reward formulations were evaluated. In the case of PPO, performance under both reward types was comparable in terms of final success rate. However, the same hyperparameter configuration did not generalize well across reward types. Specifically, PPO with sparse rewards required additional tuning to reach optimal performance.

SAC and DDPG, in their vanilla forms, achieved relatively poor results, with success rates not exceeding 40% on average. SAC-HER showed improved convergence and final performance compared to SAC, confirming the benefits of hindsight relabeling in goal-conditioned settings. Nonetheless, PPO significantly outperformed all other methods in this task, achieving the highest success rate and fastest convergence.

B. Push Skill

In contrast, the **Push** skill revealed a clear advantage for HER-based methods. SAC-HER consistently achieved success rates of approximately 95%, outperforming all other algorithms by a large margin. The observed variance in transition dynamics is attributed to differences in random seed initialization and will be further investigated in future experiments.

Other algorithms, including PPO, SAC, and DDPG, failed to reach success rates above 40%, consistent with findings reported in prior work. These results underscore the importance of HER in sparse-reward tasks that involve contact-rich manipulation and delayed feedback.

Further improvements are anticipated through continued hyperparameter tuning, curriculum learning strategies, and better initialization procedures.

IV. Insights

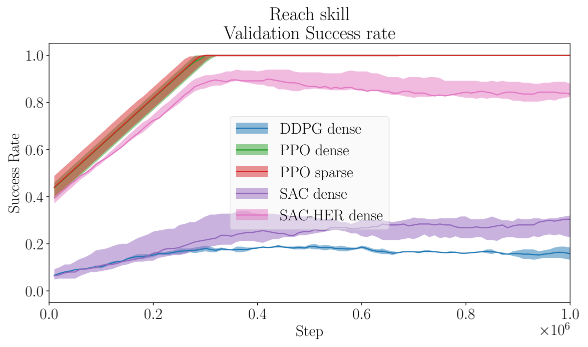
The experiments highlight important distinctions in how different RL algorithms respond to sparse vs. dense reward formulations across manipulation tasks.

For the **Reach** skill, PPO outperforms other methods in both convergence speed and final success rate. This can be attributed to its stable on-policy updates and relatively smooth reward landscape in this task, which aligns well with advantage-based gradient estimation. The Reach task is geometrically simple and does not require exploration of contact-based interactions, making PPO's exploration strategy sufficient.

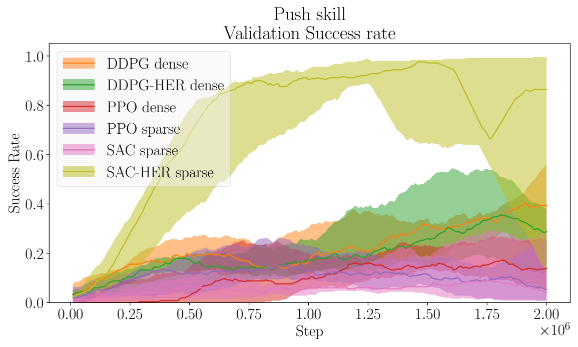
In contrast, tasks involving object manipulation—such as **Push**—present significant learning challenges due to sparse success signals and delayed rewards. In these cases, PPO and vanilla SAC struggle to discover useful behaviors, particularly under sparse rewards. SAC's entropy regularization alone is insufficient to guide the agent toward the rare reward events.

The effectiveness of **SAC-HER** in the Push skill illustrates the power of goal relabeling for credit assignment in sparse-reward settings. By converting failed episodes into pseudo-successful experiences, HER significantly increases the density of training signals without changing the environment. This accelerates learning and improves robustness, especially during early exploration.

Interestingly, the performance of SAC-HER also shows sensitivity to initial conditions and seed variance. This suggests a need for more adaptive goal sampling strategies and potentially curriculum-based shaping to stabilize training further.



(a) Reach skill



(b) Push skill

Fig. 2: Success rate comparison for the Skill Reach (a) and Skill Push (b) tasks. Shaded regions indicate 95% confidence intervals over five random seeds.

Overall, our findings reaffirm the importance of matching algorithmic properties to task demands. On-policy methods like PPO perform well in tasks with dense, smooth rewards. Off-policy methods combined with HER are better suited for sparse, goal-driven tasks involving contact and delay.

Future work will explore whether these findings hold for more complex tasks involving visual feedback and multi-stage manipulation using hierarchical learning architectures.

Conclusion

This work-in-progress study benchmarks five reinforcement learning algorithms across two foundational robotic manipulation skills: Reach and Push. The Reach skill has been successfully learned using both sparse and dense reward formulations, validating our simulation setup, training pipeline, and basic algorithmic configurations.

In contrast, the Push skill—despite showing promising results with SAC-HER—highlights the limitations of current reward-based state representations when contact-rich object manipulation is required. More complex tasks such as Slide and Pick remain under active development and will require advanced techniques to be mastered effectively.

Based on our observations, several key directions emerge for future research:

- **Extended training and scaling:** skills involving interaction with objects typically require longer training, often exceeding 4 million timesteps, due to sparse feedback and high task variability.
- **Systematic hyperparameter tuning:** performance is highly sensitive to reward scaling, exploration noise, and HER configurations. Automated tuning (e.g., Bayesian optimization) will be essential for stable learning.
- **Informed environment initialization:** kinematically-aware goal and object sampling can reduce ineffective roll-outs and accelerate convergence.
- **Hierarchical and curriculum learning:** for long-horizon tasks like Pick-and-Move, decomposing the task into sequential subgoals and gradually increasing complexity will be critical.
- **Vision-based feedback:** for tasks where the agent must manipulate or track objects (e.g., Push, Slide, Pick), relying solely on proprioceptive and positional input is insufficient. We plan to integrate camera-based perception and train policies with visual state representations to enable generalization to novel object poses and environmental conditions.
- **Sim-to-real transfer:** we aim to deploy trained policies on a physical UR10/Ur3e platform, closing the simulation-to-reality gap and validating the transferability of shaped reward functions and learned control strategies.

These challenges and opportunities define the roadmap for future development. The ultimate goal is to develop a generalizable and sample-efficient learning pipeline for real-world robotic manipulation driven by reinforcement learning.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

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

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[5] 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," in *arXiv preprint arXiv:1509.02971*, 2015.

[6] 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*, vol. 30, 2017.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[8] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," https://github.com/DLR-RM/stable-baselines3, 2021, accessed: 2024-04-07.