**Challenge 11 – GPU Acceleration of FrozenLake Q-Learning**

**1. Objective**

The purpose of this challenge was to accelerate a classical Q-learning reinforcement learning algorithm using GPU computation and benchmark its performance against the original CPU implementation. The algorithm was trained in the OpenAI Gym FrozenLake-v1 environment, with the aim of demonstrating computational gains using GPU-based tensor operations.

**2. Tools and Technologies**

* **Environment**: FrozenLake-v1 (is\_slippery=False)
* **Language**: Python 3.10
* **Libraries**: NumPy, PyTorch, Gym
* **Hardware**: NVIDIA GPU (CUDA-enabled)
* **Development**: Jupyter Notebook + Python Scripts

**3. Original Implementation (CPU)**

The baseline version used NumPy for all operations:

* The Q-table was implemented as a 2D NumPy array.
* The training loop followed the classical Q-learning update rule.
* Epsilon-greedy policy was used for exploration.
* Executed entirely on the CPU.

**4. GPU-Accelerated Implementation (LLM-Assisted)**

I used an LLM to assist with converting the CPU-based code to a PyTorch-based implementation optimized for GPU:

* Replaced NumPy arrays with PyTorch tensors.
* Moved tensors to GPU using .to("cuda").
* Rewrote argmax, max, and arithmetic using PyTorch functions.
* Preserved logic for fair comparison.

This GPU-accelerated implementation ensures that all computation-heavy parts (argmax, Q-value updates) are executed on the GPU, reducing CPU-GPU data transfer bottlenecks.

**5. Benchmarking Strategy**

To evaluate performance differences, I developed a benchmarking script (LLM-assisted) that:

* Automatically runs both CPU and GPU versions.
* Times each run using time.time().
* Repeats each version 5 times for consistency.
* Calculates and prints average time and GPU speed-up factor.

**The benchmarking script was executed using:**  !python3 benchmark.py

**6. Benchmark Results**

|  |  |  |
| --- | --- | --- |
| **Version** | **Average Time (s)** | **Speed-up** |
| CPU (NumPy) | 0.0583 | 1.00x |
| GPU (PyTorch) | 0.0543 | **1.07x** |

**7. Analysis and Insights**

* **Minimal gain** is expected due to small state-action space (16×4) and lightweight computation.
* GPU kernel launch overhead becomes noticeable in tiny workloads.
* GPU acceleration is more effective for larger environments or when deep neural networks are involved (e.g., DQNs).
* Despite the small gain, this challenge validates a **scalable workflow** for future GPU-backed reinforcement learning.

**8. Files and Artifacts**

|  |  |
| --- | --- |
| **File** | **Description** |
| frozenlake\_cpu.py | Original CPU Q-learning code |
| frozenlake\_gpu\_opt.py | GPU-optimized PyTorch implementation |
| benchmark.py | Automated benchmark runner |
| benchmark\_output.png | Jupyter output screenshot (proof of execution) |
| benchmark.ipynb | Notebook for running all tests |

**9. Reflections**

This challenge helped me:

* Translate CPU logic into GPU tensor-based computation
* Use an LLM to streamline PyTorch conversion
* Design and implement reproducible benchmarking
* Evaluate speed-up realistically for a constrained problem

I now feel more confident in identifying when GPU acceleration is worthwhile and how to measure performance impact systematically.

A screenshot of a graph

AI-generated content may be incorrect.

A white rectangular object with blue lines

AI-generated content may be incorrect.