For Challenge #11, the objective was to explore GPU acceleration by optimizing a Q-learning-based FrozenLake environment. The initial implementation was a pure Python version based on a GitHub repository by Ronan Murphy, which used tabular Q-learning on a small 5x5 grid. The goal was to translate this logic into a GPU-accelerated version using CUDA, benchmark both versions, and evaluate the speed-up.

I started by reviewing the Python version, which is inherently sequential due to its per-episode control flow and dictionary-based Q-table updates. While functional, the Python version suffers from significant interpreter overhead during large numbers of episodes. To benchmark its performance, I modified the script to measure execution time across varying episode counts (1000, 5000, 10000) and output the results to a file (cpu\_time\_log.txt) for later comparison.

To create the GPU version, I turned to ChatGPT for guidance. I asked for a CUDA-based rewrite of the Q-learning logic that could be compiled using nvcc in Google Colab. Since Q-learning is not naturally parallel due to its sequential decision-making and temporal dependencies, the GPU version was implemented with a single thread executing the learning loop. Despite this, it offered an opportunity to test how efficiently a GPU handles tight computational kernels with minimal memory and control overhead. With ChatGPT’s help, I structured the CUDA kernel to perform tabular Q-value updates and measured kernel execution time using cudaEvent timers. The GPU version logged its timings to gpu\_time\_log.txt.

One obstacle I faced was ensuring valid comparisons between the CPU and GPU versions. The GPU version was initially reporting unrealistically small execution times (in the microseconds), prompting a review of what was being measured. ChatGPT clarified that the GPU benchmark was only timing kernel execution, excluding memory allocation, initialization, and transfer overhead, which led to misleading performance claims. With that understanding, I took steps to capture full workloads and interpret results more critically.

Finally, I used Python and matplotlib to read both benchmark logs and plot the CPU vs GPU execution times on a logarithmic scale. The graph clearly showed the GPU version outperforming the Python version, but the context provided by ChatGPT helped interpret this result carefully. The speed-up was significant numerically, but largely due to Python’s inherent slowness and the minimal kernel size, not due to massive GPU parallelism.

Overall, ChatGPT was instrumental in helping me port the Q-learning logic to CUDA, benchmark both versions accurately, and understand the architectural reasons behind the performance difference. This challenge taught me that GPU acceleration isn’t always straightforward—true gains require restructuring algorithms to exploit massive parallelism—and that benchmarking must be done thoughtfully to yield meaningful insights.