Sparse-Reward Reinforcement Learning for Robotic Manipulation

MAE 547 Course Project

Runyang Li
Supervisor: Prof. Wanxin Jin

December 7, 2025

Abstract

Robotic manipulation with even relatively simple parallel grippers remains challenging due to contact-rich dynamics, model uncertainty, and high-dimensional state and action spaces. Reinforcement learning (RL) provides a data-driven way to synthesize control policies directly from interaction, but its practical success critically depends on the design of reward functions. In many manipulation tasks, feedback is naturally sparse and binary, whereas dense rewards require extensive manual shaping and are prone to reward exploitation and reward misspecification. This report investigates a sparse-reward reinforcement learning framework for a Franka Panda3 arm push task implemented in MuJoCo simulation enviornment. Specifically, We focus on a goal-conditioned, multi-goal RL setting with binary success/failure rewards and compare three off-policy actor-critic algorithms: Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), and Truncated Quantile Critics (TQC). We formalize the sparse-reward objective, derive the update rules for each algorithm, and evaluate their performance on the Franka Push object task. The results show that DDPG suffers from Q-value overestimation and poor exploration, leading to low success rates, while SAC and especially TQC achieve near-perfect performance with stable convergence, highlighting the advantages of entropy-regularized and distributional off-policy methods in sparse-reward robotic manipulation.

1 Introduction

Robotic manipulation is a central capability for autonomous systems in industrial, service, logistics, and household scenarios. Even when using a simple parallel gripper rather than a fully dexterous hand, tasks such as grasping, pushing, object reorientation, and precise placement involve complex contact interactions, non-smooth dynamics, and significant model uncertainties. These characteristics challenge classical model-based and heuristic control approaches, which typically require accurate system identification, task-specific tuning, and careful handling of contact events.

Reinforcement learning (RL) offers an appealing alternative paradigm by framing control as sequential decision-making in a Markov decision process (MDP). An agent learns a policy that maps state information to actions so as to maximize an expected cumulative reward. For robotic manipulation, this enables directly optimizing the policy from data collected in simulation or on real hardware, potentially bypassing the need for accurate analytic models. Recent work has demonstrated promising results on various manipulation tasks using RL and sim-to-real transfer techniques.

However, the effectiveness of RL in practice is heavily influenced by the reward design. In real-world robotic tasks, feedback is often naturally expressed as a binary success/failure signal:

an insertion either succeeds or fails; an object is either placed within a tolerance of the target pose or not. While dense rewards—e.g., negative distances, velocity penalties, or shaped intermediate objectives—are frequently introduced to accelerate learning, they come at several costs: (i) they require substantial manual engineering and domain knowledge, (ii) they can be misaligned with the actual task objective, and (iii) they are vulnerable to reward hacking, where the learned policy exploits loopholes in the reward instead of truly solving the task. As a result, reward engineering can become more difficult than the learning problem itself.

Sparse rewards provide a conceptually simpler and more natural formulation for many goal-oriented manipulation tasks. A sparse reward can be defined as *success yields zero (or positive) reward and all other outcomes yield a negative reward*. In goal-conditioned RL and multi-goal RL frameworks, this typically corresponds to rewarding the agent only when the achieved goal lies within a prescribed tolerance of the desired goal. Although such rewards are challenging from a credit-assignment and exploration perspective, techniques such as Hindsight Experience Replay (HER) together with sample-efficient off-policy algorithms have shown that sparse-reward RL can be both practical and competitive with heavily shaped dense rewards.

In this course project, we study sparse-reward RL for a representative robotic manipulation task: pushing an object on a table to a desired target location with a Franka Emika Panda arm in MuJoCo. We adopt an open-source environment built on MuJoCo Menagerie and the Gymnasium Robotics API, which follows the Multi-Goal RL design with goal-conditioned observations and both sparse and dense reward options. We deliberately focus on the sparse binary reward formulation in order to avoid dense reward engineering and to better reflect realistic robot feedback.

Our contributions in this report are threefold:

- We formalize the Franka push task as a goal-conditioned MDP with a sparse binary reward and specify the corresponding observation, action, and goal spaces.
- We present detailed formulations of three off-policy RL algorithms—DDPG, SAC, and TQC—including their critic and actor objectives in the sparse-reward, multi-goal setting.
- We empirically compare the algorithms on the FrankaPush environment and analyze why TQC and SAC succeed while DDPG fails, from the perspectives of value overestimation, entropy-regularized exploration, and distributional critics.

2 Related Work

RL for robotic manipulation has been studied in a variety of settings, including grasping, pushing, insertion, and assembly tasks, often leveraging high-fidelity physics simulators for data collection and policy training. Sim-to-real methods have shown that policies trained entirely in simulation can transfer to real hardware when appropriate domain randomization and robust control techniques are used.

Multi-goal RL and goal-conditioned policies enable reusing learned behaviors across different target configurations by conditioning the policy and value functions on the desired goal. Hindsight Experience Replay (HER) dramatically improves learning under sparse rewards by relabeling unsuccessful episodes with alternative, achieved goals so that more transitions become informative training data.

On the algorithmic side, off-policy actor-critic methods such as DDPG and SAC have become standard for continuous control. DDPG learns a deterministic policy and a single Q-function, but is known to suffer from Q-value overestimation and instability. SAC addresses several of these

issues by employing twin Q-networks with clipped double-Q learning and optimizing a maximum-entropy objective to promote exploration. Building on SAC, TQC introduces a distributional critic with truncated quantile mixing to further control overestimation bias and improve stability on challenging benchmarks.

3 Reinforcement Learning Formulation with Sparse Rewards

We first formalize the Franka push task as an MDP and specify the sparse-reward structure.

3.1 Goal-Conditioned MDP

We consider a finite-horizon, goal-conditioned MDP defined by the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{G}, p, r, \gamma, \rho_0),$$

where:

- $\mathcal{S}$ is the state space;
- $\mathcal{A}$ is the continuous action space;
- $\mathcal{G}$ is the goal space (desired object positions on the table);
- $p(s_{t+1} \mid s_t, a_t)$ is the transition probability induced by the MuJoCo simulator;
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$ is the reward function;
- $\gamma \in [0, 1]$ is the discount factor;
- ρ_0 is the distribution over initial states and goals.

At each time step t, the environment provides a goal-conditioned observation consisting of:

- $o_t \in \mathbb{R}^{d_o}$: the low-level state observation of the robot and the object;
- $g_t^{\text{ach}} \in \mathbb{R}^3$: the achieved goal, i.e., the current object position;
- $g^{\text{des}} \in \mathbb{R}^3$: the desired goal, i.e., the target position on the table.

The agent receives $(o_t, g_t^{\text{ach}}, g^{\text{des}})$, chooses an action $a_t \in \mathcal{A}$, and transitions to s_{t+1}, with a corresponding next observation and goal information.

A stochastic policy $\pi_\theta(a_t \mid o_t, g^{\text{des}})$ with parameters θ aims to maximize the expected return

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, g^{\text{des}})\right], \tag{1}$$

where the expectation is over the initial state distribution, the environment dynamics, and the policy.

3.2 Sparse and Dense Reward Functions

Let g_t^{ach} denote the object position at time t, and let g^{des} denote the desired goal. We define the distance

$$d_t = \left\| g_t^{\text{ach}} - g^{\text{des}} \right\|_2, \tag{2}$$

and a threshold $\epsilon > 0$ that specifies the radius of the target region.

The *sparse binary reward* used in this project is given by

$$r_{\text{sparse}}(s_t, a_t, g^{\text{des}}) = \begin{cases} 0, & \text{if } d_t \leq \epsilon, \\ -1, & \text{otherwise.} \end{cases} \tag{3}$$

An episode is deemed successful if $d_t \leq \epsilon$ for some t within the horizon T.

For comparison, the environment also implements a *dense* distance-based reward of the form

$$r_{\text{dense}}(s_t, a_t, g^{\text{des}}) = -d_t, \tag{4}$$

or a scaled variant thereof. In this report, we deliberately focus on the sparse reward (3), reflecting the natural success/failure structure of the task and avoiding potential misalignment introduced by dense reward shaping.

3.3 Hindsight Experience Replay

To mitigate the exploration difficulties introduced by sparse rewards, we employ Hindsight Experience Replay (HER). During training, for each collected episode with an original desired goal g^{des}, we additionally generate relabeled transitions by replacing g^{des} with alternative goals sampled from the set of achieved goals $\{g_t^{\text{ach}}\}$ in the same episode. For a relabeled goal $\tilde{g}^{\text{des}}$, the reward is recomputed according to (3), thus turning previously unsuccessful episodes into successful ones for the new goal. This significantly increases the fraction of informative transitions in the replay buffer.

4 Environment: Franka Push Task

4.1 Simulation Setup

The Franka push task, referred to as `FrankaPush`, is based on a Franka Emika Panda arm model from MuJoCo Menagerie and implemented using the MuJoCo physics engine and the Gymnasium Robotics API. The task requires the robot to push a cuboid object lying on a table to a target location on the tabletop. At the start of each episode, the robot arm is reset to a nominal configuration, and the initial object and target positions are randomized within specified bounds on the table surface.

The Franka arm has 7 revolute joints; the parallel gripper is present but locked for this task, so there is no gripper control. A mocap body is attached to the end-effector to maintain a fixed orientation, and the underlying joint angles are solved implicitly via inverse kinematics. The simulation timestep is set to 2 ms to balance accuracy and computational efficiency.

4.2 Observation and Action Spaces

The low-level observation o_t contains:

- The Cartesian position and linear velocity of the end-effector, $x_e, v_e \in \mathbb{R}^3$;

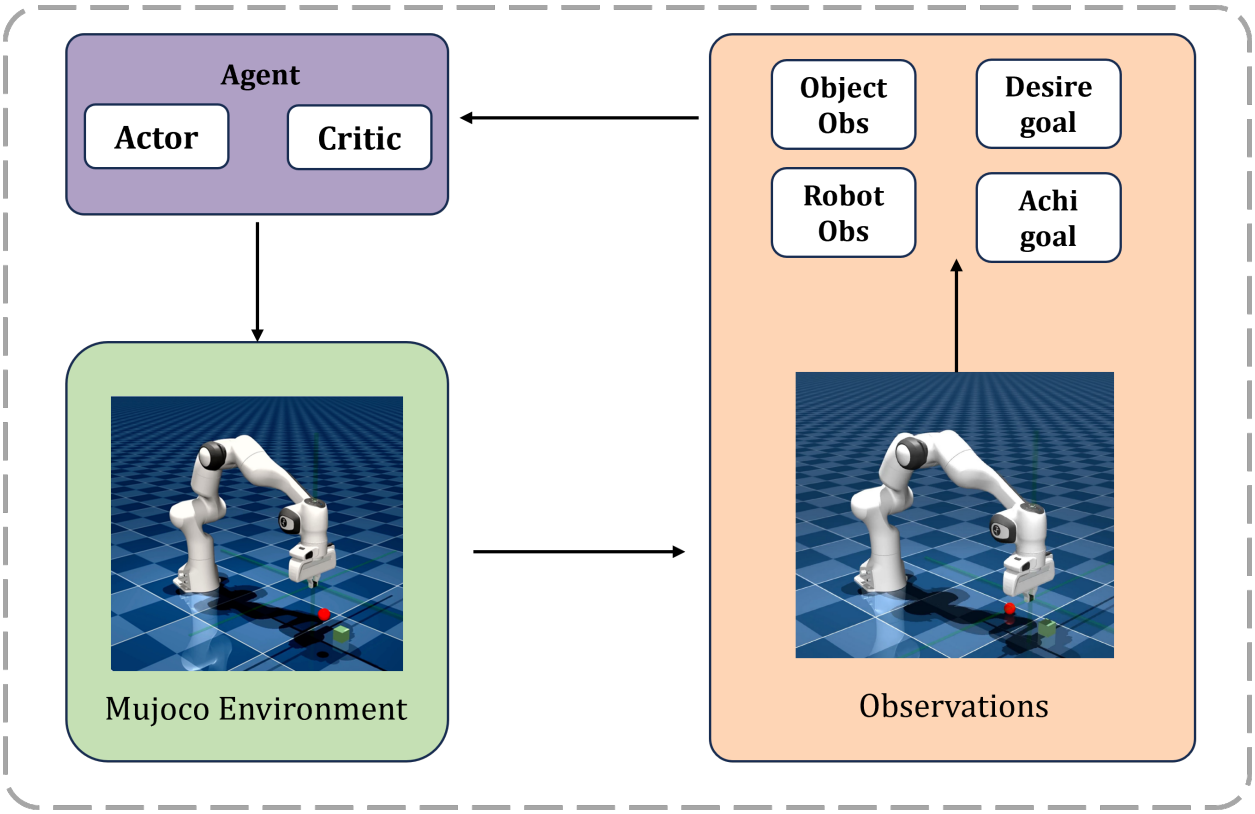

Figure 1: Simulation enviornment based on Mujoco.

- The Cartesian position, orientation (Euler angles), linear velocity, and angular velocity of the object, $x_b, e_b, v_b, w_b \in \mathbb{R}^3$.

Combined, these yield an 18-dimensional observation vector for the push task, excluding the goal fields.

The action space is 3-dimensional and corresponds to Cartesian displacements or velocity commands for the end-effector position in x, y, and z. The gripper remains fixed for the push task.

The environment returns a dictionary observation with keys `observation`, `achieved_goal`, and `desired_goal`, where `achieved_goal` and `desired_goal` are three-dimensional vectors representing g_t^{ach} and g^{des} respectively.

5 Off-Policy RL Algorithms

We now describe the three off-policy RL algorithms considered in this report: DDPG, SAC, and TQC. All algorithms are trained in an off-policy manner using a replay buffer that stores goal-conditioned transitions augmented by HER.

5.1 Deep Deterministic Policy Gradient (DDPG)

DDPG learns a deterministic policy $\mu_\theta(s)$ and a single critic $Q_\phi(s, a)$ for continuous control. In the goal-conditioned setting, we write $s = (o, g^{\text{des}})$. The critic is trained by minimizing the mean-

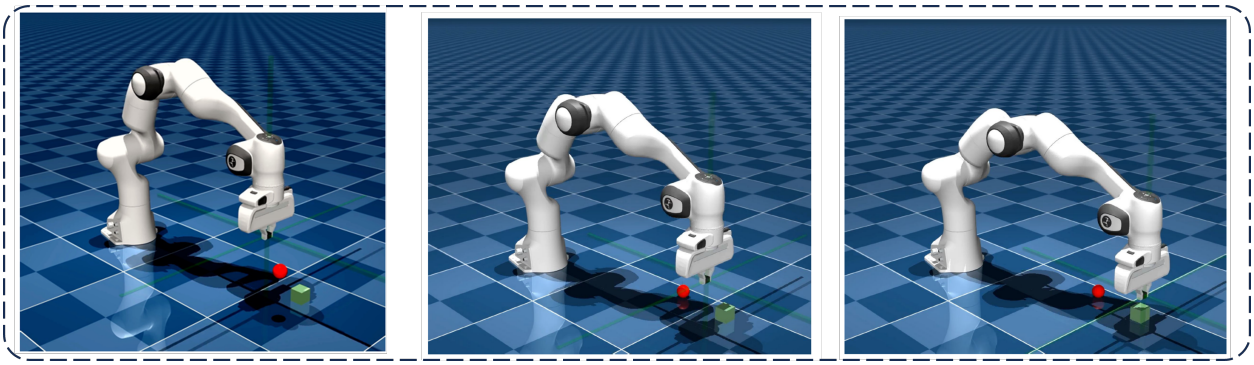

Figure 2: Push task procedure.

squared Bellman error:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},d_t)\sim\mathcal{D}}\Big[\big(Q_\phi(s_t,a_t) - y_t\big)^2\Big], \tag{5}$$

where the target y_t is given by

$$y_t = r_t + \gamma(1-d_t)\, Q_{\bar{\phi}}(s_{t+1}, \mu_{\bar{\theta}}(s_{t+1})). \tag{6}$$

Here $\bar{\phi}$ and $\bar{\theta}$ denote target network parameters updated by Polyak averaging. The termination flag $d_t \in \{0,1\}$ indicates whether the episode has ended.

The actor is updated to maximize the critic's Q-value:

$$J_\pi(\theta) = -\mathbb{E}_{s_t\sim\mathcal{D}}\big[Q_\phi(s_t, \mu_\theta(s_t))\big], \tag{7}$$

which is implemented as gradient ascent on Q_ϕ with respect to θ. During exploration, Gaussian noise is added to the deterministic policy output:

$$a_t = \mu_\theta(s_t) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2 I). \tag{8}$$

Despite its simplicity, DDPG is known to suffer from Q-value overestimation due to using a single critic in the target, which can lead to optimistic value estimates, unstable learning, and convergence to poor policies—especially under sparse rewards and complex contact dynamics.

5.2 Soft Actor-Critic (SAC)

SAC extends the actor-critic framework by introducing a maximum-entropy objective that encourages stochastic policies with high entropy. SAC maintains two critics Q_{ϕ_1} and Q_{ϕ_2} and a stochastic policy $\pi_\theta(a \mid s)$. The soft Q-learning objective for each critic is

$$J_Q(\phi_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},d_t)\sim\mathcal{D},\, a_{t+1}\sim\pi_\theta}\Big[\big(Q_{\phi_i}(s_t,a_t) - y_t\big)^2\Big], \tag{9}$$

with target

$$y_t = r_t + \gamma(1-d_t)\Big(\min_{j=1,2} Q_{\bar{\phi}_j}(s_{t+1},a_{t+1}) - \alpha \log \pi_\theta(a_{t+1} \mid s_{t+1})\Big), \tag{10}$$

where $\alpha > 0$ is the temperature parameter controlling the trade-off between return maximization and entropy. The use of the minimum of the two target critics implements clipped double-Q learning, which effectively reduces overestimation bias.

The actor is optimized to maximize the expected soft Q-value plus entropy:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D},\, a_t \sim \pi_\theta}\Big[\alpha \log \pi_\theta(a_t \mid s_t) - \min_{i=1,2} Q_{\phi_i}(s_t, a_t)\Big]. \tag{11}$$

In practice, actions are sampled via a reparameterization trick, e.g., a squashed Gaussian.

The temperature α can be fixed or adapted automatically by optimizing a separate objective that enforces a target entropy. In sparse-reward tasks like FrankaPush, entropy regularization improves exploration, making it more likely for the agent to discover successful trajectories that yield non-negative rewards.

5.3 Truncated Quantile Critics (TQC)

TQC builds on SAC but replaces the scalar critics with distributional critics that approximate the full return distribution using quantile regression. For each critic $n \in \{1, \ldots, N\}$, TQC represents the return distribution $Z_{\phi_n}(s, a)$ as a uniform mixture of M Dirac atoms:

$$Z_{\phi_n}(s, a) = \frac{1}{M} \sum_{m=1}^{M} \delta_{\theta^m_{\phi_n}(s,a)}, \tag{12}$$

where $\theta^m_{\phi_n}(s, a)$ denotes the m-th quantile value predicted by critic n.

For a transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ and a next action $a_{t+1} \sim \pi_\theta(\cdot \mid s_{t+1})$, all target atoms from all critics are pooled:

$$\mathcal{Z}_{t+1} = \left\{ \theta^m_{\phi_n}(s_{t+1}, a_{t+1}) : n = 1, \ldots, N,\ m = 1, \ldots, M \right\}.$$

These NM atoms are sorted, and the largest jN atoms are truncated, leaving kN atoms with $k = M - j$. The truncated set is denoted by $\{z_i\}_{i=1}^{kN}$.

The target distribution for critic n is then defined by

$$y_i = r_t + \gamma(1 - d_t)\Big(z_i - \alpha \log \pi_\theta(a_{t+1} \mid s_{t+1})\Big), \quad i = 1, \ldots, kN. \tag{13}$$

Each critic's quantile regression loss is computed using the Huber quantile loss between the predicted quantiles $\theta^m_{\phi_n}(s_t, a_t)$ and the target atoms $\{y_i\}$.

The actor objective in TQC is similar to that of SAC, but uses the mean of all (non-truncated) quantile estimates:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D},\, a_t \sim \pi_\theta}\left[\alpha \log \pi_\theta(a_t \mid s_t) - \frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} \theta^m_{\phi_n}(s_t, a_t)\right]. \tag{14}$$

By truncating the highest target quantiles, TQC provides a more conservative value estimate, further reducing overestimation relative to SAC and leading to improved stability on challenging tasks.

6 Experimental Setup

6.1 Training Protocol

We follow the hyperparameter settings reported for the FrankaPush environment. Key parameters are summarized in Table 1.

Table 1: Key hyperparameters for the Franka push task.

Parameter	Value
Action dimension	3
Observation dimension	18
Network architecture (actor/critic)	MLP with [256, 256, 256] units
Batch size	512
Replay buffer size	10^6 transitions
Optimizer	Adam
Action noise (DDPG, SAC)	$\mathcal{N}(0, 0.2^2 I)$
Learning rate	1×10^{-3}
Polyak update coefficient	0.05
Discount factor γ	0.95
Training steps (Push)	5×10^5
Episode length	50 steps
Evaluation interval	every 2,000 steps
Reward type	Sparse (0 / -1)
HER strategy	Future relabeling, 4 HER samples per transition
Number of critics (TQC)	2
Number of quantiles per critic (TQC)	25
Quantiles dropped per critic (TQC)	2
Entropy coefficient (SAC, TQC)	auto-tuned

At the start of each episode, the arm is reset, and the object and target positions are sampled within predefined ranges. Each episode terminates either when the time horizon of 50 steps is reached or when the object enters the goal region ($d_t \leq \epsilon$). Transitions are stored in a replay buffer together with HER-relabeled versions.

6.2 Implementation Details

All three algorithms share a common implementation backbone with identical network architectures and optimizer settings, unless required by the algorithm (e.g., number of critics, quantiles). This ensures a fair comparison, isolating algorithmic differences rather than implementation artifacts.

Training is performed on a workstation equipped with an NVIDIA RTX 3070Ti GPU and an Intel i7-12700K CPU. The MuJoCo-based FrankaPush environment is computationally lightweight and can run effectively on CPU, while GPU acceleration is mainly used for neural network training.

7 Results and Discussion

7.1 Quantitative Results on the Push Task

The success rates of the three algorithms on the FrankaPush task are summarized in Table 2. Success is defined as the object reaching the target region within the episode horizon.

DDPG exhibits very poor performance, with a success rate around 10%. In contrast, both SAC and TQC reliably solve the push task, achieving near 100% success under the sparse reward formulation.

Table 2: Success rates (%) on the Franka push task for different algorithms.

Algorithm	DDPG	SAC	TQC
Success rate (%)	11.7 ± 0.3	100.0 ± 0.0	100.0 ± 0.0

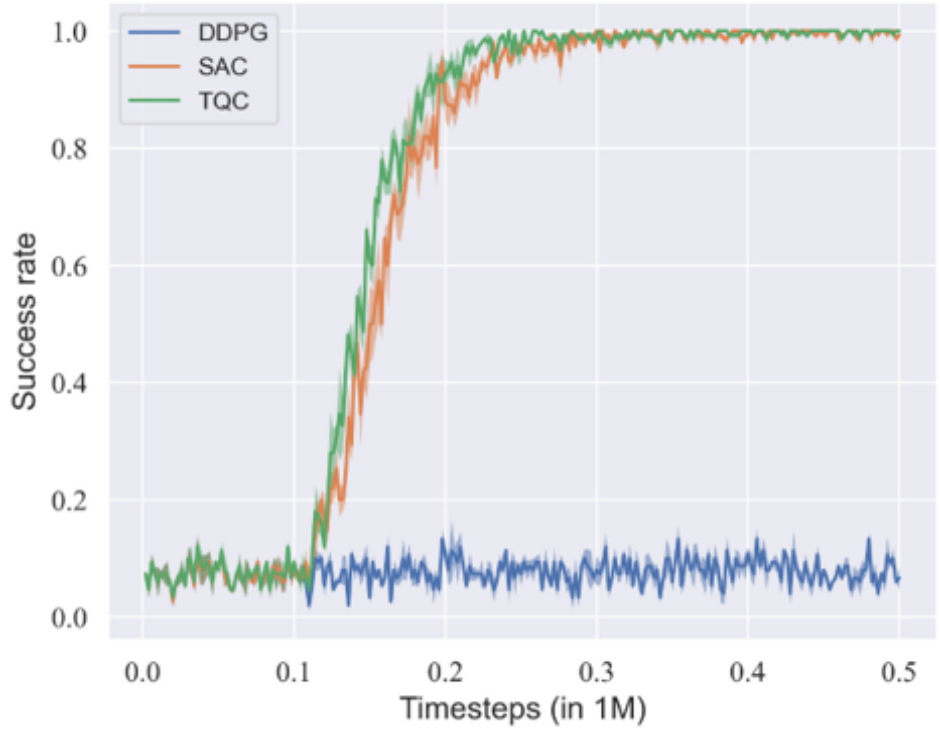

Figure 3: Learning curves of success rate for DDPG, SAC, and TQC on the FrankaPush task.

7.2 Why Do the Algorithms Perform Differently?

DDPG. DDPG's reliance on a single critic in the Bellman target makes it highly susceptible to Q-value overestimation. In a sparse-reward environment, where informative (non-negative) rewards are rare, overestimated Q-values can lead to overly optimistic policies that do not correspond to truly successful trajectories. Moreover, exploration in DDPG is driven solely by Gaussian noise added to a deterministic policy, which is often insufficient to discover the narrow set of successful pushing behaviors in a complex contact dynamics setting. As a result, DDPG frequently converges to suboptimal behaviors and fails to push the object reliably to the goal.

SAC. SAC addresses two of DDPG's main weaknesses. First, the use of twin critics and clipped double-Q learning substantially reduces Q-value overestimation. Second, the maximum-entropy objective explicitly encourages stochastic policies with high entropy, which greatly improves exploration in the presence of sparse rewards. In the FrankaPush task, SAC leverages HER to transform sparse-reward data into more informative samples and uses entropy-regularized exploration to discover robust pushing strategies, leading to fast and stable convergence to high success rates.

TQC. TQC further refines the value estimation procedure by modeling the return distribution and truncating the highest quantiles when forming the target distribution. This truncation acts

as a conservative correction that reduces the impact of overly optimistic value estimates, even beyond what clipped double-Q can achieve. The combination of distributional critics, truncation, and entropy-regularized exploration enables TQC to converge quickly and stably to near-perfect performance on the FrankaPush task. Empirically, TQC matches or slightly surpasses SAC in robustness and final success rate.

7.3 Discussion

The comparative results on the Franka push task highlight several important lessons for sparse-reward robotic manipulation:

- Off-policy RL is well-suited to settings where interaction is expensive, as it enables extensive reuse of past experience and integration with HER.
- Vanilla deterministic actor-critic methods like DDPG may be inadequate when rewards are sparse and dynamics are contact-rich, due to overestimation and weak exploration.
- Entropy-regularized and distributional methods such as SAC and TQC provide more reliable and stable learning, making them strong candidates for practical deployment in sparse-reward robotic manipulation tasks.

8 Conclusion and Future Work

This report has presented a sparse-reward reinforcement learning framework for the Franka push task, a goal-conditioned manipulation environment based on a Franka Panda arm simulated in MuJoCo. We formalized the task as a goal-conditioned MDP with a binary sparse reward, described the environment's observation and action spaces, and detailed the formulations of three off-policy RL algorithms: DDPG, SAC, and TQC.

Our experimental results demonstrate that DDPG struggles on this task, achieving low success rates due to Q-value overestimation and insufficient exploration under sparse rewards. In contrast, SAC and TQC, which incorporate clipped double-Q learning, entropy regularization, and, in the case of TQC, distributional critics with truncated quantiles, achieve near-perfect success rates with stable convergence. These findings support the use of modern off-policy RL algorithms in combination with sparse rewards and HER for robotic manipulation.

Future work may extend this framework in several directions:

- Evaluating the same algorithms on more challenging tasks such as sliding and pick-and-place, using the same sparse reward formulation.
- Incorporating more realistic control schemes (e.g., impedance control) and investigating sim-to-real transfer of learned policies to a physical Franka arm.
- Systematically comparing sparse and dense reward formulations under identical algorithms to better understand the trade-offs between sample efficiency, robustness, and reward engineering complexity.