Embodied AI: Final Project Report

Instructed by *Huazhe Xu*

Due on Jan. 11th, 2025

Peiqi Duan, Siqiao Huang, Yihan Xu

1 Introduction

Maniskill is a simulation benchmark which contains several tasks that require the agent to interact with the environment. It was first introduced in 2021 by the team of Tongzhou Mu et al[9] and later developed more, which leads to the Maniskill2 benchmark[4]. Recently, using the latest GPU simulation and rendering techniques, Tao et al released the Maniskill3[13], which supports multiprocess and more new features. The Maniskill series benchmark support diverse objects, robots, scenes, tasks and algorithms with paralleled and efficient simiulation and rendering. It also has a large-scale dataset for trajectories.

In this project, we choose several tasks by single or double robotic arms to complete and use the reinforcement learning algorithm to train the agent. We use the BC and PPO algorithm implemented in the benchmarks and try to make improvements on them, while observing suprising reward hacking during the experiments. We successfully design a new task called 'Push Can and Open the Microwave Door'. We also develop a new task generator using LLM for the Maniskill3 benchmark, which is called ManiGen. For demonstrations, please check our website: Exploring ManiSkill.

2 Preliminaries

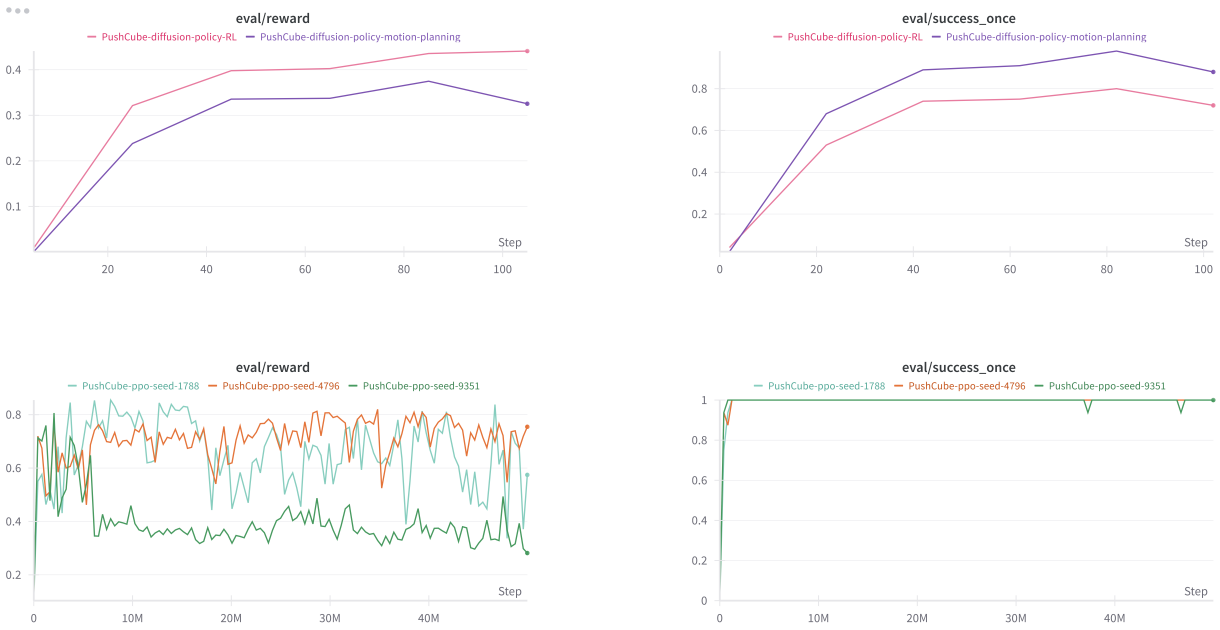
Robotics simulators are crucial tools for developing and testing robot control algorithms in a safe, cost-effective, and reproducible environment. They provide a virtual sandbox for experimenting with various robot designs, environments, and tasks without the risks and expenses associated with real-world deployment. However, a persistent challenge is the "reality gap" – the discrepancy between the simulated and physical worlds. This gap stems from simplified physics models, idealized sensor data, and the lack of unpredictable real-world factors in simulations. Overcoming this gap is a primary focus of research in robotics simulation, with techniques like domain randomization, system identification, and the incorporation of real-world data used to bridge the divide and enable effective sim-to-real transfer of learned policies. Simulators like ManiSkill3 are pushing the boundaries of realism and scalability, offering high-fidelity physics, diverse environments, and GPU-accelerated performance to facilitate the development of more robust and generalizable robot behaviors.

In this project, we choose PPO in RL and BC in IL to make improvement on. The PPO (Proximal Policy Optimization) algorithm[11] is a policy gradient method that is widely used in reinforcement learning. It is a kind of on-policy method that can be used to optimize the policy directly. The PPO algorithm is based on the trust region policy optimization method, which can prevent the policy from changing too much in each iteration. The PPO algorithm has two versions, the PPO-Clip and PPO-Penalty. The PPO-Clip uses the clipped surrogate objective function to optimize the policy, while the PPO-Penalty uses the penalty objective function. The PPO algorithm has been widely used in reinforcement learning and has achieved good results in many tasks. PPO has herited the advantages of TRPO[10] and has a better performance in the training process. Until now, PPO is still one of the most popular and powerful algorithms in reinforcement learning.

The BC (Behavioral Cloning) algorithm[14] is a kind of imitation learning algorithm that can be used to train the agent by learning from the expert's demonstration. The BC algorithm is based on the supervised learning method, which can be used to learn the policy directly. The BC algorithm has been widely used in reinforcement learning and has achieved good results in many tasks. The BC algorithm is simple and easy to implement, which makes it a good choice for the beginners in reinforcement learning.

3 Baseline Results

We run PPO and diffusion-policy on task PushCube, using the default baseline settings from ManiSkill repo. The results align with the launched benchmarks.



4 Algorithm Modification

We have made some modifications to the PPO and BC algorithms to improve their performance. The modifications are as follows:

- Change the network structure: we try to add more layers, such as observation encoder, memory module, attention mechanism, etc., to the network structure to improve the performance of the algorithm.
- Learning Rate Schedule: We intoduce learning scheduler OneCycleLR to the orginial behavior cloning, which show good results.

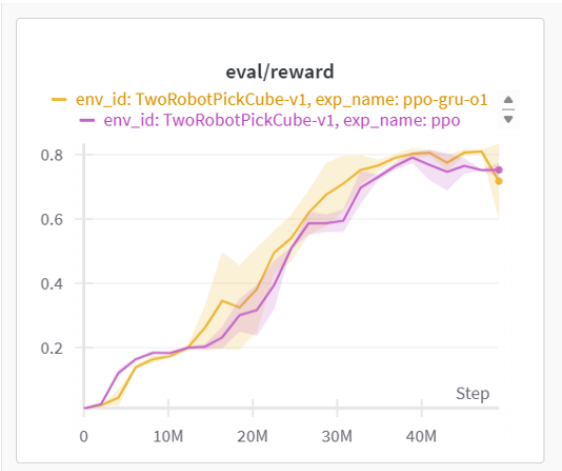
4.1 Modifications on PPO

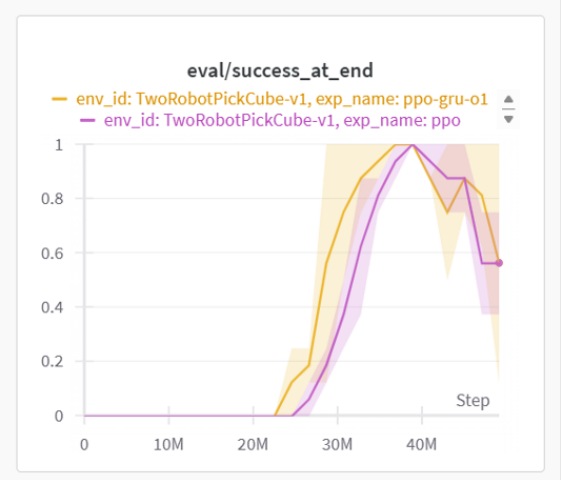
In PPO learns a history independent policy, i.e. $\Pr(a_t|s_1, ..., s_t) = \Pr(a_t|s_t)$, it may suffer from poor data efficiency as intuitively the model's next action may counteract the current action's effect. This data efficiency problem is more pronouned in long-horizon and sparse reward tasks, as the model need to build on a longer series of actions aiming in the desired direction to get a positive return. Based on this intuition, we explore the idea of leveraging the agent's history observations to achieve better performance, especially in long-horizon tasks.

We integrated Gated Recurrent Network(GRU [3]) to the observation of PPO networks, which we refer to as PPO-GRU. Specifically, we augment the current observation to a history-intergrated form $\tilde{s}_t = f(s_t, h_{t-1})$, where h_{t-1} represents the hidden state in GRU layer. We furthermore explore other model architectures to obtain three additional variants of this model:

1. PPO-LSTM: Instead of using GRU to guide the observation update, we substitute it with Long short-term memory (LSTM [5]).
2. PPO-Attention: We intergrate the attention mechanism [1] to the current structure to obtain state representations that intuitively takes into account their relationship with other states.
3. Energy-Based PPO: Instead of learning a policy $\pi(a|s) = h_\phi(s)$, we learn an energy-based model[2] $E_\phi(s, a)$, and the policy is determined by $\pi(a|s) = \frac{E_\phi(s,a)}{\sum_{a' \in A} E_\phi(s,a')}$.

Examples of the result is shown in Figure 3(a) and Figure 3(b).





However, in simulated environments, where states are fully-observed, these proposed algorithms fail to surpass PPO by a large margin. Aside from the reason that Mani-Skill 2's fully-observation environment is sufficient enough for Reinforcement Learning without need for additional state representations, we believe part of the reason for this phenomenon is the agent is hacking the hand-crafted reward [12]. An illustration in shown in figure 4.

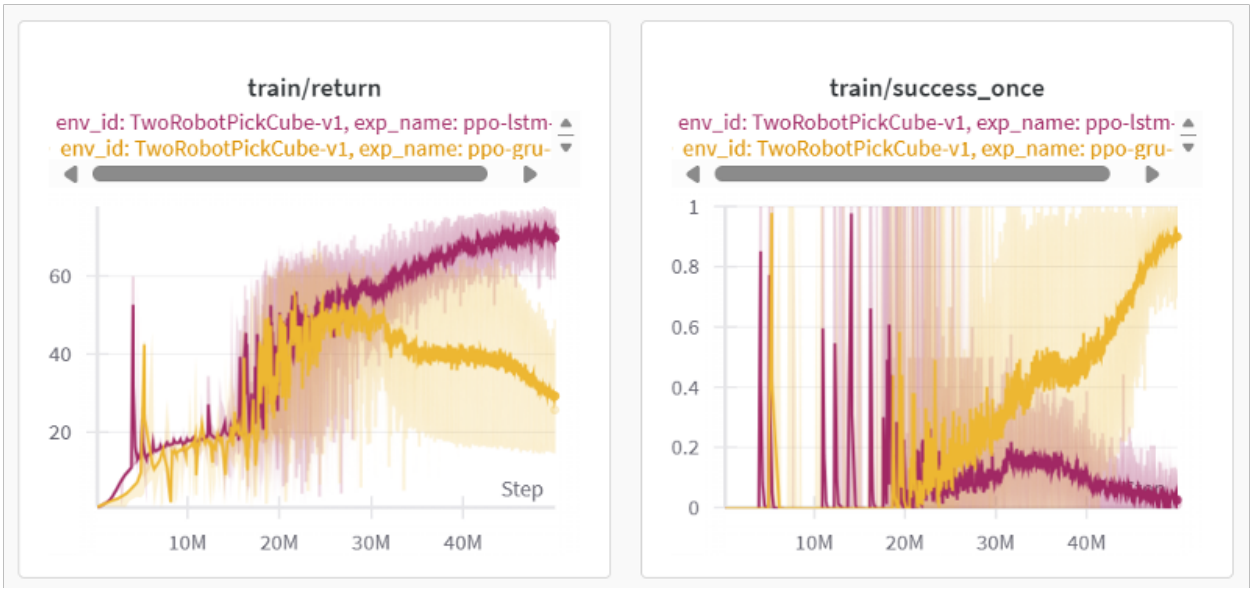


Figure 4

We believe that in Partially-Observable environments, for instance real-world robotic deployments, the current sensory observations may not be sufficient to deduce the current state, therefore leveraging history can give the agent more information to it's current state. Therefore our modification's effect may be more pronounced in POMDP environments.

4.2 Modifications on Behavior Cloning

We introduce OneCycleLR to Behavior Cloning. OneCycleLR dynamically alters the learning rate under certain schedule. The setting we choose is to first linearly increase the learning rate until certain point (0.3 * total steps), and then decrease it linearly. The basic idea is that boosting the learning rate in early stages can promote the model to learn more, while decreasing in later stages can improve the robustness. The settings we use is the default setting of the behavior cloning repo, with the same network architecture for both cases. Except for originalBC for PullCube, we run BC for 10000 steps for all cases. For PickCube, we use 1000 episodes of data from motion planning. For PullCube, we use 947 episodes. For RollBall, we use 215 episodes from reinforcement learning. Our result shows more than 10% improvement on the original BC.

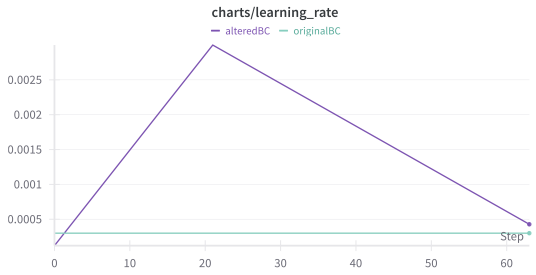
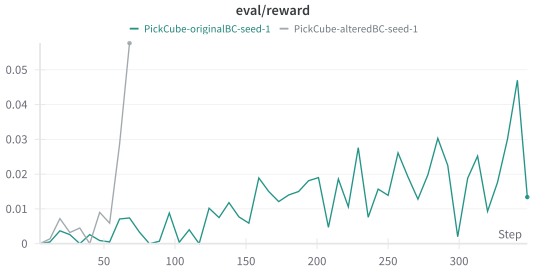
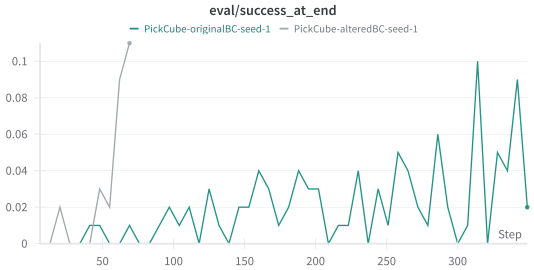
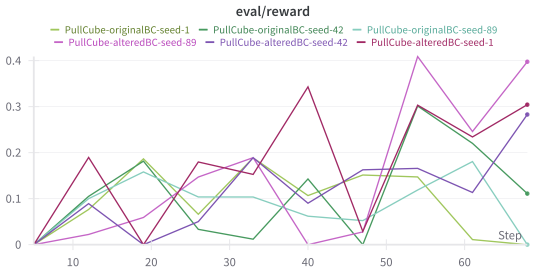
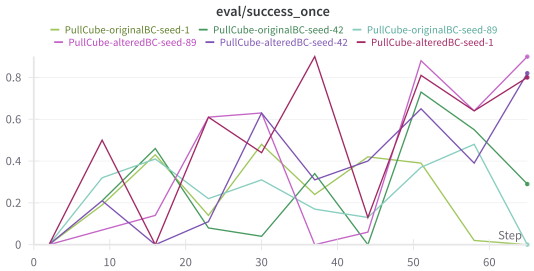


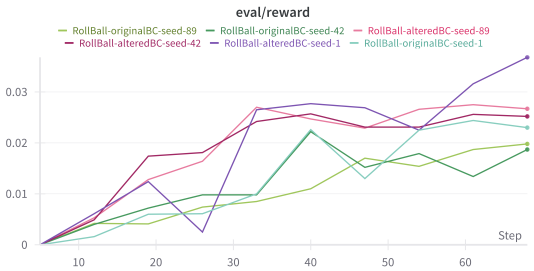
Figure 5: Learning Rate Scheduler

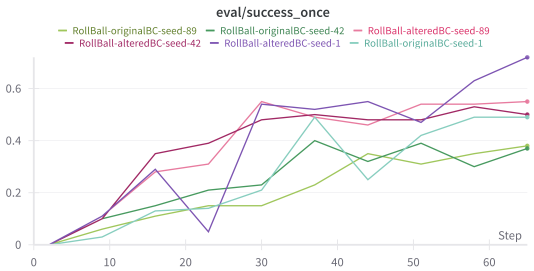

eval/reward
PickCube-originalBC-seed-1
PickCube-alteredBC-seed-1
0.05
0.04
0.03
0.02
0.01
0
50
100
150
200
250
300
Step


eval/success_at_end
PickCube-originalBC-seed-1
PickCube-alteredBC-seed-1
0.1
0.08
0.06
0.04
0.02
0
50
100
150
200
250
300
Step


eval/reward
PullCube-originalBC-seed-1
PullCube-originalBC-seed-42
PullCube-originalBC-seed-89
PullCube-alteredBC-seed-89
PullCube-alteredBC-seed-42
PullCube-alteredBC-seed-1
0.4
0.3
0.2
0.1
0
10
20
30
40
50
60
Step


eval/success_once
PullCube-originalBC-seed-1
PullCube-originalBC-seed-42
PullCube-originalBC-seed-89
PullCube-alteredBC-seed-89
PullCube-alteredBC-seed-42
PullCube-alteredBC-seed-1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
Step


eval/reward
RollBall-originalBC-seed-89
RollBall-originalBC-seed-42
RollBall-alteredBC-seed-89
RollBall-alteredBC-seed-42
RollBall-alteredBC-seed-1
RollBall-originalBC-seed-1
0.03
0.02
0.01
0
10
20
30
40
50
60
Step


eval/success_once
RollBall-originalBC-seed-89
RollBall-originalBC-seed-42
RollBall-alteredBC-seed-89
RollBall-alteredBC-seed-42
RollBall-alteredBC-seed-1
RollBall-originalBC-seed-1
0.6
0.4
0.2
0
0
10
20
30
40
50
60
Step

5 New Task Proposal

The new task we design here is called 'Push Can and Open Microwave Door'. The robot first needs to push away the can in front of the microwave door to some certain point, so that opening the door will not let the object fall off. Then it needs to open the door of the microwave.

We use PPO for the task training. The reward function we design is divided into two stages. The first reward function is responsible for pushing the object to certain range. We penalize the distance between the robot end effector and the object, as well as the distance between the object and the center of the goal range. The second reward function is for opening the microwave door, and is only valid when the first stage returns success. It first penalizes the distance between end effector and handle link, and then rewards the opening angle of the door.

Through PPO for 35 million steps we achieve a success-at-once rate of 0.88. We judge final success by two measurements: 1) whether the object is 0.2m within the goal center; 2) whether the opening angle of the door is larger than 0.75 of its maximum range. Our randomization contains the initial pose of the microwave, which is in a 0.05m * 0.05m region, the initial pose of the robot relative to the microwave, which is 2.5-3m away and -9 to 9 degrees from the microwave, the position of the object, which is in a 0.25m range on the y-axis, and the orientation of the object.

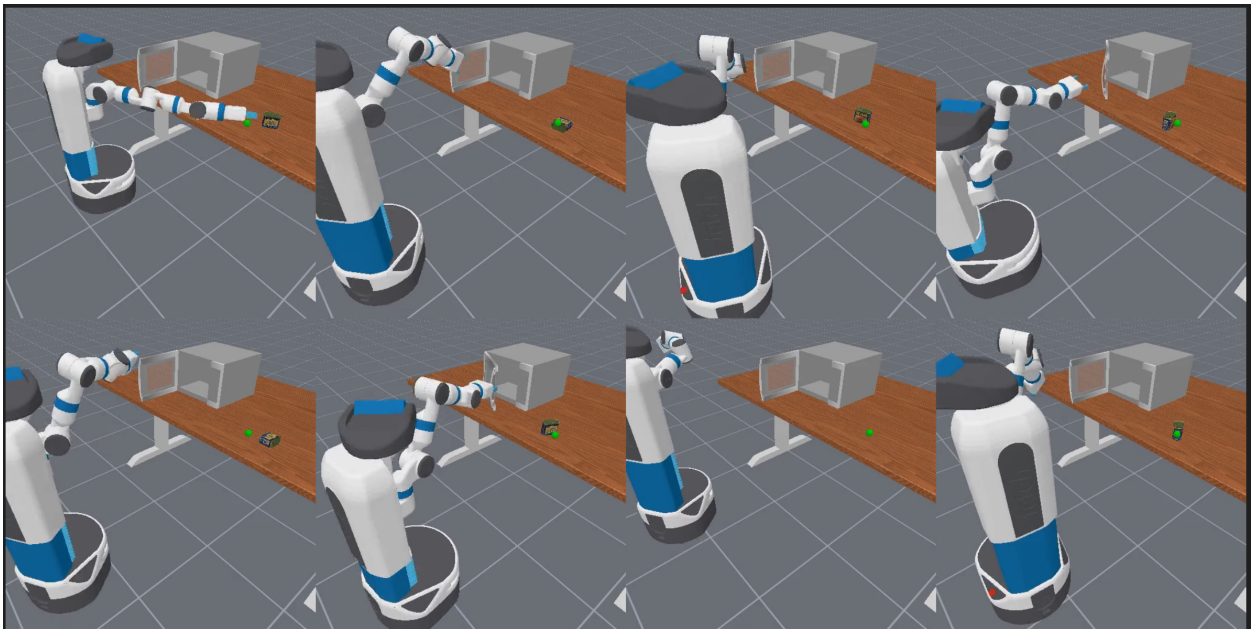


Figure 9: Success Rate with 7/8

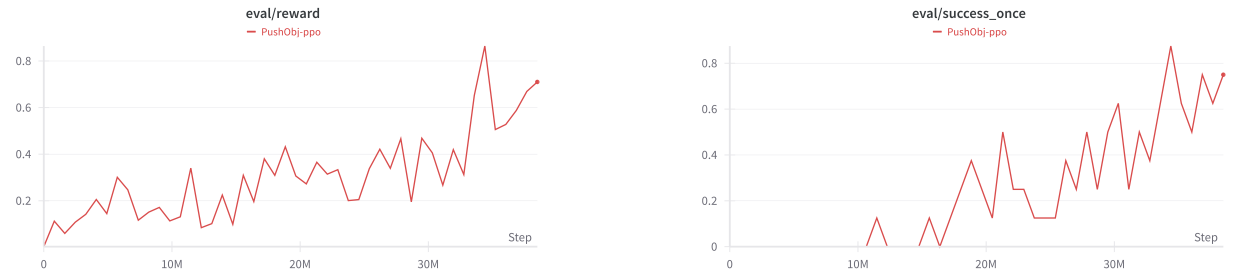


Figure 10: training curves

6 Generative Simulation for ManiSkill: ManiGen

The name ManiGen follows the style of GenSim series[6] and RoboGen[15]. We follow the idea of using LLM to generate new, diverse tasks and design reward functions. It is fully compatible with the Maniskill3 benchmark, and makes full use of its advantage: Maniskill3 provides user-friendly interface for scene formulation and new task design, enables query for link and mesh positions with their names or types, and provides fast simulation and training.

The pipeline is mainly composed of 4 parts: First, it randomly samples objects from the asset dataset(we use the GenSim2 dataset, which is mainly inherited from Parnet-Mobility Dataset). At most two rigid objects, and one articulated object are available. Then, it queries LLM (we use gpt-3.5) for possible task names, descriptions, decompositions and scene properties, including whether the scene uses table, how many objects of each type it contains, the names of the links, joints and meshes whose information is needed for the task, as well as whether the task requires to initialize the joint angle. The prompt contains all link, mesh and joint information of the objects, and promote LLM to think about the use of each part. After that, the LLM is queried again to generate code for four functions: get_obs_extra, evaluate, compute_dense_reward and set_goal_and_parameter. The prompt contains the previously generated task configurations. Finally, the functions are integrated with a task template that enables automatic object loading and scene generation. Usually we can directly start training on the generated code.

We consider three scene settings: only rigid bodies, only articulated bodies and mixed, and write prompt for task and code generation for each one of them. Thus, our pipeline allows diverse scene generations as there can be unrelevant objects on the scene.

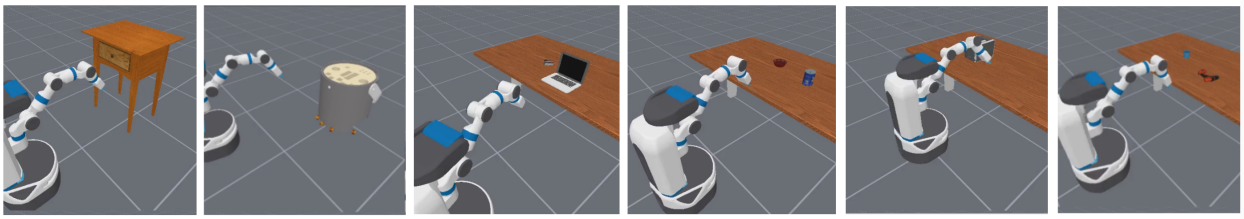
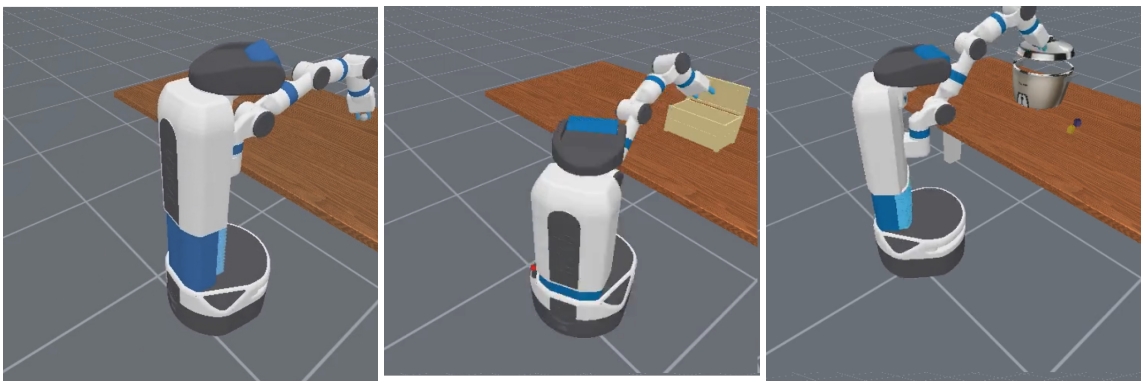


Figure 11: Example Scenes

The main problem generative simulation research tries to tackle is how to acquire robot learning data with high efficiency and low human participation. We believe that a generative simulation



Transfer Golf Ball | Open the Box | Remove the Lid

Figure 12: Example Tasks (See our website for demos)

pipeline is promising to scale up the robot learning dataset in the Maniskill environment. Meanwhile, we identify some remaining challenges in our system, and some of them are fundamental to this field and not only exist in our projects. We list them as follows:

- Lack of Task Diversity. While currently, generative simulation pipelines can usually guarantee the diversity of scene by sampling distracting bodies and scene configurations, in our pipeline, as well as in our experience of implementing other generative simulation pipelines like Gen2Sim[7], we all observe that LLM always output similar tasks with similar asset inputs, especially for articulated objects. For example, given a microwave, the pipelines can only generate combinations of opening and closing the door. This is mainly due to the limitation of LLM itself, and partially due to human prompt and asset labeling.

  For rigid bodies, the imagination of LLM is broader. For instance, in our pipeline, given bottles and balls, other than primitive pick-and-place, LLM generates a task called bottle-bowling, using the ball to knock the bottles over. However, the problem here is the high difficulty, beyond the capability of LLMs and RL.

- LLM's inability to generate good reward functions, especially for long-horizon tasks. There are three main challenges here: the first is that LLM may not always output correct and runnable code; the second is that LLM may fail to understand the time and logic order of the tasks, and its output may be self-contradictory (i.e. for a long-horizon task that requires multiple picking and placing, it may put all reaching, grasping and placing rewards together), which we solve to some extent by urging it to consider the order in our prompt; the third is that it tends to fail when the task is complex, and explicit reward concerning positions and velocities cannot easily solve it. For the latter case, we guess incorporating reward-refinement system like Eureka[8] can help.

- Unsatisfiable performance of RL systems and reward hacking, coupled with insufficient essential information in the raw asset dataset. In many cases, a pure-RL task solver is incapable or inefficient. In our project, we rarely succeed with long-horizon generated tasks, even with seemingly correct and resonable generated reward functions. Some existing generative simulation pipelines look to motion planning for help. RoboGen uses motion

planning to first guide the robot to an area near the goal object, while GenSim2 uses a kPAM motion planner as its main solver. While these approaches significantly enhance the performance of the tasks, there's still potential for improvement, to both lift performance and reduce human efforts if other solver systems can be added.

Meanwhile, we find that the information of objects agent can acquire from raw asset dataset and the simulator is sometimes not enough for task completion. For example, to open a laptop, the simulator can only provide the center of the laptop screen mesh, but the edge center coordinate is what matters the most. GenSim2 labels the articulated objects with keypoint coordinate, but the fixed coordinate cannot cope with rotation and scaling of objects. We need more modules to be imported, to improve the agent's knowledge and capability.

We hope that there will be great advance in enhancing data quality in the field of generative robotic simulation.

7 Conclusion

In this project, we explored the capabilities and challenges of Reinforcement Learning and Imitation Learning algorithms using Maniskill2 benchmark for robotic simulation tasks. We evaluated the performance of baseline algorithms. Furthermore, we proposed algorithmic improvements on both PPO and BC algorithms by introducing architectural designs in PPO and integrating OneCycleLR learning rate scheduler into the in BC. Through empirical study, we demonstrated the effectiveness of our approach.

As for new task proposal, we designed 'Push Can and Open Microwave Door' task, a manipulation task that evaluates the ability for algorithms to learn in complex and long-horizon environments. We then introduce ManiGen, a generative simulation tool that utilizes the generative powers of large language models to automate the creation of diverse and scalable tasks. Despite the challenges remaining in our simulation pipeline, we believe this work verifies the potential of using generative simulation pipelines to scale up robot learning, as well as offering a valuable resource for future research in the field.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2014.

[2] Ruifeng Chen, Chengxing Jia, Zefang Huang, Tian-Shuo Liu, Xu-Hui Liu, and Yang Yu. Offline transition modeling via contrastive energy learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 6992–7014. PMLR, 21–27 Jul 2024.

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv: 1406.1078*, 2014.

[4] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659*, 2023.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[6] Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. *arXiv preprint arXiv:2410.03645*, 2024.

[7] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023.

[8] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2024.

[9] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021.

[10] John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

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

[12] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[13] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[14] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[15] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.