

# Lab7 Report

## Deep Learning

### Policy-Based Reinforcement Learning

姓名：陳科融

學號：314551010

Aug 12, 2025

## 1. Introduction

本次作業聚焦於兩種廣泛應用於深度強化學習領域的基於策略（Policy-Based）演算法：Advantage Actor-Critic（A2C）與 Proximal Policy Optimization（PPO）。在強化學習中，基於策略的方法能夠直接學習將狀態映射至動作的策略，特別適用於處理高維度且連續的動作空間問題，然而，傳統策略梯度方法往往因更新步長過大而導致訓練過程不穩定，限制了其效能。

為了解決此問題，A2C 作為一種同步且確定性的 Actor-Critic 方法，透過 Critic 網路估計的優勢函數（Advantage Function）引導 Actor 網路策略更新，有效降低策略梯度的變異性。而 PPO 則在此基礎上進行改進，透過引入裁剪代理目標函數（Clipped Surrogate Objective）限制每次策略更新幅度，提升訓練的穩定性與樣本利用效率，成為目前最流行且穩健的強化學習演算法之一。

## 2. Please briefly explain your implementation

### 2.1 How do you obtain the stochastic policy gradient and the TD error for A2C?

在 A2C（Advantage Actor-Critic）演算法的實作中，Actor 與 Critic 的更新是同步進行的，其核心在於時間差分誤差（Temporal Difference Error, TD Error）的計算與應用。

#### TD 誤差（Temporal Difference Error）

TD 誤差是 Critic 網路學習的基礎，用於衡量當前價值估計與更準確的 TD 目標之間的差距。TD 目標結合了即時獎勵  $r$  與下一步狀態的折扣價值估計  $\gamma \cdot V(s')$ 。

## 1. TD 目標 (TD Target) :

$$\text{TD Target} = r + \gamma \cdot V(s') \cdot (1 - \text{done})$$

## 2. TD 誤差 (Advantage Function) :

$$\text{Advantage} = \text{TD Target} - V(s)$$

其中，Advantage 直觀地表示在狀態  $s$  採取某動作後，比預期好多少。

## 隨機策略梯度 (Stochastic Policy Gradient)

Actor 的目標是最大化期望回報，其梯度更新依賴 Critic 提供的 Advantage。

計算流程如下：

- Actor 網路輸出一個常態分佈，代表在當前狀態下動作的機率分佈。
- 從該分佈中採樣一個動作  $a$ ，並計算對數機率  $\log \pi(a|s)$ 。

Actor Loss 的計算公式如下：

$$L_{\text{actor}} = -\log \pi(a|s) \cdot \text{Advantage}$$

```
with torch.no_grad():
    # 計算TD目標
    td_target = rewards_t + self.config.gamma * self.critic(next_states_t) * (1 -

    # 取得當前狀態估計
    state_values = self.critic(states_t)
    critic_loss = F.smooth_l1_loss(state_values, td_target)
    self.critic_optimizer.zero_grad()
    critic_loss.backward()
    nn.utils.clip_grad_norm_(self.critic.parameters(), self.config.max_grad_norm)
    self.critic_optimizer.step()

    # TD 誤差
    advantage = (td_target - state_values).detach()
    policy_dist = self.actor(states_t)

    # 取得採樣動作的對數機率  $\log(\pi(a|s))$ 
    log_probs = policy_dist.log_prob(actions_t)
    entropy = policy_dist.entropy().mean()

    # 3. 計算 Actor Loss (包含策略梯度和熵獎勵)
    actor_loss = -(log_probs * advantage).mean() - self.config.entropy_beta * entropy
    self.actor_optimizer.zero_grad()
    actor_loss.backward()
    nn.utils.clip_grad_norm_(self.actor.parameters(), self.config.max_grad_norm)
    self.actor_optimizer.step()
```

程式碼 1. TD error 程式碼

## 2.2 How do you implement the clipped objective in PPO?

PPO 的核心創新在於其裁剪代理目標函數 (Clipped Surrogate Objective)，它旨在限制每次策略更新的幅度，防止因過大的更新導致性能崩潰，從而提升訓練的穩定性。

實作步驟如下：

1. 計算機率比 (Probability Ratio)：首先，計算新策略與舊策略（收集數據時的策略）對於同一個動作的機率比：

$$Ratio = \exp(\log(p_{\text{new}}) - \log(p_{\text{old}}))$$

2. 計算兩個代理目標：

- 目標一（無裁剪）： $\text{surr1} = \text{ratio} * \text{advantage}$
- 目標二（有裁剪）： $\text{surr2} = \text{clip}(\text{ratio}, 1 - \epsilon, 1 + \epsilon) * \text{advantage}$

其中  $\epsilon$  (`clip_coef`) 是一個超參數（通常為 0.2），它定義了一個裁剪區間。`clip` 函數會將 `ratio` 強制限制在  $[1 - \epsilon, 1 + \epsilon]$  的範圍內。

3. 取最小值：PPO 的最終目標函數是取這兩個代理目標中的較小值，並加上負號（因為我們要最大化它）：

$$\text{policy}_{\text{loss}} = -\min(\text{surr1}, \text{surr2})$$

這個機制的巧妙之處在於：

- 當  $\text{advantage} > 0$  (好動作) 時，`ratio` 被限制在  $1 + \epsilon$ ，防止策略過於激進地增加該動作的機率。
- 當  $\text{advantage} < 0$  (壞動作) 時，`ratio` 被限制在  $1 - \epsilon$ ，防止策略過於激進地降低該動作的機率。

```
# 計算機率比Ratio
log_ratio = new_log_probs - mb_log_probs_old
ratio = torch.exp(log_ratio)

# 目標一 無裁切
surr1 = mb_advantages * ratio

# 目標二 有裁切
surr2 = mb_advantages * torch.clamp(ratio, 1 - self.config.clip_coef, 1 + self.config.clip_coef)

# 取最小值作為Loss
pg_loss = -torch.min(surr1, surr2).mean()

# 加上熵獎勵，構成最終的 Actor 損失函數
actor_loss = pg_loss - self.config.entropy_beta * entropy
```

程式碼 2. PPO Clipped 程式碼

## 2.3 How do you obtain the estimator of GAE?

廣義優勢估計 (Generalized Advantage Estimation, GAE) 是一種在 TD 誤差的高偏差 (bias) 和蒙地卡羅 (Monte Carlo) 的高變異 (variance) 之間取得平衡的技術。它透過超參數  $\lambda$  (gae\_lambda) 來調節這個平衡。

實作是從一批經驗數據的最後一步倒推回來計算每一步的 GAE。

其遞迴公式為：

$$\text{GAE}(t) = \delta(t) + \gamma \cdot \lambda \cdot \text{GAE}(t+1)$$

其中  $\delta(t) = r(t) + \gamma \cdot V(s_{t+1}) - V(s_t)$  正是 TD 誤差。

具體計算流程如下：

1. 從  $t = T-1$  (rollout 的最後一步) 開始，倒序遍歷到  $t = 0$ 。
2. 在每一步  $t$ ，計算該步的 TD 誤差  $\delta$ 。
3. 使用  $\delta$  和上一步計算出的  $\text{GAE}(t+1)$ ，來計算當前的  $\text{GAE}(t)$ 。
4. 將計算出的  $\text{GAE}(t)$  存儲起來，並更新 `last_gae_lam` 以供下一步使用。

最終，我們得到了一個比單步 TD 誤差更穩定且準確的優勢估計值。

```
num_steps = len(rewards)
advantages = np.zeros_like(rewards, dtype=np.float32)
last_gae_lam = 0

# 1. 從 rollout 的最後一步開始，倒序遍歷
for t in reversed(range(num_steps)):
    if t == num_steps - 1:
        next_non_terminal = 1.0 - last_done
        next_value = last_value
    else:
        next_non_terminal = 1.0 - dones[t + 1]
        next_value = values[t + 1]
    # 2. 計算 TD 誤差 delta
    # 公式:  $\delta_t = r_t + \gamma \cdot V(s_{t+1}) - V(s_t)$ 
    delta = rewards[t] + config.gamma * next_value * next_non_terminal - values[t]

    # 3. 使用遞迴公式計算 GAE
    # 公式:  $A_t = \delta_t + \gamma \cdot \lambda \cdot A_{t+1}$ 
    advantages[t] = last_gae_lam = delta + config.gamma * config.gae_lambda * next_non_terminal

memory['advantages'] = advantages
memory['returns'] = advantages + values
```

程式碼 3. GAE 程式碼

## 2.4 How do you collect samples from the environment?

用 On-Policy 的樣本收集方式，更新模型的數據必須是由當前最新的策略產生的。

收集流程如下：

1. 初始化：在訓練開始或一個回合結束時，重置環境以獲得初始狀態  $s$ 。
2. 互動迴圈：在一個固定長度的迴圈中（由 `batch_size` 參數定義）：
  - 選擇動作：將當前狀態  $s$  輸入 Actor 網路，得到一個動作的機率分佈，並從中採樣一個動作  $a$ 。
  - 執行動作：將動作  $a$  輸入環境，獲得下一個狀態  $s'$ 、獎勵  $r$ 。
  - 儲存經驗：將這一步的完整經驗 ( $s, a, r, done, \log\_prob, value$ ) 儲存到一個暫時的記憶體 (buffer) 中。
  - 更新狀態： $s = s'$ 。
4. 完成收集：當迴圈結束後，這個 buffer 中的所有數據就被用於一次模型更新，更新完成後，這個 buffer 會被清空，然後開始下一輪的數據收集。

```
for _ in range(num_steps):
    if self.total_steps >= self.max_total_steps: break

    # 採樣動作
    with torch.no_grad():
        state_tensor = to_tensor(self.state, device).unsqueeze(0)
        policy_dist = self.actor(state_tensor)
        action = policy_dist.sample()
        log_prob = policy_dist.log_prob(action).sum(dim=-1)
        value = self.critic(state_tensor)

    action_np = action.squeeze(0).cpu().numpy()
    clipped_action = np.clip(action_np, self.env.action_space.low, self.env.action_space.high)

    # 與環境互動
    next_state, reward, terminated, truncated, _ = self.env.step(clipped_action)
    done = terminated or truncated

    # 儲存經驗
    memory['states'].append(self.state)
    memory['actions'].append(action_np)
    memory['rewards'].append(reward)
    memory['dones'].append(done)
    memory['log_probs'].append(log_prob.cpu().numpy())
    memory['values'].append(value.cpu().numpy().flatten())

    # 更新狀態
    self.state = next_state
    self.total_steps += 1
    self.episode_reward += reward
```

程式碼 4. On policy 程式碼

## 2.5 How do you enforce exploration?

儘管 A2C 和 PPO 是 On-Policy 方法，它們的探索機制是內建於隨機策略 (Stochastic Policy)。具體來說，Actor 網路輸出的不是一個確定的動作，而是一個常態分佈 (Normal Distribution) 的參數 (平均值  $\mu$  和標準差  $\sigma$ )。

1. 採樣實現探索：在訓練過程中，不是直接選擇平均值  $\mu$  作為動作，而是從  $\text{Normal}(\mu, \sigma)$  這個分佈中採樣一個動作，這個採樣過程本身就引入了隨機性，使得 Agent 有機會嘗試那些不是當前最優但可能有潛力的動作，從而實現了探索。
2. 熵獎勵 (Entropy Bonus)：為了進一步鼓勵探索，防止策略過早地收斂到次優解 (即分佈的標準差  $\sigma$  變得過小)，在 Actor 的損失函數中加入了一項熵 (Entropy) 的獎勵。

$$\text{actor\_loss} = \text{policy\_loss} - \text{entropy\_beta} * \text{entropy}$$

熵衡量了機率分佈的隨機性，透過最大化熵，鼓勵策略保持一定的隨機性，從而進行更充分的探索。entropy\_beta 用於控制探索獎勵的強度。

```
def forward(self, state_tensor: torch.Tensor) -> torch.distributions.Normal:
    x = self.network(state_tensor)
    action_means = self.mean_head(x)
    action_stds = torch.clamp(self.log_stds.exp(), 1e-3, 10.0)
    return torch.distributions.Normal(action_means, action_stds)
```

程式碼 5. 定義分佈

```
# 採樣動作
with torch.no_grad():
    state_tensor = to_tensor(self.state, device).unsqueeze(0)
    policy_dist = self.actor(state_tensor)

# 從分佈中「採樣」一個動作
action = policy_dist.sample()
log_prob = policy_dist.log_prob(action).sum(dim=-1)
value = self.critic(state_tensor)
```

程式碼 6. 採樣動作

```
# 將熵作為獎勵加入最終的 Actor 損失函數
# 乘以 config.entropy_beta 來控制強度
actor_loss = pg_loss - self.config.entropy_beta * entropy
```

程式碼 7. 熵獎勵

## 2.6 Explain how you use Weight & Bias to track model performance and the loss values (including actor loss, critic loss, and the entropy).

為了確保訓練過程的數據可視化一致性，在 W&B 中使用 `wandb.define_metric()` 將 Global Step（環境總步數）定義為統一的 X 軸。

在訓練過程中，透過 `wandb.log()` 記錄關鍵指標，包括：




- 損失函數：Train/actor\_loss、Train/critic\_loss、Train/entropy（模型更新）。
- 訓練表現：charts/episodic\_return（每回合結束時記錄）。
- 評估表現：Eval/avg\_episode\_reward（獨立評估後記錄，作為衡量模型最終性能的主要指標）。

```
if wandb.run:
    wandb.log({
        "Train/actor_loss": pg_loss.item(),
        "Train/critic_loss": v_loss.item(),
        "Train/entropy": entropy.item(),
        "Global Step": current_step
    })
```

程式碼 8. Train 紀錄 loss

```
if wandb.run:
    wandb.log({
        "Eval/avg_episode_reward": avg_reward,
        "Eval/episode": current_episode_count,
        "Global Step": runner.total_steps
    })
    last_eval_episode = current_episode_count
```

程式碼 9. 評估訓練分數

<p>A2c</p>	 <p>The A2c training metrics show the following trends:</p> <ul style="list-style-type: none"> <li><b>Eval/episode:</b> Increases linearly from approximately 200 to 1000 over 200k global steps.</li> <li><b>Eval/avg_episode_reward:</b> Starts at -1200, rises sharply after 50k steps, and stabilizes around -200.</li> <li><b>Train/entropy:</b> Starts at 1.4 and decreases steadily to about 0.8.</li> <li><b>Train/critic_loss:</b> Starts at 10 and decreases with high-frequency noise to around 2.</li> <li><b>Train/actor_loss:</b> Starts at 10 and decreases with high-frequency noise to around 0.</li> </ul>
<p>PPO Task2</p>	 <p>The PPO Task2 training metrics show the following trends:</p> <ul style="list-style-type: none"> <li><b>Eval/avg_episode_reward:</b> Starts at -1000, rises sharply after 25k steps, and stabilizes around -200.</li> <li><b>Train/critic_loss:</b> Starts at 6 and decreases with high-frequency noise to around 1.</li> <li><b>Train/actor_loss:</b> Starts at 0.6 and decreases with high-frequency noise to around 0.</li> </ul>
<p>PPO Task3</p>	 <p>The PPO Task3 training metrics show the following trends:</p> <ul style="list-style-type: none"> <li><b>losses/value_loss:</b> Starts at 1.5 and decreases to around 0.2.</li> <li><b>losses/policy_loss:</b> Starts at -0.005 and decreases to around -0.02.</li> <li><b>losses/entropy_loss:</b> Starts at 8.4 and decreases to around 7.2.</li> <li><b>losses/critic_lr:</b> Starts at 0.0005 and decreases to around 0.0001.</li> <li><b>losses/clipfrac:</b> Starts at 0.2 and decreases to around 0.08.</li> <li><b>losses/approx_kl:</b> Starts at 0.012 and decreases to around 0.006.</li> <li><b>charts/episodic_return:</b> Starts at 0 and increases to around 4000.</li> <li><b>charts/avg_20_ep_return:</b> Starts at 0 and increases to around 3000.</li> <li><b>charts/SPS:</b> Starts at 50 and increases to around 150.</li> </ul>



### 3. Analysis and discussions

#### 3.1 Plot the training curves (evaluation score versus environment steps)

##### Task1

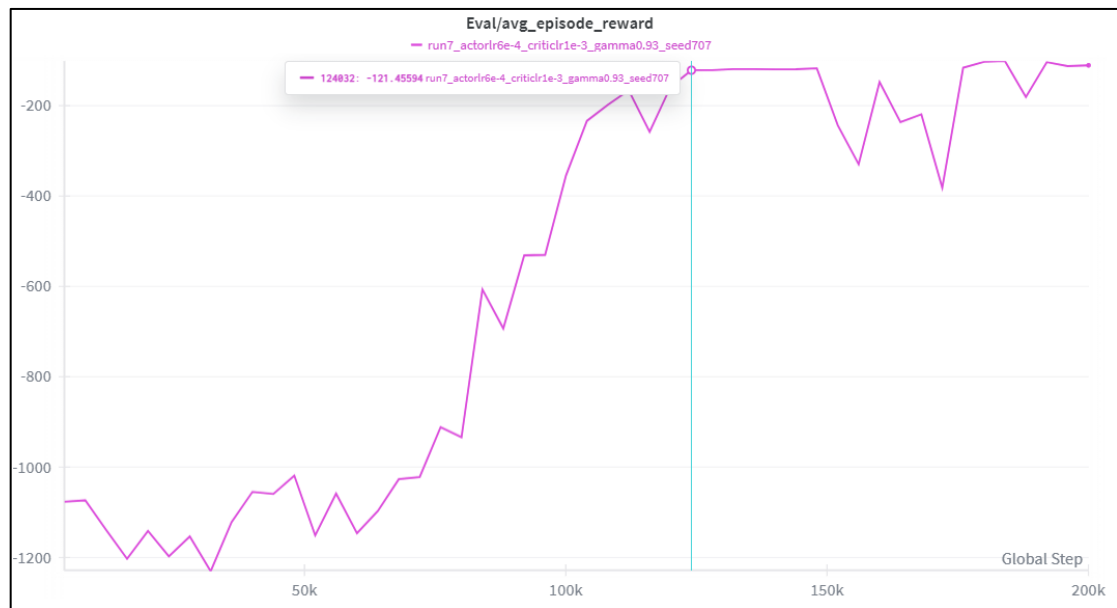


圖 1. A2C in Pendulum 訓練平均數據

##### Task2

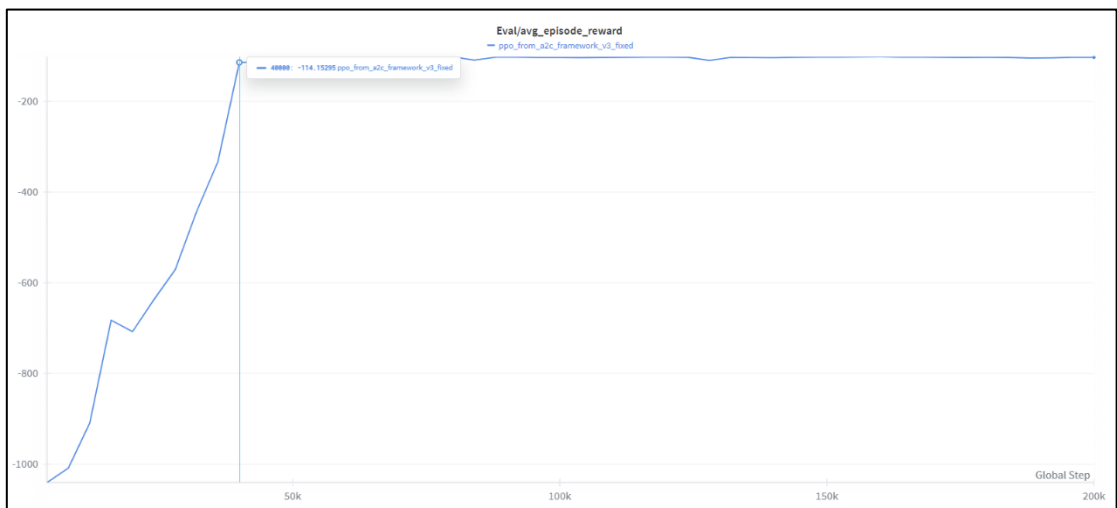


圖 2. PPO in Pendulum 訓練平均數據

### Task3

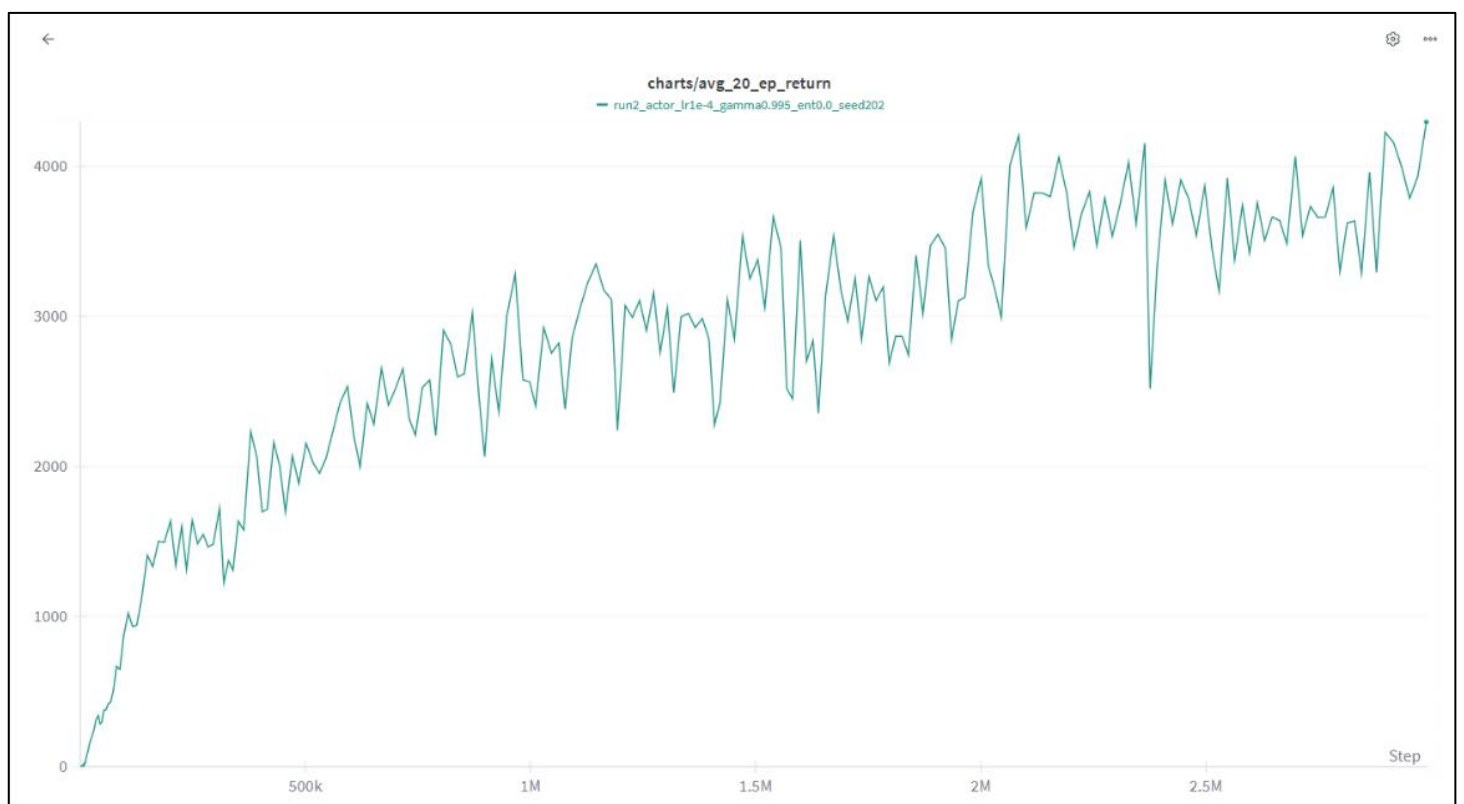


圖 3. PPO in Walker 訓練平均數據

## 3.2 Test result

### Task1

--- 測試結果報告 ---

測試的平均分數 (在 20 個固定種子上): -101.31

```
Terminal Local (2) × + ∨
--- 啟動測試模式 (Inference Mode) ---
使用裝置: cuda:0
載入模型: .\LAB7_314551010_task1_a2c_pendulum.pt
正在還原儲存的觀測狀態正規化 (obs_rms) 物件...
還原成功!
開始在固定的 20 個種子 (0 to 19) 上執行測試...
測試回合 708/20 (seed=707), 分數: -128.34
測試回合 709/20 (seed=708), 分數: -0.70
測試回合 710/20 (seed=709), 分數: -127.52
測試回合 711/20 (seed=710), 分數: -121.83
測試回合 712/20 (seed=711), 分數: -0.12
測試回合 713/20 (seed=712), 分數: -129.73
測試回合 714/20 (seed=713), 分數: -129.88
測試回合 715/20 (seed=714), 分數: -0.96
測試回合 716/20 (seed=715), 分數: -123.45
測試回合 717/20 (seed=716), 分數: -116.53
測試回合 718/20 (seed=717), 分數: -1.35
測試回合 719/20 (seed=718), 分數: -125.31
測試回合 720/20 (seed=719), 分數: -128.38
測試回合 721/20 (seed=720), 分數: -121.09
測試回合 722/20 (seed=721), 分數: -129.75
測試回合 723/20 (seed=722), 分數: -0.31
測試回合 724/20 (seed=723), 分數: -126.32
測試回合 725/20 (seed=724), 分數: -257.71
測試回合 726/20 (seed=725), 分數: -128.89
測試回合 727/20 (seed=726), 分數: -128.09

--- 測試結果報告 ---
測試的平均分數 (在 20 個固定種子上): -101.31
測試影片已儲存至: task1_results
--- 測試結束 ---
(dlp) PS C:\Users\kramer120\Desktop\深度學習\Lab_hw\Lab7>
```

## Task2

--- 測試結果報告 ---

測試的平均分數 (在 20 個固定種子上): -101.91

```
載入模型: .\LAB7_314551010_task2_ppo_pendulum.pt
正在還原儲存的觀測狀態正規化 (obs_rms) 物件...
還原成功!
D:\anaconda3\envs\dlp\lib\site-packages\gymnasium\wrappers\
rying a different `video_folder` for the `RecordVideo` wrap
logger.warn(
開始在固定的 20 個種子 (707 to 726) 上執行測試...
測試回合 1/20 (seed=707), 分數: -128.24
測試回合 2/20 (seed=708), 分數: -1.03
測試回合 3/20 (seed=709), 分數: -127.46
測試回合 4/20 (seed=710), 分數: -120.87
測試回合 5/20 (seed=711), 分數: -0.45
測試回合 6/20 (seed=712), 分數: -127.41
測試回合 7/20 (seed=713), 分數: -128.72
測試回合 14/20 (seed=720), 分數: -120.34
測試回合 15/20 (seed=721), 分數: -127.82
測試回合 16/20 (seed=722), 分數: -0.64
測試回合 17/20 (seed=723), 分數: -126.42
測試回合 18/20 (seed=724), 分數: -281.45
測試回合 19/20 (seed=725), 分數: -129.00
測試回合 20/20 (seed=726), 分數: -128.15

--- 測試結果報告 ---
測試的平均分數 (在 20 個固定種子上): -101.91
測試影片已儲存至: task2_videos\inference_run_fixed_seeds
--- 測試結束 ---
```

### Task3

Task3 (1M)	測試回合 15/20, 分數: 3930.46 測試回合 16/20, 分數: 3504.39 測試回合 17/20, 分數: 1969.38 測試回合 18/20, 分數: 2770.65 測試回合 19/20, 分數: 2743.74 測試回合 20/20, 分數: 3483.24  測試平均分數: 2778.83	
Task3 (1.5M)	測試回合 15/20, 分數: 3883.04 測試回合 16/20, 分數: 686.82 測試回合 17/20, 分數: 2159.51 測試回合 18/20, 分數: 3931.91 測試回合 19/20, 分數: 3839.01 測試回合 20/20, 分數: 3316.18  測試平均分數: 3296.18	
Task3 (2M)	測試回合 15/20, 分數: 4286.10 測試回合 16/20, 分數: 4326.71 測試回合 17/20, 分數: 4358.86 測試回合 18/20, 分數: -4.86 測試回合 19/20, 分數: 4422.55 測試回合 20/20, 分數: 2211.05  測試平均分數: 3391.52	
Task3 (2.5M)	測試回合 15/20, 分數: 2047.61 測試回合 16/20, 分數: 4031.63 測試回合 17/20, 分數: 4244.83 測試回合 18/20, 分數: 4137.21 測試回合 19/20, 分數: 1860.56 測試回合 20/20, 分數: 3491.32  測試平均分數: 3208.56	
Task3 (3M)	測試回合 15/20, 分數: 3227.19 測試回合 16/20, 分數: 4494.61 測試回合 17/20, 分數: 4398.85 測試回合 18/20, 分數: 4459.63 測試回合 19/20, 分數: 4420.57 測試回合 20/20, 分數: 4550.95  測試平均分數: 4159.30	

### 3.3 Compare the sample efficiency and training stability of A2C and PPO.

透過比較 Task 1 (A2C) 和 Task 2 (PPO) 在 Pendulum-v1 上的訓練曲線，可以觀察到以下幾點：

#### 訓練穩定性：

- PPO 的訓練曲線（圖 2）明顯比 A2C 的曲線（圖 1）更加平滑，波動更小。
- A2C 在訓練過程中容易出現性能的大幅震盪，這可能是因為某些批次的數據導致了過大的策略更新，破壞了已學到的良好策略。
- 相比之下，PPO 的裁剪目標函數有效抑制了這種破壞性的更新，使得學習過程更加穩定。

#### 樣本效率：

- 在 Pendulum-v1 環境中，兩種演算法的樣本效率（達到相似性能所需的步數）差距不大，PPO 略佔優勢。
- PPO 能夠更快地收斂到一個較高的分數，且後期性能更穩定，這得益於其更高效的數據利用方式（在同一個 rollout 上進行多次更新）和更穩定的更新策略。

### 3.4 Perform an empirical study on the key parameters, such as clipping parameter and entropy coefficient

#### Task1

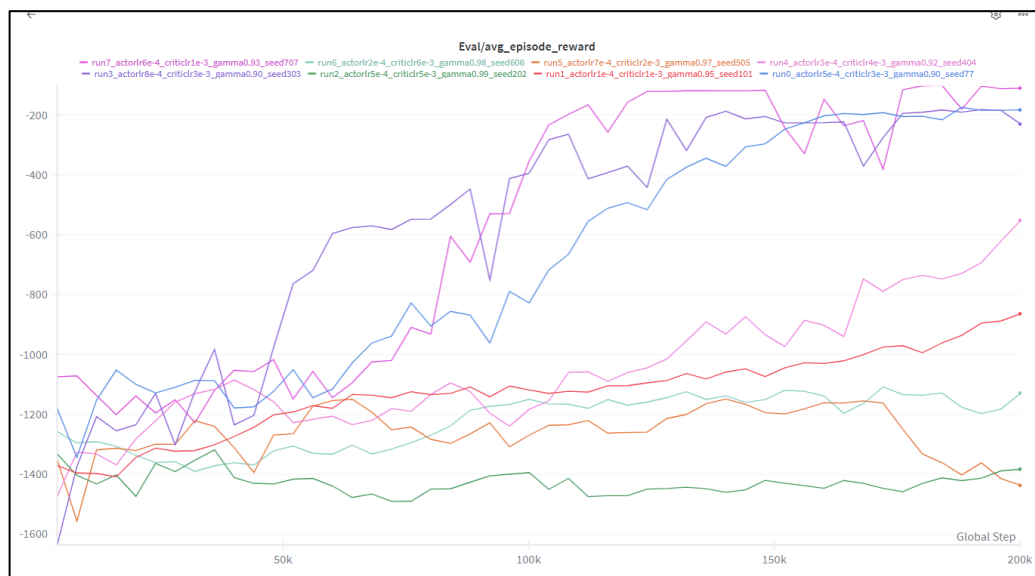


圖 4. Task1 不同參數的影響

Run	Actor LR	Critic LR	Gamma	Seed	最終表現評估
run7	6.E-04	1.E-03	0.93	707	最佳，學習快，獎勵最高
run3	8.E-04	3.E-03	0.9	303	表現優異，但後期波動較大
run0	5.E-04	3.E-03	0.9	77	表現良好，穩定上升
run5	7.E-04	2.E-03	0.97	505	中等，學習較慢
run4	3.E-04	4.E-03	0.92	404	中等偏下，後期略有下降
run1	1.E-04	1.E-03	0.95	101	學習緩慢
run6	2.E-04	6.E-03	0.98	606	表現不佳
run2	5.E-04	5.E-03	0.99	202	最差，幾乎沒有學習

#### 1. Critic Learning Rate (critic\_lr) 的影響

critic\_lr 決定了 Critic（價值函數近似器）在每次更新時的學習步伐大小。這個參數直接影響 Critic 對狀態價值（Value Function）的估計穩定性與準確性。若學習率太低，Critic 更新過於緩慢，無法及時反映環境變化；若學習率太高，則容易造成價值估計震盪或發散。由於 Critic 所提供的「優勢函數」（Advantage）是 Actor 策略更新的關鍵依據，Critic 的穩定性與準確性會顯著影響整體學習表現。

## 2. Gamma ( $\gamma$ , 折扣因子) 的影響

gamma 決定了 Agent 在決策時對未來獎勵的重視程度，也就是其「遠見」的長短。較低的 gamma 使 Agent 更偏向追求短期回報，有助於降低學習的變異性 (variance) 並加速信用分配，提升穩定性；而較高的 gamma 則鼓勵 Agent 考慮長期效益，但會增加學習難度與不穩定性，尤其在回報稀疏或環境複雜時更為明顯。

## 3. Actor Learning Rate (actor\_lr) 的影響

actor\_lr 控制策略網路 (Actor) 在每次更新時的步伐大小。較高的學習率能加速策略的探索與改進，在 Critic 穩定的前提下，有助於提升學習效率；相對地，過低的學習率會使策略更新過慢，即使其他條件理想，整體學習仍可能進展遲緩。該參數需平衡探索速度與策略穩定性。

## Task2

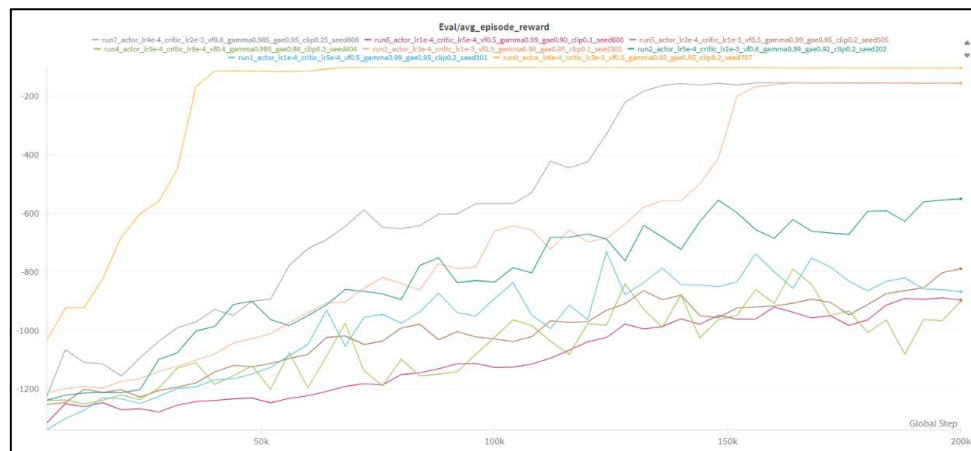


圖 5. Task2 不同參數的影響

Run	Actor LR	Critic LR	Gamma	GAE Lambda	Clip Coef	VF Coef	Seed	最終表現評估
run0	6.E-04	3.E-03	0.93	0.95	0.2	0.5	707	最佳，幾乎複製了第一次實驗的優異表現，再次證明該參數組合的穩健性。
run2	5.E-04	2.E-03	0.99	0.95	0.2	0.6	202	表現良好，學習曲線穩步上升。
run5	2.E-04	1.E-03	0.99	0.95	0.2	0.5	505	中等，學習相對緩慢但穩定，最終達到一個可接受的水平。
run3	3.E-04	1.E-03	0.98	0.95	0.1	0.5	303	中等，較小的 clip_coef 限制了學習速度，後期波動較大。
run4	3.E-04	8.E-04	0.995	0.98	0.3	0.4	404	中等偏下，較大的 clip_coef 和 gae_lambda 組合可能導致學習不穩定。
run1	1.E-04	5.E-04	0.99	0.92	0.2	0.5	101	學習緩慢，低 actor_lr 和低 gae_lambda 共同導致了性能不佳。
run7	4.E-04	2.E-03	0.985	0.95	0.25	0.6	808	表現不佳，學習曲線波動劇烈。
run6	1.E-04	5.E-04	0.99	0.9	0.1	0.5	606	最差，保守參數(低 actor_lr, gae_lambda, clip_coef) 疊加，幾乎無法學習。



## Task3



圖 6. Task3 不同參數的數據圖

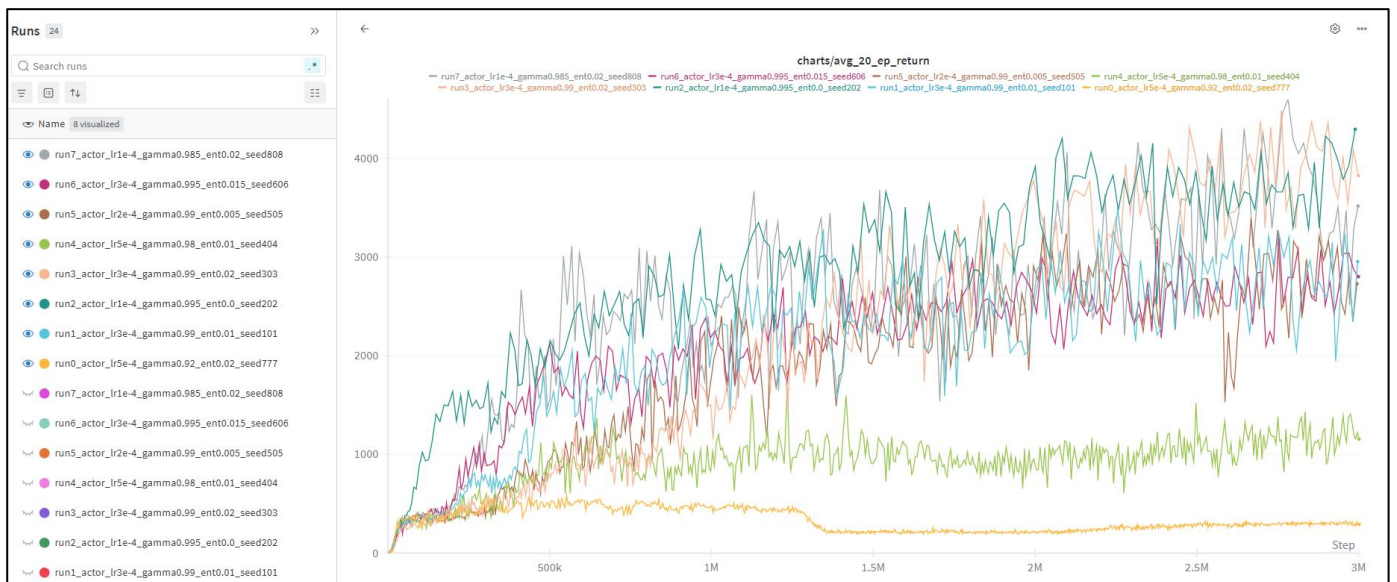


圖 7. Task3 不同參數對分數的影響圖

### 1. GAE Lambda ( $\lambda$ , gae\_lambda)

- 功能：控制偏差 (Bias) 與變異 (Variance) 的平衡，影響優勢函數估計。
- 趨近 0：低變異、高偏差 (單步 TD 誤差，穩定但依賴 Critic 準確性)。
- 趨近 1：低偏差、高變異 (MC 回報，真實但不穩定)。
- 建議值：0.95，普遍穩健。

### 2. Clipping Coefficient ( $\epsilon$ , clip\_coef)

- 功能：限制新舊策略更新幅度，維持穩定性 (Trust Region)。
- 小值 (如 0.1)：更新保守，穩定但慢。
- 大值 (如 0.3)：更新激進，快但易崩潰。
- 建議值：0.2，效率與穩定性平衡。

### 3. 更新輪數與小批量 (n\_epochs, num\_minibatches)

- n\_epochs：同批數據重複學習次數 (提升利用率，但過高會過擬合)。
- num\_minibatches：每輪訓練切分批數 (提升穩定性與泛化能力)。
- 建議做法：兩者配合使用以提高數據利用效率。

### 4. 價值函數係數 (vf\_coef)

- 功能：控制 Critic 損失在總損失中比重，平衡 Actor、Critic 學習重點。
- 過高：偏重 Critic，Actor 學習受抑制。
- 過低：Critic 學習不足，優勢估計含噪聲。
- 建議值：0.5 作為起點。

## 4. Additional analysis on other training strategies

### 4.1 網路初始化與權重衰減 (Network Initialization and Weight Decay)

良好的網路權重初始化是穩定訓練的基礎。程式碼中採用了正交初始化 (Orthogonal Initialization)，這是一種被證明在 RL 中能有效避免梯度爆炸或消失的技術。同時，在 Adam 優化器中加入了輕微的權重衰減 (Weight Decay)，作為一種 L2 正規化，有助於防止模型過擬合，提升泛化能力。

```
def layer_init(layer, std=np.sqrt(2), bias_const=0.0):
    """對神經網路的層進行正交初始化 (Orthogonal Initialization)"""
    torch.nn.init.orthogonal_(layer.weight, std)
    torch.nn.init.constant_(layer.bias, bias_const)
    return layer
```

程式碼 10. Orthogonal 程式碼

## 4.2 學習率線性衰減 (Learning Rate Annealing)

在訓練初期，較大的學習率有助於快速探索；而在訓練後期，較小的學習率則有助於策略收斂到穩定的最佳解。為此，實作了學習率線性衰減機制，讓 Actor 和 Critic 的學習率隨著總訓練步數的增加而線性下降至零。

```
# 學習率調度
if args.anneal_lr:
    frac = 1.0 - (global_step - 1.0) / args.max_steps
    lr_now = frac * args.actor_lr
    agent.actor_optimizer.param_groups[0]["lr"] = lr_now
    lr_now = frac * args.critic_lr
    agent.critic_optimizer.param_groups[0]["lr"] = lr_now
```

程式碼 11. 學習率調整方式

## 4.3 價值函數損失裁剪 (Value Function Loss Clipping)

與 PPO 的策略損失裁剪類似，對價值函數 (Critic) 的損失也進行裁剪。這個技巧可以防止因單次 TD 目標的誤差過大而導致價值函數產生劇烈更新，從而穩定 Critic 的學習過程。一個更穩定的 Critic 能為 Actor 提供更可靠的優勢函數估計。

```
# --- Critic 更新 (含價值裁剪) ---
if self.args.clip_vloss:
    v_loss_unclipped = (new_value - return_) ** 2
    v_clipped = old_value + torch.clamp(
        new_value - old_value, -self.args.clip_coef, self.args.clip_coef
    )
    v_loss_clipped = (v_clipped - return_) ** 2
    v_loss = 0.5 * torch.max(v_loss_unclipped, v_loss_clipped).mean()
else:
    v_loss = 0.5 * ((new_value - return_) ** 2).mean()
```

程式碼 12. 價值修剪

## 4.4 KL 散度監控與自適應學習率 (KL Divergence)

**KL 散度監控與早停：**KL 散度衡量了新舊策略之間的差異。如果在一次更新中，這個差異過大（超過 `kl_threshold`），意味著策略更新過於激進。此時，可以提前終止當前批次數據的更新，防止策略崩潰。

```
# 計算KL散度 (近似)
log_ratio = new_log_prob - old_log_prob
ratio = torch.exp(log_ratio)

with torch.no_grad():
    # 近似KL散度
    approx_kl = ((ratio - 1) - log_ratio).mean()
    epoch_approx_kls.append(approx_kl.item())

    # 裁剪比例統計
    clipfrac = ((ratio - 1.0).abs() > self.args.clip_coef).float().mean()
    clipfracs.append(clipfrac.item())
```

程式碼 13. 計算 KL 散度

**自適應學習率：**根據近期 KL 散度的平均值動態調整 Actor 的學習率。若 KL 散度持續偏高，則降低學習率；若持續偏低，則可以適度提高學習率。

```
# 自適應學習率調整
final_kl = np.mean(approx_kls[-5:]) if len(approx_kls) >= 5 else np.mean(approx_kls)
if final_kl > self.target_kl * 1.5:
    # KL太大, 降低學習率
    for param_group in self.actor_optimizer.param_groups:
        param_group['lr'] *= self.lr_decay_factor
elif final_kl < self.target_kl * 0.5:
    # KL太小, 可以稍微提高學習率
    for param_group in self.actor_optimizer.param_groups:
        param_group['lr'] = min(param_group['lr'] * self.lr_restore_factor,
                                self.initial_actor_lr)
```

程式碼 14. KL 自適應

## 4.5 動作標準差衰減 (Action Standard Deviation Decay)

此策略旨在平衡「探索-利用」。在訓練初期，較大的動作標準差能鼓勵探索。隨著訓練進行，可以逐步減小標準差，讓動作更集中在均值附近，從而進行更精細的利用並提升後期穩定性。

```
# 動作標準差衰減 (可選)
if args.action_std_decay:
    decay_factor = max(0.05, 1.0 - global_step / args.max_steps)
    with torch.no_grad():
        agent.actor.actor_logstd.data = torch.clamp(
            agent.actor.actor_logstd.data,
            -20,
            np.log(decay_factor)
        )
```

程式碼 15. 平衡動作選擇

## 5. Instruction

### 5.1 Train

Task1	<code>python .\LAB7_314551010_Code\a2c_pendulum.py</code>
Task2	<code>python .\LAB7_314551010_Code\ppo_pendulum.py</code>
Task3	<code>python .\LAB7_314551010_Code\ppo_walker.py</code>

### 5.2 Test(產生測試結果與影片)

Task1	<code>python .\LAB7_314551010_Code\a2c_pendulum.py --inference --load_model_path .\LAB7_314551010_task1_a2c_pendulum.pt</code>
Task2	<code>python .\LAB7_314551010_Code\ppo_pendulum.py --inference --load_model_path .\LAB7_314551010_task2_ppo_pendulum.pt</code>
Task3 (1M)	<code>python .\LAB7_314551010_Code\ppo_walker.py --inference --load_model_path .\LAB7_314551010_task3_ppo_1m.pt</code>
Task3 (1.5M)	<code>python .\LAB7_314551010_Code\ppo_walker.py --inference --load_model_path .\LAB7_314551010_task3_ppo_1p5m.pt</code>
Task3 (2M)	<code>python .\LAB7_314551010_Code\ppo_walker.py --inference --load_model_path .\LAB7_314551010_task3_ppo_2m.pt</code>
Task3 (2.5M)	<code>python .\LAB7_314551010_Code\ppo_walker.py --inference --load_model_path .\LAB7_314551010_task3_ppo_2p5m.pt</code>
Task3 (3M)	<code>python .\LAB7_314551010_Code\ppo_walker.py --inference --load_model_path .\LAB7_314551010_task3_ppo_3m.pt</code>

### 5.3 T 參考資料

[1] **jayin92**. *NYCU Deep Learning*. GitHub repository. Available: <https://github.com/jayin92/NYCU-deep-learning>

- [2] **c1uc**. *2025 Spring Deep Learning Labs*. GitHub repository. Available: [https://github.com/c1uc/2025\\_Spring\\_Deep-Learning-Labs](https://github.com/c1uc/2025_Spring_Deep-Learning-Labs)
- [3] **Part-time-Ray**. *DLP*. GitHub repository. Available: <https://github.com/Part-time-Ray/DLP>
- [4] **Shukkai**. *DLP-2025*. GitHub repository. Available: <https://github.com/Shukkai/DLP-2025>
- [5] **alu98753**. *NYCU Deep Learning 2025*. GitHub repository. Available: <https://github.com/alu98753/NYCU-Deep-Learning-2025>
- [6] **hastuma**. *2025\_DLP*. GitHub repository. Available: [https://github.com/hastuma/2025\\_DLP](https://github.com/hastuma/2025_DLP)
- [7] **shizheng\_Li**. *强化学习中的 GAE 原理与实现*. CSDN blog. Available: [https://blog.csdn.net/shizheng\\_Li/article/details/144436495](https://blog.csdn.net/shizheng_Li/article/details/144436495)
- [8] **YungHuiHsu**. *RL 演算法 — PPO*. HackMD. Available: <https://hackmd.io/@YungHuiHsu/SkUb3aBX6>
- [9] **Zhihu**. *強化學習相關討論*. Available: <https://www.zhihu.com/question/629107126>
- [10] **CSDN**. *深度學習與強化學習技術文章*. Available: [https://blog.csdn.net/weixin\\_45526117/article/details/126333385](https://blog.csdn.net/weixin_45526117/article/details/126333385)
- [11] **Zhihu**. *專欄. 強化學習文章*. Available: <https://zhuanlan.zhihu.com/p/28223597805>
- [12] **Vocus**. *強化學習文章*. Available: <https://vocus.cc/article/5af9cacafd89780001100b22>

## 5.4 使用的 AI 工具：

- ✧ **ChatGPT-4o** (由 OpenAI 提供)  
用於問答、報告草稿撰寫、英文轉中文及論文說明。
- ✧ **Google AI Studio**  
作為補充問答工具，用於模型概念、架構或函式撰寫思路輔助。
- ✧ **VSCode + Google Gemini 2.5 Pro**  
協助撰寫與補全 Python 程式碼，並支援簡易 debug。
- ✧ **VSCode GitHub Copilot (GPT-4o)**  
用於實作過程中進行即時程式補全、語法建議與除錯輔助。