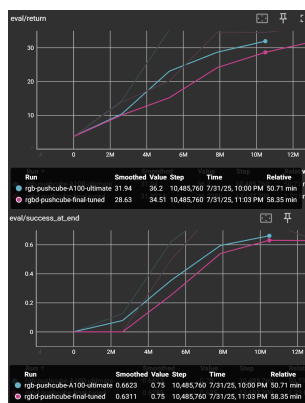


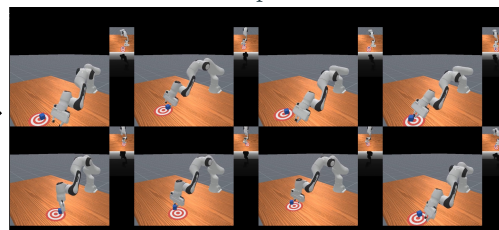
## 总结 Part 2:

### 7. The Limited Performance Gain from Depth Data

---Is More Sensory Data Always Better?



The Final Result of RGB-Depth (After 20,000,000 timesteps)



eval\_success\_once\_mean=1.0

"This strongly suggests that simply adding more sensory data, even when correctly pre-processed, is not a guaranteed path to better performance."

(蓝线只用 RGB 粉线: RGB-D)

我们比较了单纯使用 RGB 视觉输入和使用 RGB+Depth (RGB-D) 输入的 PPO 强化学习训练性能。在 1200 万步的训练中, RGB 模式在累计回报与最终成功率上均优于 RGB-D 模式。即便在 2000 万步后, RGB-D 的性能也未超过 RGB。这一结果提示我们: 在 RL 中, 感知模态的增加会带来更大的特征学习和计算负担, 若新增模态与任务关联度不高, 反而可能抵消其潜在收益, 因此更多数据并不必然带来更好表现。

# Part 3: Demonstration Data

这部分的目的是解释——在 ManiSkill 中我们是如何保存、组织和理解任务执行数据的，因为这些数据将作为后续 模仿学习（Imitation Learning）、调试和错误分析的重要基础。

下面这部分我认为 PPT 写的很好详细，大致生成点内容看着讲解就行了

标题：H5 file format & different traj

- H5/HDF5 是一种层级化数据存储格式（Hierarchical Data Format），常用于科学计算中存储大规模、多类型数据。支持把数组、数据表、元数据等放在一个文件里，就像“文件系统嵌套在单个文件中”。

在本项目中的用途：

- 我们把每一次 Agent 完成任务的全过程（称为一条 轨迹 **trajectory**）保存到一个 `.h5` 文件中。
- 每一条轨迹的命名规范：`traj_XXX`（XXX 是编号），每一条就是一次完整的 **episode** 的数据。

每条轨迹包含的内容：

1. **obs (Observations)**：观测数据

- 可以是 state 向量，也可以是 RGB 图像 / RGBD / segmentation / point cloud。
- 保存每个时间步 Agent 接收到的输入。

2. **actions**：动作数据

- 在每个时间步 Agent 对环境执行的动作（可以是离散动作 ID，也可以是连续控制量）。

3. **terminated**：是否“自然结束”（包括成功或失败正常结束）



4. **truncated**：是否被“截断结束”（比如超时、中途强制打断）

5. **success**：是否真正完成了任务目标


6. **env\_states**：环境的底层状态信息（通常是仿真引擎提供的特权信息，比如物体精确位置姿态）

7. **rewards**：每步的奖励值


 这些全部按时间步顺序存储——这样就能在事后回放、分析、训练模仿学习模型。



Demonstration Data




Field Name	Description
terminated	Indicates whether the task ended normally (e.g., completed successfully or failed)
truncated	Indicates whether the task was forcibly interrupted due to timeout
success	Indicates whether the task truly accomplished its objective


 **success = True ≠ terminated = True**


In ManiSkill, after a task is successfully completed, the agent may continue to execute a few more stabilizing actions before the episode ends or is interrupted.


A motion planning task:  
the agent might succeed but not terminate in time -> both success=True & truncated=True

## 看三个经典案例：











Peg Insertion



Stack Cube



Draw Triangle

Successful Episode  
Or  
Failed Episode







```
from mani_skill_trajectory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file='demo/PegInsertionSide-v1/retocomplanning/trajectory_rgbd_pd_joint_delta_pos_physx_cpu_H5')
data = dataset[332]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"])
axs[1].imshow(data["sensor_data"]["hand_camera"]["depth"])
axs[0].set_title("Success")
fig.suptitle("Episode 332 - ('Success' if success else 'Failure')", fontsize=10)
plt.show()
```

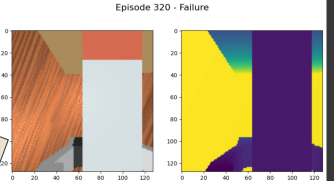
**Episode 332 - Success**




**Episode 332 (Success)**  
Peg fully inserted into the hole  
(red section completely entered the hole)


```
from mani_skill_trajectory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file='demo/PegInsertionSide-v1/retocomplanning/trajectory_rgbd_pd_joint_delta_pos_physx_cpu_H5')
data = dataset[320]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"])
axs[1].imshow(data["sensor_data"]["hand_camera"]["depth"])
axs[0].set_title("Failure")
fig.suptitle("Episode 320 - ('Success' if success else 'Failure')", fontsize=10)
plt.show()
```

**Episode 320 (Failure)**  
Peg missed the hole or deviated  
(red section still visible)



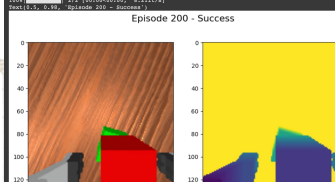






```
from mani_skill_trajectory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file='demo/StackCube-v1/retocomplanning/trajectory_rgbd_pd_joint_delta_pos_physx_cpu_H5')
data = dataset[200]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"])
axs[1].imshow(data["sensor_data"]["hand_camera"]["depth"])
axs[0].set_title("Success")
fig.suptitle("Episode 200 - ('Success' if success else 'Failure')", fontsize=10)
plt.show()
```

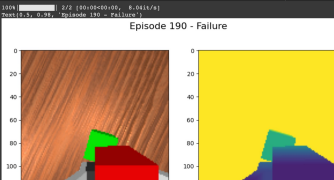
**Episode 200 - Success**



**Episode 200 (Success)**  
Red block stacked on green block in  
correct position  
Significant overlap visible in depth map

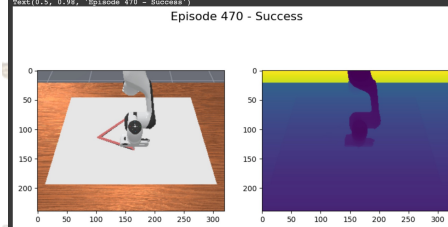
```
from mani_skill_trajectory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file='demo/StackCube-v1/retocomplanning/trajectory_rgbd_pd_joint_delta_pos_physx_cpu_H5')
data = dataset[190]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"])
axs[1].imshow(data["sensor_data"]["hand_camera"]["depth"])
axs[0].set_title("Failure")
fig.suptitle("Episode 190 - ('Success' if success else 'Failure')", fontsize=10)
plt.show()
```


**Episode 190 (Failure)**  
Blocks visibly separated (no stacking)  
Clear gap observed in depth map





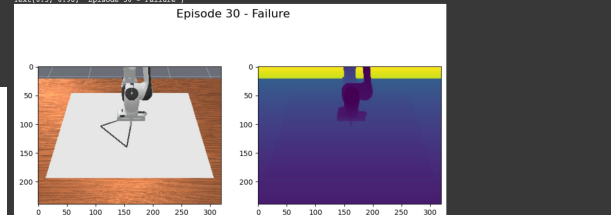
```
from mani_skill.trajecory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file="demos/DrawTriangle-v1/motionplanning/trajecory.rgbd_pd_joint_delta_pos.physx_cpu.h5")
data = dataset[478]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"]["sensor_data"]["base_camera"]["rgb"])
axs[1].imshow(data["obs"]["sensor_data"]["base_camera"]["depth"])
success = data["success"]
fig.suptitle("Episode 478 - ('Success' if success else 'Failure')", fontsize=16)
```



Episode 470 (Success )

Pink lines form a closed complete triangle  
Continuous action trajectory

```
from mani_skill.trajecory.dataset import ManiSkillTrajectoryDataset
import matplotlib.pyplot as plt
dataset = ManiSkillTrajectoryDataset(dataset_file="demos/DrawTriangle-v1/motionplanning/trajecory.rgbd_pd_joint_delta_pos.physx_cpu.h5")
data = dataset[30]
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(data["obs"]["sensor_data"]["base_camera"]["rgb"])
axs[1].imshow(data["obs"]["sensor_data"]["base_camera"]["depth"])
success = data["success"]
fig.suptitle("Episode 30 - ('Success' if success else 'Failure')", fontsize=16)
```



## The significance of collecting and visualizing trajectory data

directly used for the supervised training of **imitation learning** algorithms:

precisely extract successful trajectories as training samples

Success:

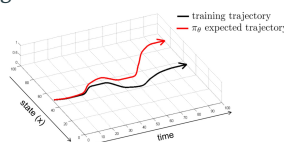
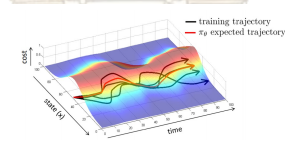
skip the misleading behaviors in failed cases

**robust & reliable**

Failure:

Analyze at which stages the agent often makes mistakes

Design more effective reward functions or intervention mechanisms



Vereshchaka, A. (2019)

## Part 4: Brief Summary



37

Summary



System Environment Preparation

1. From environment setup, rendering to point cloud generation, fully mastered the basic workflow of ManiSkill.
2. Multi-task and multi-view visualization provides the foundation for in-depth observation of agent behaviors.

Reinforcement Learning Training  
Hyperparameter Analysis

1. Explored the training effects of both state-based and vision-based policies on the **PushCube-v1** task
2. Establishes a practical hyperparameter tuning strategy based on a systematic sensitivity analysis, and quantifies the change in training efficiency when using RGB-D versus RGB data under identical settings.

Trajectory Collection & Visualization

1. For three tasks (Peg Insertion, Cube Stacking, Triangle Drawing), we collected and visualized both successful and failed trajectories.
2. Successful trajectories serve as data for imitation learning training; failed ones help with diagnosis and reward redesign their combination improves both robustness and generalization of the learned policy.



"The road ahead is long and boundless;  
We shall search up and down unceasingly."

38

Limitation

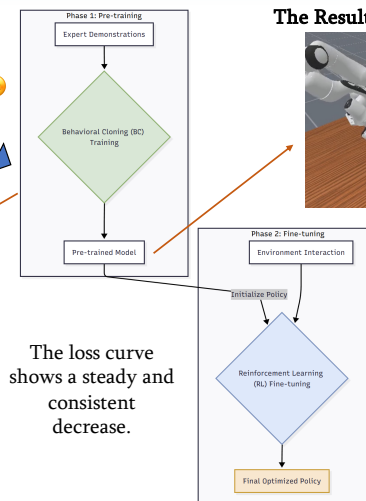
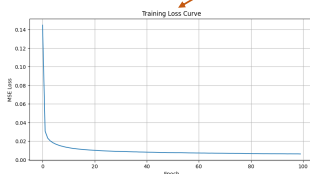


### Challenges in Validating the BC+RL Framework

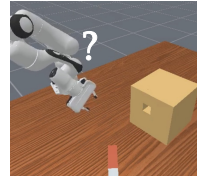
We have finished 😊

您已订阅"Colab Pro"。了解详情  
可用: 5.89个计算单元  
使用率: 大约每小时 5.97 个  
您有 1 个有效会话。

We have almost running out of  
compute units on Colab in task 2



### The Result of the Behavioral Cloning(BC) Training



It seems to do  
something that  
human could not  
understand?

We haven't finished yet 😞

Something wrong with the environment

```

Checking PyTorch connection to GPU...
PyTorch successfully connected to GPU: NVIDIA L4
Attempting to create GPU physics simulation environment...
Environment creation failed. Root error message: 'physx_gpu'
    
```

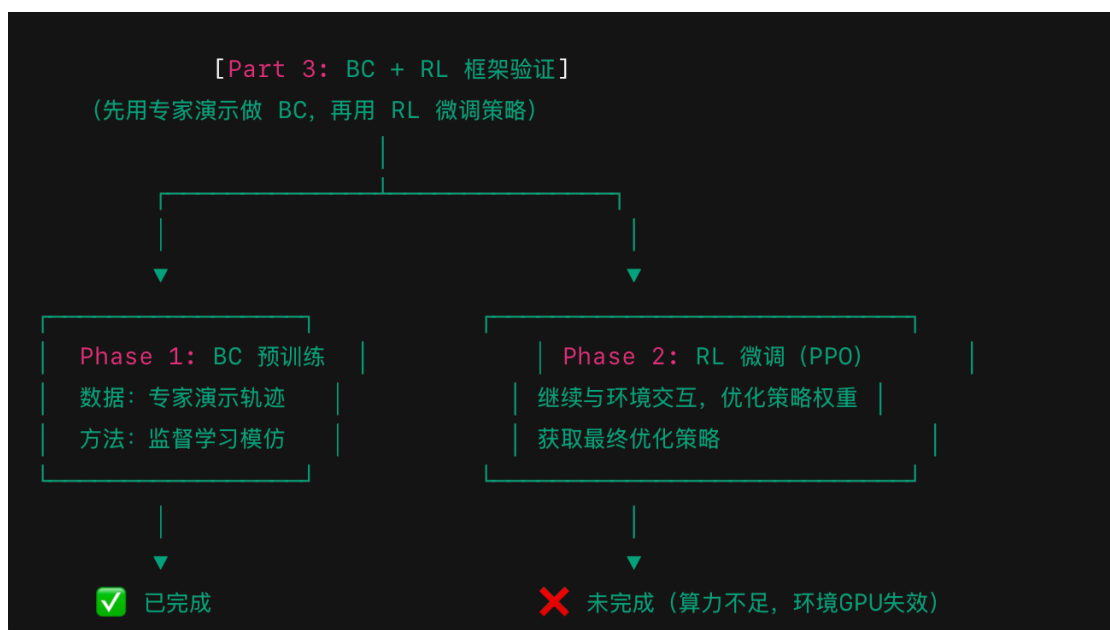
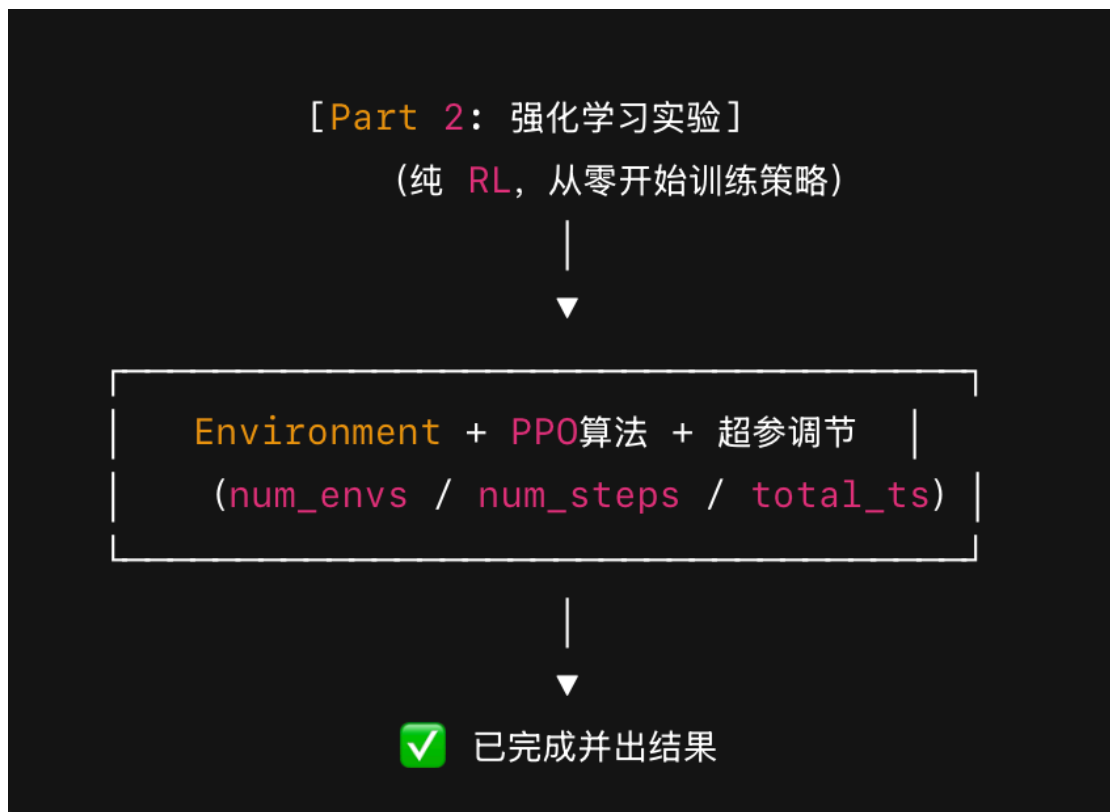
ManiSkill's GPU acceleration failed

```

SPS: 50
SPS: 48
SPS: 51
    
```

Training the model on the CPU is too slow.  
(About 50 sample per second)

关于 limitation:



可以发现我们 Part 3 属于行为克隆的一种, 我们在完成整个 Project Requirements 后想继续在 BC 基础上加上 RL 微调, 但是迫于时间压力及算力不足, 无法完成

GPU 加速失效，被迫用很慢的 CPU（约 50 样本/秒），再加上 Colab 算力快用完，导致 RL 阶段没法跑完！

# 致谢

衷心感谢 香港大学暑期学院 及 香港大学同心基金会数据科学研究院 在本项目期间给予的悉心指导与教学支持。

同时也特别感谢 **Group 6** 每一位同学在整个项目中的辛勤付出与无私协作。

本文内容基于我对结题汇报 PPT 的整理与复盘，旨在梳理项目思路与主要成果。由于个人理解有限，文中难免存在偏差或疏漏，敬请各位批评指正并欢迎讨论交流。

文中配图来源于结题汇报 PPT，该 PPT 由 **Group 6 全体成员** 共同制作，相关图片及内容版权归小组全体成员所有。

—— *Jackson (Chenxu Liu)*