НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Reinforcement Learning

Recording Agent Behavior: Saving videos and performance data

Семінар

Виконав(ла):
Богдан Заранік
Група:
ІТ-41мн
Курс:
2

30 листопада 2025 р.

1 Recording Agent Behavior: Saving videos and performance data

1.1 Gymnazium

Gymnasium — це сучасна бібліотека для навчання з підкріпленням, яка є наступником OpenAI Gym. [1] Її головна мета — надати зручний та уніфікований інтерфейс для роботи з різними середовищами. Бібліотека визначає стандартний цикл взаємодії: агент отримує стан, обирає дію, після чого середовище повертає новий стан, винагороду та ознаку завершення епізоду. Gymnasium підтримує широкий набір середовищ, забезпечує відтворюваність експериментів і дозволяє легко будувати та тестувати різні алгоритми RL.

1.2 Lunar Lander v3

LunarLander — це двовимірне середовище, у якому агент повинен посадити місячний модуль у визначену зону. [2] Стан складається з кількох параметрів: положення, швидкостей, кута та інформації про торкання опор поверхні. Дії дискретні — увімкнення основного або бічних двигунів. Винагорода стимулює стабільне зниження швидкості, правильну орієнтацію та точну посадку, а аварія або сильне відхилення дають штраф. Завдання є типовою задачею керування, де агент вчиться балансувати між стабільністю та точністю посадки.

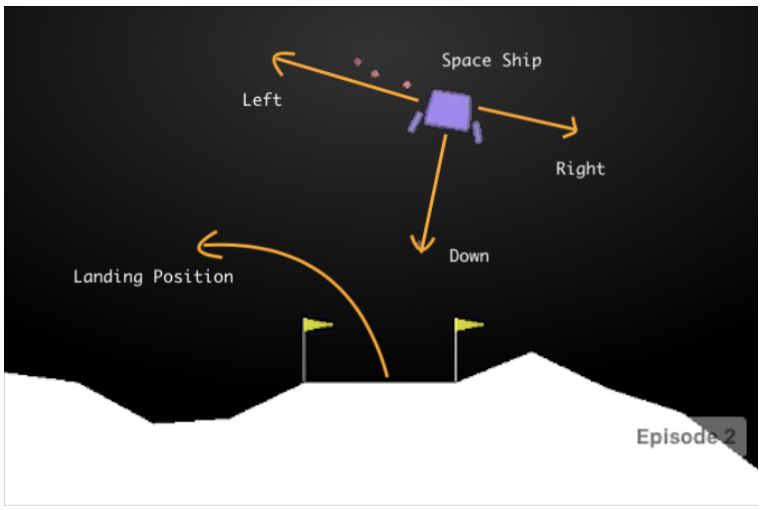

Рис. 1: Приклад роботи середовища LunarLander.

1.3 Принцип максимуму Понтрягіна

Принцип максимуму Понтрягіна формулює необхідні умови оптимального керування динамічними системами. Нехай система описується рівнянням стану

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

а цільова функція має вигляд

$$J = \int_0^T L(x(t), u(t))\, dt. \tag{2}$$

Тоді вводиться гамільтоніан

$$H(x, u, \lambda) = L(x, u) + \lambda^\top f(x, u), \tag{3}$$

і оптимальне керування повинно задовольняти умову

$$u^*(t) = \arg\max_u H(x(t), u, \lambda(t)). \tag{4}$$

Загалом, формули складні і описують загальний випадок. Головний висновок принципу для нашої задачі полягає в тому, що в оптимальному режимі керування система не використовує "середні" або проміжні значення керуючих сигналів. Оптимальна дія найчастіше є *bang–bang* типу: або максимальне керування, або його відсутність.

У задачі `LunarLander` це означає, що оптимальна стратегія схильна вмикати основний двигун на повну потужність, коли потрібно різко стабілізувати швидкість падіння, або повністю його вимикати, коли додаткова тяга шкодить точності посадки. Така поведінка узгоджується як з фізичною моделлю системи, так і з принципом максимуму Понтрягіна, де оптимальне керування лежить на межі допустимих значень.

1.4 Чому для задачі використовується навчання з підкріпленням

Посадка місячного модуля є динамічною задачею керування, де агент повинен приймати рішення в реальному часі та враховувати довгострокові наслідки кожної дії. Модель середовища є стохастичною, має неперервну фізичну динаміку та складні залежності між станами. У таких умовах неможливо аналітично вивести оптимальну траєкторію або точно розв'язати задачу класичними методами оптимального керування. Тому підхід навчання з підкріпленням є природним: агент самостійно вивчає політику, що максимізує сумарну винагороду, взаємодіючи зі середовищем.

1.5 Чому задача не підходить для Q-learning

Класичний Q-learning ґрунтується на дискретизації як простору дій, так і простору станів. У середовищі `LunarLander` простір станів є неперервним та багатовимірним (позиції, швидкості, кут та інші параметри). Щоб застосувати табличний Q-learning, потрібно було б розбити кожен параметр на велику кількість дискретних інтервалів, що призводить до експоненційного зростання таблиці станів. Це робить навчання практично неможливим: таблиця Q стає занадто великою, а збіжність алгоритму — надто повільною.

Крім того, поведінка ландера є складною з точки зору динаміки: невеликі зміни у швидкості або куті можуть радикально змінити результат посадки. Q-learning не може ефективно узагальнювати такі стани, бо не має функціонального апроксіматора.

Тому для задачі використовують алгоритми на основі параметризованих політик, наприклад PPO або Actor–Critic, які безпосередньо працюють з неперервним простором станів і вчаться узагальнювати складні залежності між станом та оптимальною дією.

1.6 Реалізація алгоритму DQN

Для розв'язання задачі посадки у середовищі `LunarLander-v3` було реалізовано власну версію алгоритму DQN. Функція $Q(s, a)$ апроксимується повнозв'язною нейронною мережею з двома прихованими шарами по 128 нейронів із активацією ReLU. Вихідний шар має розмірність, що дорівнює кількості дискретних дій у середовищі. Архітектура моделі має вигляд:

```
self.model = nn.Sequential(
  nn.Linear(obs_dim, 128),
  nn.ReLU(),
  nn.Linear(128, 128),
  nn.ReLU(),
  nn.Linear(128, act_dim)
)
```

Для стабілізації навчання використовується цільова мережа Q_{target}, яка періодично копіює ваги основної моделі. Зберігання минулих переходів реалізовано через буфер повторів (Replay Buffer), що дозволяє формувати випадкові міні-батчі та зменшує кореляцію між послідовними зразками.

Оновлення параметрів моделі виконується за допомогою цільового значення

$$y = r + \gamma \max_{a'} Q_{\text{target}}(s', a'), \tag{5}$$

де Q_{target} не оновлюється кожен крок, що робить навчання більш стабільним. Фактичне значення мережі $Q(s, a)$ комбінується з цільовим значенням через MSE втрату:

$$\mathcal{L} = (Q(s, a) - y)^2 . \tag{6}$$

Для дослідження середовища використано ε-жадібну політику. Значення ε плавно зменшується від 1.0 до 0.05, що дозволяє спочатку активно досліджувати дії, а пізніше — покладатися на політику моделі. Оновлення ваг здійснюється методом Adam з швидкістю навчання 10^{-3}, а коефіцієнт дисконтування вибрано $\gamma = 0.99$.

Однак, отримана реалізація дозволяє досягати стабільного навчання і поступового зростання сумарної винагороди впродовж епізодів лише при відносно довгому навчанні (> 1500 епох), збільшення кількості шарів нейронної мережі частково вирішує проблему точності роботи, але також і збільшує обєм обчислень.

```
Episode 394 | Reward: -106.8 | Epsilon: 0.138
Episode 395 | Reward: -35.2 | Epsilon: 0.137
Episode 396 | Reward: -87.5 | Epsilon: 0.137
Episode 397 | Reward: 59.4 | Epsilon: 0.136
Episode 398 | Reward: -14.2 | Epsilon: 0.135
Episode 399 | Reward: -187.7 | Epsilon: 0.135
Episode 400 | Reward: 37.0 | Epsilon: 0.134
Episode 401 | Reward: -198.4 | Epsilon: 0.133
Episode 402 | Reward: -118.0 | Epsilon: 0.133
Episode 403 | Reward: -129.5 | Epsilon: 0.132
Episode 404 | Reward: -93.8 | Epsilon: 0.131
Episode 405 | Reward: -160.4 | Epsilon: 0.131
Episode 406 | Reward: -307.4 | Epsilon: 0.130
Episode 407 | Reward: 38.0 | Epsilon: 0.129
Episode 408 | Reward: -145.7 | Epsilon: 0.129
Episode 409 | Reward: 38.0 | Epsilon: 0.128
Episode 410 | Reward: -109.5 | Epsilon: 0.127
Episode 411 | Reward: -264.3 | Epsilon: 0.127
Episode 412 | Reward: -48.3 | Epsilon: 0.126
Episode 413 | Reward: 61.1 | Epsilon: 0.126
Episode 414 | Reward: -188.1 | Epsilon: 0.125
Episode 415 | Reward: -180.5 | Epsilon: 0.124
Episode 416 | Reward: -163.5 | Epsilon: 0.124
Episode 417 | Reward: -74.1 | Epsilon: 0.123
Episode 418 | Reward: 27.6 | Epsilon: 0.122
Episode 419 | Reward: -76.9 | Epsilon: 0.122
```

Рис. 2: Приклад логу навчання з DQN.

Як видно з логу, навіть на 500 епохах модель навчається не "гладко а стрибками, при тому, що обчислювальні витрати достатньо суттєві. До того ж після 500 епох середнє значення reward не переходить за 0, що вказує на те, що насправді, даний підхід є тупіковим, тож розглянемо інший підхід.

1.7 Proximal Policy Optimization

PPO (Proximal Policy Optimization) — це метод навчання з підкріпленням, який навчає агента прямо, а не через оцінку Q-функції. Ідея проста: алгоритм пробує трохи змінити політику, дивиться, чи це покращує поведінку, і не дозволяє політиці змінюватися занадто різко. Завдяки цьому навчання виходить стабільним і плавним. [3, 4]

На відміну від DQN, який оцінює значення кожної дії, PPO працює з ймовірностями дій і оновлює їх так, щоб агент поступово ставав кращим. Спеціальний механізм “clipping” обмежує занадто великі стрибки в навчанні, тому модель не “ламається”, навіть якщо винагороди шумні.

Для `LunarLander` PPO підходить краще, бо це середовище з тонкою динамікою: невеликі зміни у швидкості чи куті можуть сильно вплинути на результат. PPO швидше знаходить стабільну поведінку і рідше збивається в нестійкі стани, тому агент навчається приземлятись набагато надійніше.

```
1  model = train_ppo(epochs=200)
```

```
Epoch 13: Reward=-42.9
Epoch 14: Reward=-39.2
Epoch 15: Reward=-53.1
Epoch 16: Reward=-18.6
Epoch 17: Reward=-21.1
Epoch 18: Reward=-41.5
Epoch 19: Reward=-25.6
Epoch 20: Reward=18.6
WARNING:imageio_ffmpeg:IMAGEIO FFMPEG_WRITER WARNING: input image is no
[SAVE] Video + metrics saved for epoch 20
Epoch 21: Reward=27.3
Epoch 22: Reward=-16.5
Epoch 23: Reward=-8.9
Epoch 24: Reward=-7.3
Epoch 25: Reward=11.7
Epoch 26: Reward=-20.0
Epoch 27: Reward=-5.1
Epoch 28: Reward=2.5
Epoch 29: Reward=22.1
Epoch 30: Reward=10.1
Epoch 31: Reward=8.1
Epoch 32: Reward=86.1
Epoch 33: Reward=62.8
Epoch 34: Reward=61.1
Epoch 35: Reward=38.3
Epoch 36: Reward=52.4
Epoch 37: Reward=55.7
Epoch 38: Reward=41.0
Epoch 39: Reward=62.5
Epoch 40: Reward=79.7
```

Рис. 3: Приклад логу навчання з PPO.

Як видно з логу навчання, модель дуже швидко виходить на позитивну оцінку reward, що свідчить про те, що даний підхід є більш пристосованим для даної задачі.

Для дослідження середовища використано -жадібну політику. Значення плавно зменшується від 1.0 до 0.05, що дозволяє спочатку активно досліджувати дії, а пізніше — покладатися на політику моделі. Оновлення ваг здійснюється методом Adam з швидкістю навчання 103, а коефіцієнт дисконтування вибрано = 0.99.

1.8 Проблема зависання та застосування власного environment wrapper

Під час навчання як DQN, так і PPO виникла проблема: агент часто знаходив стратегію “зависання” на певній висоті. Це відбувається через те, що при нульових або малих швидкостях агент отримує стабільну, хоча й низьку, винагороду, і модель починає віддавати перевагу стоянню на місці замість активного зниження та посадки. Така стратегія локально оптимальна, але не приводить до успішного виконання завдання.

Щоб це виправити, було додано власну обгортку тренуального середовища (wrapper), яка модифікує винагороду. У ній реалізовано кілька простих правил “reward shaping”, що підштовхують агента до коректної траєкторії:

- бонус за зменшення відстані до посадкової площадки;
- штраф за зависання, коли горизонтальна та вертикальна швидкості надто малі;
- додаткова винагорода за успішне торкання поверхні без аварії.

Це дозволяє уникнути ситуацій, коли агент знаходить вигідним залишатися у повітрі, не приймаючи рішень щодо посадки. На тестове середовище, звичайно, дані модифікації на впливають.

Нижче наведено класс LunarLanderShaped wrapper-а:

```python
class LunarLanderShaped(gym.Wrapper):
  def __init__(self):
    super().__init__(gym.make("LunarLander-v3", render_mode="rgb_array"))
    self.prev_dist = None

```

```
  def reset(self, **kwargs):
    obs, info = self.env.reset(**kwargs)
    self.prev_dist = np.sqrt(obs[0]**2 + obs[1]**2)
    return obs, info

  def step(self, action):
    obs, reward, term, trunc, info = self.env.step(action)

    x, y, vx, vy, angle, vangle, leg1, leg2 = obs

    # distance to landing pad
    dist = np.sqrt(x*x + y*y)

    # shaping: reward for moving closer
    reward += 2.0 * (self.prev_dist - dist)
    self.prev_dist = dist

    # shaping: penalty for hovering with almost zero movement
    if abs(vx) < 0.03 and abs(vy) < 0.03:
      reward -= 0.05

    # shaping: encourage touching down
    if term and not trunc:
      reward += 50.0   # successful landing bonus

    return obs, reward, term, trunc, info
```

Після додавання цього враппера поведінка агента стала більш стабільною: він перестав "висіти" на місці та почав цілеспрямовано рухатися вниз до посадкової зони. Проблема була характерною для обох підходів (DQN і PPO), тому reward shaping виявився важливим кроком для коректного навчання. На прикладах роботи навченої моделі результатом роботи даного враппера є те, що апарат направляється до землі із дуже високою швидкістю, максимізуючи функцію reward, та знижуючи швидкість лише перед самою землею.

1.9 Зняття метрик під час навчання

Для контролю процесу тренування важливо регулярно фіксувати метрики, а саме: середню винагороду, стабільність поведінки агента, збіжність політики та ефективність оновлень. Знімати метрики щокроку недоцільно, оскільки значення дуже шумні, особливо на ранніх етапах навчання. Тому метрики знімаються періодично — наприклад, кожні N епох, що дозволяє отримати більш репрезентативну статистику.

У реалізації PPO періодичний запис результатів організовано через виклик функції `record_eval_video` кожні 20 епох. Це дозволяє зберегти відеозапис поведінки агента та оцінити, як змінюється якість посадки з часом. Саме такий підхід відповідає поширеним рекомендаціям: не записувати кожен епізод, а фіксувати навчання через рівні інтервали для аналізу довгострокових тенденцій.

Відповідний фрагмент коду:

```
if (epoch + 1) % 20 == 0:
    record_eval_video(model, epoch + 1)
```

Наступний фрагмент коду надає функцію recordevalvideo, яка дозволяє провести один замір на даній моделі, згенерувати відео та зберегти його разом із файлом метрик до вказаної папки

```
def record_eval_video(model, epoch, folder="training_stats"):
  import imageio, base64
  from IPython.display import HTML

  env = gym.make("LunarLander-v3", render_mode="rgb_array")
  state, _ = env.reset()

  frames = []
  total_reward = 0
  final_obs = None

```

```
  # ---- 1 episode rollout ----
  for _ in range(1000):
    st = torch.tensor(state, dtype=torch.float32).unsqueeze(0).to(device)
    logits, _ = model(st)
    probs = torch.softmax(logits, dim=-1)
    action = torch.argmax(probs, dim=-1).item()

    next_state, reward, term, trunc, _ = env.step(action)

    base = env.render()
    panel = draw_instrument_panel(state, action, base)
    frames.append(panel)

    total_reward += reward
    state = next_state

    if term or trunc:
      final_obs = next_state
      break

  env.close()

  # ---- determine success ----
  def is_soft_landing(obs):
    x, y, vx, vy, angle, ang_vel, l1, l2 = obs
    return (
      l1 == 1 and l2 == 1 and
      abs(vx) < 0.6 and      # relaxed
      abs(vy) < 0.6 and
      abs(angle) < 0.6
    )

  success = is_soft_landing(final_obs) if final_obs is not None else False

  # ---- save video ----
  filename = f"{folder}/epoch_{epoch}_eval.mp4"
  imageio.mimsave(filename, frames, fps=60)

  # ---- save small metrics ----
  import json
  metrics = {
    "epoch": epoch,
    "reward": float(total_reward),
    "success": bool(success),
    "final_state": [float(x) for x in final_obs] if final_obs is not None else None
  }

  with open(f"{folder}/epoch_{epoch}_metrics.json", "w") as f:
    json.dump(metrics, f, indent=2)

  print(f"[SAVE] Video + metrics saved for epoch {epoch}")

  return filename
```

Це означає, що після кожних 20 епох тренування запускається окрема процедура оцінювання, яка виконує один епізод та зберігає відео з метриками у окремому файлі. Таке періодичне зняття метрик дозволяє відстежувати зміни у стабільності польоту, використанні двигунів і якості посадки без надлишкового навантаження на систему та дисковий простір.

Також у порядку ініціативи було розроблено імпровізовану панель приборів, що відображається від відео самої посадки від третьої особи для кожного запису посадки апарату, таких як: поворотний авіагоризонт, показники вертикальної та горизонтальної швидкостей, шкала висоти, індикатори роботи двигунів та індикатор посадки (справа внизу). Дане табло дає більше розуміння що саме відбувається у кожний момент часу із апаратом. При необхідності до даної панелі можна додати і числові значення потрібних приборів.

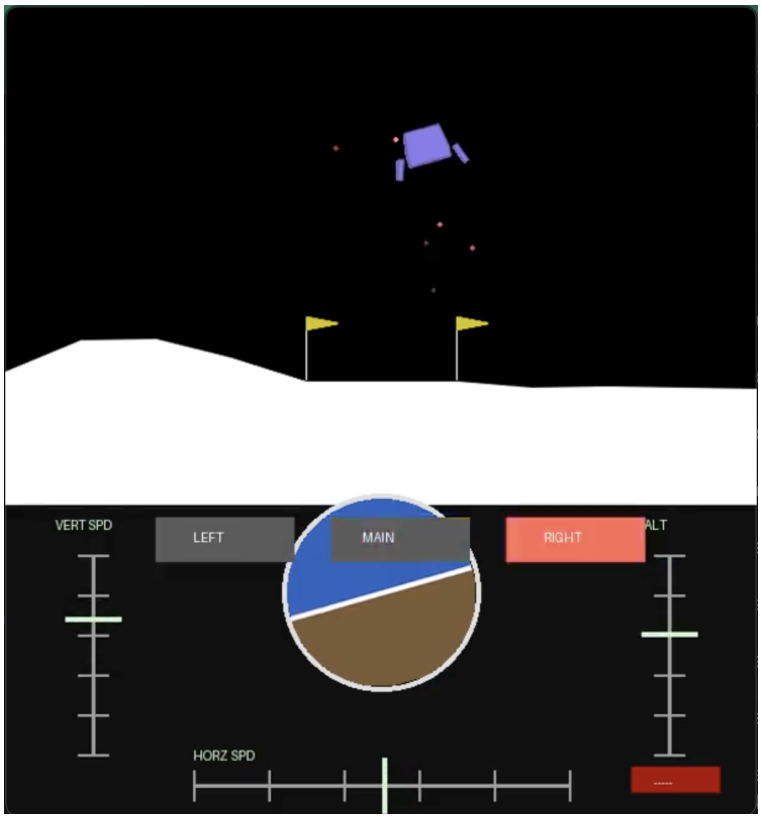

Рис. 4: Приклад записаного відео роботи агента після 160 епох навчання

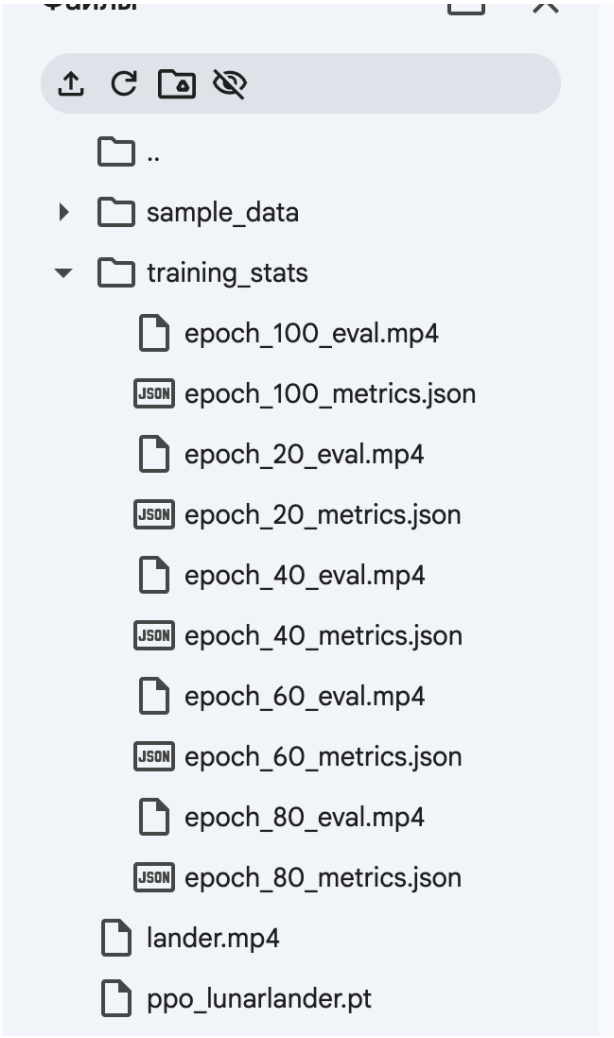

Рис. 5: Поступово зняті метрики кожні 20 епох

1.10 Результати експериментів

Під час навчання агентів було проведено базові експерименти як для DQN, так і для PPO. Через високу варіативність середовища оцінювання проводилося на декількох епізодах після кожних 20 епох тренування.

Для алгоритму DQN середня винагорода на початку становила приблизно від -200 до -150. Після 600 епізодів навчання модель стабільно виходила на значення в діапазоні $0 \dots 50$, однак траєкторії польоту залишалися нестабільними, а успішні посадки траплялися нерегулярно.

У випадку PPO прогрес був значно швидшим. Уже після 50–60 епох середня винагорода перевищувала 100, а після близько 150 епох модель почала демонструвати впевнені посадки та стійку поведінку без різких коливань. У кінцевих епізодах середня винагорода коливалася в діапазоні $180 \dots 230$, що відповідає коректній посадці з мінімальними помилками.

Додаткове використання wrapper-а з покращеною функцією винагороди усунуло проблему зависання на місці. Після введення "reward shaping" обидва алгоритми частіше рухалися до посадкової площадки, а не утримували стабільний стан у повітрі. Особливо помітно це вплинуло на DQN, який після модифікації винагороди став частіше завершувати епізоди саме посадкою.

1.11 Висновки

У роботі було досліджено задачу посадки місячного модуля та застосовано декілька підходів навчання з підкріпленням. Проведені експерименти показали, що методи, побудовані на безпосередній оптимізації політики, зокрема PPO, значно краще справляються з динамікою середовища порівняно з підходами на основі Q-функції. DQN продемонстрував здатність навчатися базовим стратегіям, однак його поведінка залишалася нестабільною, а успішні посадки траплялися нерегулярно.

Використання додаткового reward shaping через власний wrapper дозволило усунути проблему зависання апарата на фіксованій висоті. Після модифікації винагороди агенти почали активніше прямувати до посадкової площадки та рідше залишалися у "зручних" локально оптимальних станах.

У процесі навчання було знято метрики з кожної 20-ї епохи, що дозволяє краще знаходити помилки у коді та наочно продемонструвати, які саме дії виконує агент.

Загалом PPO продемонстрував швидшу збіжність, стабільнішу поведінку та кращу здатність до узагальнення. Отримані результати підтверджують, що для задач із плавною фізичною динамікою та складними залежностями між станами оптимальним вибором є політикові методи, тоді як Q-learning та його глибинні модифікації менш надійні в таких умовах.

Література

[1] F. Foundation, "Gymnasium documentation," 2023. [Online]. Available: https://gymnasium.farama.org/

[2] ——, "Lunarlander-v3 environment documentation," 2023. [Online]. Available: https://gymnasium.farama.org/environments/box2d/lunar_lander/

[3] OpenAI, "Proximal policy optimization (ppo)," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[4] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.