Step 2: Change num_envs

`num_envs` 是什么?

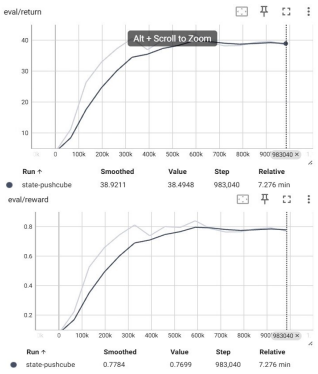
全称：**Number of parallel environments**（并行环境数量）

含义：训练时同时运行多少个独立的环境实例，让 Agent 可以并行收集数据。

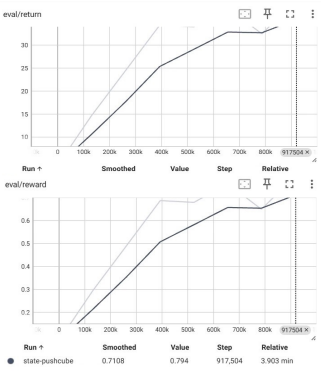
控制效果：

- **数据收集速度**：`num_envs` 越大，每秒拿到的样本越多（FPS更高），总 timesteps 累积得更快。
- **数据多样性**：并行环境起点、随机扰动不一样，可以增加经验的多样性，减少过拟合。
- **但有风险**：
  - 太大：单个环境交互步数减少（策略更新可能更频繁地基于浅轨迹），GPU/CPU 内存压力大。
  - 太小：数据收集慢，训练时间长，更新频率低。

1. Change num_envs

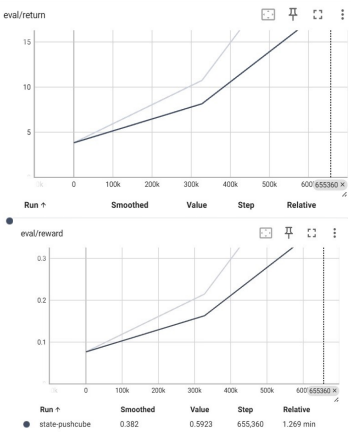


num_envs=256

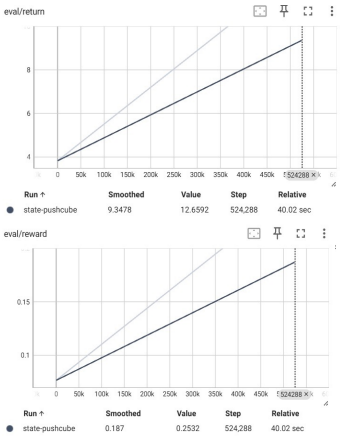


num_envs=512

1. Change num_envs--eval/reward



num_envs=1280



num_envs=2048

(2048failed)

可以看到黑线随着并行数量环境数量增加，总的效果降低

Q1：在 2048　failed 失败怎么看出来的?

(return 图)

数值一直很低，没有明显爬升趋势，可能训练到结束都没超过一个比较合理的水平。这说明智能体几乎没学到有效策略。

（reward 图)

数值也一直保持在接近 0.18 左右（也就是说，任务成功率很低)。在一些 RL benchmark 中，如果 reward 长期处于低位且无明显提升，就会认为这次 training “没有收敛”。

Q2: 为什么在 2048　failed 失败?

策略延迟（Policy lag)：

每次更新前，2048 环境同时收集到的大量样本涉及非常多步的轨迹，而这些轨迹可能是在“旧策略”下生成的 —— 当拿来更新时，策略早就变了，这些数据和当前策略不匹配，降低了学习效率。

大 batch 过于平滑：

PPO 更新时梯度噪声太小，探索信号不够，容易卡在一个劣解（suboptimal policy）。

计算资源分散：

2048 环境并行时，单环境步频率降低（资源竞争），可能导致 FPS 下降，实际更新次数少。

数据相关性下降：

轨迹太短，收集到的数据对策略的改进信息不足。

直观比喻：

想象你让一个小孩学推方块，num_envs 相当于给多少个小孩同时上这个课。

256 个/512 个的时候，大家还能学到差不多的推法，而且平均水平在上升。

2048 个的时候，可能课堂太大、老师精力完全分散，反而没几个真的学会（算法里面的原因就是“policy lag”和过大 batch 导致学习信号稀释）。

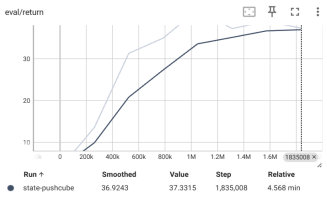
最终分数（reward/return）低，说明任务没完成。

Step 3: Change total_timesteps

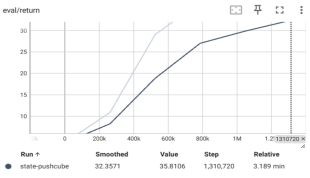
Total timesteps: 整个训练过程中 Agent 和环境交互的总步数

2.Change total_timesteps

eval/return:

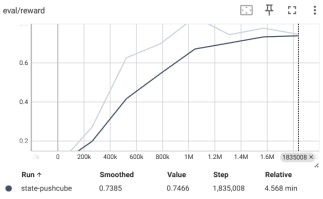


total_timesteps=2000_000

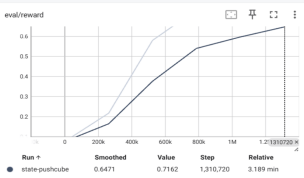


total_timesteps=1500_000

eval/reward:



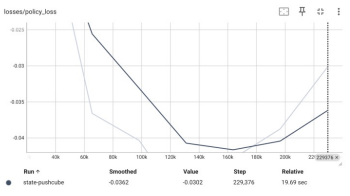
total_timesteps=2000_000

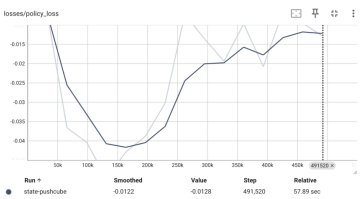


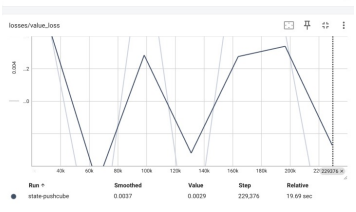
total_timesteps=1500_000

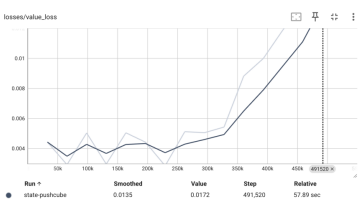
这个可以看到 total_timesteps 越大，整个模型效果越好！

结论：步数更多，模型表现更好，reward 提升明显，说明多给数据可以让策略进一步收敛，但代价是训练时间更长（200 万步比 150 万步多用约 1.3 分钟）。









total_timesteps=2300_00

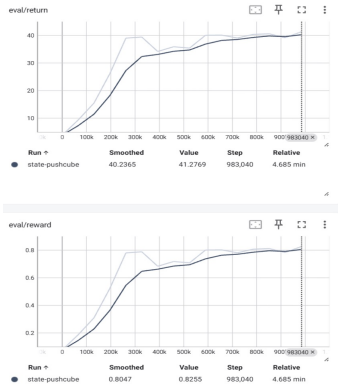
total_timesteps=5000_000

在 PPO 中，policy_loss 和 value_loss 越接近 0，通常说明策略更新幅度越小、值函数预测越准，训练越稳定、收敛程度越高。可以看到，在两个设置下 loss 都持续收敛且波动小，说明训练过程健康；而 500 万步版本的 policy_loss 更接近 0，表明策略更新已经基本停止，收敛更“彻底”，验证了增加训练总步数可以让策略更稳定。

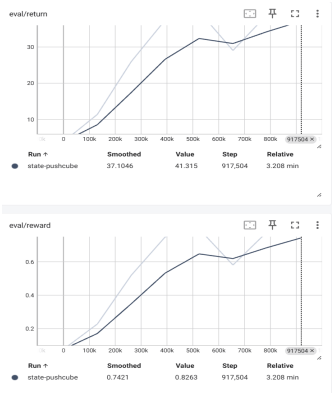
同样逻辑看 Step 4

Step 4: Change total_timesteps

3.Change num_steps

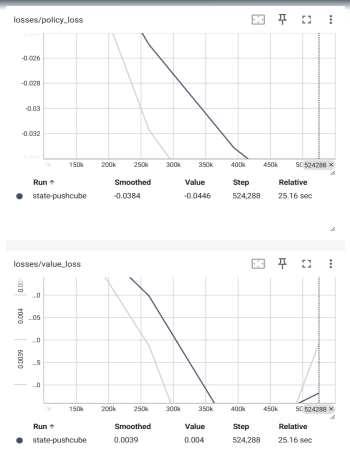


num-steps=8



num-steps=16

num-steps=64



num_steps=128

定义：

在一次策略更新（policy update）之前，每个环境会运行多少步来收集数据。

如果你用 PPO 算法，通常每次更新会把 `num_envs × num_steps` 份数据打成一个 batch 参与优化。

影响：

- **num_steps 小** → 频繁更新，批次样本数相对小，梯度更新更及时，但单次更新的样本代表性可能不足，波动可能更大。
- **num_steps 大** → 单次更新批量大（更新更平滑），梯度估计方差小，可能更稳定，但更新频率降低，策略会更慢地适应新变化（尤其是环境快速变化时）。

可以类比成：

- num_steps 小：学生每天学一点马上复习（即时反馈）
- num_steps 大：攒很多笔记一个月才复习一次（批量信息很多，但反馈滞后）

关于第一幅图：

num_steps=8 收敛效果和最终表现略优于 16

num_steps=8 更新频率更高（虽然单批样本量小），但可能有助于快速适应任务

num_steps=16 虽然每批样本多，训练时间短，但最终成绩差了一点

关于第二幅图：

从 loss 角度看：128 步的批量更大，梯度噪声更小，训练过程更平稳、收敛更稳定；64 步的 value_loss 有明显抖动

不过，步数越大，不一定性能最好——可能会因更新频率降低而失去适应性（得在最终 reward 曲线中验证）