Reinforcement Learning for BESS

4 June 2025

Part of the Scada Automation AI Initiative

วัตถุประสงค์

- โค้ดนี้เป็นการสร้างโมเดล Q-learning สำหรับควบคุมแบตเตอรี่ (Battery Energy Storage System - BESS) โดยมีวัตถุประสงค์เพื่อ:

- 1 ลดภาระโหลดจากกริด (Peak Shaving)
- 2 ลดค่าไฟฟ้า โดยเลือกเวลาในการชาร์จและปล่อยพลังงานจากแบตเตอรี่อย่างเหมาะสม แบ่งเป็น ชาร์จ, ดีสชาร์จ และ ไม่ทำอะไร

⚙️ ส่วนที่ 1: การตั้งค่าระบบ (Settings)

- HOURS = 24: จำลอง 24 ชั่วโมง
- SOC_LEVELS = 6: แบ่งสถานะแบตเตอรี่เป็น 6 ระดับ
- ACTIONS: กำหนด 3 การกระทำที่เป็นไปได้ในแต่ละชั่วโมง

```
HOURS = 24
SOC_LEVELS = 6  # มี 6 ระดับ SoC: 0%, 20%, ..., 100%
ACTIONS = ['Do Nothing', 'Charge', 'Discharge']
```

📋 ส่วนที่ 2: ข้อมูลต้นทาง

- electricity_prices: ราคาค่าไฟต่อชั่วโมงแบบ TOU (Time of Use)
- load_profile: กำลังไฟที่ใช้ของผู้ใช้ต่อชั่วโมง
- HIGH_LOAD_THRESHOLD: ตั้งเป้าว่าไม่ควรให้โหลดเกิน 6 kW

```
electricity_prices = [...]
load_profile = [...]
HIGH_LOAD_THRESHOLD = 6.0  # kW
```

🧠 ส่วนที่ 3: Q-learning

- 🎯 **เป้าหมาย:**
- เพื่อให้แบตเตอรี่เรียนรู้ว่า:
- ชาร์จตอนไหนคุ้ม
- ปล่อยพลังงานตอนไหนช่วยลดพีค
- หลีกเลี่ยงพฤติกรรมสิ้นเปลืองหรืออันตราย (เช่น discharge ตอนแบตหมด)

💡 หลักการคำนวณรางวัลในโค้ด

- รางวัลรวม =(- ค่าพลังงานที่ดึงจากกริด)
- ได้โบนัสหากลดโหลดพีคได้
- ค่าปรับ (penalty) ถ้าทำผิด เช่น charge ตอนแพง

```
def get_reward(hour, action, soc_level, battery_power, grid_power):
```

🧮 รายละเอียดที่ระบบให้รางวัล (และลงโทษ)

- ✅ **กรณีที่ได้รางวัล (positive reward)**
- ปล่อยพลังงานช่วงราคาสูง → ลดค่าไฟ
- ปล่อยพลังงานช่วงโหลดสูง → ลด peak load
- ชาร์จช่วงราคาต่ำ (เช่นตอนกลางคืน)
- ปล่อยพลังงานช่วง 14:00-17:00 → มีโบนัสเพิ่มเติม (high penalty time if load exceeds)

- action คือการกระทำของระบบในช่วงเวลานั้น (0 = do nothing, 1 = charge, 2 = discharge)
- hour คือชั่วโมงของวัน (ใช้เพื่อดูราคาค่าไฟและโหลด)
- soc คือระดับพลังงานของแบตเตอรี่ในหน่วยระดับ เช่น 0 %, 20%,... 100% เต็ม

```
def get_reward(action, hour, soc):
```

- ดึงราคาค่าไฟ (price) และ โหลดไฟฟ้า (load) สำหรับชั่วโมงนั้น

```
price = electricity_prices[hour]
load = load_profile[hour]
```

- ค่าเริ่มต้นของรางวัล

```
base_reward = 0.0
```

💤 กรณีที่ 1: action == 0 (ไม่ทำอะไร)

- ถ้าโหลดสูงเกิน (กำลังเกินจากกริด) แต่มีแบตให้ใช้ แต่ action == 0 → ลงโทษ -20 เพราะคุณควรปล่อยแบตออกมาช่วย
- ถ้าโหลดไม่สูง หรือไม่มีพลังในแบต → รางวัลเป็น 0 (กลางๆ ไม่ดีไม่แย่)

```
if action == 0:
    if load > HIGH_LOAD_THRESHOLD and soc > 0:
        base_reward = -20.0
    else:
        base_reward = 0
```

🔋 กรณีที่ 2: action == 1 (ชาร์จแบตเตอรี่)

- ถ้า SOC ยังไม่เต็ม → ชาร์จได้
  - ลงโทษตาม -price (ค่าไฟ) → เพื่อสอนว่าอย่าชาร์จตอนราคาแพง
  - ถ้าโหลดเกินหรือค่าไฟ = 4.00 (แพงที่สุด) → ลงโทษเพิ่มอีก -100 จุด
- ถ้า SOC เต็มอยู่แล้วยังพยายามชาร์จ → ลงโทษ -10 (เป็น action ที่ไม่ควรทำ)

```
elif action == 1:
    if soc < SOC_LEVELS - 1:
        base_reward = -price
        if load > HIGH_LOAD_THRESHOLD or price == 4.00:
            base_reward -= 100.0
    else:
        base_reward = -10.0
```

⚡ กรณีที่ 3: action == 2 (จ่ายไฟจากแบตเตอรี่)

- ถ้าแบตไม่ว่าง → ให้รางวัล = price (ยิ่งใช้แบตตอนราคาแพงยิ่งดี)

  - ถ้าโหลดเกิน → เพิ่มรางวัล +15 (ช่วยลดโหลด)
  - ถ้าราคาแพงที่สุด (4.00) → บวกเพิ่มอีก +2

- ถ้าแบตว่างแล้วยังพยายามจ่ายไฟ → ลงโทษ -10 (invalid action)

```
elif action == 2:
    if soc > 0:
        base_reward = price
        if load > HIGH_LOAD_THRESHOLD:
            base_reward += 15.0
        if price == 4.00:
            base_reward += 2.0
    else:
        base_reward = -10.0
```

✅ สรุปแนวคิดของโค้ด

การกระทำ	เงื่อนไขที่ดี	รางวัลที่ได้	เงื่อนไขที่แย่	โทษที่ได้รับ
Do nothing	โหลดไม่เกิน	0	โหลดเกินแต่ไม่จ่ายแบต	-20
Charge	SOC ไม่เต็ม + ช่วงไฟถูก	-น้อย	SOC เต็ม หรือช่วงไฟแพง	-10 ถึง -100
Discharge	SOC > 0 + ช่วงไฟแพง + โหลดเกิน	+price ถึง +17	SOC = 0	-10

🔄 ลูปการฝึก (Training Loop)

- สุ่ม SoC เริ่มต้น
- ฝึกให้ Q-table ค่อยๆ ดีขึ้น
- ลดค่า epsilon ทีละน้อยเพื่อเปลี่ยนจาก exploration → exploitation
- วนซ้ำฝึกรอบละ 1 วัน (episode = 1 วันมีหลายชั่วโมง)

🔁 เริ่มต้นแต่ละ episode

- ปรับ ε (epsilon) สำหรับการ explore/exploit
- เริ่มต้น ε สูง (สุ่มเยอะ) แล้วค่อยๆ ลดลง (มั่นใจมากขึ้นใน Q-table)

```
for episode in range(EPISODES):
    current_epsilon = EPSILON_END + (EPSILON_START - EPSILON_END) * np.exp(-EPSILON_DECAY_RATE * episod
```

- เริ่มต้นที่ชั่วโมง 0
- สุ่มระดับพลังงานแบต (SOC) เริ่มต้นในแต่ละรอบ เพื่อให้ระบบเรียนรู้จากสถานการณ์ต่างๆ

```
total_reward = 0
hour = 0
soc = random.randint(0, SOC_LEVELS - 1)
```

🔄 วนลูปแต่ละชั่วโมงในวันนั้น

```
while hour < HOURS:
```

✅ เลือก action จากนโยบาย Q-table + exploration

```
action = choose_action(hour, soc, current_epsilon)
```

🎁 คำนวณรางวัลเบื้องต้น

- เรียกฟังก์ชัน get_reward() ที่ให้มาก่อนหน้านี้ เพื่อคำนวณรางวัลตามกติกา SOC, ราคาไฟ, โหลด ฯลฯ

```
base_reward_from_func = get_reward(action, hour, soc)
```

⚠ คำนวณ กำลังที่ต้องการจาก grid

- ถ้าชาร์จ → กำลังที่ต้องการจากกริด = โหลด + พลังชาร์จที่แบตต้องการ
- ถ้าจ่ายไฟจากแบต →กำลังที่ต้องการกริด = โหลด - พลังที่แบตจ่าย (แต่ไม่น้อยกว่า 0)

```
if action == 1 and soc < SOC_LEVELS - 1: # Charging
    instantaneous_grid_power = current_load + BATTERY_POWER_CHARGE
elif action == 2 and soc > 0: # Discharging
    instantaneous_grid_power = max(0, current_load - BATTERY_POWER_DISCHARGE)
else:
    instantaneous_grid_power = current_load
```

เพิ่มโทษหากโหลดเกิน

- ถ้ากำลังที่ต้องการจากกริดมากเกินกว่า HIGH_LOAD_THRESHOLD → ลงโทษเพิ่ม
- ช่วง 16:00 น. โทษ x5, 17:00 น. โทษ x2 (ช่วง peak สำคัญ)

```
if instantaneous_grid_power > HIGH_LOAD_THRESHOLD:
    base_threshold_penalty = -50.0
    if hour == 16:
        additional_penalty = base_threshold_penalty * 5.0
    elif hour == 17:
        additional_penalty = base_threshold_penalty * 2.0
    else:
        additional_penalty = base_threshold_penalty
```

✍️ รวมรางวัล

```
reward = base_reward_from_func + additional_penalty
```

```
total_reward += reward
```

🔄 อัปเดตสถานะถัดไป (hour และ soc)

- ถ้าชาร์จ → SOC เพิ่ม
- ถ้าจ่ายไฟ → SOC ลด
- ถ้าไม่ทำอะไร → SOC เท่าเดิม

```
next_hour = hour + 1
next_soc = soc
if action == 1 and soc < SOC_LEVELS - 1:
    next_soc += 1
elif action == 2 and soc > 0:
    next_soc -= 1
```

📘 อัปเดตค่า Q-table (Bellman update)

- หาค่า **Q สูงสุดในสถานะถัดไป** (hour+1, soc ใหม่)

```
if next_hour < HOURS:
    best_future_q = np.max(Q[next_hour, next_soc])
else:
    best_future_q = 0
```

- คำนวณค่า TD Target (รางวัล + ความคาดหวังอนาคต)
- ปรับค่า Q ด้วยอัตราการเรียนรู้ ALPHA

```
td_target = reward + GAMMA * best_future_q
td_error = td_target - Q[hour][soc][action]
Q[hour][soc][action] += ALPHA * td_error
```

⏩ ไปยังชั่วโมงถัดไป

```
hour = next_hour
soc = next_soc
```

📊 สรุปรางวัลแต่ละรอบ

```
episode_rewards.append(total_reward)
```

🖨 แสดงผลทุก 10,000 รอบ

```
if (episode + 1) % 10000 == 0:
    print(f"Episode {episode + 1}/{EPISODES}, Epsilon: {current_epsilon:.4f}, Total Reward:
```

🧠 สรุปใจความสำคัญ

จุดสำคัญ	คำอธิบาย
Q-table	เก็บค่าคาดหวัง (Q-value) สำหรับแต่ละ state-action
SOC	บ่งบอกพลังงานคงเหลือในแบตเตอรี่
Epsilon	ควบคุมความสมดุลระหว่างการสุ่มลอง (explore) กับใช้ค่าที่ดีที่สุด (exploit)
Penalty & Reward	ถูกออกแบบอย่างรัดกุม เช่น โทษหนักหากโหลดเกินช่วง peak
Goal	สร้างนโยบายควบคุมแบตเตอรี่อัตโนมัติ เพื่อลด peak load และใช้พลังงานอย่างชาญฉลาด

📚 ส่วนที่ 4: สร้าง Policy และจำลองวันจริง

- โค้ดส่วนนี้ทำหน้าที่ สรุปนโยบายการตัดสินใจ จากผลลัพธ์ของการเรียนรู้ด้วย Q-learning ซึ่งก็คือการ “ดึงคำแนะนำที่ดีที่สุด” ออกมาจาก Q-table

  - Q-table คือ ตารางที่บันทึกว่า "ในสถานการณ์นี้ ควรทำอะไรดี" จากประสบการณ์ที่เรียนรู้มา เช่น ชั่วโมงที่ 16 และ แบตมีอยู่ครึ่งหนึ่ง จะชาร์จดีไหม ปล่อยดีไหม หรือไม่ทำอะไรเลย?

- ดังนั้น การ “สร้างนโยบาย (Policy Extraction)" ก็คือ:
  - การตรวจดูใน Q-table ทุกจุด (แต่ละชั่วโมง x ระดับแบตเตอรี่)
  - แล้วเลือกแอคชันที่มีคะแนนดีที่สุดในแต่ละจุด
  - สร้างเป็นนโยบายว่า ณ สถานะนั้น ควรทำอะไร เพื่อให้ได้ผลตอบแทนรวมดีที่สุดในระยะยาว

- สมมุติ Q-table ในสถานะหนึ่งเป็นแบบนี้:

Action	Q-value
Do Nothing	5.2
Charge	2.1
Discharge	8.7

- ระบบจะเลือก Discharge เพราะมี Q-value สูงสุดดังนั้น ณ จุดนั้น นโยบายคือ "ปล่อยประจุ"

ทำไมต้องทำสิ่งนี้?

- เพื่อจะสามารถ นำไปใช้งานจริง ได้ในขั้นตอนสุดท้าย เช่น ในระบบควบคุมแบตเตอรี่ จะได้รู้ว่าควรสั่งให้ชาร์จ ปล่อย หรืออยู่เฉยๆ
- ไม่ต้องเรียนรู้ซ้ำแล้วในช่วงใช้งานจริง — เพียงแค่ใช้ตารางนี้เลือกแอคชันตามสถานะ

- สร้างแมทริกซ์ policy ขนาด (ชั่วโมง x ระดับ SOC) เพื่อเก็บชื่อของแอคชัน (เช่น "Charge", "Discharge") ที่ควรทำในแต่ละสถานะ
- ค่าเริ่มต้นเป็นสตริงว่าง ''

```
# === Extract Policy === #
policy = np.full((HOURS, SOC_LEVELS), '', dtype=object)
```

- action_colors ใช้สำหรับแสดงผลภาพ (เช่น heatmap หรือ plot) โดยแทนแต่ละแอคชันด้วยสีที่ต่างกัน
- ACTION_COLORS เป็นพจนานุกรมที่กำหนดว่าแต่ละแอคชันจะถูกแทนด้วยหมายเลขใด

```
action_colors = np.zeros((HOURS, SOC_LEVELS))
ACTION_COLORS = {'Do Nothing': 0, 'Charge': 1, 'Discharge': 2}
```

- ลูปนี้วนผ่านทุกชั่วโมง (h) และทุกระดับ SOC (s)
- Q[h][s] คือเวกเตอร์ค่าคะแนน Q ของ 3 แอ็คชัน (Do Nothing, Charge, Discharge) ที่สถานะนั้น
- np.argmax() คืนตำแหน่งของแอคชันที่มี Q-value สูงที่สุด = สิ่งที่ควรทำ

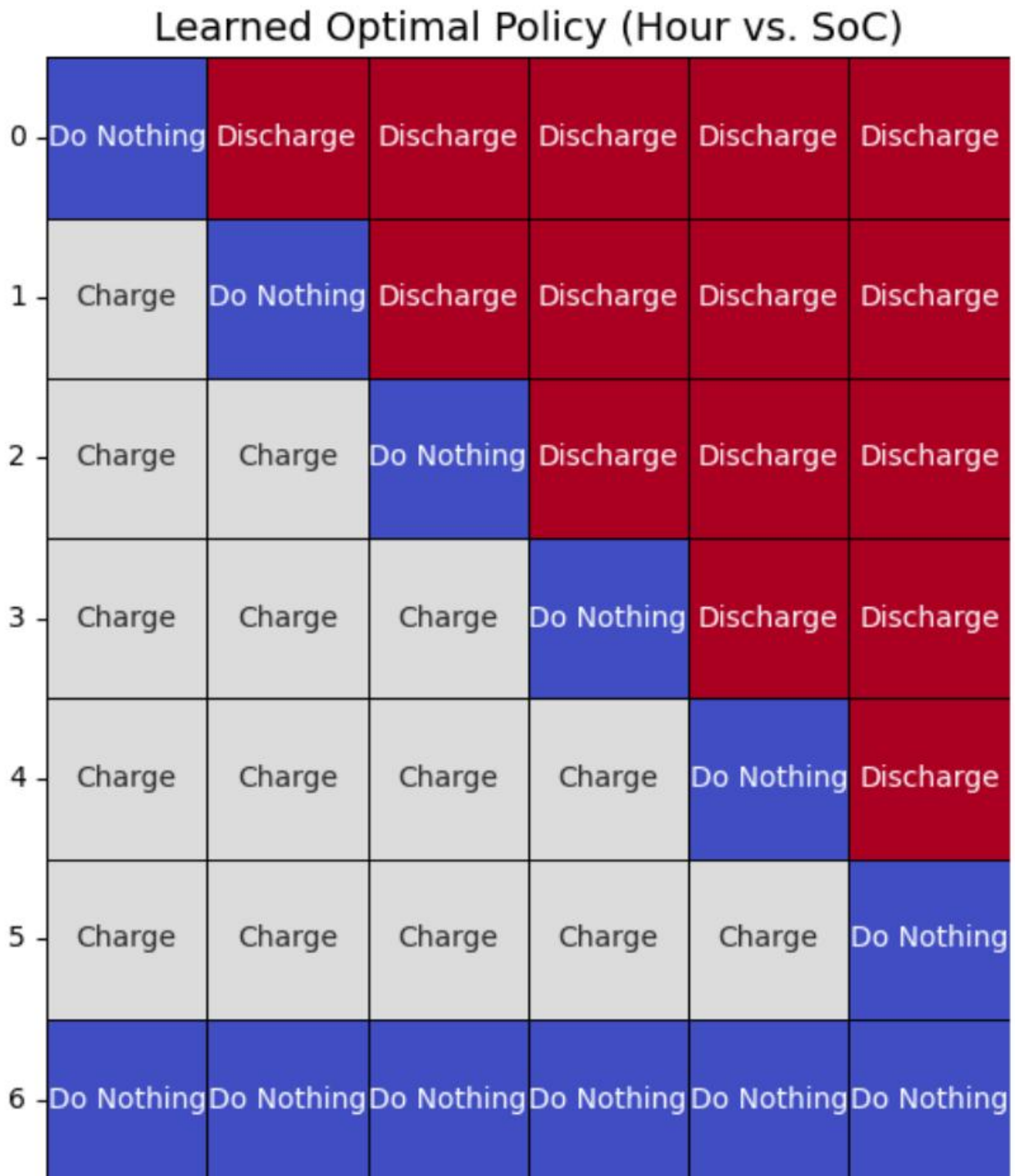
```
for h in range(HOURS):
    for s in range(SOC_LEVELS):
        best_action = np.argmax(Q[h][s])
```

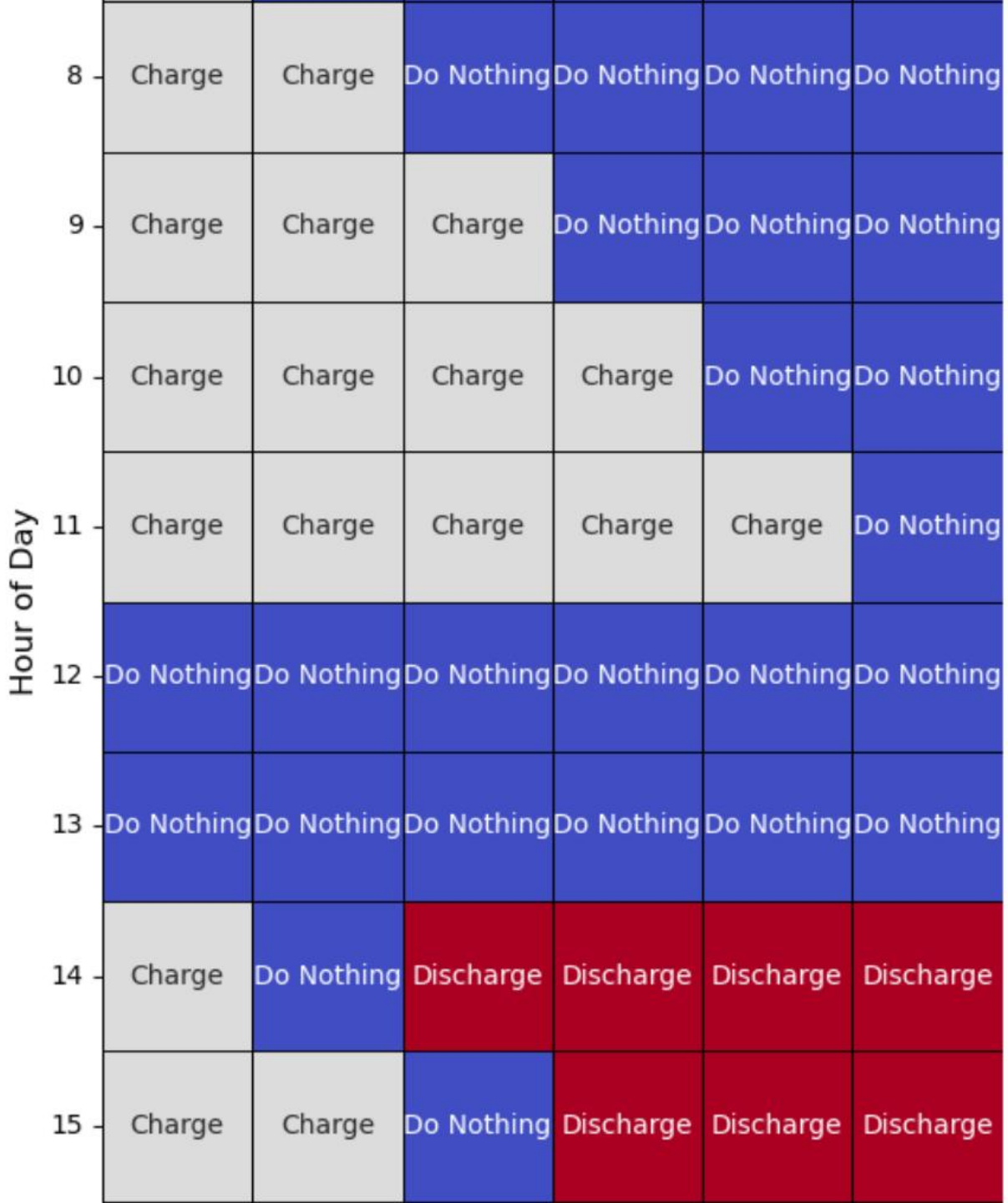
- ดึงชื่อแอคชันออกมาเก็บในแมทริกซ์ policy ตามตำแหน่งนั้น เช่น “Charge”, “Discharge” ฯลฯ

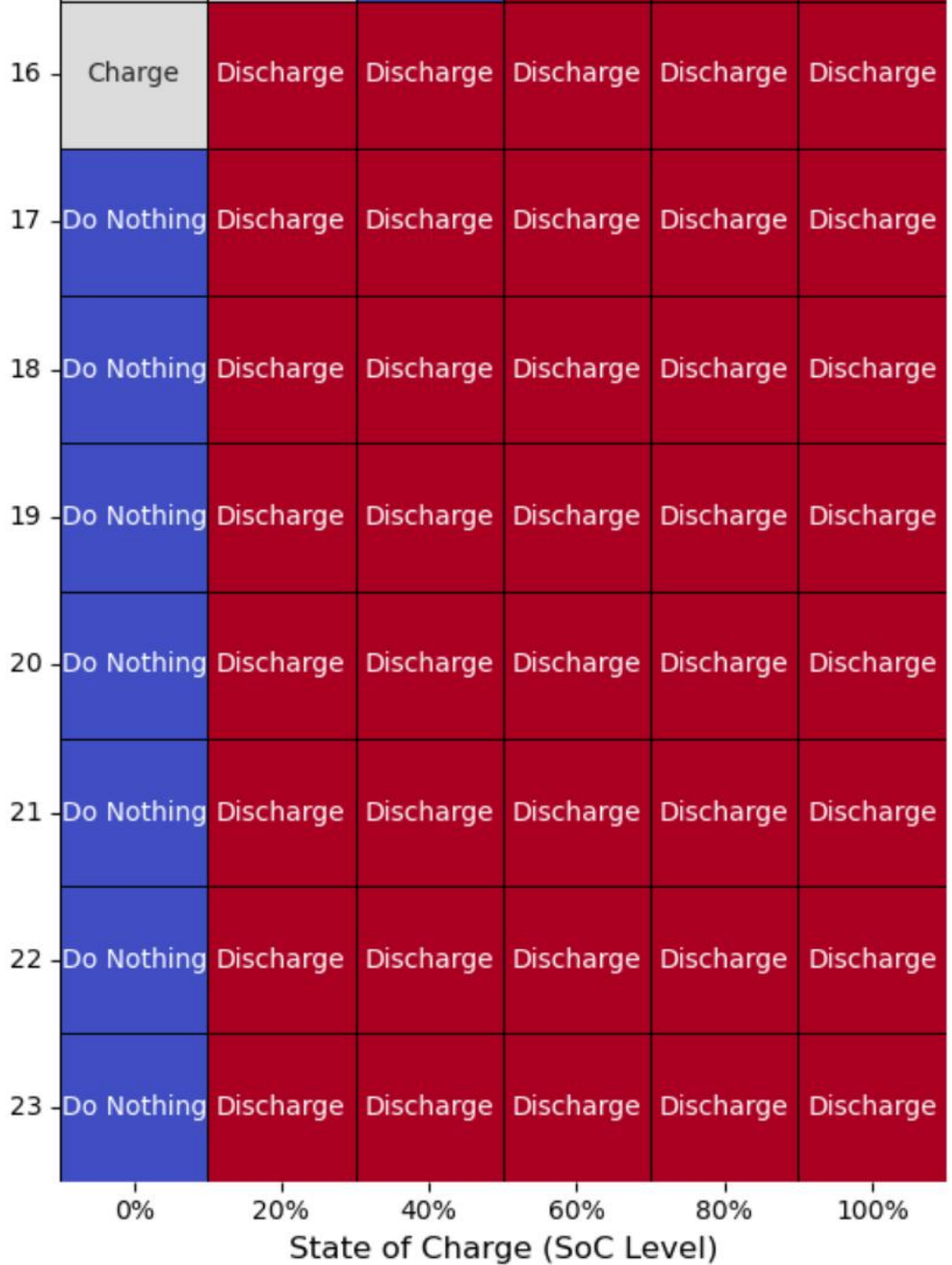
```
policy[h][s] = ACTIONS[best_action]
```

- เก็บค่าตัวเลขที่แทนแอคชันลงใน action_colors เพื่อใช้วาดกราฟต่อไป (แต่ละค่าจะมีสีเฉพาะ)

```
action_colors[h][s] = ACTION_COLORS[ACTIONS[best_action]]
```


Learned Optimal Policy (Hour vs. SoC)
0
1
2
3
4
5
6
Do Nothing
Discharge
Discharge
Discharge
Discharge
Discharge
Charge
Do Nothing
Discharge
Discharge
Discharge
Discharge
Charge
Charge
Do Nothing
Discharge
Discharge
Discharge
Charge
Charge
Charge
Do Nothing
Discharge
Discharge
Charge
Charge
Charge
Charge
Do Nothing
Discharge
Charge
Charge
Charge
Charge
Charge
Do Nothing
Do Nothing
Do Nothing
Do Nothing
Do Nothing
Do Nothing
Do Nothing


Hour of Day
8
Charge Charge Do Nothing Do Nothing Do Nothing Do Nothing
9
Charge Charge Charge Do Nothing Do Nothing Do Nothing
10
Charge Charge Charge Charge Do Nothing Do Nothing
11
Charge Charge Charge Charge Charge Do Nothing
12
Do Nothing Do Nothing Do Nothing Do Nothing Do Nothing Do Nothing
13
Do Nothing Do Nothing Do Nothing Do Nothing Do Nothing Do Nothing
14
Charge Do Nothing Discharge Discharge Discharge Discharge
15
Charge Charge Do Nothing Discharge Discharge Discharge


16 Charge Discharge Discharge Discharge Discharge Discharge
17 Do Nothing Discharge Discharge Discharge Discharge Discharge
18 Do Nothing Discharge Discharge Discharge Discharge Discharge
19 Do Nothing Discharge Discharge Discharge Discharge Discharge
20 Do Nothing Discharge Discharge Discharge Discharge Discharge
21 Do Nothing Discharge Discharge Discharge Discharge Discharge
22 Do Nothing Discharge Discharge Discharge Discharge Discharge
23 Do Nothing Discharge Discharge Discharge Discharge Discharge
0% 20% 40% 60% 80% 100%
State of Charge (SoC Level)

📊 ส่วนที่ 5: การแสดงผล (Visualization)

- กราฟค่าไฟและโหลด
- Heatmap ของนโยบายที่เรียนรู้
- กราฟรางวัลต่อรอบการเรียนรู้
- กราฟแสดงพลังงานจากแบตเตอรี่, โหลด, และโหลดจากกริด

- โค้ดส่วนนี้คือ **การจำลองการทำงานของระบบแบตเตอรี่ตลอด 1 วัน** โดยใช้ **นโยบายที่เรียนรู้มาแล้วจาก Q-learning** เพื่อดูว่าระบบจะตัดสินใจอะไรในแต่ละชั่วโมง และผลที่ได้เป็นอย่างไรบ้าง เช่น ระดับแบตเปลี่ยนยังไง พลังงานที่จ่ายหรือรับจากแบตมีเท่าไร และภาระที่เหลือต้องรับจากกริดมีมากน้อยแค่ไหน

- **เตรียมตัวแปรสำหรับบันทึกผลการจำลอง**

```
sim_soc_levels = []          # เก็บระดับแบตเตอรี่ (State of Charge) ในแต่ละชั่วโมง
sim_battery_power = []       # เก็บพลังงานเข้า/ออกจากแบตเตอรี่ในแต่ละชั่วโมง
sim_grid_power = []          # เก็บพลังงานที่ดึงจากกริดในแต่ละชั่วโมง
```

- ตั้งค่าเริ่มต้น
  - เราจะเริ่มการจำลองในกรณีที่แบตเตอรี่มีพลังงานเต็ม 100%

```
current_soc = SOC_LEVELS - 1  # เริ่มต้นจากแบตเต็ม (สมมุติว่า 100%)
```

- วนลูปทีละชั่วโมง (0–23)
  - จำลองการใช้แบตทีละชั่วโมง ตลอด 24 ชั่วโมง

```
for h in range(HOURS):
```

- เลือกแอคชันจาก Q-table ที่เรียนรู้มา
  - ดู Q-table ว่า ณ ชั่วโมง h และระดับแบต current_soc แอคชันใดดีที่สุด
  - เช่น อาจได้ว่า "ควรปล่อยประจุ"

```
action_index = np.argmax(Q[h, current_soc])
action = ACTIONS[action_index]
```

- โหลดและคำนวณพลังงาน

```
load = load_profile[h]  # ความต้องการโหลดจากอาคาร


# ค่าพลังงานจากแบต และพลังงานที่ยังต้องดึงจากกริด
battery_power_at_hour = 0.0
grid_power_at_hour = 0.0
```

🔋 ถ้าเลือกชาร์จแบต

- แบตต้องไม่เต็ม
- ระดับ SOC เพิ่มขึ้น
- กำลังไฟจากกริดต้องรับโหลดอาคาร + พลังงานที่ใช้ชาร์จแบต

```
if action == 'Charge':
    if current_soc < SOC_LEVELS - 1:
        current_soc += 1
        battery_power_at_hour = -BATTERY_POWER_CHARGE
        grid_power_at_hour = load + BATTERY_POWER_CHARGE
    else:
        grid_power_at_hour = load
```

⚡ ถ้าเลือก ปล่อยประจุ:

- แบตต้องไม่หมด
- SOC ลดลง
- กำลังไฟจากแบตมาช่วยแบกรับโหลด ส่วนที่เหลือดึงจากกริด

```
elif action == 'Discharge':
    if current_soc > 0:
        current_soc -= 1
        battery_power_at_hour = BATTERY_POWER_DISCHARGE
        grid_power_at_hour = max(0, load - BATTERY_POWER_DISCHARGE)
    else:
        grid_power_at_hour = load
```

💤 ถ้าเลือก **ไม่ทำอะไร**:

- โหลดทั้งหมดรับจากกริด

```
else: # Do Nothing
    grid_power_at_hour = load
```

บันทึกผลการจำลอง

- เก็บข้อมูลไว้ใช้วิเคราะห์ หรือสร้างกราฟผลการจำลองต่อไป

```
sim_soc_levels.append(current_soc)
sim_battery_power.append(battery_power_at_hour)
sim_grid_power.append(grid_power_at_hour)
```

🧠 Q-learning Configuration & BESS Control Simulation (สรุปพารามิเตอร์)

- ⚙️ 1. การตั้งค่าระบบแบตเตอรี่ (BESS Settings)

พารามิเตอร์	ค่าที่ตั้ง	ความหมาย
`HOURS`	`24`	จำลองทั้งวันแบบรายชั่วโมง
`SOC_LEVELS`	`6`	ระดับ SoC: 0%, 20%, 40%, 60%, 80%, 100%
`ACTIONS`	`['Do Nothing', 'Charge', 'Discharge']`	ตัวเลือกการกระทำ
`NUM_ACTIONS`	`3`	จำนวนการกระทำทั้งหมด

- 💸 **2. Time-of-Use Tariff & Load Profile**
- 🔌 **Electricity Price ต่อชั่วโมง (หน่วย: THB/kWh)**
- แบ่งเป็น 3 ช่วง:
- **Off-Peak**: 00:00–05:00 & 23:00 → 2.50
- **Mid-Peak**: 06:00–11:00 & 18:00–22:00 → 3.00
- **On-Peak**: 12:00–17:00 → 4.00

- 🔋 **โหลดผู้ใช้ (User Load) ต่อชั่วโมง (หน่วย: kW)**
- โหลดสูงสุดที่ประมาณ 14:00–17:00 (สูงถึง 9.0 kW)

พารามิเตอร์	ค่า	ความหมาย
`HIGH_LOAD_THRESHOLD`	`6.0 kW`	เป้าหมายจำกัดโหลดจากกริดไม่ให้เกิน

- 🔄 **3. กำลังของแบตเตอรี่ในการชาร์จและคายประจุ**

พารามิเตอร์	ค่า	ความหมาย
`BATTERY_POWER_CHARGE`	`5.0 kW`	กำลังที่ดึงจากกริดเมื่อต้องการชาร์จหนึ่งระดับ SoC
`BATTERY_POWER_DISCHARGE`	`5.0 kW`	กำลังที่จ่ายให้โหลดเมื่อคายประจุหนึ่งระดับ SoC

- 🧮 **4. Q-learning Hyperparameters**

พารามิเตอร์	ค่า	ความหมาย
`EPISODES`	`100000`	จำนวนรอบการฝึก Q-learning
`ALPHA`	`0.1`	อัตราการเรียนรู้ (Learning Rate)
`GAMMA`	`0.98`	ค่าการลดความสำคัญของอนาคต (Discount Factor)
`EPSILON_START`	`1.0`	เริ่มต้นการสุ่มการกระทำ (Exploration)
`EPSILON_END`	`0.01`	ค่าต่ำสุดของ Epsilon
`EPSILON_DECAY_RATE`	`0.000045`	อัตราการลดของ Epsilon แบบ exponential decay

- 🧠 5. โครงสร้าง Q-table

- ขนาด Q[hour, soc_level, action] → (24, 6, 3)
- แต่ละ state คือ (ชั่วโมง, ระดับ SoC)
- Q-value คือค่าคาดหวังผลตอบแทนหากเลือก action ที่สถานะนั้น

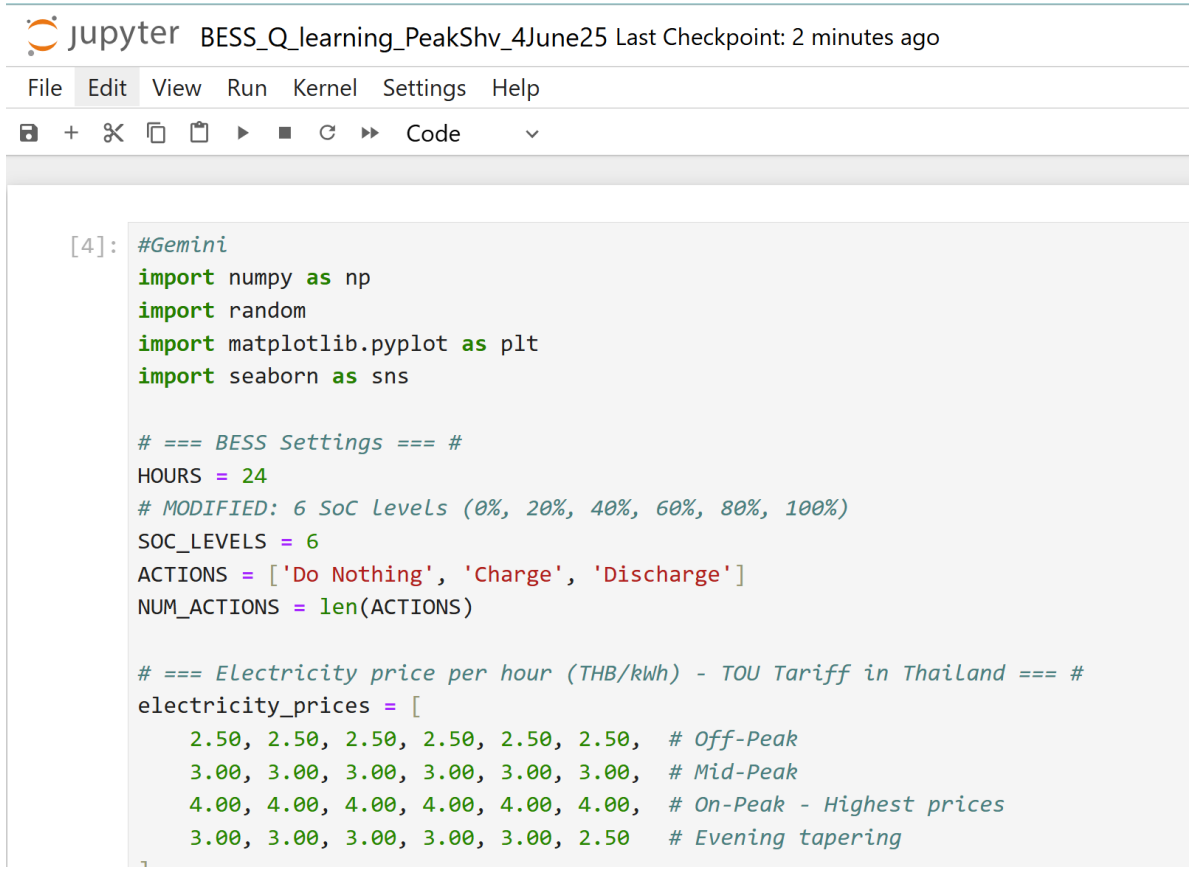
- 📜 **6. การให้รางวัล (Reward)**

การกระทำ	เงื่อนไข	รางวัล
`Do Nothing`	ถ้าโหลด > threshold และมี SoC	-20
	อื่น ๆ	0
`Charge`	ถ้า SoC < max และโหลดหรือราคาสูง	-ราคา -100
	ถ้า SoC เต็ม	-10
`Discharge`	ถ้ามี SoC	+ราคา +15 ถ้าโหลดสูง +2 ถ้าราคา 4.00
	ถ้า SoC = 0	-10

penalty เพิ่ม (ปกติ: -50)

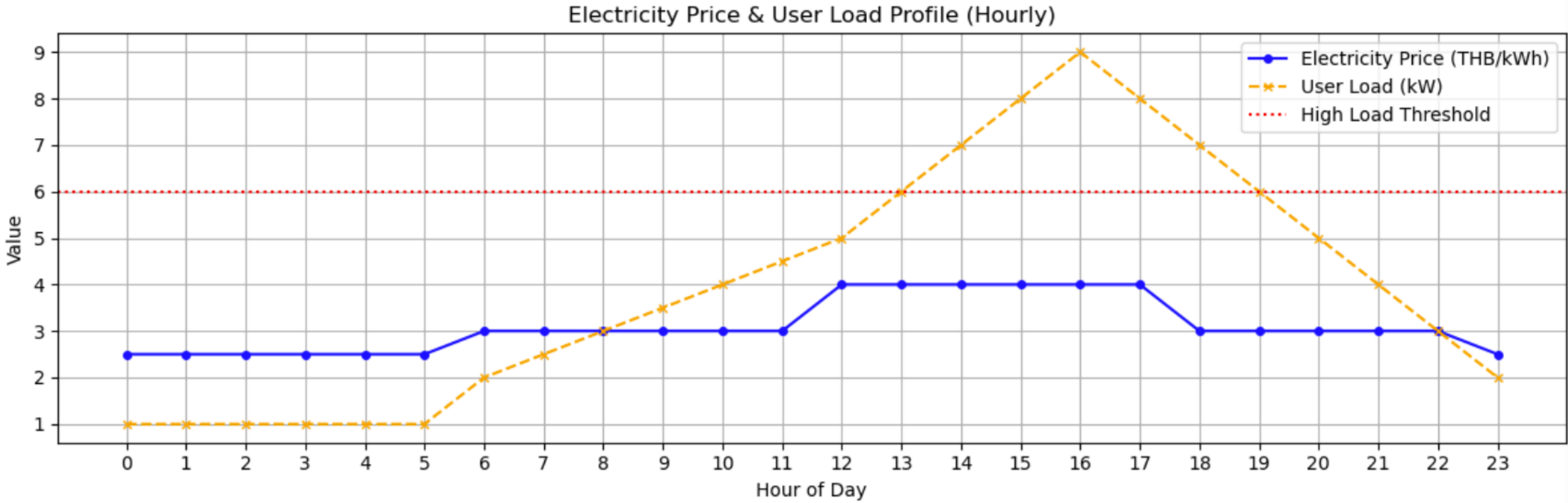
- นอกจากนี้ จะมี penalty เพิ่มถ้า:
- ต้องดึงพลังงานกริด (load + charge หรือ load - discharge) > threshold → มี penalty เพิ่มเติม: ช่วง peak พิเศษ:
  - 16:00 → -50 *5 = -250
  - 17:00 → -50 *2 = -100

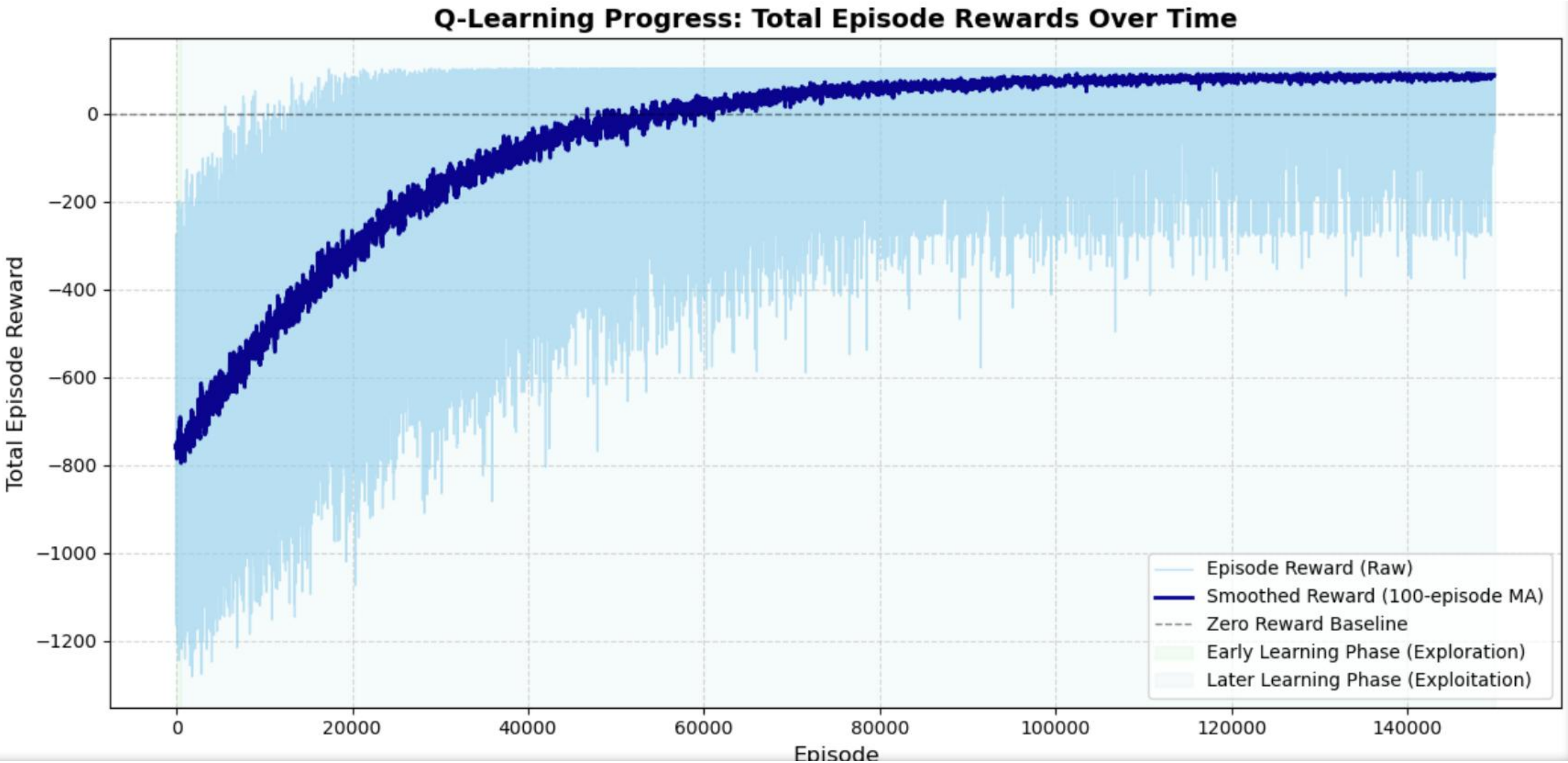
BESS_Q_learning_PeakShv_4June25.ipynb



https://github.com/DrHammerhead/SoC-estimation/blob/486bd3566e998cb2a33e49f06e9f11396f9c0919/BESS_Q_learning_PeakShv_4June25.ipynb

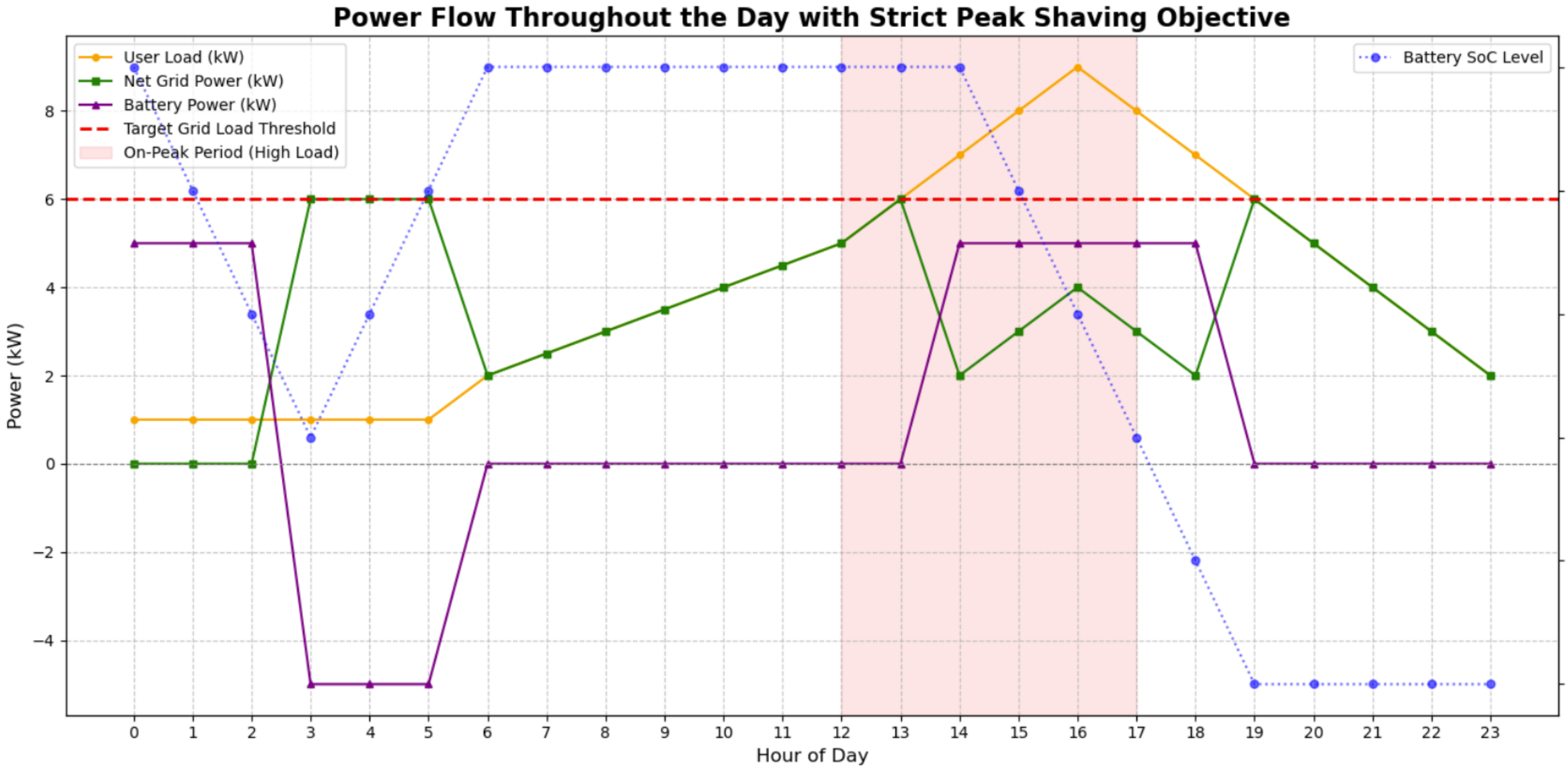
กราฟค่าไฟฟ้า (TOU) โหลดโปรไฟล์ และ ค่าพีคเทรชโฮลด์




Q-Learning Progress: Total Episode Rewards Over Time
Total Episode Reward
0
−200
−400
−600
−800
−1000
−1200
0
20000
40000
60000
80000
100000
120000
140000
Episode
Episode Reward (Raw)
Smoothed Reward (100-episode MA)
Zero Reward Baseline
Early Learning Phase (Exploration)
Later Learning Phase (Exploitation)

=== Optimal Policy (Hour x SoC) - With Enhanced Peak Shaving Focus ===

Hour	0% (Empty)	20%	40%	60%	80%	100% (Full)
0	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
1	Charge	Do Nothing	Discharge	Discharge	Discharge	Discharge
2	Charge	Charge	Do Nothing	Discharge	Discharge	Discharge
3	Charge	Charge	Charge	Do Nothing	Discharge	Discharge
4	Charge	Charge	Charge	Charge	Do Nothing	Discharge
5	Charge	Charge	Charge	Charge	Charge	Do Nothing
6	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing
7	Charge	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing
8	Charge	Charge	Do Nothing	Do Nothing	Do Nothing	Do Nothing
9	Charge	Charge	Charge	Do Nothing	Do Nothing	Do Nothing
10	Charge	Charge	Charge	Charge	Do Nothing	Do Nothing
11	Charge	Charge	Charge	Charge	Charge	Do Nothing
12	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing
13	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing	Do Nothing
14	Charge	Do Nothing	Discharge	Discharge	Discharge	Discharge
15	Charge	Charge	Do Nothing	Discharge	Discharge	Discharge
16	Charge	Discharge	Discharge	Discharge	Discharge	Discharge
17	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
18	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
19	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
20	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
21	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
22	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge
23	Do Nothing	Discharge	Discharge	Discharge	Discharge	Discharge


Power Flow Throughout the Day with Strict Peak Shaving Objective
User Load (kW)
Net Grid Power (kW)
Battery Power (kW)
Target Grid Load Threshold
On-Peak Period (High Load)
Battery SoC Level
Power (kW)
Hour of Day
Battery SoC Level (0=0%, 1=20%, 2=40%, 3=60%, 4=80%, 5=100%)