Carbon-Aware LLM

Artem Koval

December 19, 2025

Keywords: cloud, LLM, scheduling, carbon intensity, temporal shifting, serverless

Project Links

Web App URL — GitHub: Web App — GitHub: Scheduler — GitHub: Scheduler Simulation

1 Motivation and Overview

OpenAI's ChatGPT sees over 2.5 billion requests daily [4]. In contrast with traditional cloud computing, every Generative AI prompt executes immediately "on-demand", providing the user with a response without consideration for power grid capacity or carbon intensity conditions.

With the rise of Agentic AI, LLM tasks become more complex, computationally intensive, and time-consuming. OpenAI's **o3-deep-research** model can generate responses containing up to 100,000 tokens, and deep research tasks may take between 5 and 30 minutes to complete their work [3]. I argue that such tasks often do not require being on demand, and significant environmental benefit can be derived from carbon-aware scheduling of such tasks.

The project implements a full-stack web application between a user and an LLM. The app allows the user an option to delay (temporally shift) prompts before they are entered into an LLM. The prompts are executed by the scheduler to run within a user-defined timeframe (e.g., "in the next 24 hours") based on the Carbon Intensity of the electricity grid associated with the processing data center. The performance of the developed scheduler and the reduction in carbon footprint are simulated based on historical data from 2024.

2 Overview of what was built

Web App Link

The LLM is served to users via Azure Static Web App with Azure Serverless Functions (Python). Azure Functions here accomplish the following:

- Use the Electricity Maps API to get the current carbon intensity and the past 24-hour carbon intensity.

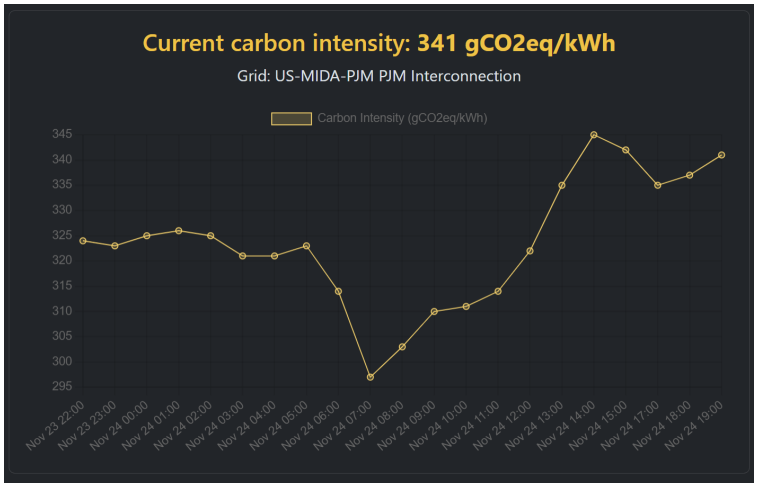

Figure 1: Example of reporting the carbon intensity 24-hour history to the front-end

- "Send" button: routes the prompt to Azure OpenAI models, and LLM responses back to the user. Models the typical interaction between a user and an LLM.
- "Schedule" button: stores the prompt and current carbon intensity in the Azure Data Table, which will be accessed by the scheduler. See **Appendix A** for schema.
- Additionally, currently scheduled and completed prompts are displayed.

Scheduler Link

- A scheduler is implemented as a serverless Azure Function on a timer. For each pending prompt, the scheduler executes it if conditions are favorable. See **Scheduler** section for details.

Scheduler Simulation Link

- To test the performance of my scheduler, I ran a simulation using real carbon intensity data from year 2024, exported from Electricity Maps. See **Simulation & Results** for details.

3 Methodology

I will only consider operational carbon during execution of the LLM query, and approximate the energy per query. To quantify the benefit of temporal shifting, the amount of carbon savings is measured:

$$\text{Saved } CO_2 \text{ (g)} = \text{Immediate } CO_2 \text{ (g)} - \text{Scheduled } CO_2 \text{ (g)}$$

Where:

- Immediate carbon is the footprint of an LLM query as if it was not scheduled. This is a default, on-demand LLM execution.
- Scheduled carbon is the footprint of an LLM query after a temporal shift.

The carbon footprint of a single query is calculated in the following way:

$$CO_2 \text{ (g)} = \text{Energy per query} \times \text{Carbon Intensity} = \frac{E_{query} \text{ (Wh)}}{1000} \times CI \text{ (gCO}_2\text{/kWh)}$$

The methodology for quantifying energy per prompt is presented by Jegham et al [1]. Two models are considered: **GPT-4o-mini** and **GPT-5** (which I implement and test).

For **GPT-4o mini**, energy consumption varies with prompt size. Because the target of my paper is the agentic tasks, I will assume long prompts (10,000 input, 1,500 output), which consume 2.106 ± 0.477 Wh [1]. Therefore, for **GPT-4o mini** $E_{query} = 2.106 \pm 0.477$ Wh.

For **GPT-5**, which is a much larger model, this equates to $E_{query} = 2.106 \pm 0.477$ Wh.

Under this model, the reduction in carbon intensity is directly proportional to the reduction in carbon footprint. Therefore, if the scheduler executes a prompt at 70% lower carbon intensity, the result is a 30% redduction in operational carbon footprint.

The plot below shows typical weekly Carbon Intensity metrics in the local power grid region (PJM Interconnection), demonstrating significant variation. This suggests that a temporal shift of LLM queries by as little as an hour could potentially reduce the carbon footprint by up to 30%.

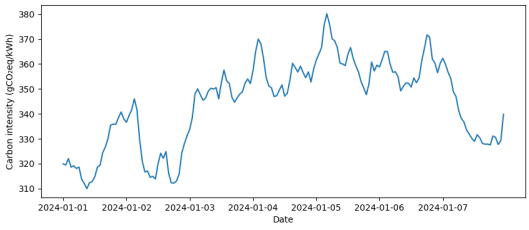

Figure 2: PJM Interconnection Carbon Intensity variation in a week of October 2024

Scheduler

The core of the project is the scheduler, which runs every hour, queries the pending prompts, and decides whether to execute them.

The scheduler uses a probabilistic approach, where the probability of executing a given prompt is determined by:

- **Historical trends**. I put the carbon intensity data into bins using the previous 3 days as a reference window. When the current CI falls into the lowest bin—matching the minimum observed over those 3 days—the scheduler executes immediately, since CI is unlikely to drop further. This approach is better than binning on global min and max carbon intensity, since it attempts to account for local and seasonal trends.

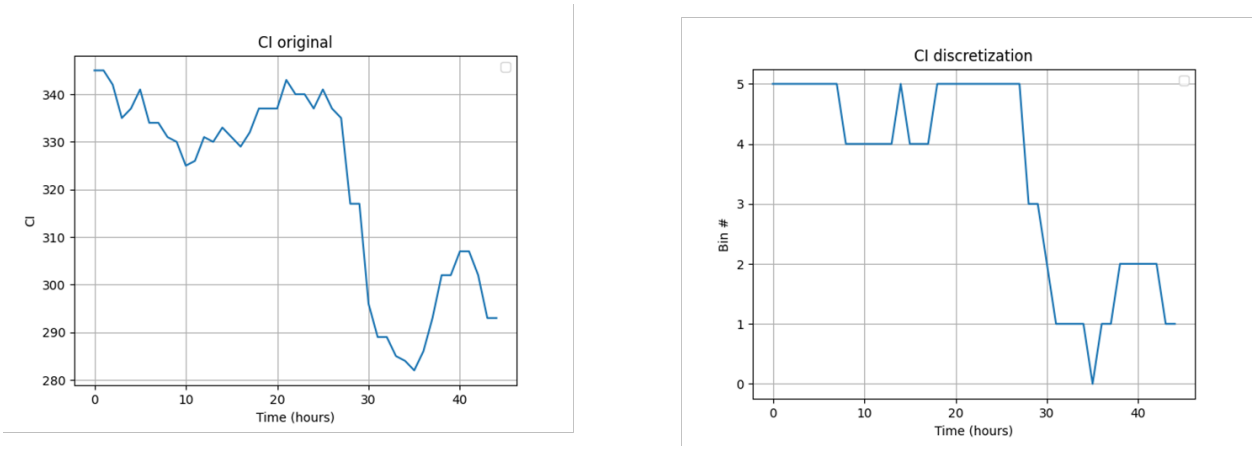

Figure 3: CI Discretization approach

- **Deadlines**. As the deadline approaches, the likelihood of execution increases.
- **Benefit**. Higher reduction in carbon has a higher probability of execution. Benefit is calculated using bins mentioned above (benefit = $\text{bin}_{\text{at schedule time}} - \text{bin}_{\text{at execution time}}$). Jobs only execute when the current CI is lower than one at submission. If no such opportunity arises before the deadline, the prompt executes late at expiration.
- **Rates of change**. The scheduler becomes more aggressive as carbon intensity increases. This is done by looking at the carbon intensity at the last two hours: $\Delta_{CI} = CI_{t-1} - CI_t$. If Δ_{CI} is positive, carbon intensity is going down, and we can wait.

When these strategies are combined, execution probability follows the equations and curves shown on the next page. See **Appendix C** for details.

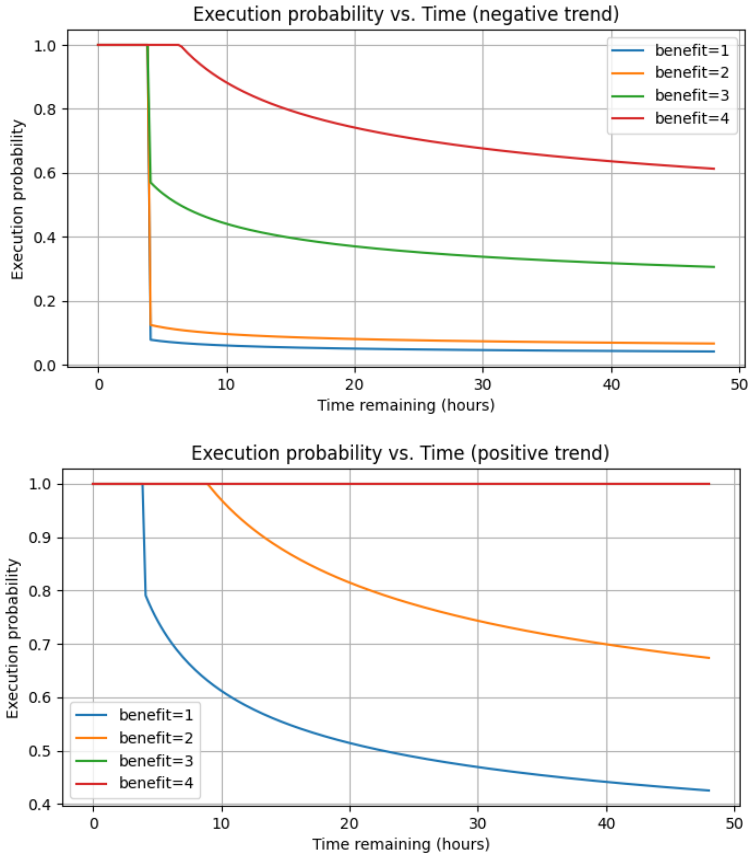

Figure 4: Probability to execute a prompt when CI rate of change is negative (top, less aggressive) and positive (bottom, more aggressive)

There are some significant limitations to my scheduler. First, jobs are executed only if there is a reduction in carbon intensity and, consequently, a decrease in carbon footprint, which is not guaranteed. For example, suppose a user wants to schedule a prompt in a 6-hour window. If carbon intensity only increases within that window, the prompt will execute when it expires, possibly at a much higher carbon intensity.

Second, my scheduler is very naive and simple. There are existing carbon-based schedulers that could be incorporated into the app. This project can be further developed by enhancing the simulated performance of the scheduler and testing it in a live environment. Additional tools, such as the Electricity Maps forecasting feature (which is not free), could be incorporated to reliably reduce operational carbon.

Simulation & Results

Testing the live version of the scheduler would take a significant amount of time. To evaluate the performance, the hourly carbon intensity data was exported from Electricity Maps for the year 2024.

Then, I simulate a user submitting jobs to the scheduler at random times throughout the year, with varying deadlines. The simulation runs the scheduler every hour until completion, exactly like the live system.

I ran a total of 300 jobs with varying deadlines (ranging $[6, 12, 24, 48]$ hours from submission time). A summary of the results is seen in Figure 5 below. Note that savings and penalties are in terms of carbon intensity values. These values could be converted to carbon quantity directly using the formulas provided in the methodology section. I decided that reporting performance in terms of carbon intensity is more robust than using estimated prompt-to-carbon values for the following reasons:

- Estimated values per prompt are unreliable, since they are reliant on model size, prompt size, and do not apply to reasoning models.
- Reduction in carbon intensity at execution is directly proportional to the reduction in operational carbon footprint.

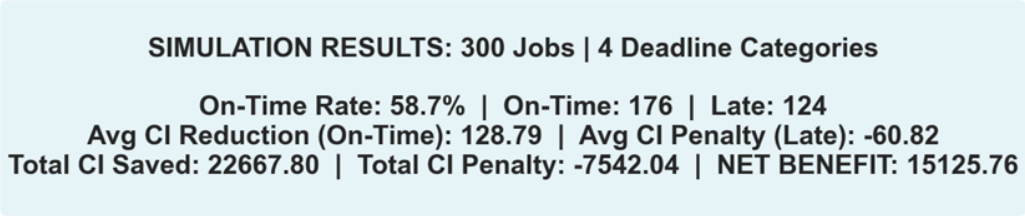

Figure 5: Simulation results summary

As shown above, the scheduler achieves meaningful carbon reductions. 58% of jobs arrived on time, meaning that they executed at lower carbon intensity.

Next figure shows the breakdown of on-time vs late jobs. On-time jobs are split into two categories:

- **Benefit**: These are the jobs triggered by the probability roll.
- **Low CI**: Jobs executed when carbon intensity reached the lowest bin—matching the minimum observed in the previous 3 days. This prevents the scheduler from waiting indefinitely for unrealistic improvements.

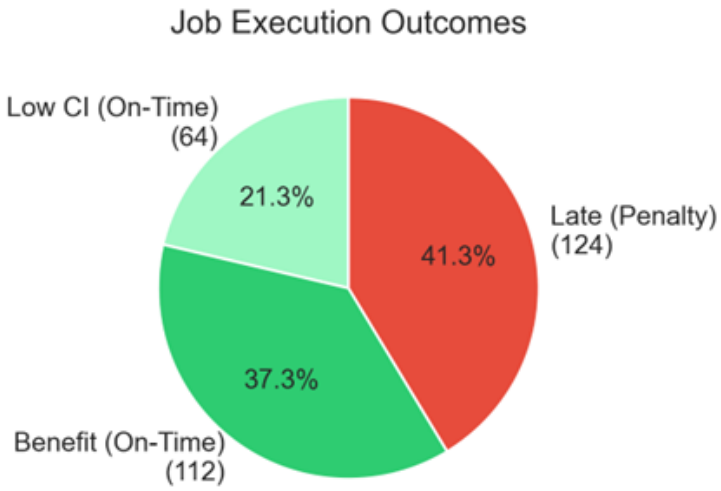

Figure 6: On-Time vs Late jobs breakdown. Most jobs executed on-time

Next plot shows the distribution of late vs on-time jobs by user-defined deadline. It shows that scheduling prompts to execute within 6 hours and 12 hours is too low to get reliable carbon benefits. This makes sense: carbon intensity usually fluctuates every 24 hours, and typically we need to wait longer than that to achieve a consistent reduction in carbon footprint.

Figure 7: On-Time vs Late jobs by deadline. Jobs with a deadline > 24 hours consistently executed at lower carbon intensity

Next plot shows how long before the deadline jobs tend to execute. Notable result is at 48-hour deadline. It demonstrates that the scheduler tends to execute closer to the deadline (as intended by the scheduler's logarithmic probabilities). It shows that the scheduler is more aggressive in the beginning, willing to wait longer to execute at a higher benefit.

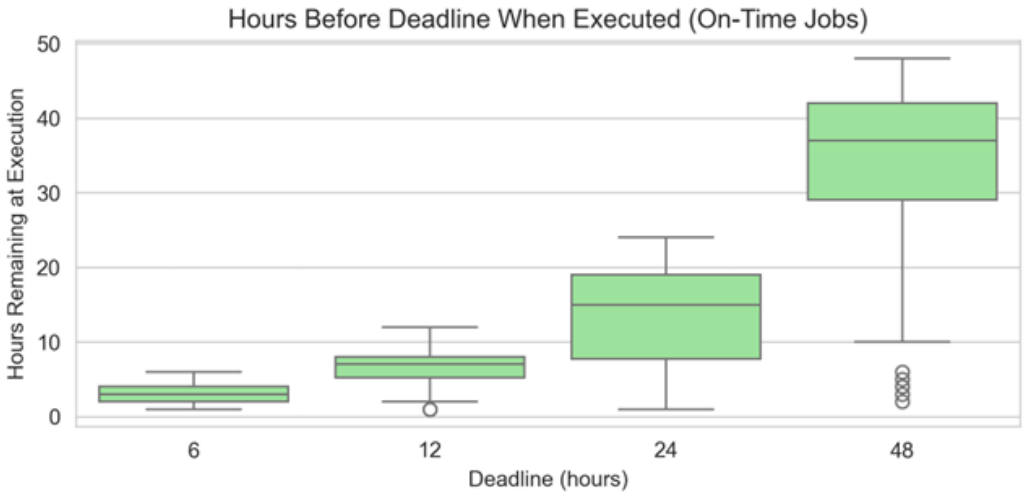

Figure 8: Hours before deadline when executed

Next plot shows the distribution of benefits and penalties for on-time and late jobs. The plot appears more negative because it includes 6 and 12 hour deadlines, when most jobs are executed late at a penalty.

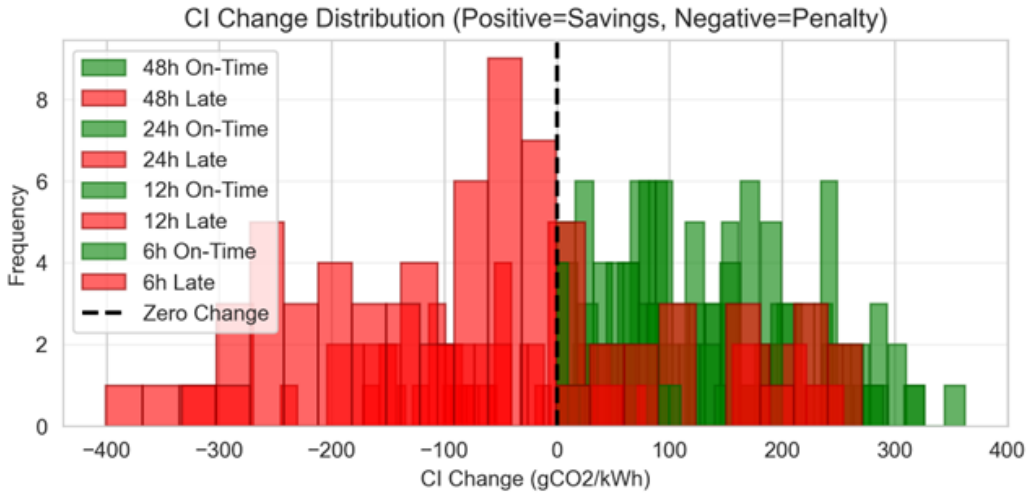

Figure 9: CI Benefit/Penalty distribution

Next plot two plots show additional metrics on on-time vs late jobs savings and penalties.

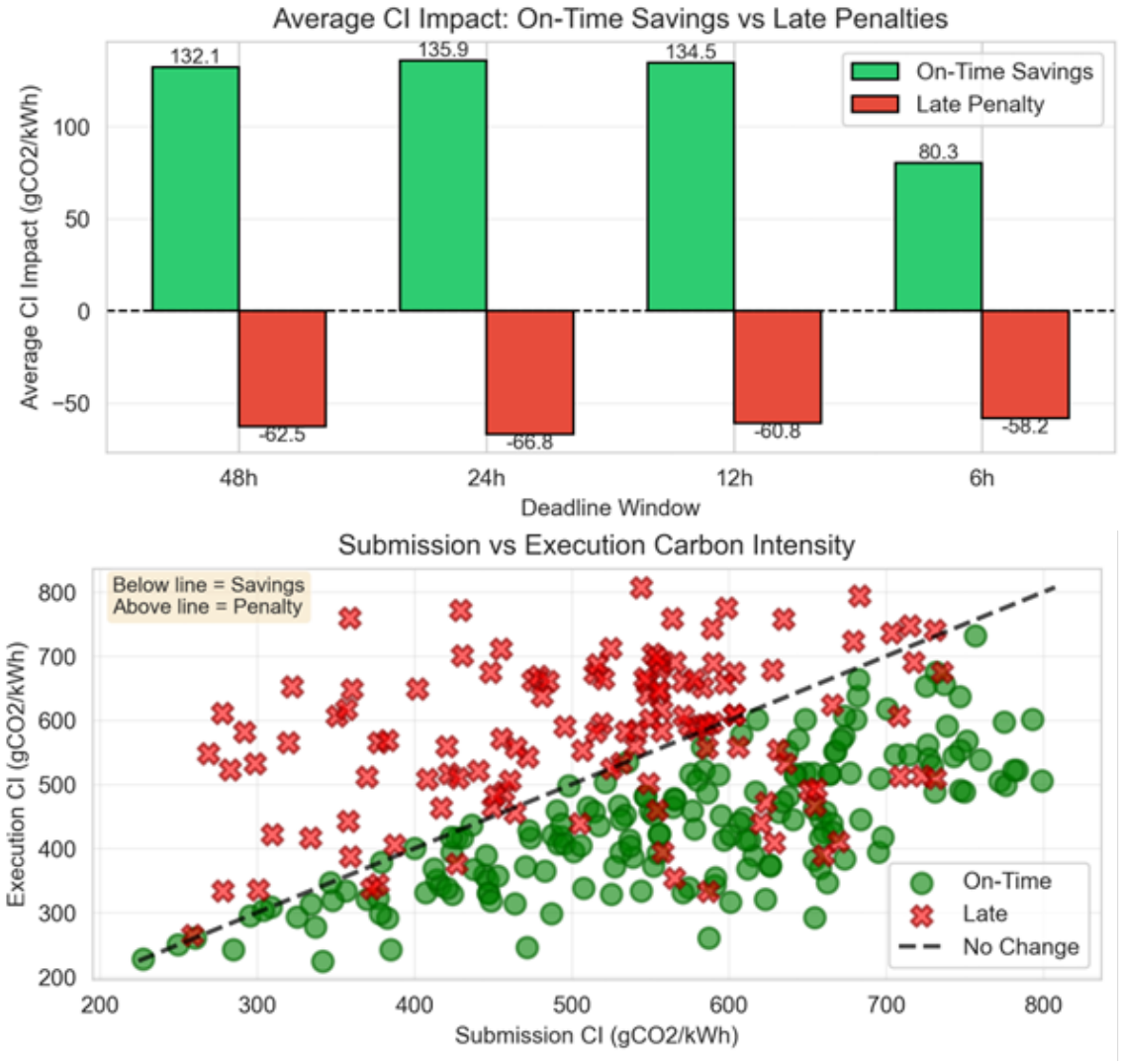

Figure 10: CI benefits and penalties for on-time vs late jobs

Conclusion

As generative AI scales, operational carbon emissions will become increasingly significant. This project demonstrates that temporal delays of at least 24 hours can be consistently translated into carbon reductions without degrading model quality or requiring infrastructure changes. This approach is also model-agnostic. For workloads that can tolerate such delays, carbon-aware scheduling represents a practical pathway toward more sustainable AI systems.

4 Appendix A: Architecture

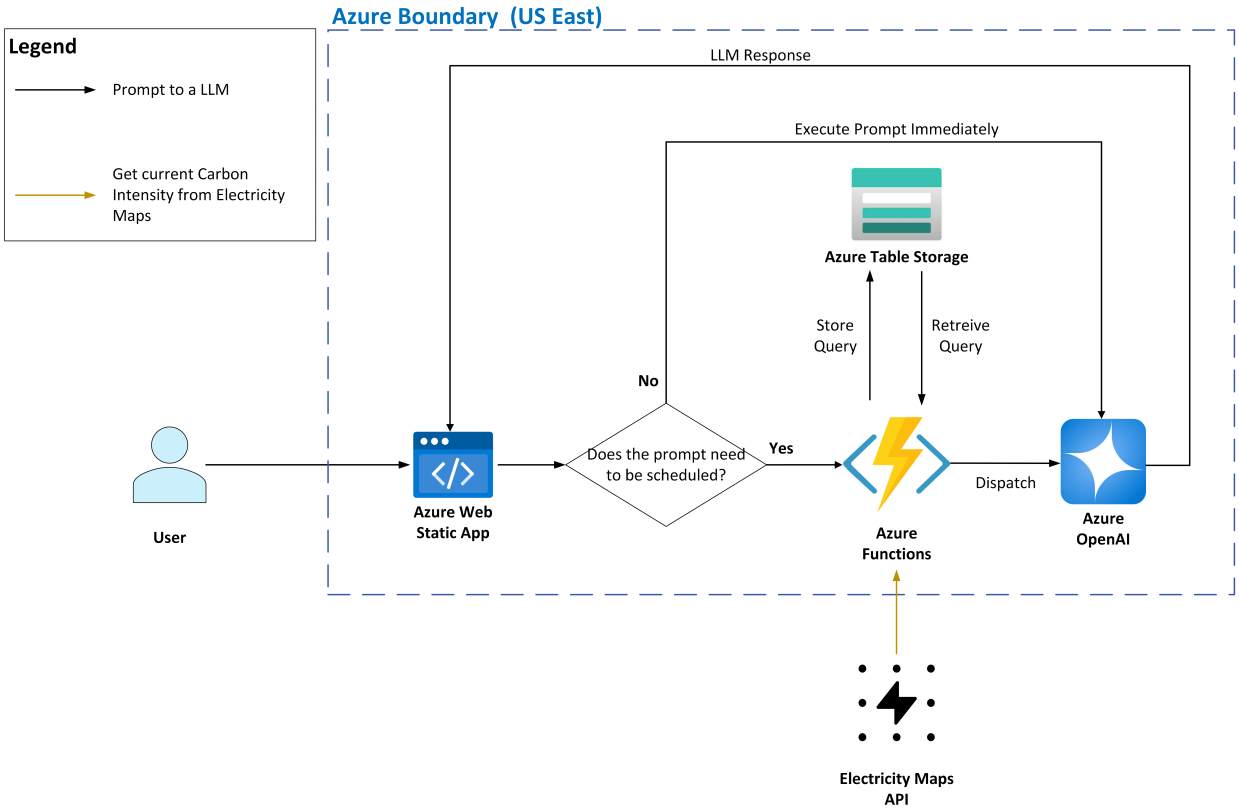

Figure 11: Enter Caption

Front-end

- *Technology:* Azure Static Web App
- *Description:* Serves the user interface. Accessed via:
- wonderful-flower-0e3ecf80f.2.azurestaticapps.net

LLM

- *Host:* Azure OpenAI
- *Model:* GPT-4o-mini, GPT-5
- *Description:* The project aims to optimize the carbon footprint of queries dispatched to these models.

Storage

- *Technology:* Azure Table Storage

- *Description:* A user writes a prompt, schedules it to run within 6, 12, 24, or 48 hours. It then gets dispatched to an LLM when Carbon Intensity conditions are favorable.

Scheduling & Query Dispatch

- *Technology:* Azure Functions
- *Description:* Microsoft's Serverless application solution. Responsible for adding queries from the frontend to an Azure Queue and dispatching queries from the Queue to the LLM. Implemented as a probabilistic scheduler.

Field	**Type**	**Description**
PartitionKey	String	Partition identifier
RowKey	UUID	Unique ID for each entry
Prompt	String	The prompt text to be processed
Status	String	Processing status (e.g., "pending")
Model	String	AI model to use
Schedule	String	When to execute the prompt
CreatedAt	ISO 8601	Timestamp when entry was created
ExpirationDate	ISO 8601	When the entry should expire
CarbonIntensity_s	Float	Carbon intensity at submission time
CarbonIntensity_c	Float	Carbon intensity at completion time
CompletedAt	ISO 8601	Timestamp when processing finished
Response	String	Generated response from the model
OutTokens	Integer	Number of output tokens generated

Table 1: Database schema for prompt scheduling system

Appendix B: Note on LLM model providers

Originally, I attempted to develop an application using NIM (NVIDIA Inference Microservice) as my LLM interface. I discovered that NIM microservices are deeply integrated with AWS, and that my LLM traffic was routed to the AWS Global Accelerator edge node in Seattle. From there, I found it impossible to determine which data center my LLM traffic is routed to, as AWS Global Accelerator does not disclose the underlying compute locations. This lack of transparency represents a classic challenge in sustainable computing, as it prevents accurate tracking of carbon intensity.

This led me to use Microsoft Azure OpenAI as my LLM platform. Azure OpenAI allowed me to explicitly select the region where my LLM requests are processed (for my project, I arbitrarily chose **eastus2**). Electricity Maps, in turn, explicitly maps each Azure cloud region to its corresponding Electricity Maps zone (for **eastus2**, a corresponding zone is **PJM Interconnection**). This setup allowed me to deterministically identify both the region where my LLM traffic is processed and its associated carbon intensity.

Appendix C: Scheduler equations

$$P_{\text{exec}}(b, t, \Delta_{\text{CI}}) = \begin{cases} 0 & \text{if benefit } b \leq 0 \\ 1 & \text{if time remaining } t < 4 \\ 0.001 & \text{if } \Delta_{\text{CI}} > 1 \text{ and } b < 4 \\ p_{\text{base}}(b) \cdot u(t, \Delta_{\text{CI}}) & \text{otherwise} \end{cases}$$

Where:

b = carbon intensity bin benefit

t = hours remaining until deadline

Δ_{CI} = recent trend $CI_{t-1} - CI_t$

Base probability from benefit p_{base}:

$$p_{\text{base}}(b) = \begin{cases} 1.0 & \text{if } b \geq 4 \\ 0.5 & \text{if } b = 3 \\ 0.1 \ln(b+1) & \text{if } 0 < b < 3 \end{cases}$$

Urgency factor u:

$$u(t, \Delta_{\text{CI}}, b) = \begin{cases} \frac{3}{1+\ln(t+1)} \times 10 & \text{if } \Delta_{\text{CI}} < -1 \text{ and } b \geq 1 \\ \frac{3}{1+\ln(t+1)} & \text{otherwise} \end{cases}$$

References

[1] N. Jegham, M. Abdelatti, L. Elmoubarki, and A. Hendawi, *How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference*, May 2025. DOI: 10.48550/arXiv.2505.09598.

[2] N. Jegham, M. Abdelatti, C. Y. Koh, L. Elmoubarki, and A. Hendawi, "How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference," *arXiv preprint arXiv:2505.09598*, 2025, v5, 11 Nov 2025. arXiv: 2505.09598 [cs.CY]. [Online]. Available: https://arxiv.org/abs/2505.09598.

[3] OpenAI. "Deep research. "[Online]. Available: https://openai.com/index/introducing-deep-research/.

[4] E. Roth. "Openai says chatgpt users send over 2.5 billion prompts every day." Accessed October 8, 2025, Accessed: Oct. 8, 2025. [Online]. Available: https://www.theverge.com/news/710867/openai-chatgpt-daily-prompts-2-billion.