МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Обучение с подкреплением»

Тема: Реализация PPO для среды MountainCarContinuous-v0

Студент гр. 0310 _____ Нагибин И.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2025

Содержание

Цель работы

Написать алгоритм PPO для обучения агента в среде MountainCarContinuous-v0.

Задание

1. Измените длину траектории (steps).
2. Подберите оптимальный коэффициент clip_ratio.
3. Добавьте нормализацию преимуществ.
4. Сравните обучение при разных количествах эпох.

Выполнение работы

Гиперпараметры среды заданы по умолчанию переменной HYPERPARAMS:

```
HYPERPARAMS = {
    "env_name": "MountainCarContinuous-v0",
    "steps_per_trajectory": 1024,
    "clip_ratio": 0.2,
    "gamma": 0.99,
    "lam": 0.95,
    "epochs": 10,
    "batch_size": 64,
    "lr_actor": 3e-4,
    "lr_critic": 1e-3,
    "entropy_coef": 0.01,
    "num_episodes": 300,
    "normalize_advantages": True
}
```

Параметры экспериментов переменной EXPERIMENT_PARAMS:

```
EXPERIMENT_PARAMS = {
    "steps_variants": [512, 1024, 2048],
    "clip_ratio_variants": [0.1, 0.2, 0.3],
    "epoch_variants": [5, 10, 20]
}
```

Таким образом, будут исследованы кол-во шагов на траекторию, различные параметры clip_ratio и кол-во эпох на одну траеторию.

Архитектуры Actor и Critic заданы строго в коде и не имеют отдельных параметров для настройки вне класса.

Архитектуры Actor и Critic идентичны и имею два линейный слоя чередующиеся с слоями гиперболического тангенса.

Для оценки потери используется формула суммарной потери:

total_loss = actor_loss + 0.5 * critic_loss - entropy_coef * entropy_loss

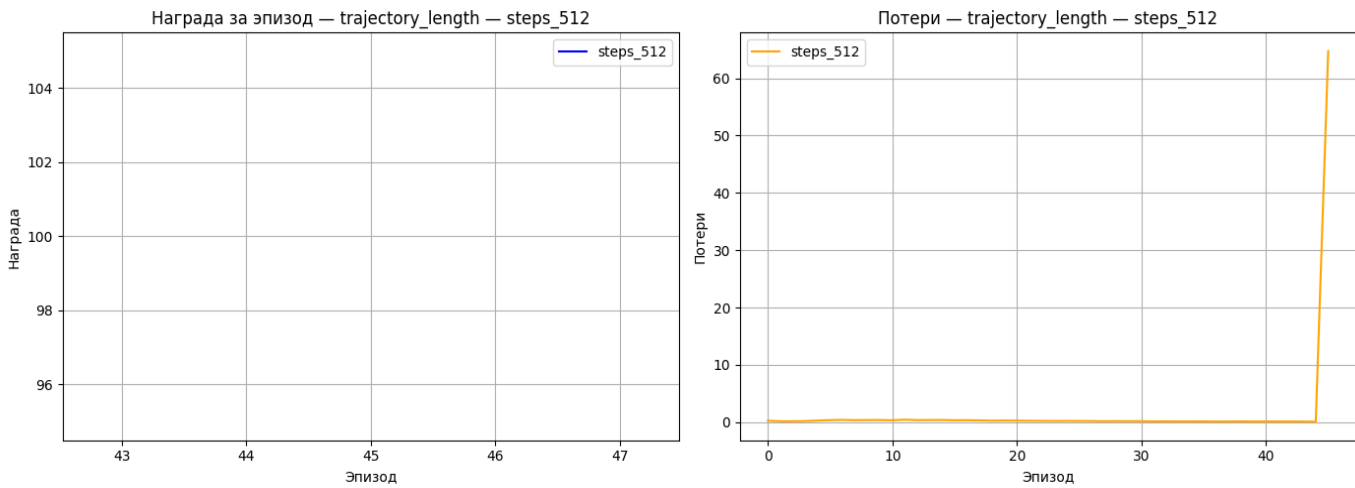

Рисунок 1 — График награды и потери; траектория поделена на 512 шагов
*Примечание: PPO с 512 шагами сошелся за 7 шагов, matplotlib некорректно отображает награду.

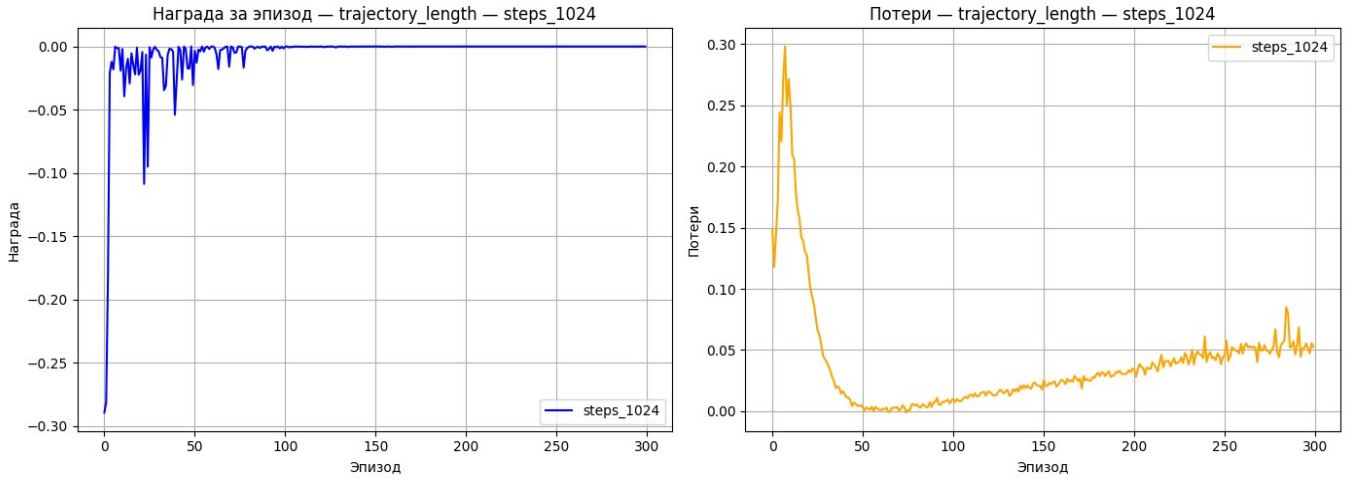

Рисунок 2 — График награды и потери; траектория поделена на 1024 шагов

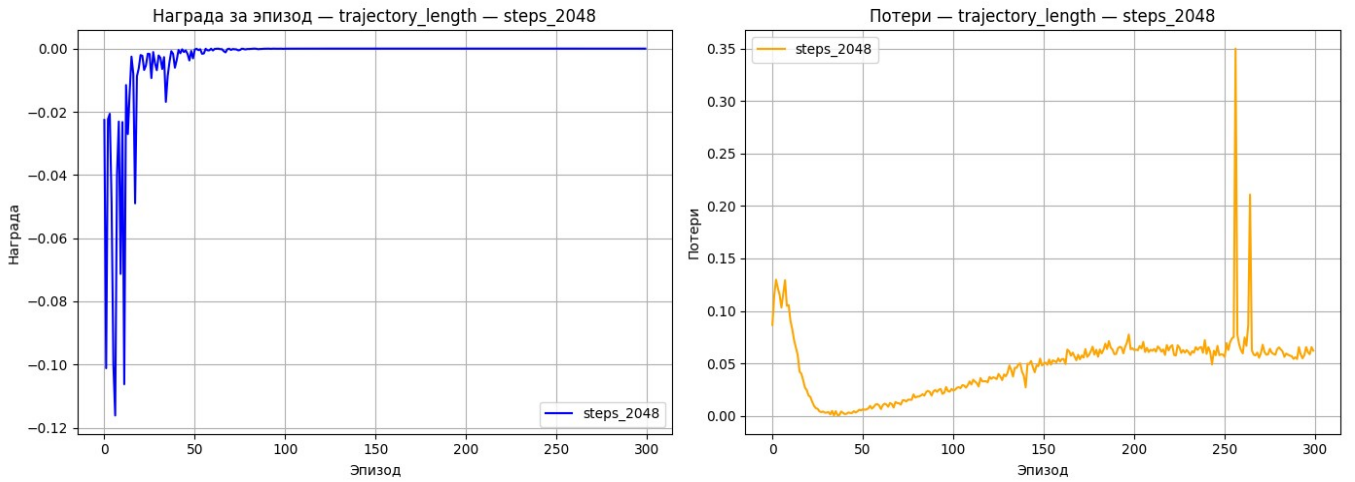

Рисунок 3 — График награды и потери; траектория поделена на 2048 шагов

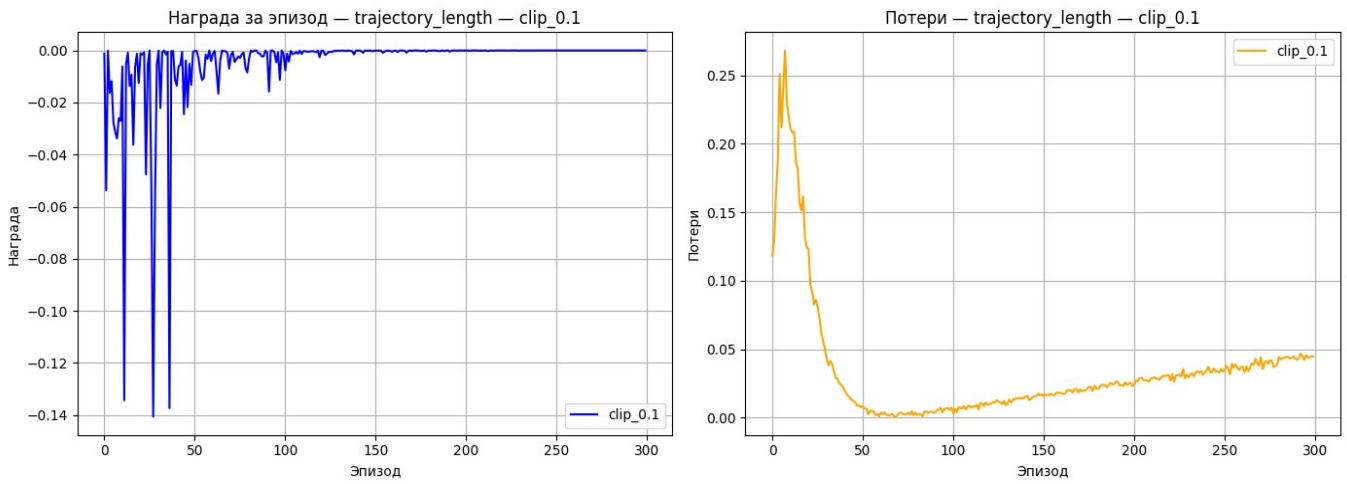

Рисунок 4 — График награды и потери; параметр clip = 0.1

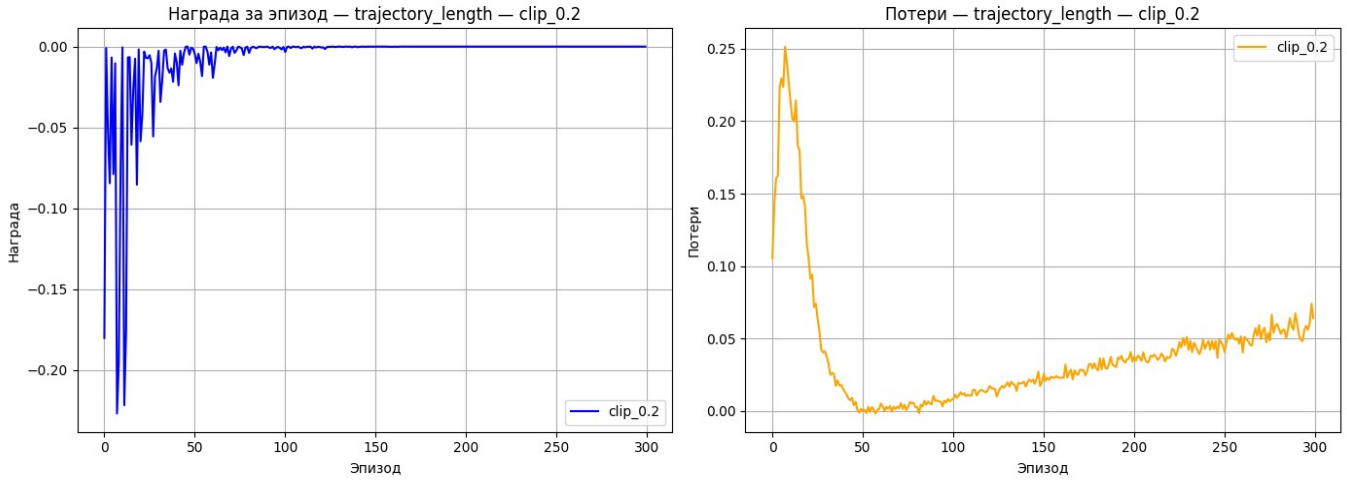

Рисунок 5 — График награды и потери; параметр clip = 0.2

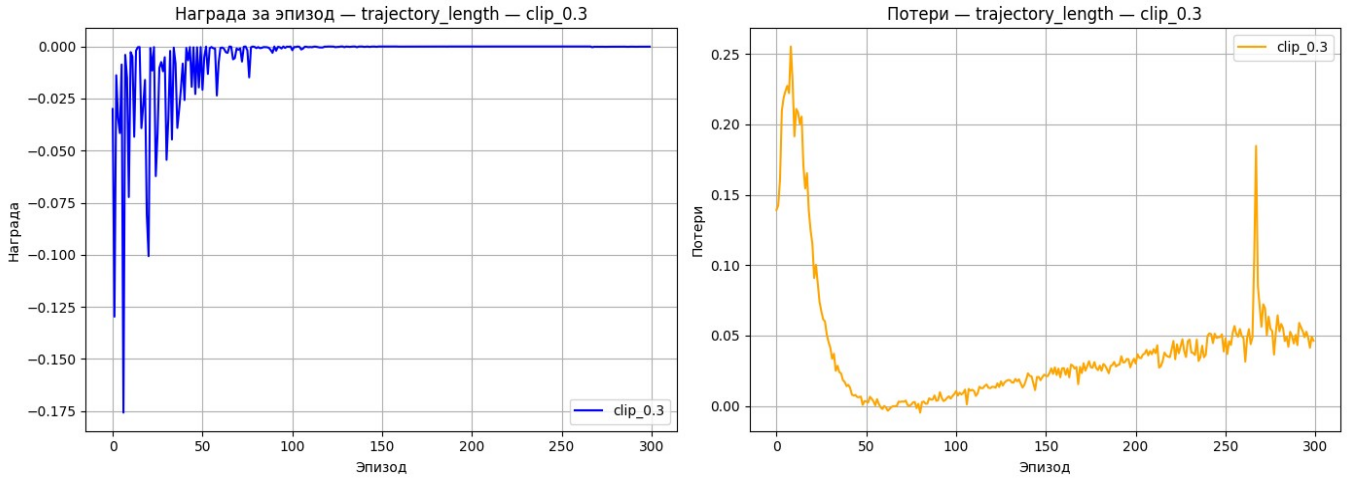

Рисунок 6 — График награды и потери; параметр clip = 0.3

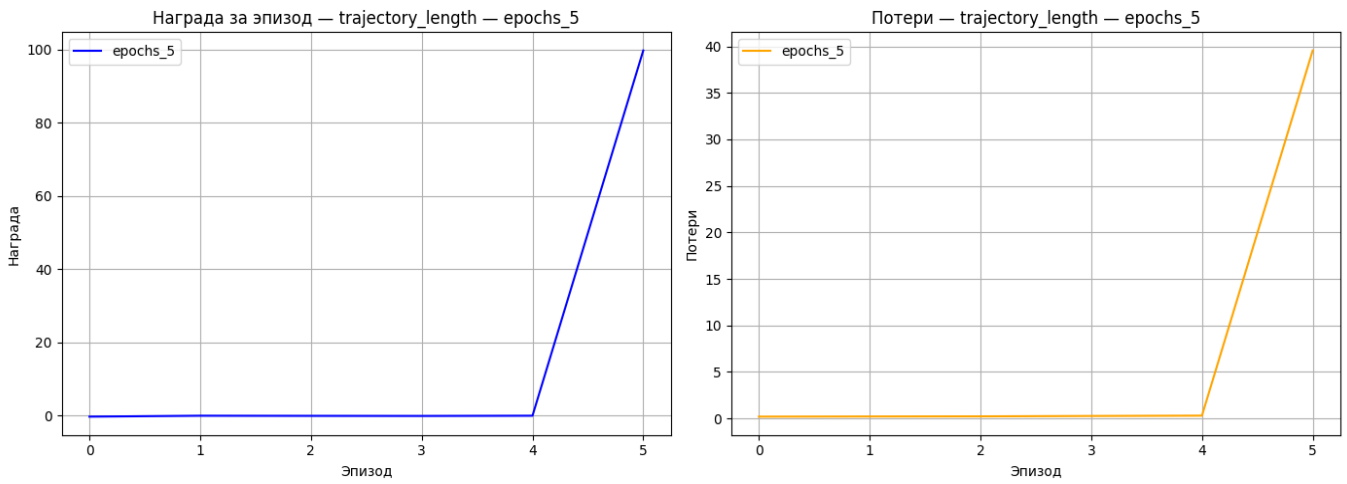

Рисунок 7 — График награды и потери; 5 эпох на траекторию

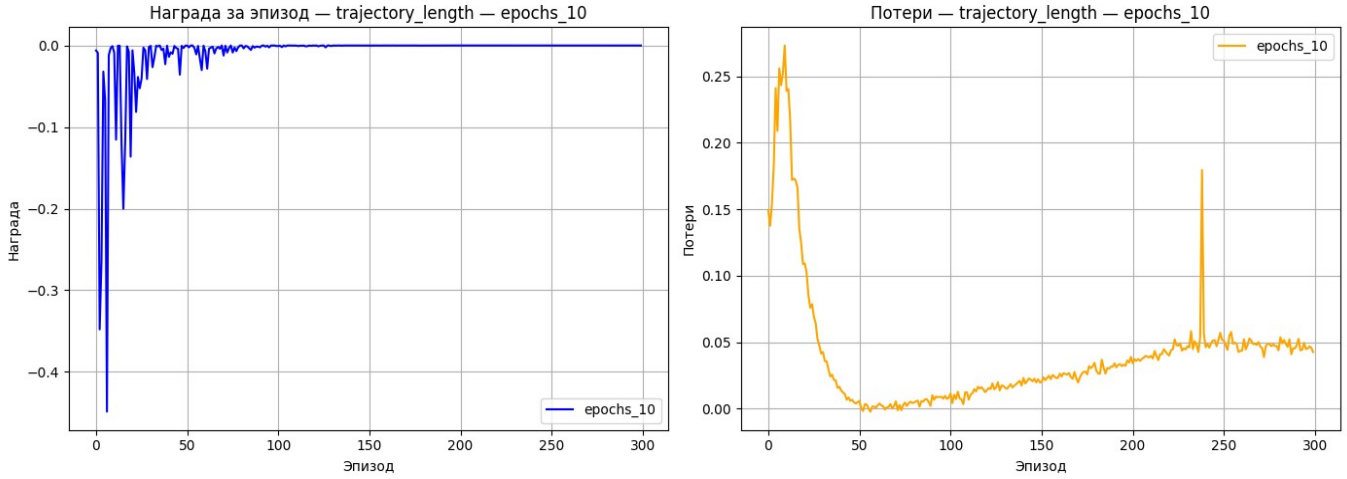

Рисунок 8 — График награды и потери; 10 эпох на траекторию

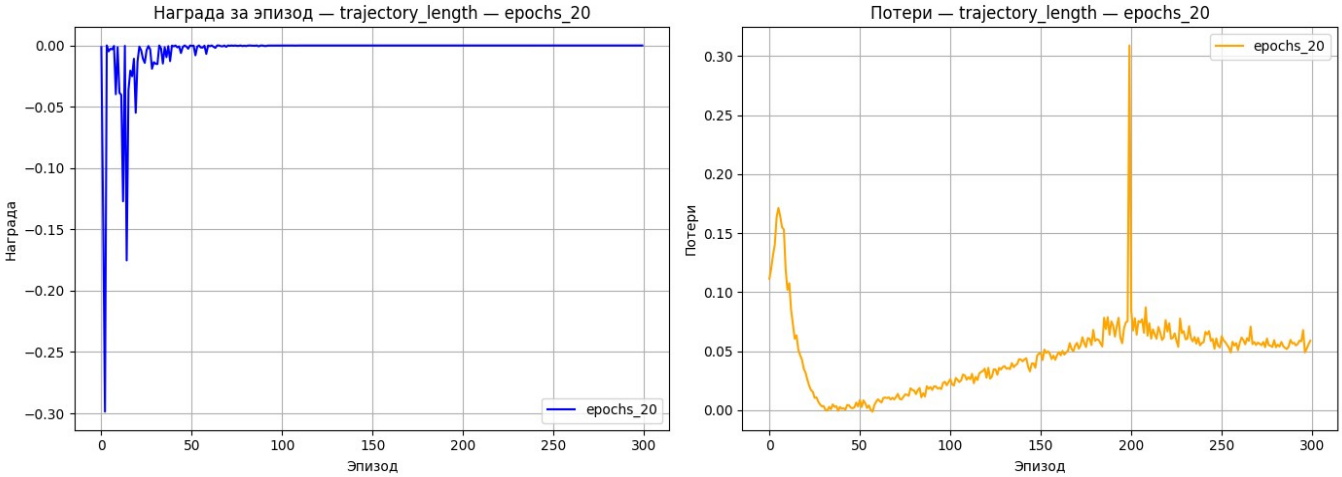

Рисунок 9 — График награды и потери; 20 эпох на траекторию

Выводы

В ходе лабораторной работы был реализован алгоритм PPO для обучения агента в среде MountainCarContinuous-v0.

Таблица 1. Анализ разбиения траектории

Параметр	Награда	Потери	Анализ
steps=512	Быстро получает заветные 100 очков при достижении конца	Резкий скачок после 40 эпизодов	Короткие траектории не обеспечивают достаточной информации для обучения
steps=1024	Награда достигает стабильного 0 на 100 эпизоде	Скачко образно уменьшается и затем медленно растет	Оптимальный баланс между качеством данных и скоростью обучения
steps=2048	Награда достигает стабильного 0 на 50 эпизоде	Потери колеблются, но есть тенденция на стабильные потери 0.05	Длинные траектории увеличивают вычислительные затраты с малым улучшением результата

Таблица 2. Анализ параметра clip_ratio

Параметр	Награда	Потери	Анализ
clip=0.1	Награда достигает стабильного 0 на 100 эпизоде	После 50 эпизода потери медленно растут	Слишком мягкое ограничение замедляет обучение
clip=0.2	Награда достигает стабильного 0 ранее 100 эпизода	После 50 эпизода потери растут быстрее	Оптимальное значение для стабильного и быстрого обучения
clip=0.3	Награда колеблется, но приближется кк 0 ранее чем clip=0.2	Потери нестабильны, скачки	Жёсткое ограничение приводит к нестабильным обновлениям политики

Параметр	Награда	Потери	Анализ
epochs=5	Быстро получает заветные 100 очков при достижении конца	-	Недостаточно итераций для глубокого обучения на данных
epochs=10	Награда достигает 0 примерно на 80 эпизоде, стабильна	Потери сходятся	Оптимальное значение для предотвращения переобучения/недообучения
epochs=20	Награда достигает 0 примерно на 50 эпизоде, стабильна	Потери нестабильны, есть скачки	Переобучение на старых данных, модель "зацикливается"

ПРИЛОЖЕНИЕ А (Исходный код программы)

```
import os
import gymnasium as gym
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.distributions import Normal
from torch.utils.tensorboard.writer import SummaryWriter
from tqdm import tqdm

HYPERPARAMS = {
    "env_name": "MountainCarContinuous-v0",
    "steps_per_trajectory": 1024,
    "clip_ratio": 0.2,
    "gamma": 0.99,
    "lam": 0.95,
    "epochs": 10,
    "batch_size": 64,
    "lr_actor": 3e-4,
    "lr_critic": 1e-3,
    "entropy_coef": 0.01,
    "num_episodes": 300,
    "normalize_advantages": True
}

EXPERIMENT_PARAMS = {
    "steps_variants": [512, 1024, 2048],
    "clip_ratio_variants": [0.1, 0.2, 0.3],
    "epoch_variants": [5, 10, 20]
}
```

```
device = torch.device("cpu")
torch.set_num_threads(os.cpu_count())
torch.set_num_interop_threads(os.cpu_count())

class Actor(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_size=64):
        super(Actor, self).__init__()
        self.shared_net = nn.Sequential(
            nn.Linear(state_dim, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, hidden_size),
            nn.Tanh(),
        )
        self.mean_net = nn.Linear(hidden_size, action_dim)
        self.log_std = nn.Parameter(torch.zeros(action_dim))

    def forward(self, x):
        shared_features = self.shared_net(x)
        mean = self.mean_net(shared_features)
        return mean, self.log_std.exp()

    def get_distribution(self, state):
        mean, std = self.forward(state)
        return Normal(mean, std)

    def act(self, state):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        with torch.no_grad():
            dist = self.get_distribution(state)
            action = dist.sample()
            log_prob = dist.log_prob(action).sum(dim=-1)
```

```
        return action.cpu().numpy().flatten(), log_prob.item()

class Critic(nn.Module):
    def __init__(self, state_dim, hidden_size=64):
        super(Critic, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, 1),
        )

    def forward(self, state):
        return self.net(state).flatten()

def compute_gae(rewards, values, dones, gamma, lam):
    advantages = np.zeros_like(rewards)
    last_gae = 0

    for t in reversed(range(len(rewards))):
        if t == len(rewards) - 1:
            next_value = 0
        else:
            next_value = values[t+1]

        delta = rewards[t] + gamma * next_value * (1 - dones[t])
- values[t]
        advantages[t] = last_gae = delta + gamma * lam * (1 -
dones[t]) * last_gae

    returns = advantages + values
    return returns, advantages
```

```
def collect_trajectories(env, policy, critic,
steps_per_trajectory):
    states, actions, log_probs, rewards, dones = [], [], [], [],
[]
    episode_rewards = []

    state, _ = env.reset()

    for _ in range(steps_per_trajectory):
        action, log_prob = policy.act(state)
        next_state, reward, terminated, truncated, _ =
env.step(action)
        done = terminated or truncated

        states.append(state)
        actions.append(action)
        log_probs.append(log_prob)
        rewards.append(reward)
        dones.append(done)

        state = next_state

        if done:
            episode_rewards.append(reward)
            state, _ = env.reset()

    states = np.array(states)
    actions = np.array(actions)
    log_probs = np.array(log_probs)
    rewards = np.array(rewards)
    dones = np.array(dones)
```

```
    with torch.no_grad():
        values = critic(torch.FloatTensor(states).to(device)).cpu().numpy()

    returns, advantages = compute_gae(rewards, values, dones, HYPERPARAMS["gamma"], HYPERPARAMS["lam"])

    if HYPERPARAMS["normalize_advantages"]:
        advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)

    return {
        "states": states,
        "actions": actions,
        "log_probs": log_probs,
        "returns": returns,
        "advantages": advantages,
        "episode_rewards": np.array(episode_rewards)
    }

def train_ppo(config):
    env = gym.make(config["env_name"])
    state_dim = 2
    action_dim = 1

    actor = Actor(state_dim, action_dim).to(device)
    critic = Critic(state_dim).to(device)

    actor_optimizer = optim.Adam(actor.parameters(), lr=config["lr_actor"])
    critic_optimizer = optim.Adam(critic.parameters(), lr=config["lr_critic"])

    all_rewards = []
```

```
    all_losses = []

    for episode in tqdm(range(config["num_episodes"])):
        batch = collect_trajectories(env, actor, critic,
config["steps_per_trajectory"])

        states = torch.FloatTensor(batch["states"]).to(device)
        actions = torch.FloatTensor(batch["actions"]).to(device)
        old_log_probs =
torch.FloatTensor(batch["log_probs"]).to(device)
        returns = torch.FloatTensor(batch["returns"]).to(device)
        advantages =
torch.FloatTensor(batch["advantages"]).to(device)

        dataset_size = states.size(0)
        indices = np.arange(dataset_size)

        iteration_loss = []

        for epoch in range(config["epochs"]):
            np.random.shuffle(indices)

            for start in range(0, dataset_size,
config["batch_size"]):
                end = start + config["batch_size"]
                if end > dataset_size:
                    end = dataset_size

                mini_indices = indices[start:end]
                mini_states = states[mini_indices]
                mini_actions = actions[mini_indices]
                mini_old_log_probs = old_log_probs[mini_indices]
                mini_returns = returns[mini_indices]
```

```
                mini_advantages = advantages[mini_indices]

                dist = actor.get_distribution(mini_states)
                new_log_probs =
dist.log_prob(mini_actions).sum(dim=-1)

                ratio = torch.exp(new_log_probs -
mini_old_log_probs)

                surrogate1 = ratio * mini_advantages
                surrogate2 = torch.clamp(ratio, 1 -
config["clip_ratio"], 1 + config["clip_ratio"]) *
mini_advantages
                actor_loss = -torch.min(surrogate1,
surrogate2).mean()

                value_estimates = critic(mini_states)
                critic_loss = ((value_estimates - mini_returns)
** 2).mean()

                entropy_loss = dist.entropy().mean()

                total_loss = actor_loss + 0.5 * critic_loss -
config["entropy_coef"] * entropy_loss

                actor_optimizer.zero_grad()
                critic_optimizer.zero_grad()
                total_loss.backward()
                actor_optimizer.step()
                critic_optimizer.step()

                # all_losses.append(total_loss.item())

                iteration_loss.append(total_loss.item())
```

```
        avg_loss = np.mean(iteration_loss)
        all_losses.append(avg_loss)
        avg_reward = np.mean(batch["episode_rewards"])
        all_rewards.append(avg_reward)

        if avg_reward >= 90:
            print(f"\nЗадача выполнена за {episode} эпизодов!")
            break

    return {"reward": all_rewards, "loss": all_losses}

def plot_results(results, title_suffix=""):
    os.makedirs("fig", exist_ok=True)

    for name, data in results.items():
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

        ax1.plot(data["reward"], label=name, color="blue")
        ax1.set_title(f"Награда за эпизод — {title_suffix} —
{name}")
        ax1.set_xlabel("Эпизод")
        ax1.set_ylabel("Награда")
        ax1.grid(True)
        ax1.legend()

        ax2.plot(data["loss"], label=name, color="orange")
        ax2.set_title(f"Потери — {title_suffix} — {name}")
        ax2.set_xlabel("Эпизод")
        ax2.set_ylabel("Потери")
        ax2.grid(True)
        ax2.legend()
```

```
        plt.tight_layout()
        fig.savefig(f"fig/{title_suffix}_{name}.png")
        plt.close(fig)

def run_experiments():

    steps_results = {}
    for steps in EXPERIMENT_PARAMS["steps_variants"]:
        config = HYPERPARAMS.copy()
        config["steps_per_trajectory"] = steps
        config["env_name"] = "MountainCarContinuous-v0"
        print(f"\nЗапуск эксперимента с {steps} шагами на
траекторию")
        result = train_ppo(config)
        steps_results[f"steps_{steps}"] = result
    plot_results(steps_results, "trajectory_length")

    clip_results = {}
    for clip in EXPERIMENT_PARAMS["clip_ratio_variants"]:
        config = HYPERPARAMS.copy()
        config["clip_ratio"] = clip
        print(f"\nЗапуск эксперимента с коэффициентом отсечения
{clip}")
        result = train_ppo(config)
        clip_results[f"clip_{clip}"] = result
    plot_results(clip_results, "trajectory_length")

    epoch_results = {}
    for epochs in EXPERIMENT_PARAMS["epoch_variants"]:
        config = HYPERPARAMS.copy()
        config["epochs"] = epochs
        print(f"\nЗапуск эксперимента с {epochs} эпохами PPO")
```

```
            result = train_ppo(config)
            epoch_results[f"epochs_{epochs}"] = result
        plot_results(clip_results, "trajectory_length")



def main():
    torch.manual_seed(42)
    np.random.seed(42)

    print("\nТренировка с параметрами по умолчанию:")
    config = HYPERPARAMS.copy()
    result = train_ppo(config)

    run_experiments()

if __name__ == "__main__":
    main()
```