Carbon-Aware Scheduling

Hacking Sustainability - Project 2

Arda Duyum (6491073) Yuvraj Singh Pathania (6495044)
Brewen Couaran (6533973) Taeyong Kwon (6283276)
Elia Jabbour (6589308)

March 2026

Abstract

1 Introduction

1.1 Context

Artificial Intelligence workloads are becoming significantly more compute-intensive, spanning large model training, repeated fine-tuning, and high-volume inference. As the scale of these workloads increases, so does the electricity required to execute them, making AI systems an increasingly important contributor to operational carbon emissions [2, 3]. Importantly, the carbon cost of the same computation is not constant over time. It depends on the electricity mix of the grid at the moment the workload runs, which can vary substantially throughout the day depending on renewable energy availability, industrial demand, and weather conditions [4]. This creates an opportunity for more sustainable software systems: if flexible workloads can be shifted to cleaner time windows, substantial emissions can be avoided without changing the workload itself.

1.2 Problem Statement

Despite this opportunity, most existing job schedulers are designed to optimize for throughput, latency, or infrastructure cost rather than carbon intensity. In practice, delay-tolerant AI jobs such as batch training, offline evaluation, and scheduled inference are often launched immediately, even when the grid is temporarily carbon-intensive. Traditional schedulers also provide little visibility into the relationship between execution timing and electricity cleanliness, making it difficult for users to make informed trade-offs between urgency and sustainability. As a result, there is a need for a scheduling system that incorporates carbon-intensity forecasts directly into execution decisions while still respecting user-defined deadlines and reliability requirements.

1.3 Proposed Solution

To address this problem, we developed Green AI Scheduler, a web-based carbon-aware scheduling platform for delay-tolerant AI workloads. The system combines a React dash-

board for visualization and job submission with a FastAPI backend that manages queued jobs, exposes carbon forecast data, and dispatches commands when grid conditions are favorable. Instead of executing every job immediately, users specify a deadline and a carbon threshold, allowing the scheduler to defer execution into cleaner energy windows when possible. To support this decision-making process, the platform also provides forecast summaries, timeline views, and optimal scheduling windows derived from a multi-year ensemble carbon oracle for the Netherlands. In this way, our solution turns carbon awareness from an abstract sustainability goal into an actionable scheduling mechanism.

2 Background & Related Work

2.1 Temporal Workload Shifting

Temporal workload shifting is the practice of delaying flexible computation so that it runs when the electricity grid is cleaner. Because carbon intensity changes over time with renewable generation, weather conditions, and demand patterns, the same AI job can produce very different emissions depending on when it is executed [4]. This makes time a useful optimization variable for delay-tolerant workloads such as model training, batch inference, and offline evaluation. Rather than reducing the amount of computation itself, temporal shifting reduces the carbon cost of that computation by aligning it with lower-carbon time windows. In carbon-aware systems, this idea is typically implemented by combining carbon forecasts with user-defined deadlines, allowing workloads to be postponed when flexibility exists and executed immediately when urgency is high.

2.2 Existing Solutions

Several existing tools partially support sustainable computing, but few provide direct carbon-aware scheduling for AI jobs. On one side, initiatives such as the Green Software Foundation and its Software Carbon Intensity (SCI) framework help developers measure and reason about software emissions at a system level [5]. On the other side, widely used schedulers such as Apache Airflow, Jenkins, and Slurm are effective at orchestrating pipelines, batch jobs, and cluster workloads, but they are primarily designed around reliability, dependency management, throughput, and resource allocation rather than real-time grid carbon intensity. As a result, carbon awareness is usually treated as an external concern instead of a built-in scheduling objective, leaving a gap for systems that can directly shift flexible workloads into cleaner execution windows.

3 Methodology & Design

3.1 System Architecture

The Green AI Scheduler follows a lightweight client-server architecture composed of three main layers: a web frontend, an application backend, and a carbon forecast data layer. The **frontend** is implemented in React and provides the user-facing dashboard for viewing carbon conditions and submitting jobs. Through this interface, a user can inspect current and forecasted grid conditions, identify cleaner execution windows, and submit a shell command together with a deadline and a maximum acceptable carbon-intensity

threshold. This makes the platform usable both as a monitoring tool and as an execution control surface for flexible AI workloads.

The **backend** is implemented in FastAPI and acts as the central orchestration layer. It exposes REST endpoints for job creation, job listing, cancellation, and carbon-related queries such as current intensity, forecasts, timeline views, and optimal execution windows. Submitted jobs are stored in an asynchronous in-memory job store, where each job maintains metadata including its command, deadline, execution status, and the carbon intensity observed at runtime. At application startup, the backend launches a background scheduler loop that periodically checks the current projected carbon intensity and compares it against all pending jobs. If a job's threshold condition is satisfied before its deadline, the backend dispatches the job as a local subprocess and records the resulting output and execution metadata.

Rather than depending directly on a live external API during runtime, the current implementation uses a local carbon oracle as its data layer. This oracle is loaded from a CSV file generated from five years of historical ElectricityMaps data and adjusted to produce a 2026 projection for the Netherlands. We chose this approach because continuous access to the live ElectricityMaps service would introduce cost, and for the scope of this project a local oracle was sufficient to validate the scheduling logic, forecast visualizations, and system behavior under realistic grid conditions. Architecturally, this design separates user interaction, scheduling logic, and carbon data access into clear components, making the system easy to test, deploy, and extend toward future integrations such as persistent queues, multi-node execution, or live grid APIs.

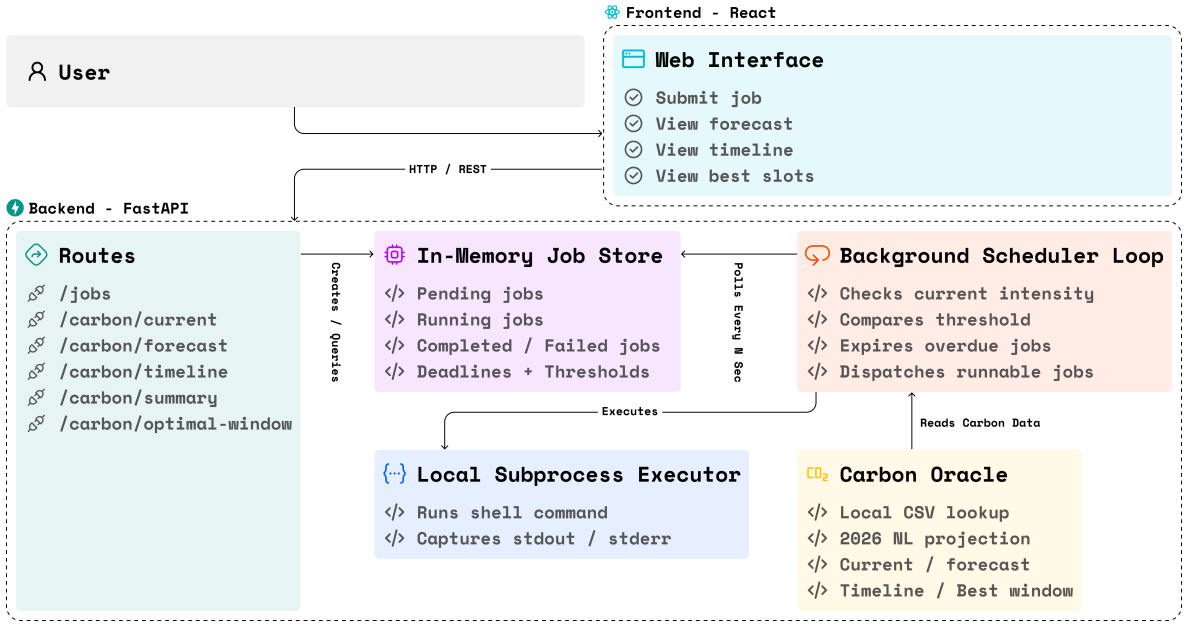

Figure 1: System Architecture of Green AI Scheduler.

3.2 Data Sources & Forecasting

We developed a Multi-Year Ensemble Oracle that acts as a Digital Twin of the Netherlands grid to ensure that our scheduler operates on realistic data for the year 2026. Relying on a single year of historical data risks "overfitting" the scheduler to specific

weather anomalies (e.g., an unusually windy month). Following the Typical Meteorological Year (TMY) methodology used in energy modeling [4], we extracted over 43,800 hourly data points spanning five years (2021-2025) of historical carbon intensity data from ElectricityMaps [1].

Our analysis identified a consistent grid decarbonization trend, with carbon intensity dropping by an average of 8.08% annually. To project a realistic "live" grid for 2026, our Oracle applies this 8.08% annual reduction to the five-year ensemble average. By grouping data by month, day, and hour, the Oracle preserves recurring weekly industrial demand cycles. This ensures that a projected Friday in 2026 accurately reflects the typical load patterns of preceding Fridays. Furthermore, to account for the inherent volatility of renewable energy [3], the Oracle maintains historical minimum and maximum bounds for every hour. This allows our system to provide a "Confidence Interval" in its forecast that visualizes weather-driven uncertainty for the user. The need for this Ensemble Oracle stems from the root cause of grid carbon volatility: the intermittent nature of renewable energy sources like wind and solar. By modeling this uncertainty through a "Digital Twin,", our solution transforms unpredictable weather patterns into a planning tool that allows AI researchers to visualize exactly when "green windows" occur and why delaying a job is positive for the environment.

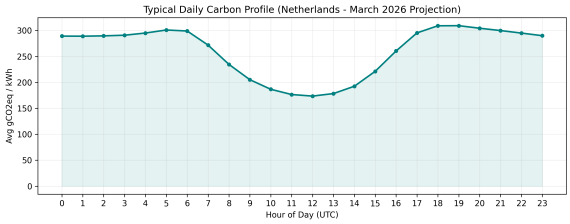

Figure 2: Projected Average Daily Carbon Profile (NL).

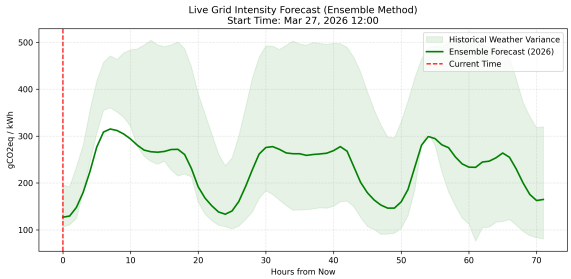

Figure 3: 72-Hour Ensemble Forecast showing variance.

3.3 Scheduling Logic

The current backend implementation of the Green AI Scheduler utilizes a "Reactive Threshold" logic. Users define a maximum acceptable carbon intensity (e.g., 150 gCO_2eq/kWh) alongside their deadline. The backend continuously polls the oracle and dispatches the job as soon as the current intensity drops below this threshold. If the deadline passes before the threshold is met, the job is marked as expired.

While this logic is computationally lightweight to implement, we hypothesized that it might lead to reliability issues. To evaluate this, our experimental validation (Section 5.1) explicitly benchmarks this implemented Reactive Threshold approach against a theoretical "Lowest Window" forecast logic to determine the optimal scheduling architecture for future iterations.

4 Implementation

4.1 Backend (FastAPI)

The backend is implemented as a FastAPI application that combines two responsibilities: exposing the platform's REST interface and running the scheduling process in the background. At startup, the application loads the local carbon oracle into memory and launches an asynchronous scheduler loop through FastAPI's lifespan hook. Cross-Origin Resource Sharing (CORS) is enabled so that the React frontend can communicate with the API during local development and containerized deployment.

The REST layer is split into two route groups. The `/jobs` endpoints allow clients to create jobs, list all submitted jobs, fetch an individual job by identifier, and cancel pending jobs. Each submitted job is represented as a structured object containing a unique identifier, the shell command to execute, a deadline, a user-defined carbon threshold, and runtime metadata such as start time, finish time, execution status, and captured standard output or error streams. The `/carbon` endpoints expose the projected current carbon intensity together with richer forecast products, including hourly forecasts, past-and-future timelines, daily summaries, and the optimal low-carbon execution window for a user-specified duration.

Job state is managed by a lightweight asynchronous in-memory store protected by an `asyncio.Lock`. This design is sufficient for the current project scope because it provides safe concurrent access without requiring an external database or message broker. The scheduler loop periodically queries the current projected intensity from the local oracle, retrieves all pending jobs, and evaluates them against two conditions: whether the deadline has already passed and whether the current intensity is below the job's carbon threshold. Jobs that have exceeded their deadline are marked as expired, while eligible jobs are dispatched for execution.

Execution is handled through Python's asynchronous subprocess interface. When a job is launched, the backend records the start time and the carbon intensity at execution, then runs the submitted shell command locally. After completion, the system captures `stdout` and `stderr`, stores the return outcome, and updates the job status to either completed or failed. This backend design keeps the implementation compact while still demonstrating the full scheduling pipeline: API ingestion, queue management, carbon-aware decision making, and local command execution.

4.2 Frontend (Vite + React)

The frontend is a single page dashboard built with React and TypeScript. All visualizations are implemented as custom React components using CSS Grid and utility classes. It separates carbon related and job related concerns into distinct modules, each containing its own API functions, types, components, and utility helpers.

The dashboard is organized as a two column layout. The sidebar displays the current carbon intensity with a color coded severity badge, a job submission form, a list of submitted jobs, and scheduling recommendations, including the best four hour execution window and the three lowest intensity hours in the next 72 hours. The main content area provides two switchable forecast views: a week calendar grid rendering a 7-day by 24-hour color coded matrix in which each cell reflects one of four intensity bands (green below 120, yellow below 200, orange below 300, and red above 300 gCO_2/kWh), and a 72-hour timeline presenting the same data as a scrollable chronological list.

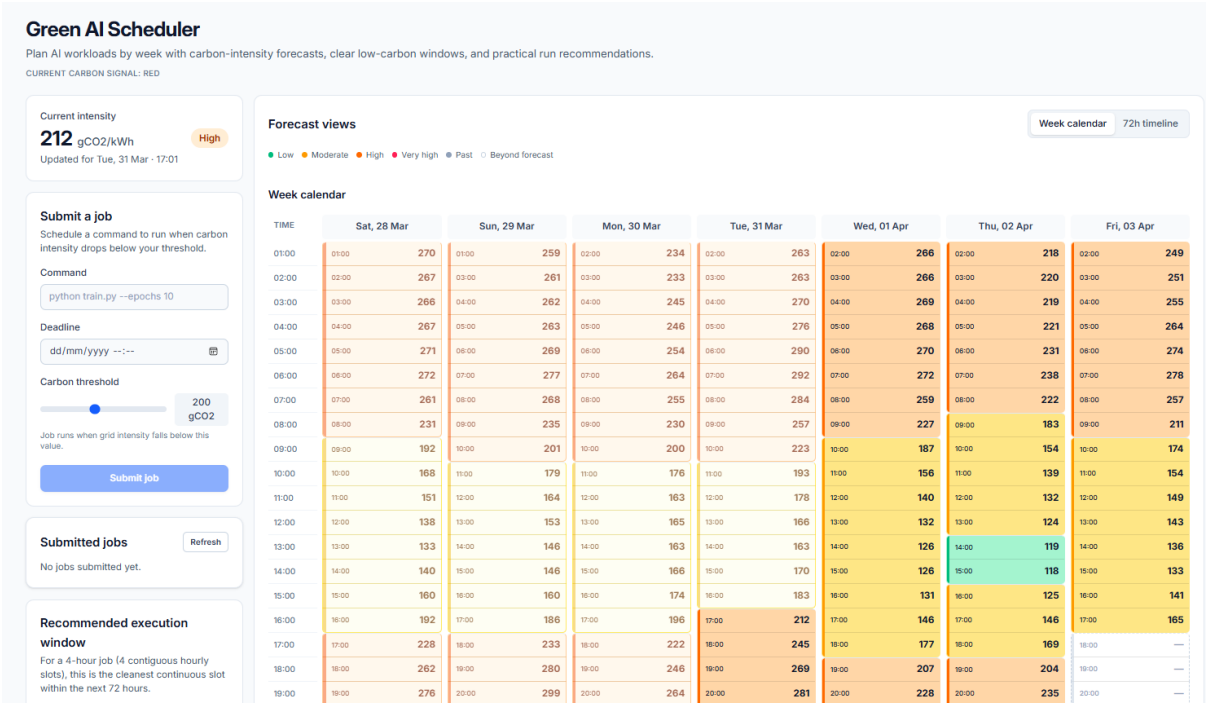

Figure 4: The Green AI Scheduler dashboard showing the sidebar with the current intensity, job submission form, and job queue alongside the week calendar grid with color coded carbon intensity forecasts.

To submit a job, users enter a shell command, select a deadline that is at least one hour ahead, and set a carbon threshold via a slider (50 to 400 gCO_2/kWh). The jobs panel lists all jobs with color coded status badges (pending, running, completed, failed, expired, or cancelled) and allows cancellation of pending jobs. Selecting a job reveals its execution timestamps and the carbon intensity at the moment of dispatch, allowing users to confirm that deferred jobs were executed during a cleaner grid window compared with the conditions visible at submission time.

The frontend polls the backend for updated carbon data every 60 seconds and refreshes the job list every 15 seconds. All API calls on initial load run in parallel via Promise.all, and a dedicated mapper layer decouples component logic from backend response formats.

4.3 Deployment

To make it as easy as possible for anyone to try out the Green AI Scheduler, we containerized the entire platform using Docker. Instead of worrying about installing the right versions of Python, Node.js, or other system dependencies, users can spin up the full stack, including both the React frontend and the FastAPI backend, with a single command.

We included two simple configurations out of the box. For tinkering and local development, running `docker-compose -f docker-compose.dev.yml up` launches the environment with hot reloading enabled, so any code changes are reflected immediately. For production deployment, running `docker-compose.prod.yml` creates a streamlined, production-ready build. This plug-and-play setup ensures that anyone, from a curious researcher to an MLOps engineer, can clone the repository and get the carbon-aware dashboard running in their own lab environment with almost zero friction.

5 Validation & Evaluation

5.1 Technical Validation (Simulation)

We validated the Green AI Scheduler using a trace-driven simulation, which is the industry standard for evaluating compute schedulers [2]. We simulated 500 synthetic AI batch jobs across a representative 31-day period (March 2026). Each job assumed a multi-GPU workload with a power draw of 1.5 kW [3] and a random duration between 2 and 8 hours.

We compared a baseline "Immediate Execution" strategy against our scheduler across varying levels of user flexibility. As shown in Figure 5, a 24-hour deadline resulted in a 28.1% reduction in total carbon emissions. Significantly, providing the scheduler with a 24-hour flexibility window does not necessitate a full 24-hour delay. Our simulations demonstrated an average actual delay of only 9.6 hours to achieve these savings, suggesting that meaningful carbon reductions can be reached without exhausting the maximum allowed deadline.

To critically evaluate our current backend architecture, we benchmarked the implemented Reactive Threshold algorithm against a theoretical "Lowest Window" forecasting algorithm (Figure 6). We found that a strict 150 gCO_2/kWh threshold in the Netherlands results in an 86.8% failure rate, as the grid rarely stays below that level for long continuous periods in March. In contrast, the Lowest Window approach achieved comparable savings with perfect reliability (0.0% failure). This simulation proves that while reactive thresholds are easy to implement, forecast-based scheduling is essential for production-grade green scheduling.

The actual impact of this solution is profound: by accepting an average delay of less than 10 hours, an organization can reduce its operational AI carbon footprint by nearly 30%. This demonstrates that significant decarbonization is achievable today without requiring new hardware or carbon offsets, simply by changing the temporal behavior of software systems. Our results prove that user flexibility is the most powerful lever in enabling sustainable AI.

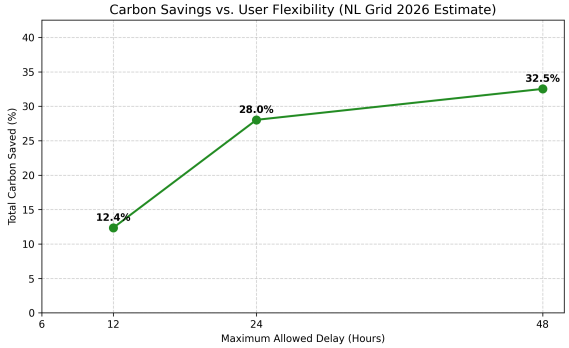

Figure 5: Carbon Saved vs. User Flexibility.

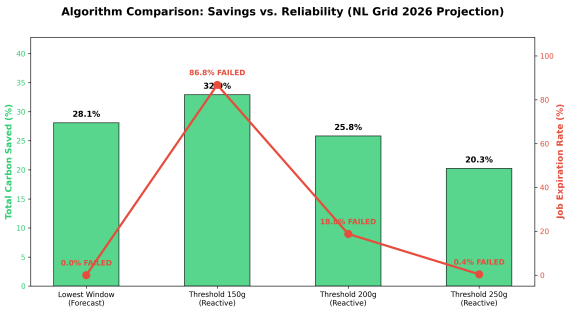

Figure 6: Algorithm Savings vs. Expiration Rate.

5.2 Usability Evaluation

To evaluate the usability, practical impact, and limitations of our platform, we developed narratives for two contrasting personas with different levels of expertise and operational priorities.

Persona 1: The Flexible Researcher (High Compute, High Flexibility)
Profile: Alice is a Senior Data Scientist. She frequently fine-tunes Large Language Models (LLMs), a process characteristic of compute-intensive "Red AI" [6]. Her priority is model accuracy, but she does not need immediate results.
Narrative: On a Friday afternoon, Alice queues five hyperparameter tuning jobs that she needs for a Monday morning review. Using the Green AI Scheduler's visual dashboard, she observes via the 72-hour ensemble forecast (Figure 3) that carbon intensity will drop during the weekend's midday solar peaks. She sets a 48-hour deadline.
Benefit: The algorithm automatically defers her jobs to the greenest contiguous blocks over the weekend. As proven by our simulations (Figure 5), this simple action reduces her models' carbon footprint by 32.5%. Because the forecast guarantees a 0.0% failure rate, Alice returns on Monday to find all jobs completed. The tool allows her to maximize sustainability with zero impact on her productivity.

Persona 2: The Time-Constrained MLOps Engineer (High Urgency, Automated)
Profile: Bob is an MLOps Engineer responsible for continuous integration (CI/CD) pipelines and Energy Regression Testing. His priority is speed; code commits must be tested rapidly to avoid blocking the development team. He has lower flexibility.
Narrative: On a Tuesday morning, Bob pushes a new model architecture. The pipeline automatically triggers a training validation job via the Green AI Scheduler's API. Because the team needs feedback today, the automated script sets a strict 6-hour deadline.
Benefit & Limitation: The scheduler scans the 6-hour window. If a minor dip in carbon intensity is forecast, it defers the job slightly, securing a modest 5% to 10% emission reduction. However, if the entire 6-hour window is highly carbon-intensive, the tool is constrained by the deadline and executes the job anyway to prevent a workflow bottleneck.

Synthesis and Limitations
Through these narratives, we identify that the Green AI Scheduler is highly effective for delay-tolerant batch processing (Alice) but faces limitations with time-critical or synchronous workloads (Bob). For users like Bob, the tool functions more as an *awareness and reporting* mechanism rather than a shifting mechanism. This highlights that while temporal workload shifting is a powerful solution for sustainable AI, its efficacy scales directly with the user's operational flexibility.

6 Dissemination & Social Impact

6.1 Open Source Availability

The Green AI Scheduler is available as an open-source project on GitHub at `https://github.com/SSE-25-26-Group15/green-ai-scheduler`. Publishing the full codebase allows others to inspect the scheduler logic, reproduce the dashboard and backend setup, and extend the platform for their own research or operational environments. Open availability also supports transparency, reuse, and future collaboration around carbon-aware software systems.

6.2 Community Adoption

Because the platform is distributed as a lightweight full-stack application with Docker-based deployment, developers can adopt it in local labs, research groups, or small server environments with minimal setup effort. The system can be used directly as a standalone dashboard for carbon-aware job submission, or extended and integrated into existing AI workflows, internal tools, and experimentation pipelines. In this way, the project lowers the barrier for practitioners who want to bring carbon awareness into everyday compute operations.

7 Discussion & Limitations

7.1 Technical Limitations

A primary limitation of our current validation is the use of a locally hosted Ensemble Oracle in place of a live production API. While the Oracle provides a scientifically valid "Digital Twin" of the 2026 grid based on five-year trends, it cannot account for real-time anomalous weather events (e.g., unforeseen grid failures or extreme storm surges) that may occur on the day of execution. In addition, our current model optimizes strictly for operational carbon emissions. It does not account for the embodied carbon of the hardware or the significant water usage associated with evaporative cooling in data centers, which are critical environmental factors in the broader Green AI landscape [8]. As discovered during our technical validation (Section 5.1), the backend's current reliance on a Reactive Threshold algorithm also creates a severe limitation regarding job reliability. Users who set ambitious carbon thresholds risk a high expiration rate during prolonged periods of fossil-fuel reliance on the grid.

7.2 Future Work

Based directly on our simulation results, the immediate next step for the Green AI Scheduler is to upgrade the backend scheduling loop from a Reactive Threshold system to the "Lowest Window" forecasting algorithm. This will eliminate job expiration while preserving carbon savings. Furthermore, we aim to integrate Spatial Flexibility [4] by routing jobs not just to a different time, but also to a different geographic data center with lower current emissions.

8 Conclusion

This project showed that carbon-aware scheduling can make AI workloads more sustainable without changing the workloads themselves. By combining a usable dashboard, a FastAPI-based scheduler, and a local carbon oracle for the Netherlands, we built a system that helps users delay flexible jobs into cleaner grid windows while still respecting deadlines. Our simulations showed that even modest temporal flexibility can lead to meaningful carbon reductions, demonstrating that execution timing is an effective lever for greener software systems. More broadly, the Green AI Scheduler illustrates how sustainability goals can be translated into concrete software design choices, making carbon awareness a practical part of everyday computing workflows.

References

[1] ElectricityMaps. (2026). *Real-time carbon intensity of the electricity grid*. `https://app.electricitymaps.com`

[2] Patterson, D., et al. (2021). *Carbon Emissions and Large Neural Network Training*. arXiv:2104.10350.

[3] Dodge, J., et al. (2022). *Measuring the Carbon Intensity of AI in Cloud Instances*. In 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT).

[4] Radovanovic, A., et al. (2022). *Carbon-aware computing for datacenters*. IEEE Transactions on Power Systems.

[5] Green Software Foundation. (2026). *Software Carbon Intensity (SCI) Standard*. `https://greensoftware.foundation`

[6] Schwartz, R., et al. (2020). *Green AI*. Communications of the ACM, 63(12), 54-63.

[7] Becker, C., et al. (2016). *Requirements: The Key to Sustainability*. IEEE Software, 33(1), 56-65.

[8] Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., et al. (2022). *Sustainable AI: Environmental Implications, Challenges and Opportunities*. Proceedings of Machine Learning and Systems (MLSys).