HW3 多臂賭徒（Multi-Armed Bandit, MAB）報告

學號:7113056041 姓名:蔡承晏

⭐演算法一: Epsilon-Greedy

1.演算法公式

若當前時間步為 t ，則選擇行動 a_t 的策略如下：

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \varepsilon \\ \arg\max_a Q_t(a), & \text{with probability } 1-\varepsilon \end{cases}$$

- ε：探索參數:通常為 0.1 或 0.01，用來控制隨機探索的比例
- $Q_t(a)$: 第 t 步對臂 a 的期望獎勵估計值
- a_t : 對 t 步選擇的拉霸機

2.ChatGPT Prompt

"請說明 Epsilon-Greedy 演算法如何在強化學習中平衡探索與利用。"

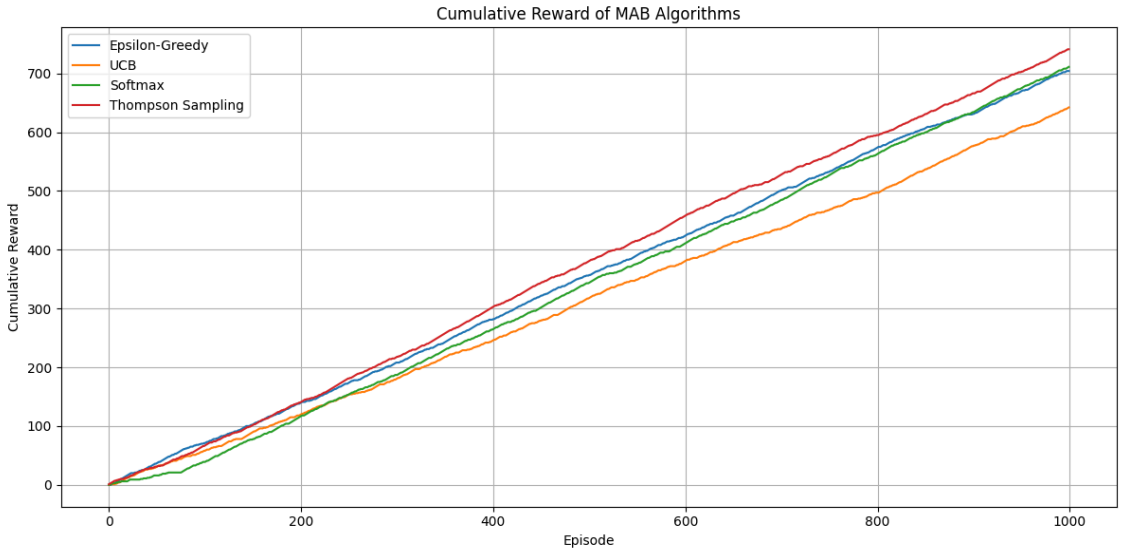
3.程式碼與圖表

- 程式碼

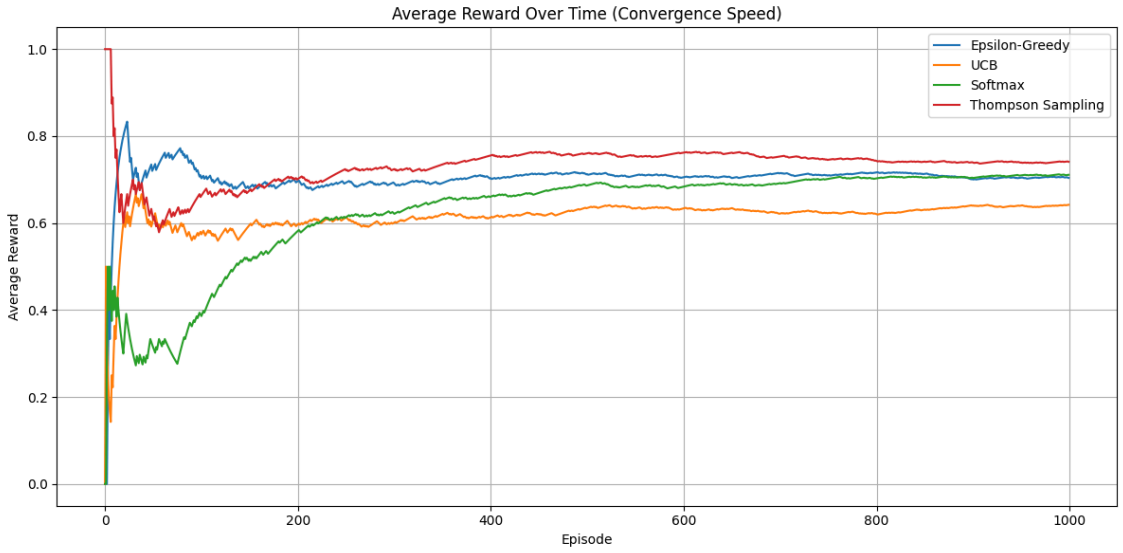
```
# ========== 1. Epsilon-Greedy ==========
def epsilon_greedy(bandit, episodes=1000, epsilon=0.1):
    counts = np.zeros(bandit.n_arms)
    values = np.zeros(bandit.n_arms)
    rewards = []

    for t in range(episodes):
        if np.random.rand() < epsilon:
            arm = np.random.choice(bandit.n_arms)
        else:
            arm = np.argmax(values)

        reward = bandit.pull(arm)
        counts[arm] += 1
        values[arm] += (reward - values[arm]) / counts[arm]
        rewards.append(reward)

    return np.cumsum(rewards)
```

- 累積獎勵圖 :
  - x-軸 : 試玩次數(1 到 1000)

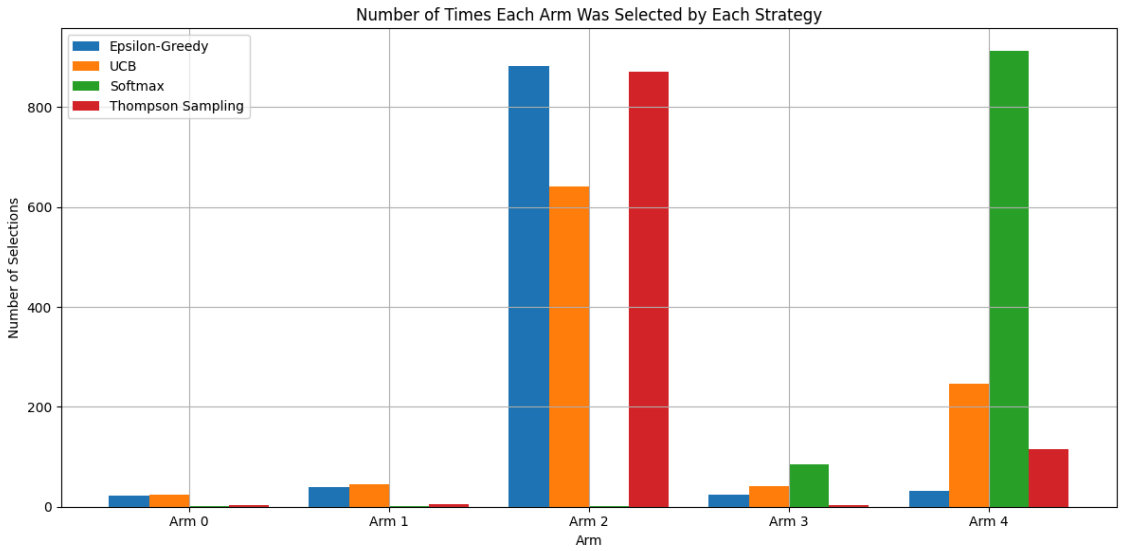
- y-軸：累積總獎勵
- 意義：顯示隨著時間累積的表現好壞



- 平均獎勵圖：
  - x-軸：試玩次數
  - y-軸：每次選擇的平均回報
  - 意義：是否收斂至最佳拉霸機



- 拉霸選擇次數圖：顯示每台拉霸機被選擇的次數



4.結果解釋

1.空間與時間的觀點：

時間面分析：

- 在演算法初期（回合數較少時），由於 Epsilon-Greedy 會以 ϵ 的機率隨機探索，每個拉霸臂都有被嘗試的機會，因此回報會波動較大。
- 隨著回合數增加，演算法會以 $1-\epsilon$ 的機率選擇目前預估回報最高的臂，導致行為漸漸趨向「利用」，平均獎勵逐漸收斂。
- 若觀察「平均獎勵」圖，可以看到曲線初期不穩，後期穩定上升並收斂，顯示演算法學會辨認最佳臂。

空間面分析：

- Epsilon-Greedy 不需儲存整個歷史資料，只需記錄每個臂的：
  - 總獎勵 $Q(a)$
  - 拉霸次數 $N(a)$
- 這使得它在空間上非常輕量級，適合處理臂數很多的場景。

2.優勢與限制分析：

✅優勢：

- **實作簡單、直觀易懂**：核心邏輯清晰，只需控制一個超參數 ϵ。
- **控制探索程度**：透過調整 ϵ，可靈活改變探索與利用的平衡。
- **適用於非貝葉斯模型**：不需假設回報分佈或使用貝葉斯更新。

❌限制：

- **固定探索率**：即使已經明顯辨認出最佳臂，仍會以 ϵ 的機率做無意義的隨機探索，導致浪費資源。
- **缺乏不確定性考量**：演算法並未考慮「對某臂的信心程度」，僅根據平均獎勵值作選擇。
- **探索效率較低**：與 UCB、Thompson Sampling 相比，Epsilon-Greedy 在探索初期表現較弱，收斂速度較慢。

3.在不同情境下的表現差異：

情境	表現說明
✅ 臂數較少	表現穩定，因為探索成本較低
⚠️ 臂數眾多	表現不佳，過多隨機探索會浪費時間
✅ 報酬穩定（stationary）環境	表現良好，因為探索後的學習結果不易變化
❌ 報酬變動（non-stationary）環境	固定 ϵ 容易導致無法適應環境變化，需引入隨時間衰減的探索率

⭐演算法二: UCB（Upper Confidence Bound）

1.演算法公式

在第 t 步，選擇行動 a_t 的規則為:

$a_t = \arg\max_a \left[Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}} \right]$

- $Q_t(a)$: 到目前為止對臂 a 的平均回報估計
- $N_t(a)$: 臂 a 被拉過的次數
- t : 當前總的步數(回合數)
- c : 控制探索程度的常數 (例如 $c = 2$)

這個公式中，第二項 :

$\sqrt{\frac{\ln t}{N_t(a)}}$ 代表不確定性(confidence bound)，當某臂被選的少($N_t(a)$小)時，這一項會變大，鼓勵探索。

2.ChatGPT Prompt

"UCB 演算法如何透過不確定性來指引決策？"

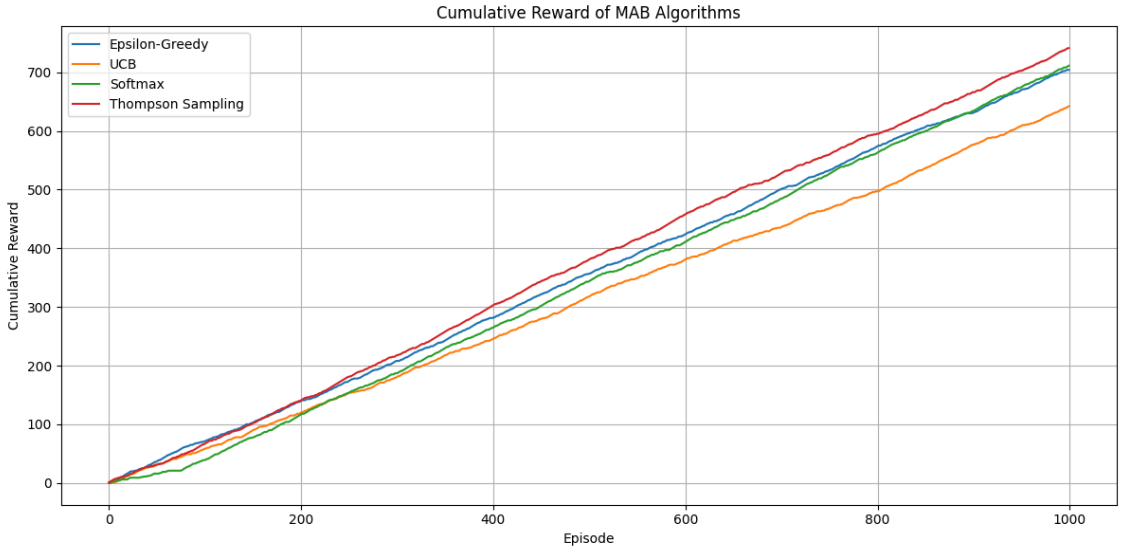
3.程式碼與圖表

- 程式碼

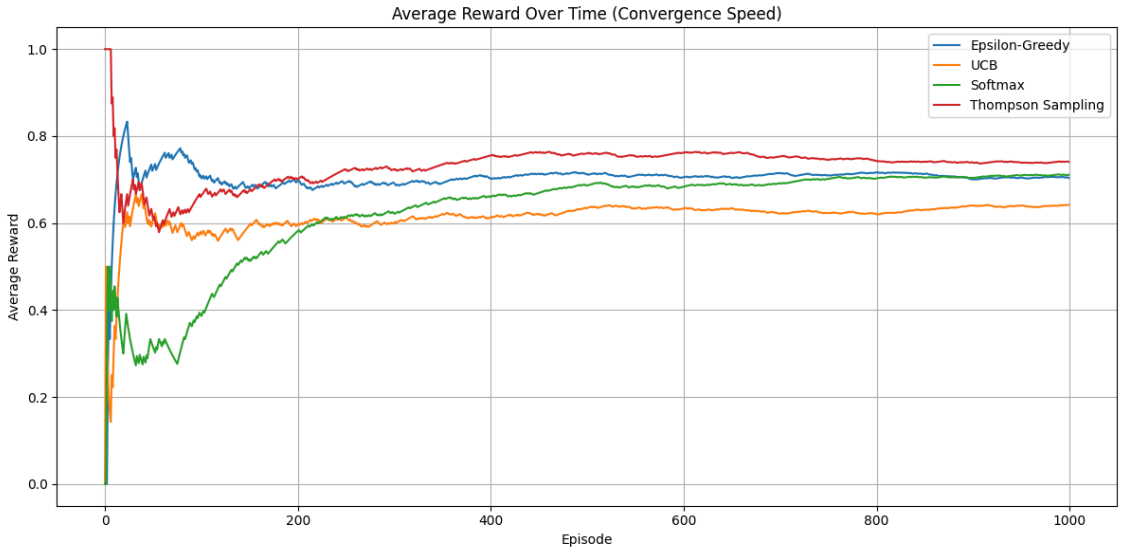
```
# ========== 2. Upper Confidence Bound (UCB) ==========
def ucb(bandit, episodes=1000):
    counts = np.zeros(bandit.n_arms)
    values = np.zeros(bandit.n_arms)
    rewards = []

    for t in range(1, episodes + 1):
        if 0 in counts:
            arm = np.argmin(counts)
        else:
            confidence_bounds = values + np.sqrt(2 * np.log(t) / counts)
            arm = np.argmax(confidence_bounds)

        reward = bandit.pull(arm)
        counts[arm] += 1
        values[arm] += (reward - values[arm]) / counts[arm]
        rewards.append(reward)
```

```
return np.cumsum(rewards)
```

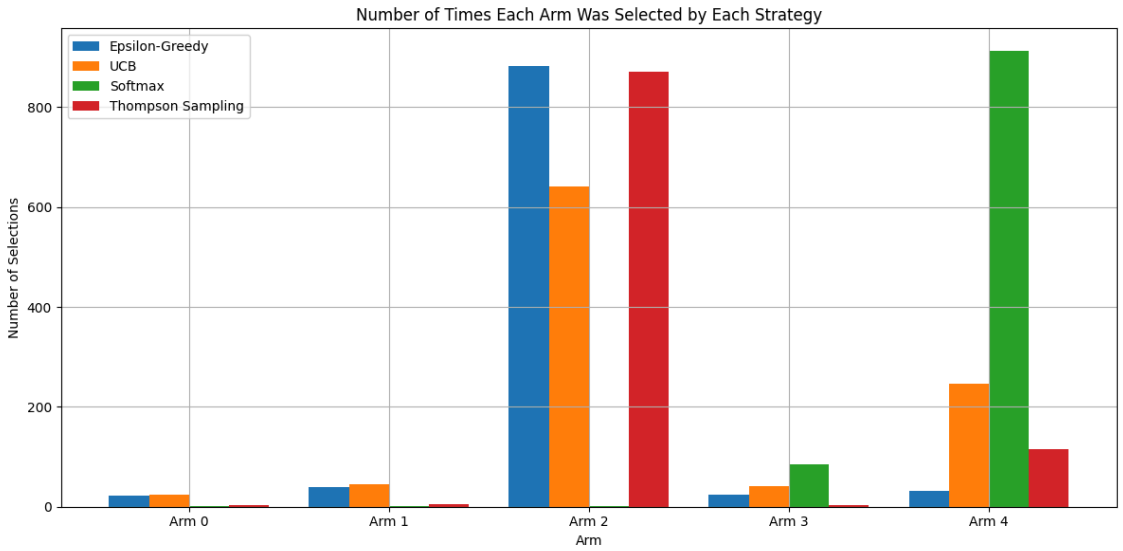
- 累積獎勵圖：
  - x-軸：試玩次數(1 到 1000)
  - y-軸：累積總獎勵
  - 意義：顯示隨著時間累積的表現好壞



- 平均獎勵圖：
  - x-軸：試玩次數
  - y-軸：每次選擇的平均回報
  - 意義：是否收斂至最佳拉霸機



- 拉霸選擇次數圖：顯示每台拉霸機被選擇的次數



4.結果解釋

1.空間與時間的觀點：

時間面分析：

- UCB在每一步都會考慮每個臂的平均回報以及其不確定性(探索項)，具體公式為：

$$UCB_i(t) = \hat{\mu}_i(t) + c \cdot \sqrt{\frac{2 \ln t}{n_i(t)}}$$

- $\hat{\mu}_i$：臂 i 的平均回報
- n_i：臂 i 被選擇的次數
- t：目前總的嘗試次數
- 第二項會隨著 n_i 增加而下降，意味著：拉得少的臂會被更常探索
  - **早期：** 所有臂都會獲得探索機會，探索項大，鼓勵廣泛探索
  - **後期：** 被選擇次數多的臂其探索項會縮小，逐漸集中於高回報臂，收斂速度加快

從「平均獎勵」與「累積獎勵」圖表中可見，UCB 的曲線收斂速度快，並且整體回報表現佳。

空間面分析：

- UCB 僅需維護每個臂的：
  - 回報總和 / 平均回報 $\hat{\mu}_i$
  - 拉霸次數 n_i

2.優勢與限制分析：

✅優勢：

- **動態調整探索：** 自動根據不確定性（拉得次數少）決定是否繼續探索。
- **理論基礎穩固：** 有明確的理論保證，UCB 在最壞情況下仍可達到良好的 regret 上限。
- **收斂快：** 在穩定環境下比 Epsilon-Greedy 更快達到穩定回報。

❌限制：

- **不適合非固定報酬（Non-stationary）環境：** 一旦臂的分佈變動，UCB 不會重新探索，可能導致卡在錯誤選擇上。
- **初期探索開銷較大：** UCB 為了保障信心上界，會對不常拉的臂做較多探索。
- **需依賴精確的回合計數（t）與拉次數（n）**：若資料稀疏或部分資訊缺失，表現會不穩。

3.在不同情境下的表現差異：

情境	表現說明
✅ 臂數中等，報酬穩定	表現非常佳，快速辨識出最佳臂
✅ 需要高探索效率的場景	藉由探索項導引探索，節省探索資源
❌ 報酬不穩定的場景（非平穩）	一旦初期誤判，後期幾乎不會修正
⚠️ 高臂數（如上百臂）	每一臂的回報估計與信心項維護仍簡單，但探索初期會較久

⭐演算法三: Softmax

1.演算法公式

在第 t 步中，選擇每個臂 a 的機率為:

$$P_t(a) = \frac{e^{Q_t(a)/\tau}}{\sum_b e^{Q_t(b)/\tau}}$$

- $P_t(a)$：在時間 t 選擇臂 a 的機率
- $Q_t(a)$：臂 a 的平均回報估計值
- τ：溫度參數(temperature)，控制探索與利用的平衡

溫度參數解釋：

- 當 $\tau \to 0$：接近貪婪策略(只選最大回報的臂)
- 當 $\tau \to \infty$：接近完全隨機選擇(所有臂機率相同)

這種方式讓回報較高的臂擁有較高機率被選中，但仍保留了一定的隨機性來探索其他臂。

2.ChatGPT Prompt

"請說明 Softmax 演算法中的溫度參數 對選擇策略的影響。"

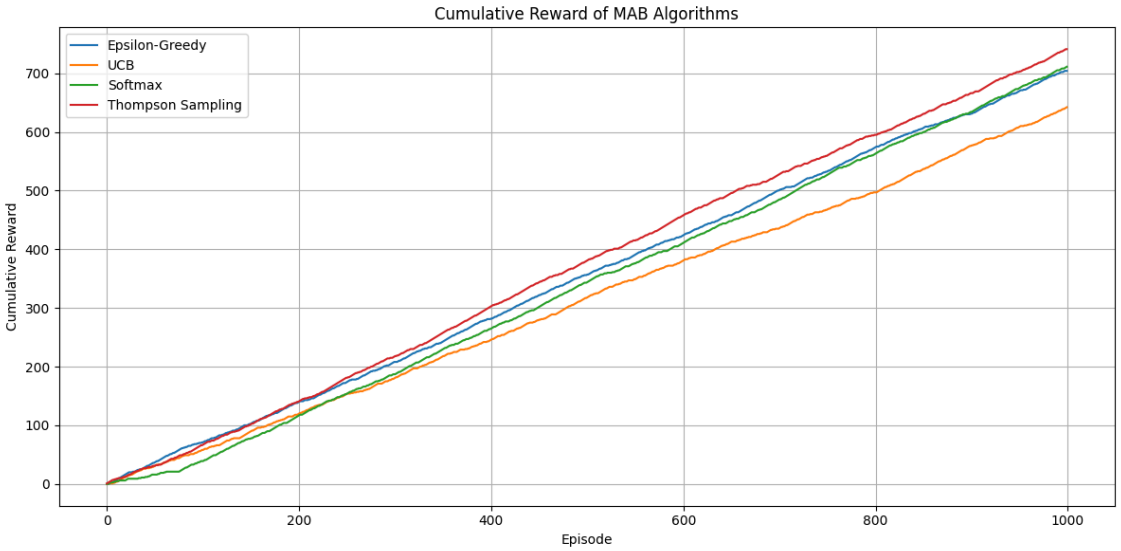
3.程式碼與圖表

- 程式碼

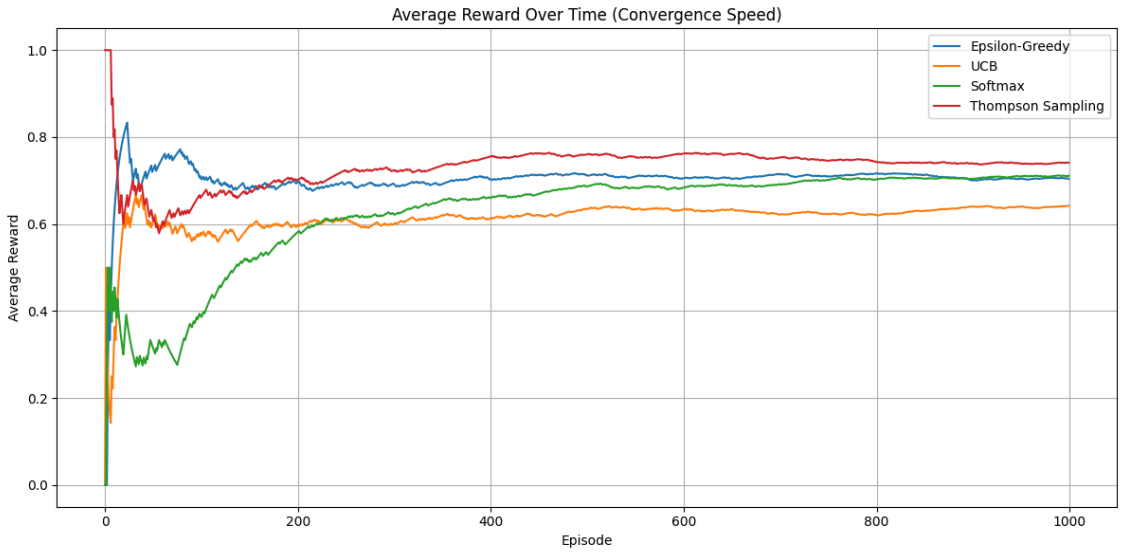
```
# ========== 3. Softmax ==========
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

def softmax_strategy(bandit, episodes=1000, tau=0.1):
    counts = np.zeros(bandit.n_arms)
    values = np.zeros(bandit.n_arms)
    rewards = []

    for _ in range(episodes):
```

```
        probs = softmax(values / tau)
        arm = np.random.choice(bandit.n_arms, p=probs)
        reward = bandit.pull(arm)
        counts[arm] += 1
        values[arm] += (reward - values[arm]) / counts[arm]
        rewards.append(reward)

    return np.cumsum(rewards)
```

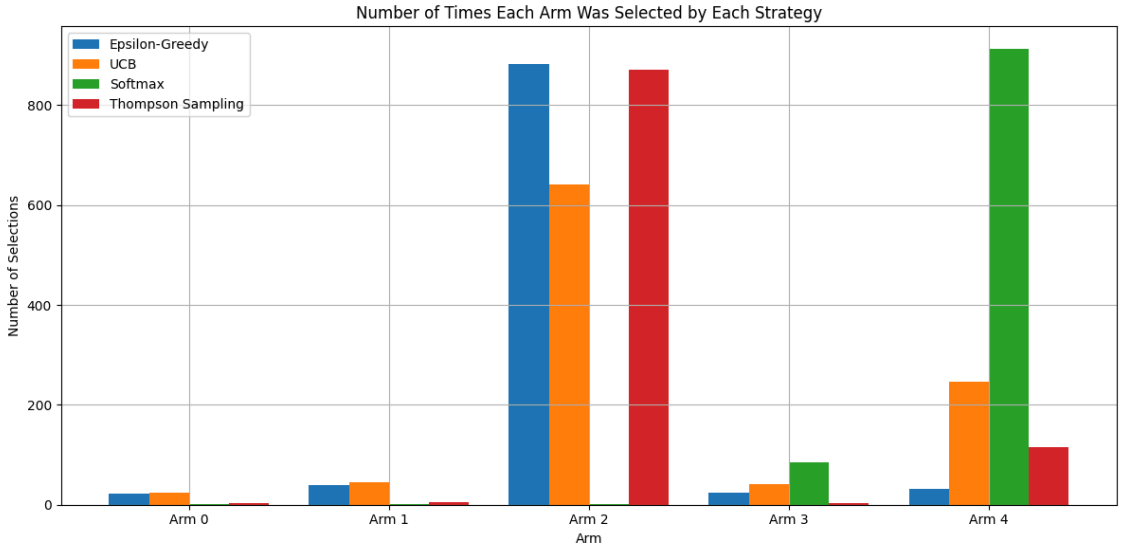
- 累積獎勵圖：
  - x-軸：試玩次數(1 到 1000)
  - y-軸：累積總獎勵
  - 意義：顯示隨著時間累積的表現好壞



- 平均獎勵圖：
  - x-軸：試玩次數
  - y-軸：每次選擇的平均回報
  - 意義：是否收斂至最佳拉霸機



- 拉霸選擇次數圖：顯示每台拉霸機被選擇的次數



4.結果解釋

1.空間與時間的觀點：

時間面分析：

- Softmax 演算法根據每個臂的「預估價值 $Q(a_i)$」計算機率，再以機率進行選擇。
- 透過控制參數 τ（溫度）來調整探索程度：
  - 高溫 $\tau \to \infty$:各臂機率接近均勻（偏重探索）
  - 低溫 $\tau \to 0$:偏好回報高的臂（偏重利用）

在「平均獎勵收斂圖」中可觀察到，Softmax 收斂速度通常介於 Epsilon-Greedy 與 UCB 之間。

空間面分析：

- 與其他演算法類似，只需儲存：

    - 每個臂的平均回報 $Q(a_i)$
    - 被選擇次數
- 空間複雜度為 $O(k)$

2.優勢與限制分析：

✅優勢：

- **平滑探索與利用平衡：** 相較 Epsilon-Greedy 的硬切換（隨機 vs 最佳），Softmax 提供連續的權重分配。
- **對高回報臂更有偏好**，但不完全放棄其他臂，較容易在 early stage 發現潛力臂。
- **可透過溫度參數 τ 動態調整策略**。

❌限制：

- **需調整溫度參數 τ** ：過高會導致太多探索、過低則幾乎等於貪婪法。
- **計算開銷略高於 Epsilon-Greedy**：每次選擇需計算 Softmax 機率分佈。
- **收斂速度受溫度影響大**：若沒選好 τ ，可能導致收斂變慢或過早收斂。

3.在不同情境下的表現差異：

情境	表現說明
✅ 臂數適中、回報接近	有機會探索所有臂，識別細微差異
✅ 需平衡探索與利用的場景	透過 τ 控制探索程度，靈活調整
❌ 高臂數、回報差異極大	高回報臂會壓倒性勝出，低回報臂幾乎無機會被選
⚠️ 非平穩環境	若沒調整 τ，早期收斂後會錯過環境變化

⭐演算法四: Thompson Samplin

1.演算法公式

對每一臂 a，根據其回報的貝葉斯後驗分佈進行抽樣，選擇具有最大抽樣回報的臂：

$\theta_a \sim \text{Beta}(\alpha_a, \beta_a)$

- θ_a :臂 a 的後經驗回報估計值。從Beta分布中抽樣
- α_a 和 β_a :Beta分布的形狀參數，代表該臂的成功和失敗的機率
    - 若獲得獎勵，則 α_a 增加1
    - 若獲得懲罰，則 β_a 增加1
- 選擇臂：在每一步，選擇 $argmax_a\theta_a$ 的臂

更新規則：

- 初始時，每個臂的 $\alpha_a = 1$ 和 $\beta_a = 1$ (Beta分布的均勻分布)
- 每次獲得回報時，根據回報更新Beta分布的參數

2.ChatGPT Prompt

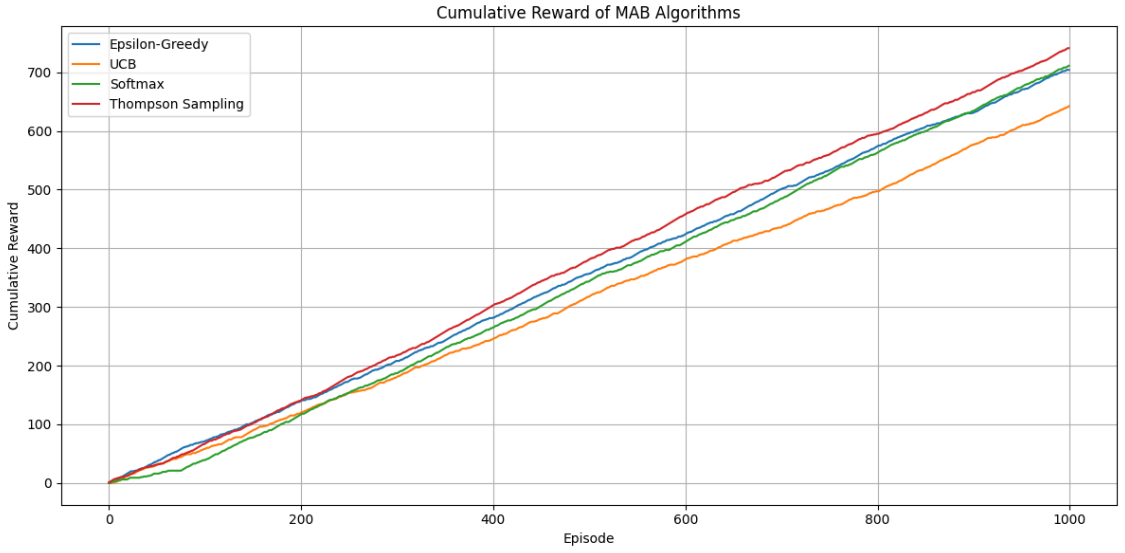
"請解釋貝葉斯觀點下的 Thompson Sampling 是如何實現探索與利用的平衡。"

3.程式碼與圖表

- 程式碼

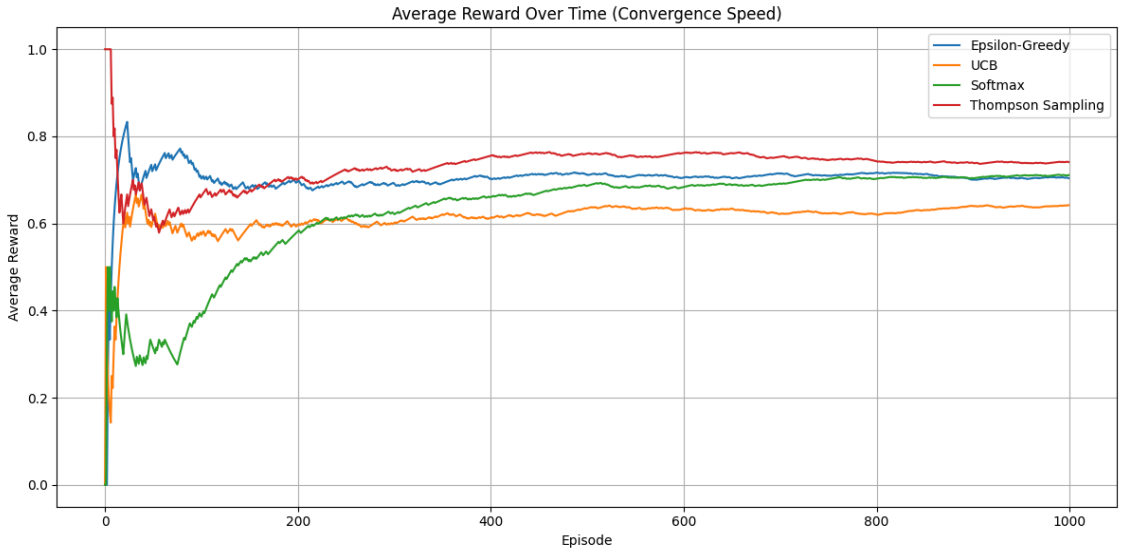
```
# ========= 4. Thompson Sampling =========
def thompson_sampling(bandit, episodes=1000):
    alpha = np.ones(bandit.n_arms)
    beta = np.ones(bandit.n_arms)
    rewards = []

    for _ in range(episodes):
        samples = np.random.beta(alpha, beta)
        arm = np.argmax(samples)
        reward = bandit.pull(arm)

        if reward == 1:
            alpha[arm] += 1
        else:
            beta[arm] += 1

        rewards.append(reward)

    return np.cumsum(rewards)
```

- 累積獎勵圖：
  - x-軸：試玩次數(1 到 1000)
  - y-軸：累積總獎勵
  - 意義：顯示隨著時間累積的表現好壞

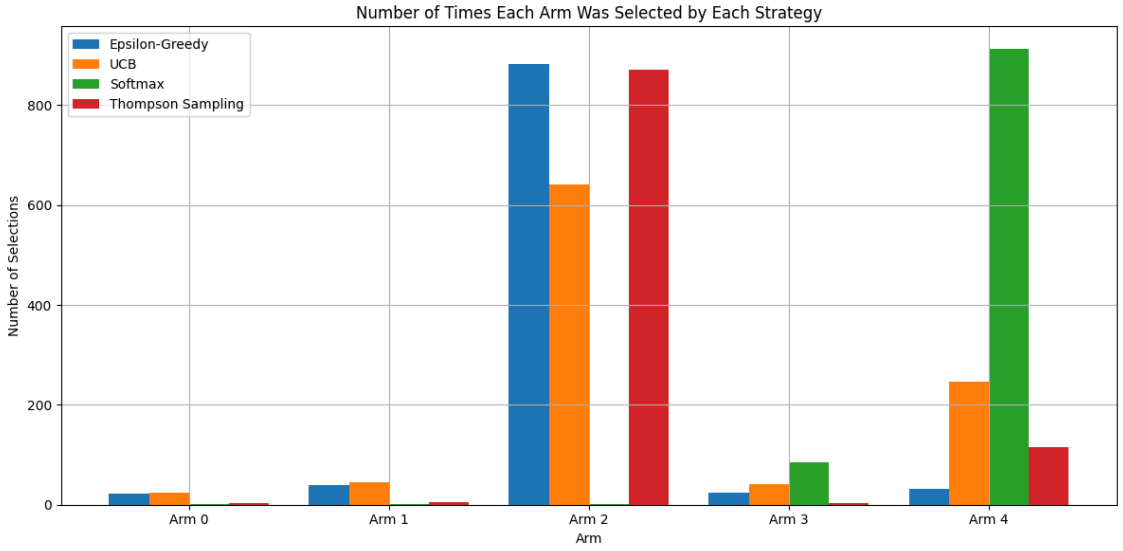


- 平均獎勵圖：
  - x-軸：試玩次數

- y-軸：每次選擇的平均回報
- 意義：是否收斂至最佳拉霸機



- 拉霸選擇次數圖：顯示每台拉霸機被選擇的次數



- 三張圖顯示效果皆最佳：累積回報高，平均獎勵收斂快，選擇正確拉霸機的次數多

4.結果解釋

1.空間與時間的觀點：

時間面分析：

- 每回合從每個臂的 Beta 分佈中抽樣一個值 θ_i，選擇最大者。
- 初期探索較多，隨著試驗次數增加，分佈越集中，策略自然趨向利用。
- 從圖表可見，**Thompson Sampling 的收斂速度與累積獎勵表現均非常優秀**，尤其在拉霸機數量多時更顯穩定。

空間面分析：

- 每個臂只需維護兩個參數：成功次數 α_i、失敗次數 β_i。
- 空間複雜度為 $O(k)$

2.優勢與限制分析：

✅優勢：

- **自然平衡探索與利用**：透過抽樣實現「樂觀初始化」與「機率式探索」。
- **收斂速度快、穩定性高**，特別在早期表現優於其他方法。
- 不需人工設定 epsilon 或 temperature 等超參數。

❌限制：

- **需明確的先驗分佈**（如 Bernoulli 對應 Beta）：若應用場景不符合，模型效果可能下降。
- **計算上需進行抽樣操作**：但現代硬體下這點開銷微乎其微。

3.在不同情境下的表現差異：

情境	表現說明
✅ 獎勵為 Bernoulli（0/1）	非常適合，Beta 分佈能有效建模回報不確定性
✅ 臂數多、資料稀疏	抽樣方式可快速辨識最佳臂，避免陷入次佳選擇
❌ 連續或高斯型回報	須改用其他先驗（如 Gaussian），需更多實作考量
⚠️ 多臂差距小	可能需較多試驗以區分最優臂，但仍能穩健收斂