Challenge #11: GPU Acceleration

1. Prompts Used

1. "Can you identify the computational bottlenecks in the FrozenLake code?"
2. "Propose a HW implementation of the biggest bottleneck."
3. "Generate a SV code for the HW implementation."
4. "Can you optimize the original code for a GPU?"
5. "Compare the pure python version and this version."
6. "If I want to calculate the speed up, what do I do?"

2. Final CPU and GPU Code

```
CPU Version (Baseline)


import gym
import numpy as np
import time


env = gym.make("FrozenLake-v1", is_slippery=False)
n_states = env.observation_space.n
n_actions = env.action_space.n


q_table = np.zeros((n_states, n_actions))


episodes = 1000
```

```
max_steps = 100
alpha = 0.8
gamma = 0.95
epsilon = 1.0
min_epsilon = 0.01
epsilon_decay = 0.995

start = time.time()

for ep in range(episodes):
    state = env.reset()
    done = False

    for _ in range(max_steps):
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(q_table[state])

        new_state, reward, done, _ = env.step(action)

        q_table[state, action] += alpha * (reward + gamma *
np.max(q_table[new_state]) - q_table[state, action])
        state = new_state

        if done:
            break
```

```
    epsilon = max(min_epsilon, epsilon * epsilon_decay)

end = time.time()
print(f"🧠 CPU Q-learning time: {end - start:.4f} seconds")
print("Q-table (rounded):\n", np.round(q_table, 2))
```

```
GPU Version (PyTorch)

import gym
import torch
import time

env = gym.make("FrozenLake-v1", is_slippery=False)
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

n_states = env.observation_space.n
n_actions = env.action_space.n

Q = torch.zeros((n_states, n_actions), dtype=torch.float32,
device=device)

episodes = 1000
max_steps = 100
alpha = 0.8
```

```
gamma = 0.95
epsilon = 1.0
min_epsilon = 0.01
epsilon_decay = 0.995

start = time.time()

for ep in range(episodes):
    state = env.reset()
    done = False

    for _ in range(max_steps):
        if torch.rand(1).item() < epsilon:
            action = torch.randint(0, n_actions,
(1,)).item()
        else:
            action = torch.argmax(Q[state]).item()

        new_state, reward, done, _ = env.step(action)

        Q[state, action] += alpha * (reward + gamma *
torch.max(Q[new_state]) - Q[state, action])
        state = new_state

        if done:
            break
```

```
    epsilon = max(min_epsilon, epsilon * epsilon_decay)

end = time.time()
print(f"⚡ GPU Q-learning time: {end - start:.4f} seconds")
print("Q-table (rounded):\n", torch.round(Q.cpu(),
decimals=2))
```

3. **Documentation**

Objective:

- Accelerate Q-learning using GPU
- Compare it with CPU baseline in terms of runtime and learning performance

Runtime Results

Version	Runtime (sec)	Learned Q-table?
CPU	**0.3050**	Yes
GPU	**15.4548**	No

Speedup Calculation:

$$\text{Speedup} = \frac{T_{CPU}}{T_{GPU}} = \frac{0.3050}{15.4548} \approx 0.0197\times$$

Key Findings

- CPU version trained successfully and quickly.
- GPU version failed to learn despite correct implementation.

- GPU underperformed due to:
    - Small state-action space (16 × 4)
    - No vectorized/batch ops to leverage GPU parallelism
    - High overhead in PyTorch and CUDA memory access for scalar operations

4. Graphs & Proofs

CPU Q-Table Snapshot:

> **Interpretation:** Non-zero Q-values prove successful learning and goal-reaching episodes.

```
🧠 CPU Q-learning time: 0.3050 seconds
Q-table (rounded):
 [[0.74 0.7  0.77 0.74]
 [0.74 0.   0.81 0.77]
 [0.77 0.86 0.77 0.81]
 [0.81 0.   0.77 0.77]
 [0.7  0.66 0.   0.74]
 [0.   0.   0.   0.  ]
 [0.   0.9  0.   0.81]
 [0.   0.   0.   0.  ]
 [0.59 0.   0.4  0.7 ]
 [0.66 0.9  0.87 0.  ]
 [0.86 0.95 0.   0.86]
 [0.   0.   0.   0.  ]
 [0.   0.   0.   0.  ]
 [0.   0.9  0.95 0.82]
 [0.9  0.95 1.   0.9 ]
 [0.   0.   0.   0.  ]]
```

GPU Q-Table Snapshot:

(Q-table contained only zeros — omitted for clarity)

> **Interpretation:** The agent **did not learn** any meaningful policy using GPU training in this configuration.

Final Takeaway

This challenge highlighted that **GPU acceleration is not always beneficial**, especially for small-scale, tabular environments like FrozenLake. While GPUs shine in high-dimensional, batch-optimized computations, traditional CPU approaches remain more efficient for simpler problems.