```
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.env_util import make_vec_env
from mani_skill.utils import gym_utils
from mani_skill.utils.wrappers.record import RecordEpisode
from mani_skill.vector.wrappers.sb3 import ManiSkillSB3VectorEnv

# Run 64 environments for parallelized learning!
ms3_vec_env = gym.make("PullCube-v1", num_envs=64, control_mode=
'pd_joint_delta_pos',max_episode_steps=100) #control_mode= 'pd_joint_delta_pos'
max_episode_steps = gym_utils.find_max_episode_steps_value(ms3_vec_env)
vec_env = ManiSkillSB3VectorEnv(ms3_vec_env)

# Define PPO agent and train the model
model = PPO("MlpPolicy", vec_env, gamma=0.8, gae_lambda=0.95, n_steps=50,
batch_size=128,
            n_epochs=8, verbose=1)
model.learn(total_timesteps=500_000) # Start learning
model.save("ppo") # Save the model as ppo
vec_env.close()
del model

model = PPO.load("ppo") # Load the saved model


# Visualize the trained agent
eval_vec_env = gym.make("PullCube-v1", num_envs=16, control_mode=
'pd_joint_delta_pos',render_mode="rgb_array") # control_mode= 'pd_joint_delta_pos',
eval_vec_env = RecordEpisode(eval_vec_env, output_dir="Videos/PPO3",
save_video=True,
                             save_trajectory=False,
max_steps_per_video=max_episode_steps)
#eval_vec_env = ManiSkillSB3VectorEnv(eval_vec_env)

# obs = eval_vec_env.reset()

# #Generate actions using the trained expert
# for i in range(max_episode_steps):
#      action, _states = model.predict(obs, deterministic=True)
#      obs, rewards, dones, info = eval_vec_env.step(action)


#evaluating the model
def eval_success_rate(model, eval_env, num_episodes=100):
    success_count = 0

    for episode in range(num_episodes):
        obs = eval_vec_env.reset()[0]
        final=False

        for i in range(max_episode_steps):
            action, _states = model.predict(obs.cpu().numpy(), deterministic=True)
            obs, rewards, dones, _, info = eval_vec_env.step(action)

            #print(f"Info at step {i}: {info}")


            for j in range(len(info)):
                if 'success' in info:
                    success_c = info['success'].cpu().numpy()
                    for k in range(len(success_c)):
                        success = success_c[k]
                        # print('Success is:', success)
                    if success:
                        final=True
                        print(f"Task was successful at step {i}")
                        break

        if not final:
            print("Task was not successfull",episode)
        else:
            print("Task successfull")
            success_count +=1

        print(f"Episode {episode + 1} is done at this stage")

    print(f"\nSuccess Count: {success_count}/{num_episodes} episodes")
    success_rate = (success_count / num_episodes) * 100
    print(f"Success Rate: {success_rate}%")

eval_success_rate(model, eval_vec_env, num_episodes=100)
```