HKU-IDS Project Report

Robotic Manipulation Learning in Simulation

Project Overview: (Authorised)

This project introduces fundamental robot learning techniques for manipulation tasks using ManiSkill as the simulation environment. We will first learn how to interact with the simulation environment, then explore Reinforcement Learning (RL) and Behavior Cloning (BC) algorithms to train robots on object manipulation tasks (e.g., picking, stacking, or assembly).

Presentation Overview:

This presentation focuses on exploring **robotic manipulation learning within simulation environments**, using the **ManiSkill** platform as the core experimental framework. The work is structured into four main parts:

1. **Running and Rendering Simulation Environments** – Setting up CPU/GPU configurations, rendering

RGB/Depth/segmentation images, converting depth data into 3D point clouds, and visualizing multi-task environments.

2. **Reinforcement Learning (RL) Experiments** – Training both state-based and vision-based RL agents on tasks like *PushCube-v1*, analyzing the influence of key hyper parameters (num_envs, num_steps, total_timesteps) on performance, efficiency, and stability.
3. **Demonstration Data Analysis** – Collecting, labelling, and visualizing successful and failed trajectories for various manipulation tasks (e.g., Peg Insertion, Cube Stacking, Triangle Drawing), with implications for imitation learning and reward design.
4. **Summary & Limitations** – Synthesising workflow mastery, experimental insights, and challenges encountered, such as GPU acceleration issues and computational constraints.

Overall, the presentation provides a **complete experimental workflow** from environment preparation to RL training, data analysis, and interpretation, aiming to **deepen understanding of learning dynamics in robotic manipulation** and offer **practical strategies for RL hyper parameters tuning**.

Part 1: Set up the environment (CPU / GPU)

由于时间限制，本项目在云端（Google Colab）初始化并运行 ManiSkill 仿真环境，包括在 CPU 和 GPU 上的运行差异和配置。

1、Setup Environment
2、PegInsertionSide-v1: Create an Environment and Run It

Utilize the "PegInsertionSide-v1" environment within mani_skill, execute a complete episode under the obs_mode="state" setting, and measure the FPS (Frames Per Second) of the simulation, which represents the number of simulation steps executed per second.

名词解读：

1. episode：（类比一局游戏）从环境初始状态开始（env.reset()）不断给动作（env.step(action)）直到任务结束（success、fail 或达到最大步数）

   运行完一局可以：测量 FPS（总步数 ÷ 总时间）并计算成功率

2. FPS (Frames Per Second):

   一帧 = 仿真环境中的一步（一步包括：Agent 给出动作 → 环境更新 → 返回新状态）。

FPS = 每秒能跑多少步。 FPS 决定了你的数据采集速度。

📌 为什么重要：

强化学习是非常“吃数据”的，可能需要几百万到几千万步的交互才能学好一个任务。

如果 FPS 低，比如 50 FPS → 1 秒采集 50 步 → 要采集 100 万步需要 20000 秒（= 5.5 小时）。

如果 FPS 高，比如 1000 FPS → 同样 1 百万步只要 1000 秒（<17 分钟）。

💡 和游戏里的 FPS 类比：

玩《csgo》时，FPS 越高画面越流畅。而这里的 FPS 越高，训练速度越快。

3. 渲染 （rendering）

渲染（Rendering）就是把 3D 场景中的模型数据（几何位置、材质、光照、颜色、相机位置等），通过计算生成二维图像的过程。

在 ManiSkill 里：

“3D 场景”就是你的机器人、桌子、方块、插销等物体的模型

“渲染器”会根据相机定义，把这些 3D 模型转换成给定类型的图像：RGB、深度图、分割图…… (为后续观测者模式奠定基础图像)

每次调用 env.render() 就相当于用相机拍一张当前场景的“照片”

4. obs_mode (observation mode 观测者模式)

作用：告诉 ManiSkill **用什么形式来提供环境的状态信息**给智能体。

参数值不同 → 返回的数据类型和维度不同 → 模型的输入形式不同。

常见 obs_mode 取值：

1） “state” —— 低维状态向量

直接告诉机器人物体位置、姿态（旋转）、速度、角度等明确的数字信息，不给真实图片或画面。

类比：

在 ROS 中，没有仿真 Rviz 我们通过监听来判断小车具体位置，往里传输的是位置坐标数据而不是具体地图等

你玩遥控车，没有盯着车看，而是电脑直接告诉你：“小车在 X=0.53 米, Y=-0.24 米, 朝向 30 度, 速度 0.2 m/s”。

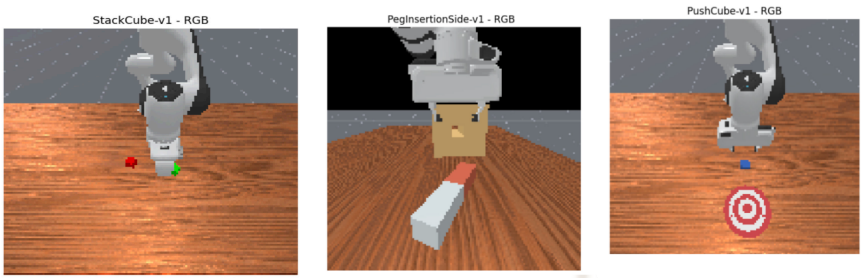
你用这些数据就能“闭着眼”控制它走到目标。

优缺点：

✅ 快速、准确、计算量低

❌ 现实中不容易获得（现实环境里没有人会实时告诉你方块的精确 XYZ 坐标）

2） “rgb” —— 彩色画面



一张三通道彩色图片（RGB：红、绿、蓝），就像普通摄像机拍到的那样。

类比：

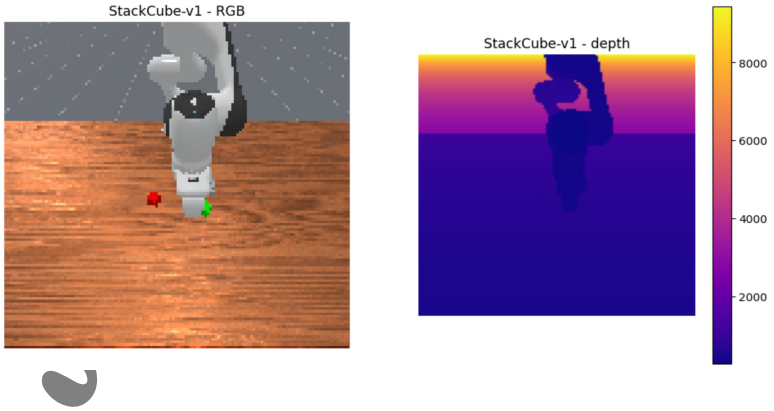
你用手机摄像头拍下桌面，有光照、颜色、纹理，只能通过看画面来推测物体位置。

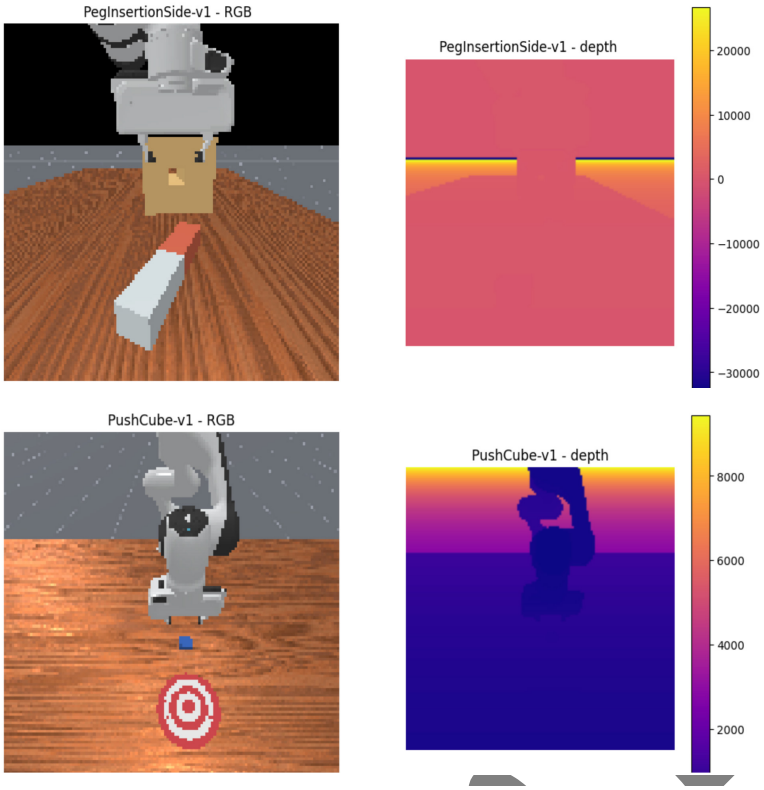
优缺点：

✅ 最贴近人类的感知方式

❌ 图像是高维数据，处理速度慢，需要深度学习模型提取特征

3） “rgbd” —— 彩色 + 深度图





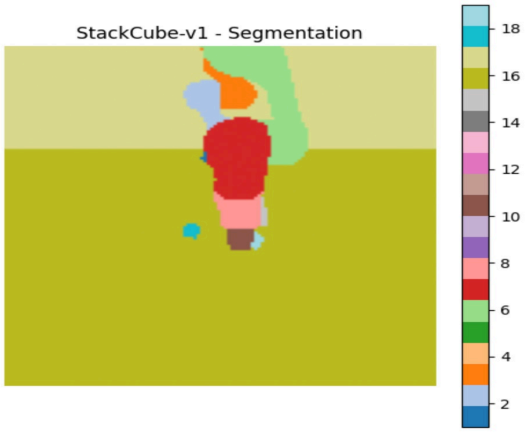
不仅有彩色图片，还多了一张深度图（每个像素到相机的距离）。 深度信息可以让机械臂感知位置是近了还是远了。

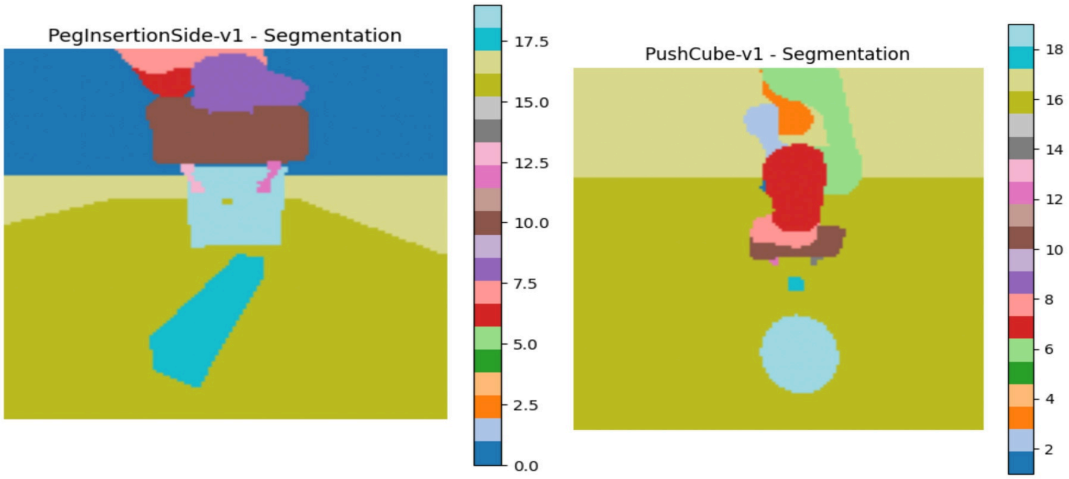
优缺点：

✅ 能直接估算物距，可以生成 3D 模型

❌ 数据比 RGB 大，还要多一个通道处理

4) “segmentation” —— 语义分割图





每个像素不是颜色，而是物体类别 ID（用不同颜色代表不同物体）

类比：

你戴上 AR 眼镜，它自动给你眼中的每个物体贴上颜色标识：

红色=桌子，蓝色=书，绿色=杯子。

优缺点：

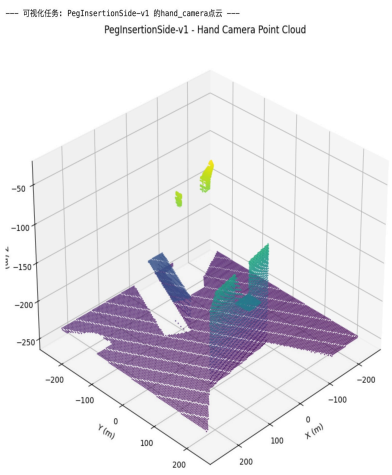
✅ 分辨物体简单，边界清晰

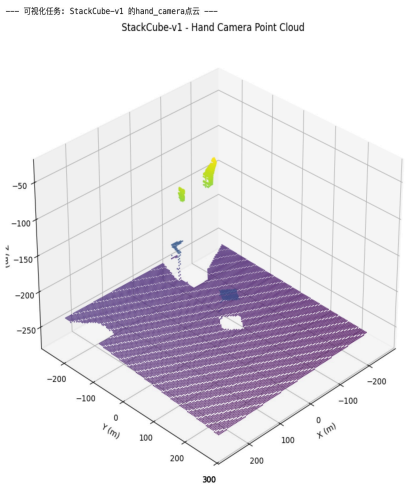
❌ 丢失真实纹理信息（材质、颜色）

5) “pointcloud” —— 三维点云图

```
# Visualize point clouds of multiple tasks
tasks = ["StackCube-v1", "PegInsertionSide-v1]
for task in tasks:
    visualize_hand_camera_pointcloud(task)
```

通过手部相机，将深度图转化为三维点云图





三维点云就是由很多很多的“三维坐标点（X, Y, Z)”组成的集合，这些点的集合可以大致勾勒出物体或场景的形状。

优缺点：

✅ 直接给你 3D 空间信息

❌ 数据处理专业化要求高，没有显式的颜色信息

经过 Part 1，我们已经完成了从环境搭建、任务运行，到渲染出多种感知模式的全过程。这意味着，机器人现在不论是通过数字状态、彩色图像、还是 3D 点云，都已经能够“看到”任务场景。那么，接下来的关键问题是：我们怎么让它“学会”在这个世界中完成任务？

这就是我们要进入的 Part 2：Reinforcement Learning（强化学习）——通过和环境不断交互、获取奖励反馈，逐步让智能体掌握最优策略。我们会先从基于状态的强化学习（State-Based RL）入手，再到基于视觉的强化学习（Visual-Based RL），比较不同观测模式下的学习表现。

Part 2: Reinforcement Learning

1. 前置知识名词：

奖励（Reward, r）

- **定义**：每做一步动作，环境给智能体的反馈评分，说它好或不好。
- **设计很关键**：奖励设计不好，Agent 可能学不到目标行为。
- **类比**：马里奥吃金币 +1 分，踩怪 +5 分，掉坑 -10 分

轨迹（Trajectory / Episode）

- **定义**：智能体从任务开始到结束的交互序（$s_1,a_1,r_1,s_2,a_2,r_2,...$）。
- **终止条件**：达到目标、超时、失败。
- **类比**：马里奥从开局到过关/死亡的整个过程。

策略（Policy, π）

- **定义**：智能体在给定状态下选择动作的概率分布/规则。
- **表示方式**：通常用神经网络建模。
- **类比**：马里奥的“人脑”——当看到怪物，就跳；看到金币，就过去吃。

PPO（Proximal Policy Optimization）

- **定义**：一种强化学习策略梯度方法，更新时会限制策略变化幅度，防止不稳定或策略崩溃。
- **类比**：你不可能一下子从开车很慢变成 F1 赛车手，中间需要适度的渐进调整。

训练超参数（在开始学习之前参数的设置值）

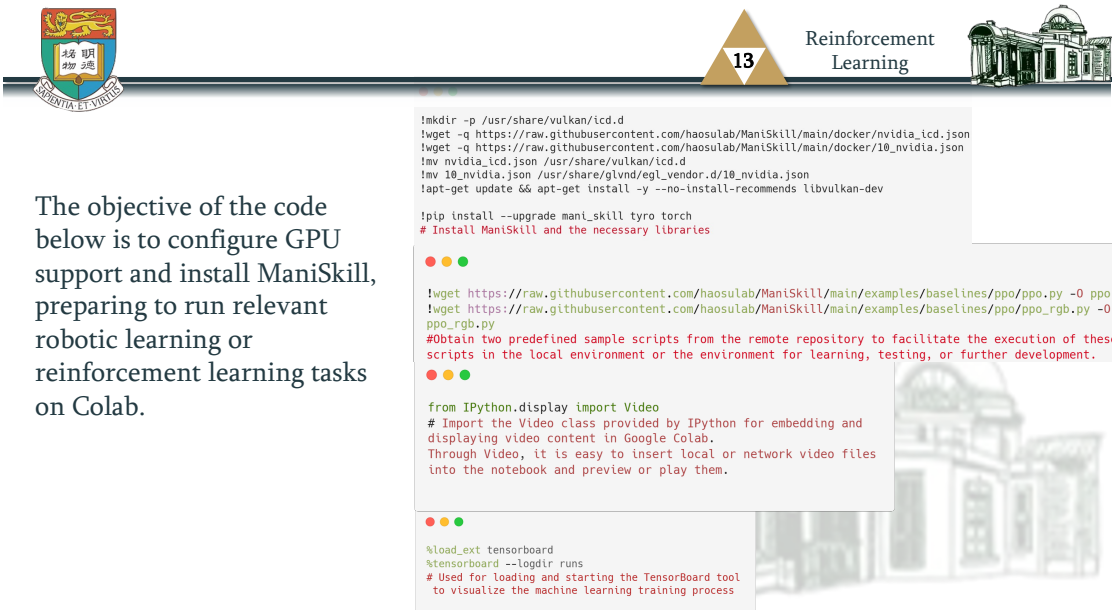
常见的：

- **num_envs**：并行环境数量 → 同时跑多少局游戏来收集数据
- **total_timesteps**：总训练步数 → AI 和环境交互的总次数
- **num_steps**：每个环境在一次更新前运行多少步
- **update_epochs**：一次参数更新时重复使用多少轮采集到的数据
- **num_minibatches**：把数据分成多少批训练

类比：

- num_envs = 同时有多少个马里奥在刷怪
- total_timesteps = 马里奥总共跳多少次
- num_steps = 每个马里奥一次跑多少地图片段后才来开会总结

2. 正式开始内容



1） State-Based RL

核心思想：不给机器人看视频帧，只给它“状态向量”（低维数值）。

优点：输入数据维度小 → 学习快，不用卷积网络 / 更少受干扰

缺点：一般无法直接迁移到现实世界（因为真实硬件里不直接有全部状态）

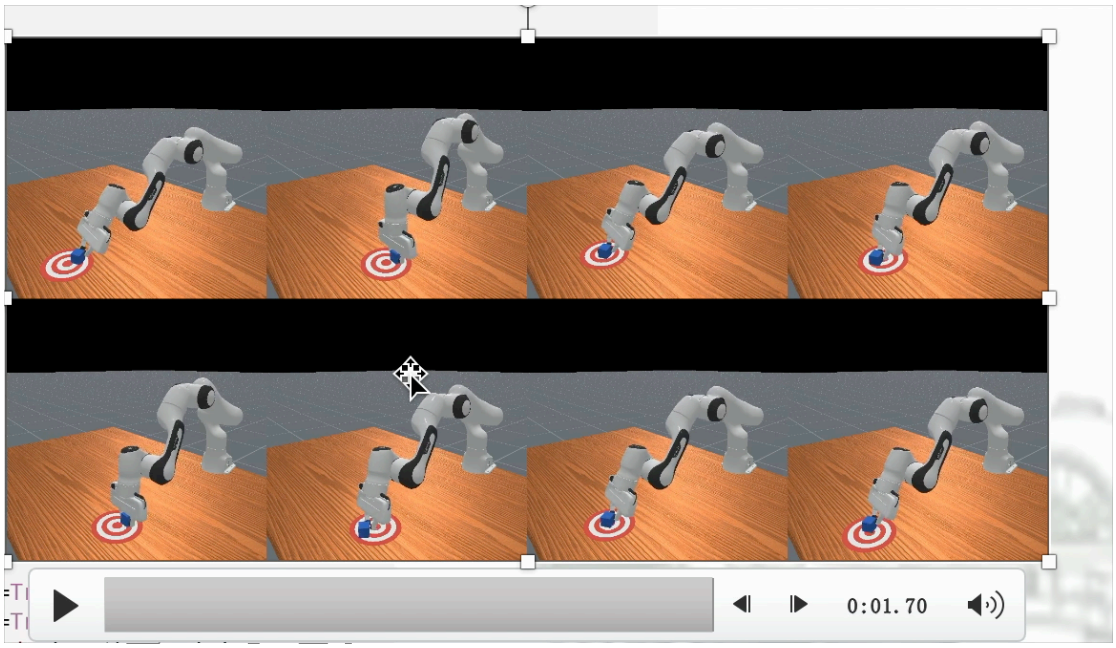
任务： PushCube-v1（推立方体到目标环）

脚本支持并行渲染 → 多环境加速训练

并行评估视频生成 → 每隔固定步数保存视频，快速验证策略效果

```
!python ppo.py --env_id="PushCube-v1" --exp-name="state-pushcube" \
  --num_envs=1280 --update_epochs=8 --num_minibatches=32 \
  --total_timesteps=2000_000 --eval_freq=8 --num-steps=128
#--env_id="PushCube-v1": The push cube task with the specified training environment as ManiSkill.
--exp-name="state-pushcube": Experiment name, used for saving logs and videos.
--num_envs=1280: The number of parallel environments (significantly accelerating training).
--update_epochs=8: The number of iterations during each strategy update.
--num_minibatches=32: The number of small batches of data divided.
--total_timesteps=2000_000: Total training steps (2 million).
--eval_freq=8: Conduct an assessment (generate a video) every 8 updates.
--num-steps=128: Conduct an assessment (generate a video) every 8 updates.
```

效果展示：　(效果还是可以的 😂)



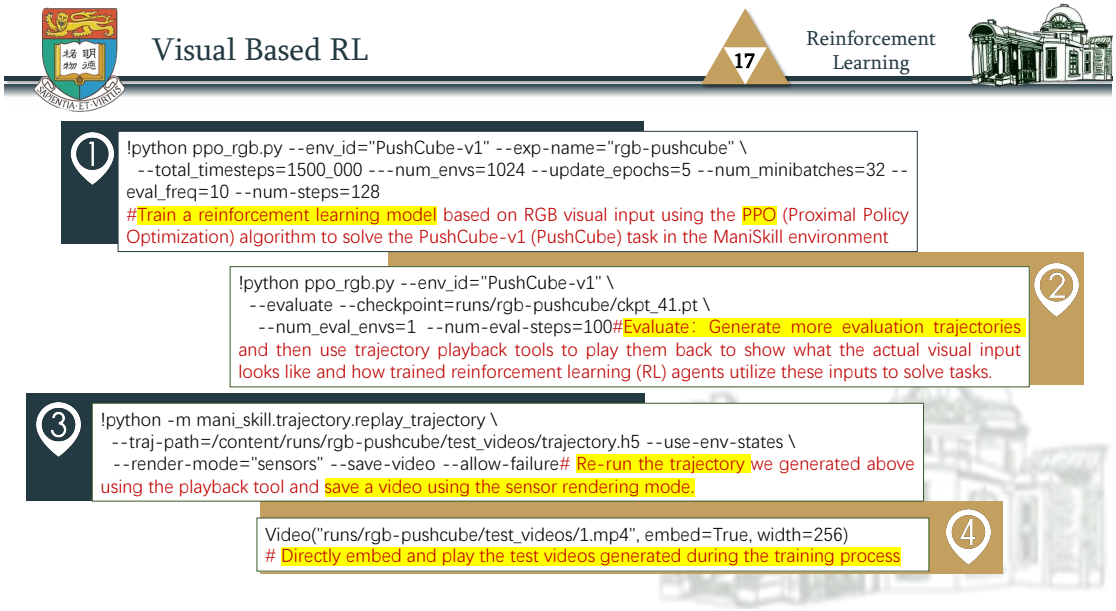
2)　Visual-Based RL

核心思想：运行基于视觉的强化学习主要关注智能体如何通过视觉观察学习策略以最大化奖励

Visual-Based 输入是摄像机画面（RGB / RGBD）

挑战：高维度（128×128×3）/ 需要卷积神经网络提取特征 / 可能需要更多数据、更长训练

优势：更贴近真实感知，能迁移到真实机器人

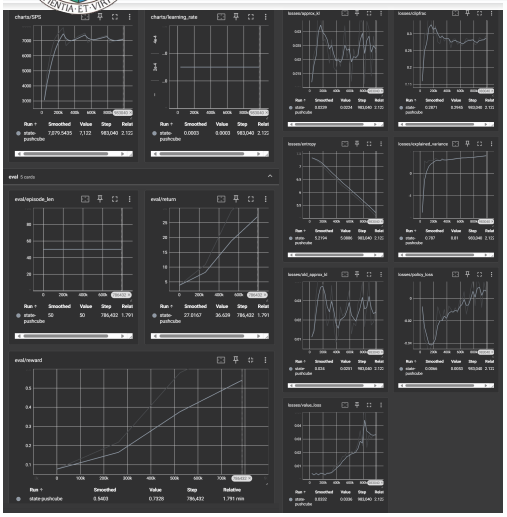


包含四个步骤：训练 -> 评估 -> 回放 + 渲染 -> 嵌入视频

3） 扩展内容：超参调整

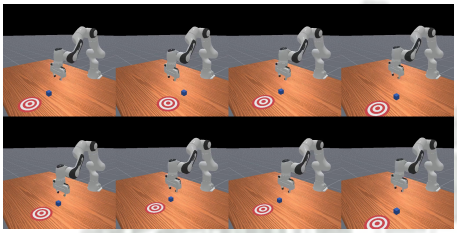
为了研究超参数对训练结果的影响，我们首先确定了一个基线。在特定的配置下，智能体既没有表现出明显的失败，也没有完全掌握，也没有表现出过度拟合的迹象。在这些中线结果的基础上，我们依次改变每个关键超参数，并可视化相应的奖励轨迹和成功率，以严格评估它们的个体影响。

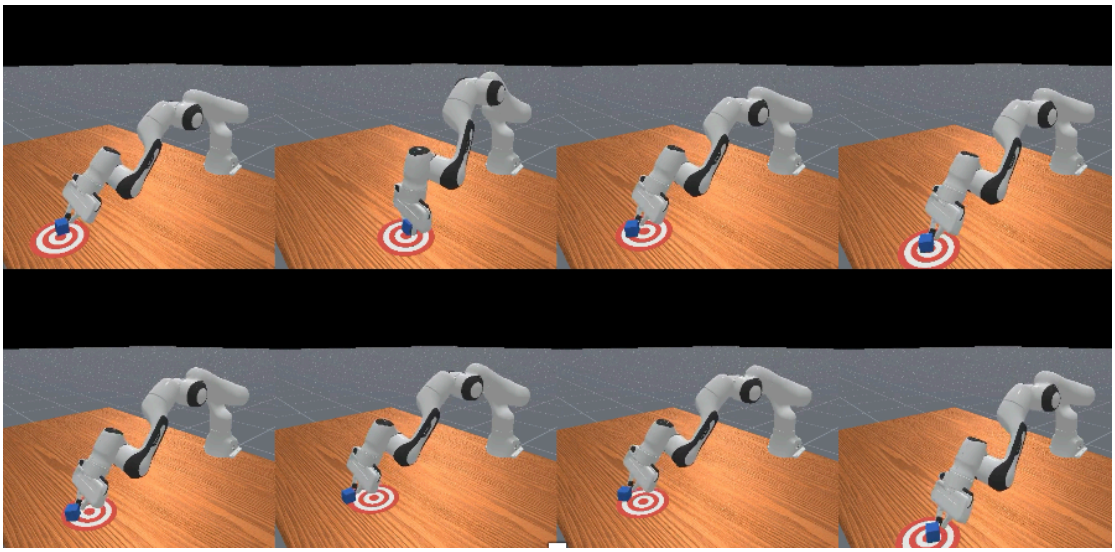
Step 1：确定基线



Initial Baseline Hyperparameters:

```
!python ppo.py --env_id="PushCube-v1" --exp-name="state-pushcube" \
  --num_envs=1024 --update_epochs=8 \
  --total_timesteps=1000_000 --num-steps=32
```





我们可以清晰看到有些还可以，有些效果不是很可以

逐个修改关键超参数，画出奖励曲线、成功率曲线，分析影响

意义：

这种单一参数敏感性分析可以明确：

哪些参数最重要

调优空间在哪；　　从而对后续提高性能很有指导性

在对六个超参数进行比较分析后，我们发现影响强化学习训练过程最显著的是 num_envs、num_steps 和 total_timesteps。其中，num_envs 决定了同时运行多少个并行环境，因此直接关系到数据收集的速度和样本的多样性；num_steps 决定了每个环境在一次策略更新前要运行多少步，这实际上在批量大小和更新频率之间做平衡——步数多，单次更新的样本批量大但更新频率低；total_timesteps 则是整个训练的总交互步数，相当于数据预算，数值越大，策略有更充分的机会收敛到稳定的最优策略。

在这些见解的基础上，我们将通过可视化和定量分析专业指标，如平均情景奖励曲线、贴现回报轨迹、价值函数和策略损失趋势、样本效率和总训练时间，来评估它们对强化学习性能的具体贡献。

（下面所有的图：黑灰线都是同一组数据，灰线是直接绘制的原始评估结果，通常会有比较大的上下波动；黑线是对灰线做了平滑）

黑灰线接近 -> 波动小；模型稳定性高

黑线的位置高低才是最关键的（高就是回报/奖励好）