

Neural Network-Based Dynamic Grasping of Moving Objects with Robotic Manipulators

Yin Cao^{1,A}, A.A. Boguslavsky^{2,B}

^A Lomonosov Moscow State University, Moscow, Russia

^B Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, Moscow, Russia

¹ ORCID: 0009-0008-7577-2327, caoyin1995@gmail.com

² ORCID: 0000-0001-7560-339X, anbg74@mail.ru

Abstract

We explore the use of the open-source library Stable Baselines3 to implement reinforcement learning via deep neural networks for controlling a manipulator of grasping moving objects along a conveyor belt. Unlike static object grasping, this task requires accounting for dynamic factors, significantly complicating the process. We provide a detailed description of the physical-kinematic modeling of the manipulator in PyBullet and the integration of both the manipulator's and the moving objects' parameters into the neural network for training. The results of this study demonstrate that the decision-making capabilities and autonomous behavior provided by reinforcement learning algorithms can be successfully applied to complex tasks, such as dynamic object grasping.

Keywords: manipulator control, reinforcement learning, application of neural network models, physical simulation, PyBullet.

1. Introduction

The utilization of robotic manipulators for grasping objects on industrial conveyor lines has been instrumental in advancing automation in transportation and logistics for decades. Automating these processes enhances productivity, reduces costs, and minimizes human intervention in continuous production settings.

With recent advancements in artificial intelligence, researchers are actively applying neural networks to robotic and industrial automation tasks [1]. One particularly challenging problem is grasping moving objects, which requires high precision, adaptability, and rapid response. Solving this problem is crucial for optimizing modern production processes and increasing efficiency.

Neural network-based robot control offers several advantages over traditional methods, including learning from data, adaptability to new conditions, and handling complex and dynamic scenarios. For instance, Pane et al. [2] demonstrated that reinforcement learning algorithms can effectively enhance the accuracy of manipulators by stabilizing various trajectories.

Traditional control methods for manipulators, such as PID controllers and rigid algorithm-based approaches, often lack the flexibility and efficiency necessary for handling moving objects. Sekkat et al. [3] showed that deep learning-based algorithms, particularly DDPG combined with YOLO-based positioning, improve manipulator performance in grasping tasks. Furthermore, Wang et al. [4] explored reinforcement learning techniques for mobile manipulators using onboard sensors, integrating deep learning algorithms with visual perception.

Tian et al. [5] introduced a digital twin-based approach for trajectory planning, which was successfully applied in dynamic object collection, such as fruit picking. Their method demonstrated significant accuracy improvements over traditional techniques.

Naresh Marturi et al. [6] proposed a method that utilizes an inverse kinematics (IK) solver combined with task space error to select the optimal grasping trajectory for moving objects in real time. However, this approach relies on multiple fixed-position cameras, an RGB-D camera mounted on the gripper, and a manually designed error function.

Iretiayo Akinola et al. [7] introduced a method that depends on a pre-generated graspable database (containing approximately 5,000 grasp candidates) and an RNN-based neural network model to predict object motion trajectories for grasping.

In this study, we propose a method that leverages the open - source Stable Baselines3 and PyBullet libraries to develop an accessible and reproducible algorithm for the grasping of moving objects by a manipulator. The presented approach enables the development of an accessible and reproducible algorithm for manipulator grasping of moving objects. Our primary research goal is to leverage neural networks to improve a manipulator's ability to grasp objects moving along a conveyor belt, ensuring greater accuracy and stability even under dynamic conditions. The key contributions of this study include:

- We employ reinforcement learning algorithms for autonomous manipulator control, thereby eliminating the need for manually programmed trajectories.
- We demonstrate the capability to grasp moving objects, a task significantly more challenging than static object grasping
- We conduct a quantitative analysis of various reinforcement learning algorithms, with Proximal Policy Optimization (PPO) achieving the best results.
- We achieve a 100% success rate in 1,000 random tests after 800 training episodes.

2. Manipulator Modeling and Problem Statement

The system under study is a 7-degree-of-freedom (7DOF) Franka Emika Panda manipulator (see Figure 1). The geometric center of the fixed support, which connects the manipulator's base to the table, coincides with the origin of the coordinate system used for modeling. This configuration simplifies both trajectory calculation and motion control.

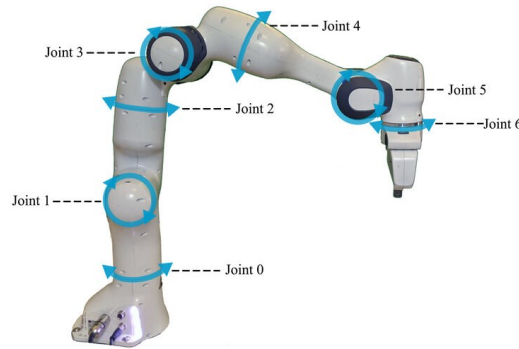


Fig. 1. Franka Emika Panda manipulator [8]

Each of the manipulator's seven joints has a specific range of rotation, ensuring high flexibility in task execution. The gripper of the manipulator can open up to a maximum width of 8 cm. The process control strategy employs a stepwise approach, in which the current joint positions determine minimal angular changes in the subsequent step, thereby ensuring smooth motion and minimal errors during complex manipulations

2.1 Problem Definition

The task involves a cylindrical object (diameter = 0.04 m, height = 0.17 m) that moves uniformly on a conveyor belt at a speed of approximately 3–5 meters per minute. The manipula-

tor is required to grasp the target within the specified area P ($0.7 \text{ m} \times 0.3 \text{ m}$) and within a time window of $T = 14$ seconds, while ensuring no contact is made with either the object or the conveyor belt.

2.2 Manipulator Modeling

PyBullet is a popular open-source physics engine that includes built-in models of various modern robots, manipulators, and other devices. It also supports loading models in formats such as URDF (Unified Robot Description Format), SDF (Simulation Description Format), and MJCF (MuJoCo XML Format). The engine is designed to simplify the transition from basic to realistic modeling, thereby facilitating convenient simulations of kinematics, optics, collision detection, and other processes. PyBullet is suitable for a wide range of robotic modeling and machine learning tasks.

In PyBullet, the simulation environment and target object are modeled (see Figure 2), where the arrow indicates the direction of the target's movement.

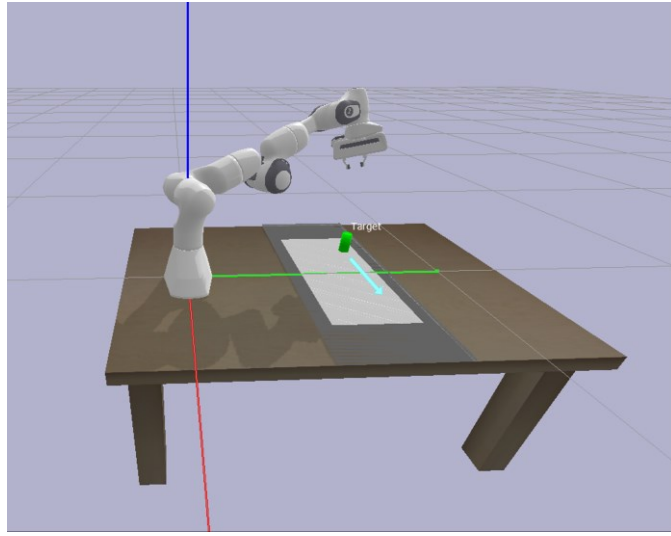


Fig. 2. Simulation environment of the task in PyBullet

3. Methodology

To address the aforementioned challenges, we first determine the position of the moving object using RGB-D camera data while simultaneously evaluating the impact of image processing latency on measurement accuracy. Subsequently, we develop a reinforcement learning pipeline for robotic arm grasping.

3.1 Target Position Calculation

A mask is created by distinguishing between the foreground and background on the conveyor belt. A segmentation network is utilized to extract the pixel coordinates (u, v) of the target along with its corresponding depth value, d . The intrinsic parameters of the camera are encapsulated in the matrix K , defined as follows:

$$K = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Where:

f_x, f_y — focal length of the camera (in pixels), calculated based on the known field of view angle (FOV);

C_x, C_y — position of the camera's optical center (typically the image center).

The transformation of pixel coordinates (u, v) into camera coordinates $P_cam: (x_cam, y_cam, z_cam)$ is as follows:

$$x_cam = \frac{(u - c_x) \cdot d}{f_x}, y_cam = \frac{(v - c_y) \cdot d}{f_y}, z_cam = d \quad (2)$$

Where:

where d represents the depth acquired from the RGB-D camera. The normalized depth value d_true is denormalized as follows:

$$d_true = \frac{z_{near} - z_{far}}{z_{far} - (z_{near} - z_{far}) \cdot d} \quad (3)$$

Where:

z_{near} — known parameter, the minimum visible distance for the camera.;

z_{far} — known parameter, the maximum visible distance for the camera;

d_{img} — known parameter, the normalized depth value.

Next, the extrinsic parameter matrix E is introduced, which transforms points from the camera coordinate system to the world coordinate system. It is defined as:

$$E = \begin{bmatrix} R & -R \cdot t \\ 0 & 1 \end{bmatrix} \quad (4)$$

Where:

t : Translation vector (known parameter), representing the camera's position in the world coordinate system, and R is the rotation matrix describing the orientation of the camera coordinate system relative to the world coordinate system.

$$R = \begin{bmatrix} Forward_x & Right_x & UP_x \\ Forward_y & Right_y & UP_y \\ Forward_z & Right_z & UP_z \end{bmatrix} \quad (5)$$

Where:

Forward = $\frac{focus_pos - camera_pos}{\|focus_pos - camera_pos\|}$, $focus_pos$, $camera_pos$ - coordinates of the camera's focus position and camera position.

Right = $\frac{foward \times camera_pos}{\|focus_pos \times camera_pos\|}$, rightward direction of the camera's orientation toward the focus point.

Up: The upward direction is computed as $Right \times Foward$

Finally, points in the camera coordinate system are transformed into the world coordinate system using the extrinsic parameter matrix. $P_world = [x_w, y_w, z_w, 1]$ using the extrinsic parameter matrix and data from RGB-D:

$$P_world = E \cdot P_cam \quad (6)$$

The entire process—from image acquisition to the completion of all calculations—requires approximately 50 ms on a standard computer (CPU: Intel Core i7-11800H, GPU: NVIDIA RTX 3060), and errors due to computational delays:

$$\frac{5 \text{ m/min} * 50 \text{ ms}}{60 * 1000} \approx 0,00416m \approx 4mm$$

Note that computational delays introduce errors, which increase with the instantaneous speed of the moving object. For instance, if an object moves at 4 m/s, the positional error may exceed 20 cm. Consequently, both our research and related studies focus on lower speeds (approximately 3–5 m/min). Given that the target objects measure about 4 cm in size, the resulting error remains within acceptable limits for successful capture.

3.2 The Process of Training Neural Network

Training a robot to successfully grasp a moving object is a challenging task. Initially, the robot's actions are largely random—it may attempt to grasp too early, too late, or even cause collisions and other unintended outcomes. During training, every attempt is carefully observed, and the system provides targeted feedback via a reward function. For instance, if the robot reaches too early, the feedback might be: “This time you reached too early; next time, try to get a little closer.” This process embodies the core principle of reinforcement learning, whereby continuous trial-and-error, combined with systematic feedback, enables the robotic arm to refine its actions for more efficient grasping of moving objects.

The core components of the reinforcement learning environment include:

Agent (Robot): The manipulator interacting with the environment.

Environment: The simulation setup (implemented in PyBullet as detailed in Section 2).

State: Defined by the observation space.

Action: Defined by the action space.

Reward: A feedback function guiding the learning process.

In our task, the observation space S is represented by a 13-dimensional vector, capturing both the manipulator's joint angles and additional sensory inputs.:

$$S_t = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, gripper_{x,y,z}, target_{x,y,z}]$$

Similarly, the action space A is defined as a 7-dimensional vector corresponding to the manipulator's degrees of freedom. Here, the components θ in S denote the current joint angles, while $\Delta\theta$ in A represent the incremental adjustments to be applied.:

$$A_t = [\Delta\theta_1, \Delta\theta_2, \Delta\theta_3, \Delta\theta_4, \Delta\theta_5, \Delta\theta_6, \Delta\theta_7]$$

These dimensions correspond to the manipulator's degrees of freedom and sensory inputs (θ in S are the current joint angles of the robotic manipulator, and $\Delta\theta$ in A mean the next time to move).

The training of the neural network involves the following stages:

Step 1. Parameter Initialization: The manipulator is moved to its initial position, and the object is introduced at a random location within the workspace.

Step 2. Observation and Action: The current state S_t , is fed as input to the network, which then outputs the adjustment A_t for the joint angles.

Step 3. State Update: After executing the action, the new state S_{t+1} is recorded as the object moves along the conveyor belt.

Step 4. Reward: The network receives a reward R_t for successfully completing the grasping task, or a penalty if it deviates from the target. The reward components include:

$$R_t = \begin{cases} Reward_{collision} = [if \ True = -100 \ else \ 0] \\ Reward_{jud_success} = [if \ True = 1000 \ else \ 0] \\ Reward_{distance} = [(\|target_xyz - gripper_xyz\|) \times 1000] \\ Reward_{time_left} = [if \ True = 0.5 * time_{left} \ else \ 0] \end{cases} \quad (7)$$

$Reward_{collision}$ — A penalty for collisions, guiding the manipulator to avoid impacts;

$Reward_{jud_success}$ — An additional reward for task completion;

$Reward_{distance}$ — A reward that encourages the gripper to approach the target;

$Reward_{time_left}$ — A reward based on the remaining time after task completion, promoting faster execution.

Step 5. Parameter adjustment: Network parameters are updated to minimize error.

Step 6. Repeat Steps 1–5 until convergence.

The task is deemed successful if the manipulator's grasping action locates the object within an error margin of 2 cm and no safety constraints are violated (e.g., no collisions with the conveyor). In our implementation, various reinforcement learning algorithms provided by the Stable Baselines3 library were evaluated—such as DDPG (Deep Deterministic Policy Gradi-

ent) [9], TD3 (Twin Delayed Deep Deterministic Policy Gradient) [10], SAC (Soft Actor-Critic) [11], and PPO (Proximal Policy Optimization) [12]. Each algorithm offers distinct advantages for continuous control tasks:

DDPG: Utilizes deterministic policy gradients for smooth control.

TD3: Enhances DDPG by mitigating overestimation errors via twin delayed updates.

SAC: Incorporates an entropy term to foster exploration.

PPO: Employs a clipped objective function to maintain stable learning.

Due to space limitations, a detailed discussion is omitted; readers are referred to [9]–[12] for further details.

The system operation diagram is shown in Figure 3.

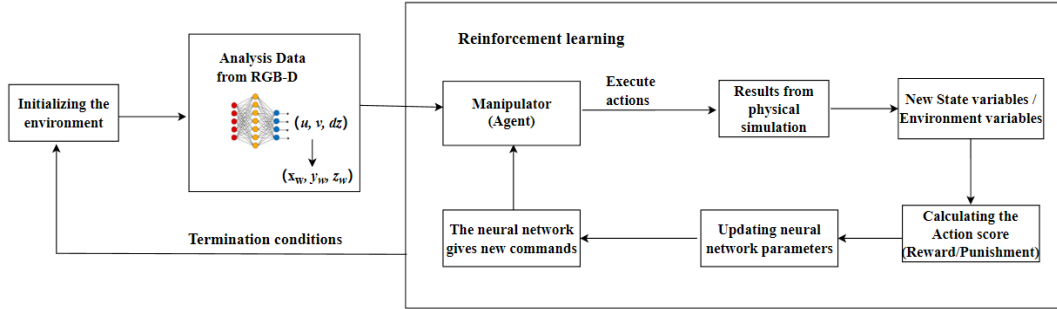


Fig. 3. Scheme of operation of the robot control system in the simulation environment

3.3 Integrated Learning Environment

Based on the aforementioned process, we have established a deep reinforcement learning framework to perform the task. However, it is essential to integrate the physical modeling with the task dynamics into a unified learning environment that can be seamlessly processed by the algorithms. Gymnasium is an open-source framework designed to standardize and integrate various reinforcement learning environments. We modified the Gymnasium framework to integrate both the physical modeling and the learning process into a single environment (see Fig. 2). Further details about Gymnasium can be found in [13].

4. Results and discussion

We conducted experiments over 800 episodes on a general-purpose computer powered by an Intel Core i7-11800H CPU. Each episode consists of 30 million time steps. To evaluate the proposed neural network-based manipulator control method, we implemented and tested four popular reinforcement learning algorithms—DDPG, TD3, SAC, and PPO.

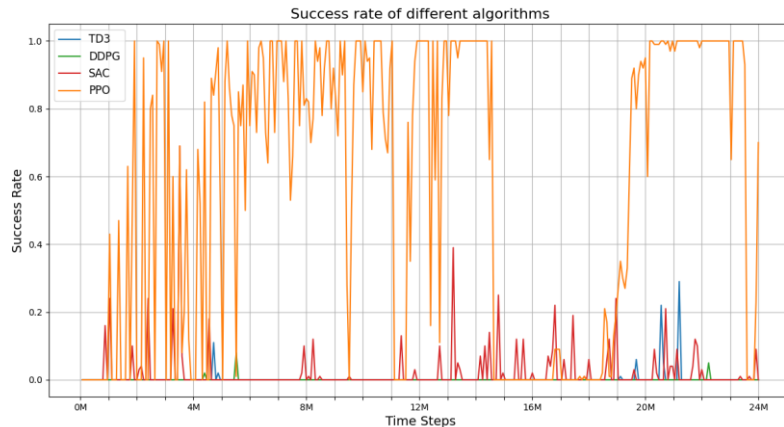


Fig. 4. Success rate of different algorithms for iterations

As illustrated in Figure 4, the following observations can be made:

PPO achieves a high success rate, with the range of episodes reaching 100% success widening as training time increases.

SAC and **TD3** exhibit moderate performance, maintaining a success rate of approximately 50%.

DDPG performs the worst, with success rates remaining below 10%.

Figure 5 illustrates the performance of the various algorithms under random testing conditions.

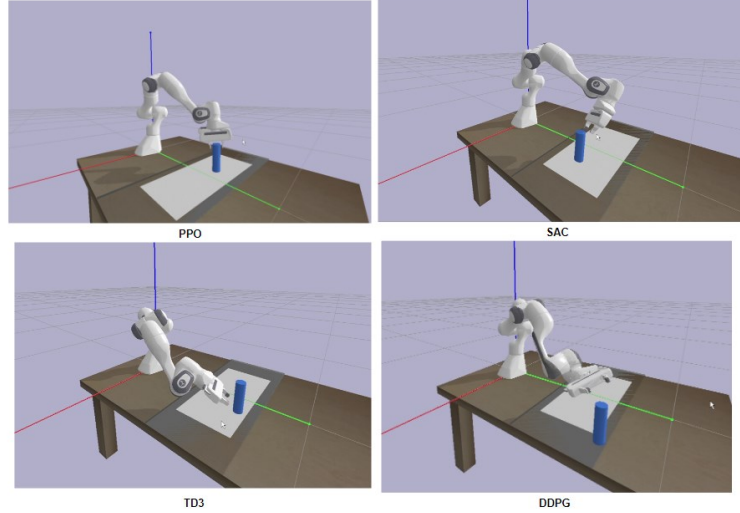


Fig. 5. Results of different algorithms in random test

Real-time recordings of the experiments are available for viewing or download at: <https://disk.yandex.ru/i/cXR4msWAKiwjow>.

We summarize the results for different algorithms in Table 1. The higher the average reward, the faster the capture task is completed and the better the overall performance. Analyzing the data from Table 1, we can conclude that the PPO algorithm shows better results in terms of training time and task completion success compared to other algorithms.

Table 1 — Comparison of performance of different algorithms

Algorithms	Reference indicators				
	Neural network inference speed(ms)	Average time steps	Success rate in 1000 times (%)	Average Reward R_t	Time of training (hhmm)
DDPG	<2	213.8	3.1	660.5	9h15m
TD3	<2	164.1	36.3	1188.6	8h51m
SAC	<2	269.1	68.1	1397.2	9h21m
PPO	<2	120.5	100	2239.9	3h39m

5. Conclusion

This article presents an approach that employs neural networks and reinforcement learning—implemented using the open-source library Stable Baselines3—to address the challenge of a manipulator capturing objects moving uniformly on a conveyor belt. In our study, the target objects were abstracted as cylinders, and the method was validated using a virtual physical simulation platform, thereby confirming its applicability and effectiveness. We employed several popular reinforcement learning algorithms, including PPO, DDPG, SAC, and TD3, and provided a detailed description of the steps necessary to solve this task using open-source neural network models. The results obtained in this study provide a solid foundation for further research and offer a valuable starting point for other researchers interested in similar

tasks. Future research directions may include investigating the use of mobile robots for capturing moving targets.

Acknowledgements

The present work was partially supported by the China Scholarship Council (CSC) N^o202108090230.

References

1. Shuzhi, S. G., Hang, C. C., & Woon, L. C. Adaptive neural network control of robot manipulators in task space // IEEE transactions on industrial electronics, 44(6), 1997, pp.746-752.
2. Pane, Y. P., Nagesh Rao, S. P., Kober, J., & Babuška, R. (2019). Reinforcement learning based compensation methods for robot manipulators // Engineering Applications of Artificial Intelligence, 78, 2019, pp.236-247.
(doi: 10.1016/j.engappai.2018.11.006) (<https://doi.org/10.1016/j.engappai.2018.11.006>)
3. Sekkat, H., Tigani, S., Saadane, R., & Chehri, A. Vision-based robotic arm control algorithm using deep reinforcement learning for autonomous objects grasping // Applied Sciences, 11(17), 2021, p.7917.(doi:10.3390/app11177917) (<https://doi.org/10.3390/app11177917>)
4. Wang, C., Zhang, Q., Tian, Q., Li, S., Wang, X., Lane, D., & Wang, S. Learning mobile manipulation through deep reinforcement learning // Sensors, 20(3), 2020, 939.
(doi:10.3390/s20030939) (<https://doi.org/10.3390/s20030939>)
5. Tian, X., Pan, B., Bai, L., Wang, G., & Mo, D. Fruit Picking Robot Arm Training Solution Based on Reinforcement Learning in Digital Twin // Journal of ICT Standardization, 11(3), 2023, pp.261-282.
(doi:10.13052/jicts2245-800x.1133) (<https://doi.org/10.13052/jicts2245-800x.1133>)
6. Marturi, N., Kopicki, M., Rastegarpanah, A., Rajasekaran, V., Adjigble, M., Stolkin, R., ... & Bekiroglu, Y. Dynamic grasp and trajectory planning for moving objects // Autonomous Robots, 43, 2019, pp. 1241-1256.
(doi:10.1007/s10514-018-9799-1) (<https://doi.org/10.1007/s10514-018-9799-1>)
7. Akinola, I., Xu, J., Song, S., & Allen, P. K. Dynamic grasping with reachability and motion awareness // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 9422-9429.
(doi:10.48550/arXiv.2103.10562) (<https://doi.org/10.48550/arXiv.2103.10562>)
8. Reed, A., Albin, D., Pasricha, A., Roncone, A., & Heckman, C. Transformer-based Learning Models of Dynamical Systems for Robotic State Prediction., 2024,
(doi:10.21203/rs.3.rs-3919154/v1) (<https://doi.org/10.21203/rs.3.rs-3919154/v1>)
9. Lillicrap, T. P. Continuous control with deep reinforcement learning // arXiv preprint, 2015, (doi:10.48550/arXiv.1509.02971) (<https://doi.org/10.48550/arXiv.1509.02971>)
10. Fujimoto, S., Hoof, H., & Meger, D. Addressing function approximation error in actor-critic methods // International conference on machine learning, PMLR, 2018, pp. 1587-1596.
(doi:10.48550/arXiv.1802.09477) (<https://doi.org/10.48550/arXiv.1802.09477>)
11. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor // International conference on machine learning, PMLR, 2018, pp. 1861-1870. (doi:10.48550/arXiv.1801.01290) (<https://doi.org/10.48550/arXiv.1801.01290>)
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. Proximal policy optimization algorithms // arXiv preprint, 2017 (doi:10.48550/arXiv.1707.06347) (<https://doi.org/10.48550/arXiv.1707.06347>)
13. Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., ... & Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint, 2024, (doi:10.48550/arXiv.2407.17032) (<https://doi.org/10.48550/arXiv.2407.17032>)