1、Clone与环境配置

第零步：ssh连接并设置学术加速

EB cloud 学术加速页面

```
# 开启加速服务
source /public/bin/network_accelerate

# 关闭加速服务
source /public/bin/network_accelerate_stop
```

第一步：创建与激活环境

这里使用了 `uv` （一个极速 Python 包管理器）来创建环境。

```
# 1. 创建名为 env_isaaclab 的虚拟环境，指定 Python 版本为 3.11
# --seed 参数用于初始化 pip 等基础包
uv venv --python 3.11 --seed env_isaaclab

# 2. 激活刚刚创建的虚拟环境
source env_isaaclab/bin/activate
```

第二步：安装核心依赖 (PyTorch & Isaac Sim)

安装特定版本的 PyTorch 和 NVIDIA Isaac Sim 的 Python 客户端。

```
# 3. 安装指定版本的 PyTorch 和 Torchvision
# --index-url 指定从 PyTorch 官方源下载 CUDA 12.8 对应的版本
pip install -U torch==2.7.0 torchvision==0.22.0 --index-url
https://download.pytorch.org/whl/cu128

# 4. 从 NVIDIA 的 PyPI 源安装 Isaac Sim 核心库
# 指定版本为 5.1.0，并包含所有组件和缓存扩展
pip install "isaacsim[all,extscache]==5.1.0" --extra-index-url
https://pypi.nvidia.com
```

第三步：克隆仓库并安装 Isaac Lab

下载源码并执行项目自带的安装脚本。

```
# 5. 从 GitHub 克隆 Isaac Lab 的源代码仓库
```

```
git clone https://github.com/isaac-sim/IsaacLab.git

# 6. 进入克隆下来的目录
cd IsaacLab

# 7. 运行 Isaac Lab 的安装脚本
# --install (或 -i) 参数表示执行安装流程
./isaaclab.sh --install
```

第四步：运行倒立摆测试

```
python scripts/reinforcement_learning/skrl/train.py /
--task=Isaac-Cartpole-v0
```

```
python scripts/reinforcement_learning/skrl/play.py /
--task=Isaac-Cartpole-v0 --headless
```

2、运行isaaclab

```
# 运行docker容器
./docker/container.py start
./docker/container.py enter
```

```
# 停止容器
python docker/container.py stop
```

3、创建新项目

```
./isaaclab.sh --new
```

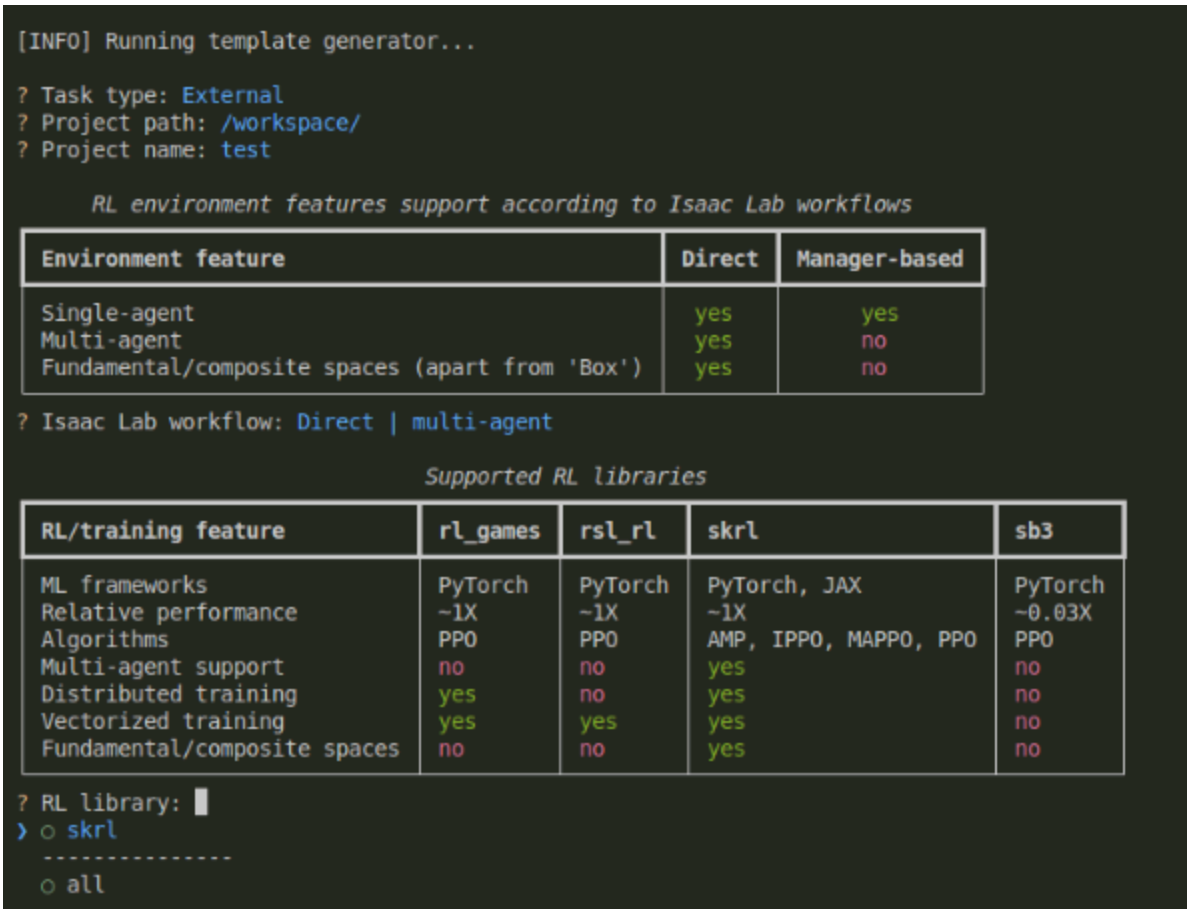
- 设置项目的保存路径
- 设置项目名称（会自动生成对应的 Python 包名）

选项	说明
External	独立于 Isaac Lab 仓库的项目，作为外部扩展运行。可以推送到自己的 GitHub 仓库，便于独立维护和更新

选项	说明
Internal	作为 Isaac Lab 仓库的一部分。仅用于向 Isaac Lab 贡献新任务

选项	说明
Direct	所有实现细节都在环境类中，抽象更少，开发路径最短，适合快速原型开发
Manager-Based	使用模块化定义（ActionManager、ObservationManager、RewardManager 等），更适合需要高度模块化和可扩展性的项目

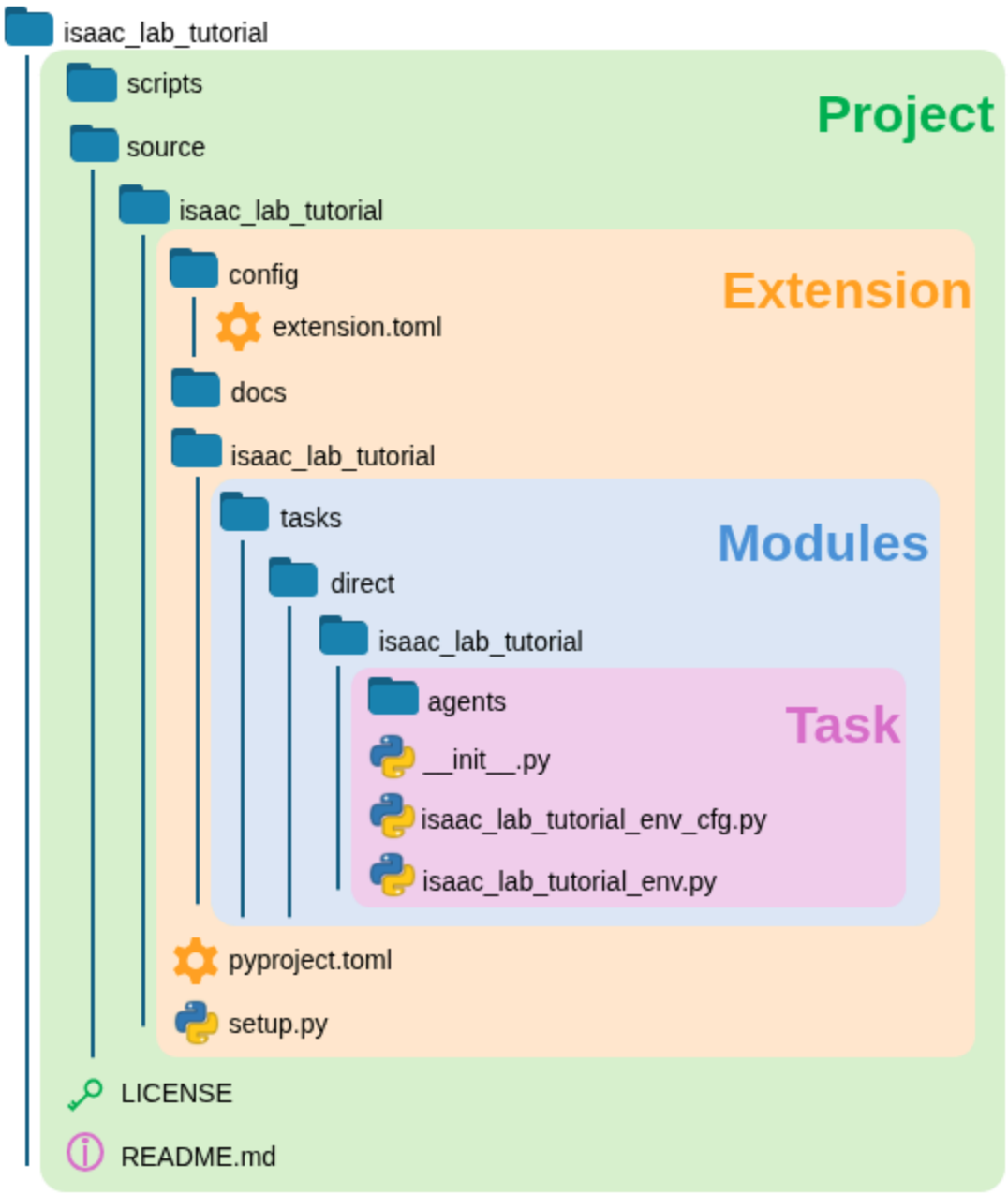
框架	说明
RSL-RL	基于 PPO 算法，支持 GPU 并行训练
SKRL	支持多种算法（PPO, A2C, SAC, TD3 等），使用 PyTorch
Stable-Baselines3	流行的 RL 库，支持 PPO, A2C, SAC 等
RL-Games	支持分布式训练，适合大规模并行



创建后安装项目：

```
python -m pip install -e source/my_robot_task
```

新项目的文件结构：



init文件：

```
gym.register(
    id="Template-Test1-Marl-Direct-v0",
    entry_point=f"{__name__}.test1_marl_env:Test1MarlEnv",
    disable_env_checker=True,
    kwargs={
```

```
        "env_cfg_entry_point": f"{__name__}.test1_marl_env_cfg:Test1MarlEnvCfg",
        "skrl_mappo_cfg_entry_point": f"{agents.__name__}:skrl_mappo_cfg.yaml",
    },
)
```

id = "XXXX" 的 XXXX为task ID

```
python scripts/reinforcement_learning/skrl/train.py /
--task=XXXX
```

```
python scripts/reinforcement_learning/skrl/play.py /
--task=XXXX --headless
```

CLI 重要参数解释：

训练参数 (train.py)

```
python scripts/reinforcement_learning/skrl/train.py \
  --task=Isaac-XXX-v0              # 任务名称（必须）
  --num_envs=64                     # 并行环境数量
  --headless                        # 无头模式（不渲染GUI）
  --video                           # 录制视频
  --enable_cameras                 # 启用离屏渲染
  --run_name my_experiment          # 运行名称（用于日志目录）
  --agent rsl_rl_cfg_entry_point    # 指定agent配置
  --device cpu                      # 设备: cpu / gpu
```

播放参数 (play.py)

```
python scripts/reinforcement_learning/skrl/play.py \
  --task=Isaac-XXX-v0              # 任务名称（必须）
  --num_envs=32                     # 环境数量
  --headless                        # 无头模式
  --use_last_checkpoint            # 使用最新checkpoint
  --checkpoint /path/to/model.pt   # 指定checkpoint路径
  --device cpu                      # 设备
```

参数说明

参数	作用
`--task`	指定要训练/播放的环境ID
`--num_envs`	并行环境数，越多训练越快但显存占用越大
`--headless`	服务器训练必须，禁用GUI渲染
`--video`	录制训练过程视频保存到logs目录
`--run_name`	区分不同实验的运行
`--agent`	选择不同的agent配置
`--use_last_checkpoint`	播放时使用最新保存的模型
`--device`	指定用CPU还是GPU

XXXX_marl_env_cfg.py

```python
# Copyright (c) 2022-2025, The Isaac Lab Project Developers
# ... (版权声明省略)

from isaaclab_assets.robots.cart_double_pendulum import CART_DOUBLE_PENDULUM_CFG
from isaaclab.assets import ArticulationCfg
from isaaclab.envs import DirectMARLEnvCfg
from isaaclab.scene import InteractiveSceneCfg
from isaaclab.sim import SimulationCfg
from isaaclab.utils import configclass

@configclass
class Test1MarlEnvCfg(DirectMARLEnvCfg):
    """
    多智能体倒立摆环境配置类
    继承自 DirectMARLEnvCfg，表示使用直接工作流（无 Manager）的多智能体环境
    """

    # ----------------------
    # 1. 基础环境设置
    # ----------------------
    decimation = 2            # 抽帧数。模拟器每跑 2 步物理计算，智能体才决策 1 次。
                              # 假设物理频率 120Hz，控制频率就是 60Hz。
    episode_length_s = 5.0    # 每一回合(Episode)的最大时长（秒）。

```

```
25      # ----------------------
26      # 2. 多智能体(MARL)定义
27      # ----------------------
28      # 定义在这个环境中有哪几个智能体。
29      # 这里把"滑块(cart)"和"摆杆(pendulum)"拆成了两个独立的智能体。
30      possible_agents = ["cart", "pendulum"]
31
32      # 动作空间维度:
33      # cart: 1个维度 (左右推力)
34      # pendulum: 1个维度 (关节力矩)
35      action_spaces = {"cart": 1, "pendulum": 1}
36
37      # 观测空间维度:
38      # cart: 看到 4 个数 (自身位置, 自身速度, 杆角度, 杆角速度)
39      # pendulum: 看到 3 个数 (相对角度等，具体看 env.py)
40      observation_spaces = {"cart": 4, "pendulum": 3}
41
42      # 全局状态空间维度。设置为 -1 通常表示不使用或者自动推断。
43      state_space = -1
44
45      # ----------------------
46      # 3. 物理模拟参数
47      # ----------------------
48      # dt=1/120: 物理引擎的步长，即每秒计算 120 次物理碰撞和受力。
49      # render_interval: 渲染间隔，与控制频率保持一致。
50      sim: SimulationCfg = SimulationCfg(dt=1 / 120,
    render_interval=decimation)
51
52      # ----------------------
53      # 4. 机器人资产 (Robot Asset)
54      # ----------------------
55      # 加载预定义的 Cart-Double-Pendulum 资产配置
56      # replace(...) 用于修改该机器人在 USD 舞台中的路径，支持正则表达式
57      # "/World/envs/env_.*/Robot" 意味着每个环境里都有一个 Robot
58      robot_cfg: ArticulationCfg =
    CART_DOUBLE_PENDULUM_CFG.replace(prim_path="/World/envs/env_.*/Robot")
59
60      # ----------------------
61      # 5. 场景设置
62      # ----------------------
63      # num_envs=4096: 并行训练的环境数量 (GPU 强的话可以开很大)。
64      # env_spacing=4.0: 每个环境之间隔开 4 米，防止机器人互相穿模干扰。
65      scene: InteractiveSceneCfg = InteractiveSceneCfg(num_envs=4096,
    env_spacing=4.0, replicate_physics=True)
66
67      # ----------------------
```

```
    # 6. 自定义参数 (用于 env.py 中调用)
    # ----------------------
    # --- 关节名称 (需要在 USD 文件里对应) ---
    cart_dof_name = "slider_to_cart"      # 滑块关节
    pole_dof_name = "cart_to_pole"        # 第一级杆关节
    pendulum_dof_name = "pole_to_pendulum" # 第二级杆关节

    # --- 动作缩放 (Action Scaling) ---
    # 神经网络输出通常是 [-1, 1]，需要乘上这个数变成真实的力(牛顿)或力矩(牛米)
    cart_action_scale = 100.0  # [N]
    pendulum_action_scale = 50.0  # [Nm]

    # --- 奖励权重 (Reward Scales) ---
    # 强化学习的核心：定义什么是“好”，什么是“坏”
    rew_scale_alive = 1.0           # 活着（没倒）就给分
    rew_scale_terminated = -2.0     # 死了（倒了）倒扣分
    rew_scale_cart_pos = 0          # 滑块位置惩罚（这里设为0，表示不限制滑块位置）
    rew_scale_cart_vel = -0.01      # 滑块速度惩罚（希望它动作平稳，不要剧烈抖动）
    rew_scale_pole_pos = -1.0       # 杆子角度惩罚（希望杆子垂直，角度越小越好）
    rew_scale_pole_vel = -0.01      # 杆子角速度惩罚
    rew_scale_pendulum_pos = -1.0
    rew_scale_pendulum_vel = -0.01

    # --- 重置条件 (Reset) ---
    # 每次重置时，给关节角度加一点随机噪声，增加鲁棒性
    initial_pendulum_angle_range = [-0.25, 0.25]  # [rad]
    initial_pole_angle_range = [-0.25, 0.25]      # [rad]
    max_cart_pos = 3.0  # 如果滑块跑出 3米 远，就视为失败重置
```

XXXX_marl_env.py

```
# ... (Imports 省略)

class Test1MarlEnv(DirectMARLEnv):
    # 指明配置类是上面定义的 Test1MarlEnvCfg
    cfg: Test1MarlEnvCfg

    def __init__(self, cfg: Test1MarlEnvCfg, render_mode: str | None =
None, **kwargs):
        super().__init__(cfg, render_mode, **kwargs)

        # ----------------------
        # 初始化：获取关节索引
```

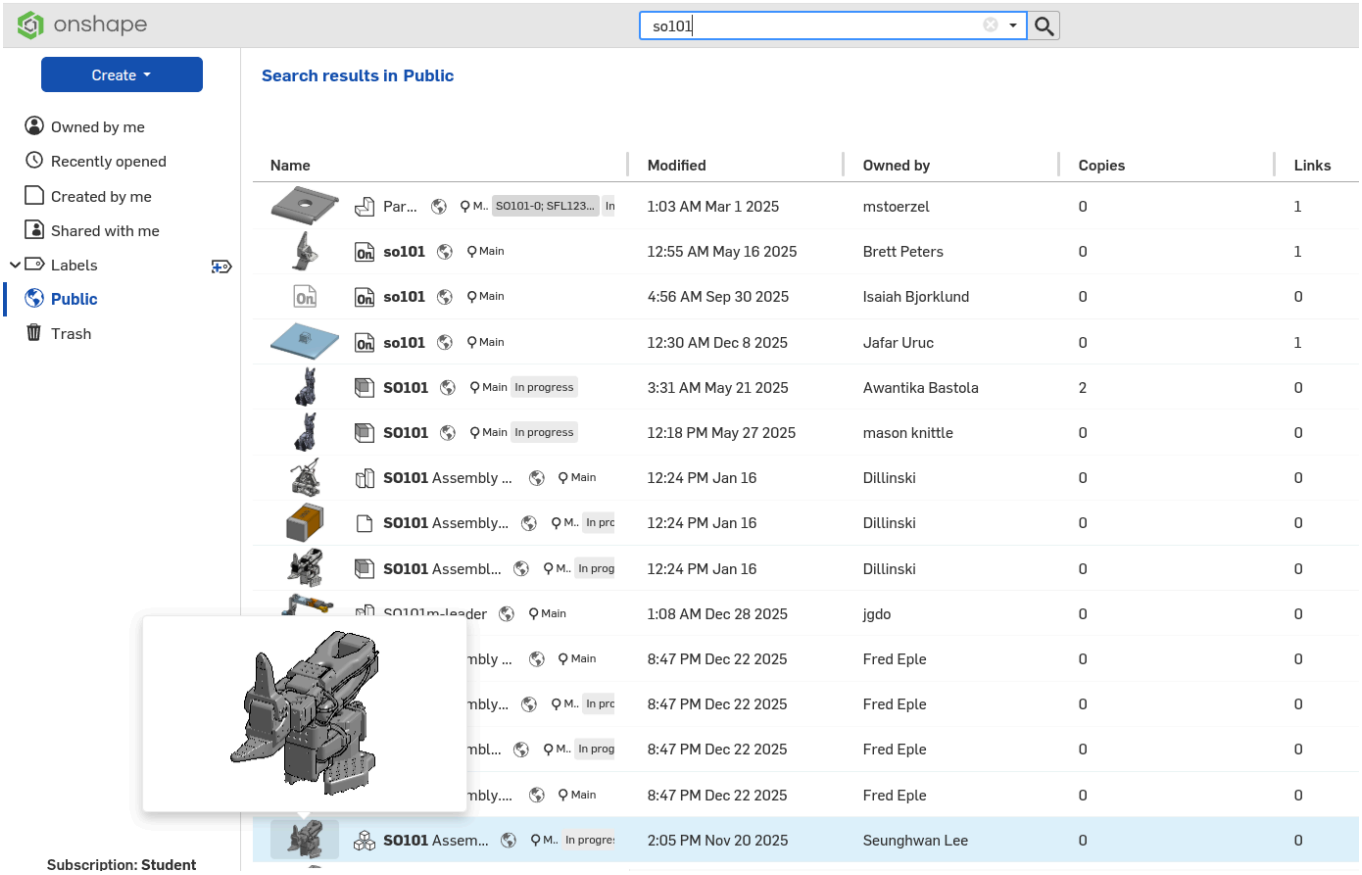
```
12          # ----------------------
13          # 为了高效计算，我们不能每一帧都用字符串去查关节。
14          # 这里一次性查好关节对应的 index (整数)，存在 self 里。
15          self._cart_dof_idx, _ =
    self.robot.find_joints(self.cfg.cart_dof_name)
16          self._pole_dof_idx, _ =
    self.robot.find_joints(self.cfg.pole_dof_name)
17          self._pendulum_dof_idx, _ =
    self.robot.find_joints(self.cfg.pendulum_dof_name)
18
19          # 创建快捷引用，方便后续直接访问关节位置和速度
20          self.joint_pos = self.robot.data.joint_pos
21          self.joint_vel = self.robot.data.joint_vel
22
23      def _setup_scene(self):
24          """场景构建函数：在仿真开始前运行一次"""
25          self.robot = Articulation(self.cfg.robot_cfg)
26          # 1. 生成地面
27          spawn_ground_plane(prim_path="/World/ground",
    cfg=GroundPlaneCfg())
28          # 2. 克隆环境 (把上面的配置复制 4096 份)
29          self.scene.clone_environments(copy_from_source=False)
30          # 3. 碰撞过滤 (CPU 模式下优化性能，GPU 模式通常不需要)
31          if self.device == "cpu":
32              self.scene.filter_collisions(global_prim_paths=[])
33          # 4. 把机器人注册到场景管理器中
34          self.scene.articulations["robot"] = self.robot
35          # 5. 添加光照
36          light_cfg = sim_utils.DomeLightCfg(intensity=2000.0, color=(0.75,
    0.75, 0.75))
37          light_cfg.func("/World/Light", light_cfg)
38
39      def _pre_physics_step(self, actions: dict[str, torch.Tensor]) -> None:
40          """物理步前钩子：保存神经网络传来的动作"""
41          self.actions = actions
42
43      def _apply_action(self) -> None:
44          """应用动作：将神经网络输出转换为物理力矩"""
45          # 给 Cart (滑块) 施加力
46          # actions["cart"] 是网络输出，乘以 cart_action_scale 变成真实的牛顿
47          self.robot.set_joint_effort_target(
48              self.actions["cart"] * self.cfg.cart_action_scale,
    joint_ids=self._cart_dof_idx
49          )
50          # 给 Pendulum (摆杆) 施加力矩
51          self.robot.set_joint_effort_target(
```

```
            self.actions["pendulum"] * self.cfg.pendulum_action_scale,
joint_ids=self._pendulum_dof_idx
        )

    def _get_observations(self) -> dict[str, torch.Tensor]:
        """获取观测：定义智能体能看到什么数据"""
        # 将关节角度归一化到 [-pi, pi] 之间
        pole_joint_pos = normalize_angle(self.joint_pos[:,
self._pole_dof_idx[0]].unsqueeze(dim=1))
        pendulum_joint_pos = normalize_angle(self.joint_pos[:,
self._pendulum_dof_idx[0]].unsqueeze(dim=1))

        observations = {
            "cart": torch.cat(
                (
                    # Cart 看到：自己的位置 + 自己的速度 + 杆的角度 + 杆的角速度
                    self.joint_pos[:,
self._cart_dof_idx[0]].unsqueeze(dim=1),
                    self.joint_vel[:,
self._cart_dof_idx[0]].unsqueeze(dim=1),
                    pole_joint_pos,
                    self.joint_vel[:,
self._pole_dof_idx[0]].unsqueeze(dim=1),
                ),
                dim=-1,
            ),
            "pendulum": torch.cat(
                (
                    # Pendulum 看到：两杆角度之和(可能是绝对角度) + 自己的相对角度
+ 自己的角速度
                    pole_joint_pos + pendulum_joint_pos,
                    pendulum_joint_pos,
                    self.joint_vel[:,
self._pendulum_dof_idx[0]].unsqueeze(dim=1),
                ),
                dim=-1,
            ),
        }
        return observations

    def _get_rewards(self) -> dict[str, torch.Tensor]:
        """计算奖励：调用下面的 JIT 函数计算"""
        # 收集所有需要的数据传给 compute_rewards 函数
        total_reward = compute_rewards(
            self.cfg.rew_scale_alive,
            self.cfg.rew_scale_terminated,
```

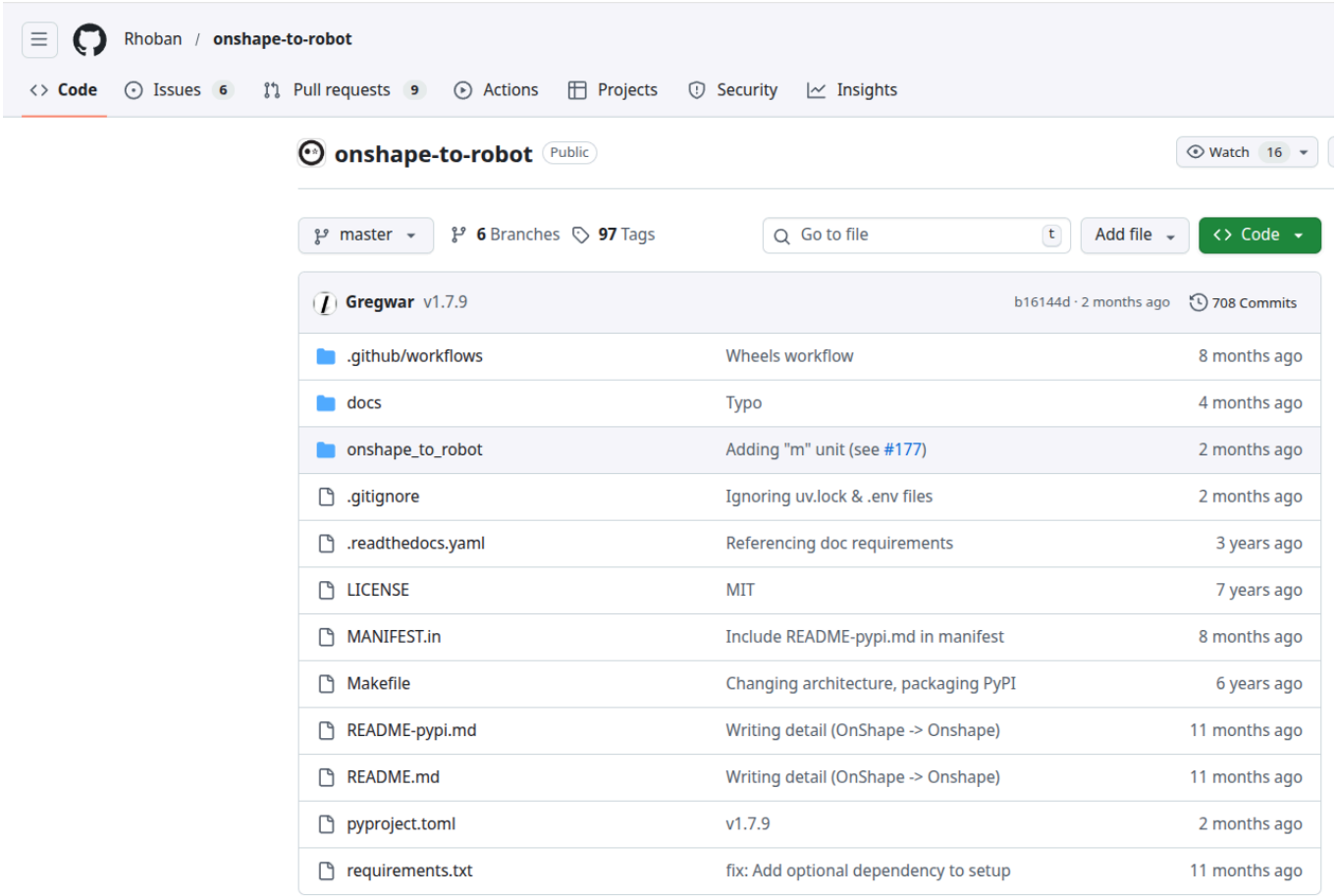
```
            self.cfg.rew_scale_cart_pos,
            self.cfg.rew_scale_cart_vel,
            self.cfg.rew_scale_pole_pos,
            self.cfg.rew_scale_pole_vel,
            self.cfg.rew_scale_pendulum_pos,
            self.cfg.rew_scale_pendulum_vel,
            self.joint_pos[:, self._cart_dof_idx[0]], # 滑块位置
            self.joint_vel[:, self._cart_dof_idx[0]], # 滑块速度
            normalize_angle(self.joint_pos[:, self._pole_dof_idx[0]]), # 杆角度
            self.joint_vel[:, self._pole_dof_idx[0]], # 杆速度
            normalize_angle(self.joint_pos[:, self._pendulum_dof_idx[0]]), # 摆角度
            self.joint_vel[:, self._pendulum_dof_idx[0]], # 摆速度
            math.prod(self.terminated_dict.values()), # 是否结束
        )
        return total_reward

    def _get_dones(self) -> tuple[dict[str, torch.Tensor], dict[str, torch.Tensor]]:
        """判断回合是否结束 (Done)"""
        # 更新一下最新的物理状态
        self.joint_pos = self.robot.data.joint_pos
        self.joint_vel = self.robot.data.joint_vel

        # 1. 超时判断：是否超过了最大时长 (5秒)
        time_out = self.episode_length_buf >= self.max_episode_length - 1

        # 2. 出界判断 (Fail):
        # - 滑块跑出 +/- 3米
        out_of_bounds = torch.any(torch.abs(self.joint_pos[:, self._cart_dof_idx]) > self.cfg.max_cart_pos, dim=1)
        # - 或者 杆子倒得太厉害 (超过 90度，即 pi/2)
        out_of_bounds = out_of_bounds | torch.any(torch.abs(self.joint_pos[:, self._pole_dof_idx]) > math.pi / 2, dim=1)

        # 构建返回值字典
        terminated = {agent: out_of_bounds for agent in self.cfg.possible_agents} # 失败结束
        time_outs = {agent: time_out for agent in self.cfg.possible_agents}       # 时间到结束
        return terminated, time_outs

    def _reset_idx(self, env_ids: Sequence[int] | None):
        """重置环境：当环境 Done 了之后如何复位"""
```

```
        if env_ids is None:
            env_ids = self.robot._ALL_INDICES
        super()._reset_idx(env_ids)

        # 1. 获取默认姿态
        joint_pos = self.robot.data.default_joint_pos[env_ids]

        # 2. 添加随机噪声 (Domain Randomization)
        # 在初始角度上加一点 [-0.25pi, 0.25pi] 的随机值，防止过拟合
        joint_pos[:, self._pole_dof_idx] += sample_uniform(
            self.cfg.initial_pole_angle_range[0] * math.pi,
            self.cfg.initial_pole_angle_range[1] * math.pi,
            joint_pos[:, self._pole_dof_idx].shape,
            joint_pos.device,
        )
        joint_pos[:, self._pendulum_dof_idx] += sample_uniform(
            self.cfg.initial_pendulum_angle_range[0] * math.pi,
            self.cfg.initial_pendulum_angle_range[1] * math.pi,
            joint_pos[:, self._pendulum_dof_idx].shape,
            joint_pos.device,
        )

        joint_vel = self.robot.data.default_joint_vel[env_ids]
        default_root_state = self.robot.data.default_root_state[env_ids]
        default_root_state[:, :3] += self.scene.env_origins[env_ids]

        # 3. 将这些修改后的状态写入到仿真器内存中
        self.joint_pos[env_ids] = joint_pos
        self.joint_vel[env_ids] = joint_vel

        self.robot.write_root_pose_to_sim(default_root_state[:, :7], env_ids)
        self.robot.write_root_velocity_to_sim(default_root_state[:, 7:], env_ids)
        self.robot.write_joint_state_to_sim(joint_pos, joint_vel, None, env_ids)


# ----------------------
# 辅助函数 (使用 JIT 加速)
# ----------------------
# @torch.jit.script 装饰器意味着这部分代码会被编译成 C++ 级运行，极大提升 GPU 上的执行速度

@torch.jit.script
def normalize_angle(angle):
```

```python
    """辅助函数：将角度标准化到 [-pi, pi] 之间"""
    return (angle + math.pi) % (2 * math.pi) - math.pi


@torch.jit.script
def compute_rewards(
    # ... 参数列表 ...
    rew_scale_alive: float,
    # ...
    reset_terminated: torch.Tensor,
):
    """
    具体的奖励计算逻辑
    """
    # 存活奖励：只要没失败(terminated)，每一步都给分。鼓励活得久。
    rew_alive = rew_scale_alive * (1.0 - reset_terminated.float())

    # 死亡惩罚：如果失败了，扣一大笔分。
    rew_termination = rew_scale_terminated * reset_terminated.float()

    # 姿态惩罚：使用 square (平方) 惩罚，意味着离目标越远惩罚越重
    # 希望 pole_pos (杆角度) 接近 0
    rew_pole_pos = rew_scale_pole_pos * torch.sum(torch.square(pole_pos).unsqueeze(dim=1), dim=-1)

    # 希望整个连杆结构也是直的
    rew_pendulum_pos = rew_scale_pendulum_pos * torch.sum(
        torch.square(pole_pos + pendulum_pos).unsqueeze(dim=1), dim=-1
    )

    # 能量/动作惩罚：希望速度小一点，不要剧烈晃动
    rew_cart_vel = rew_scale_cart_vel * torch.sum(torch.abs(cart_vel).unsqueeze(dim=1), dim=-1)
    rew_pole_vel = rew_scale_pole_vel * torch.sum(torch.abs(pole_vel).unsqueeze(dim=1), dim=-1)
    rew_pendulum_vel = rew_scale_pendulum_vel * torch.sum(torch.abs(pendulum_vel).unsqueeze(dim=1), dim=-1)

    # 汇总奖励字典
    total_reward = {
        "cart": rew_alive + rew_termination + rew_pole_pos + rew_cart_vel + rew_pole_vel,
        "pendulum": rew_alive + rew_termination + rew_pendulum_pos + rew_pendulum_vel,
    }
    return total_reward
```



https://github.com/Rhoban/onshape-to-robot.git



https://github.com/TheRobotStudio/SO-ARM100.git

main / SO-ARM100 / Simulation / SO101 /

Gregory119 Update actuator model params (#141)

Name	Last commit message
..	
assets	Add SO101 mujoco and urdf (#80)
README.md	Update URDF (#83)
joints_properties.xml	Add old and new simulation files for so101 (#86)
scene.xml	Add old and new simulation files for so101 (#86)
so101_new_calib.urdf	fix(urdf): fixing multiple issues related to the last URDF update (#117)
so101_new_calib.xml	Update actuator model params (#141)
so101_old_calib.urdf	Updated the new so101 calibration urdf with fixed direction and frame (...
so101_old_calib.xml	Update Mujoco XML files with colors (#115)