Carbon-Aware AI/ML Workload Scheduler

This project addresses the growing environmental impact of artificial intelligence and machine learning by implementing a carbon-aware workload scheduler. As the energy demands of training large-scale AI models continue to rise, so do their carbon emissions, especially when powered by fossil-fuel-heavy electricity grids. To mitigate this, the scheduler monitors real-time carbon intensity data and delays training jobs until the grid is cleaner, helping reduce emissions without compromising model performance.

The project was implemented entirely in Python 3.11, using a combination of open-source libraries and tools. The core scheduling logic uses the requests library to query the Electricity Maps API, which provides real-time carbon intensity data by region. Due to limited access to U.S. regional data under the free tier, the scheduler was configured to pull data from China (zone code CN), where consistent results were available. AI job simulation was done using torch and torchvision to create a simple PyTorch-based model training routine. Carbon intensity data is logged continuously to a CSV file, and when the threshold drops below a specified level (e.g., 300 gCO_2eq/kWh), the scheduler triggers the AI job and calculates estimated energy usage and emissions based on job duration and assumed power draw. Visualization is handled by matplotlib, and pandas was used to assist with parsing and plotting the log data.

This project was containerized using Docker. The Dockerfile installs all necessary Python packages and copies the project files into a container. The container runs the scheduler script automatically and generates a graph image (carbon_plot.png) upon exiting, which can be viewed on the host system. A shared volume (-v $(pwd):/app) ensures that logs and graph files created in the container are saved directly to the user's machine.

I faced some challenges while working on this project. Initially, the Electricity Maps API returned no data for U.S. zones, which was resolved by switching China as it was a reliable region with public access. SSL certificate errors on macOS prevented the MNIST dataset from downloading, which were fixed by running Apple's built-in certificate installer script. Visual Studio Code's Python interpreter failed to recognize certain libraries, causing yellow import warnings, but that was resolved by selecting the correct interpreter linked to the project environment. Another issue occurred when the scheduler exited too quickly inside Docker, as the carbon level remained too high to trigger the job. This was addressed by implementing a carbon-check loop with visible logging and time delays. Finally, since Docker doesn't support GUI windows by default, plotting with matplotlib had to be modified to save output as an image file instead of displaying it directly.

In the end, this project produced a robust, containerized system that dynamically responds to environmental conditions, supports real-time scheduling of AI workloads, logs and visualizes carbon data, and estimates emissions per job. It showcases how AI systems can be made more environmentally responsible through thoughtful integration of real-world data and intelligent scheduling.

** To Run on Docker:

Build image
docker build -t carbon-scheduler .

Run the docker with container
docker run -it -v $(pwd):/app carbon-scheduler

Wait for the Scheduler or Press Ctrl+C

Open the plot
open carbon_plot.png