GreenestRoute: Carbon-Aware Job Scheduling for a Greener ICT Sector

Yanzhi Chen
Delft University of Technology
Delft, The Netherlands
tomchen@tudelft.nl

Alex Hautelman
Delft University of Technology
Delft, The Netherlands
p.a.hautelman-1@student.tudelft.nl

Daniel Rugge
Delft University of Technology
Delft, The Netherlands
drugge@tudelft.nl

Sydney Kho
Delft University of Technology
Delft, The Netherlands
sydneykho@tudelft.nl

Yi Wu
Delft University of Technology
Delft, The Netherlands
y.wu-37@tudelft.nl

Abstract

Software can be run almost anywhere and at any time, yet this flexibility is under-exploited for cutting carbon emissions. **GreenestRoute** is an open-source scheduler that routes flexible compute jobs to a server location and start time that jointly minimise CO_2 emissions and server cost, making green scheduling a practical default. It integrates with the Carbon Aware SDK [1] for near-real-time and forecast grid intensity data, and converts projected emissions into a monetary carbon cost using the Social Cost of Carbon (SCC) by default. Organisations can also override this with configurable market carbon prices (e.g., from the EU ETS sourced via ICAP [2]) to weigh their options based on internal budgets. This report describes the system design, methodology, and evaluation on historical GitHub Actions workloads.

Keywords

Green Computing, Carbon-Aware Scheduling, Sustainable Software Engineering, CI/CD, Cloud Computing

ACM Reference Format:
Yanzhi Chen, Alex Hautelman, Daniel Rugge, Sydney Kho, and Yi Wu. 2026. GreenestRoute: Carbon-Aware Job Scheduling for a Greener ICT Sector. In *CS4575-Q3-26: Sustainable Software Engineering, TU Delft*. ACM, New York, NY, USA, 7 pages.

1 Introduction

The ICT sector's carbon footprint is large, growing, and increasingly hard to ignore. A 2024 joint report by the International Telecommunications Union and the World Bank [3] estimates that ICT emitted at least 567 million tonnes of CO_2equivalent in 2022 (around 1.7% of global greenhouse gas emissions) while consuming approximately 1,183 TWh of electricity (4.7% of the world total). Data centres account for roughly 13% of ICT emissions, and cloud and content data centre emissions grew by 46% between 2020 and 2022. With AI workloads, high-performance computing, and the digitalisation of every industry sector, these numbers will only grow without deliberate intervention.

The carbon intensity of electricity, or how much carbon is emitted per kWh of energy, varies by an order of magnitude across regions and throughout the day as demand and the mix of renewables and fossil generation changes. Unlike factories or vehicles, many software jobs have no fixed time or place requirement: they only need to complete before a deadline. Routing these workloads to greener locations, or delaying them until the grid is cleaner, can cut emissions substantially with no impact on outcomes. The literature confirms this: temporal shifting alone achieves 31–70% reductions [4, 5] and spatial routing adds further gains [6].

We designed a scheduler, GreenestRoute, that applies this to any flexible cloud workload: CI/CD pipelines, batch processing, model training, or scheduled data jobs. Given a job's duration, deadline, candidate server locations, and their associated costs, it queries real-time grid intensity data, estimates a job's energy use, converts projected emissions into a monetary carbon cost, and outputs the single (location, start time) pair that minimises the combined total. The key novelty is that GreenestRoute optimises both dimensions simultaneously and expresses the result in financial terms, bridging the gap between sustainability teams and the engineers and budget owners who manage spending. Additionally, it incorporates an optional active energy measurement module to refine the initial estimate for recurring jobs to achieve increasingly accurate scheduling decisions based on real-world data, in turn allowing for precise carbon reporting.

2 Related Work

Carbon-aware computing has attracted growing attention across academia and industry, with prior work spanning three complementary themes.

Temporal shifting. Delaying flexible jobs until the grid is cleaner is the most studied strategy and yields consistently strong results. Wiesner et al. [7] show that even simple heuristics reduce carbon footprints meaningfully for geo-distributed batch workloads. Claßen et al. [4] demonstrate 31.2% emission reductions in CI/CD pipelines through intelligent delay, with no change to job outcomes; CASPER [5] reports gains of up to 70% for a broader workload class, with no performance degradation.

CS4575-Q3-26, TU Delft

Spatial routing. Choosing the lowest-carbon data centre region adds a complementary dimension. LinTS [6] shows that carbon-aware routing for inter-datacenter transfers reduces emissions by up to 66% compared to the worst case and 15% against alternative approaches, while preserving all deadline constraints.

Tooling and real-world adoption. The Carbon Aware SDK [1] from the Green Software Foundation provides a mature, open-source API for real-time and forecast grid intensity across hundreds of regions, and has seen adoption in production at scale: UBS successfully reduced cloud emissions using carbon-aware scheduling principles [8], and the Zeus framework [9] demonstrates that carbon and energy signals can be integrated into ML training pipelines with minimal overhead. GreenestRoute builds on this mature tooling ecosystem while adding explicit cost conversion and full user configurability.

3 System Design

3.1 Architecture

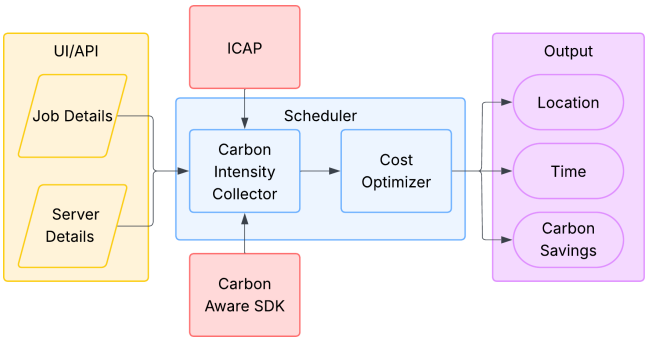

Figure 1: High-level architecture of GreenestRoute.

GreenestRoute consists of four loosely coupled components (Figure 1). The **User Interface / API** accepts a job specification: duration δ, optional energy estimate E (kWh), deadline d, candidate locations $\mathcal{R}$, per-location server cost p_r ($/h), and an optional carbon price π ($ /tCO$_2$e). The **Carbon Intensity Collector** queries the Carbon Aware SDK for forecast grid intensity $\hat{c}(r,t)$ at each $r \in \mathcal{R}$ over all feasible start times $t \in [t_{\text{now}},\ d-\delta]$. The **Cost Optimiser** converts intensity to total cost and selects the minimum (formulation below). The **Output** returns the optimal (r^*, t^*) together with estimated CO$_2$ savings (g) and monetary savings ($) relative to an immediate, unoptimised baseline.

3.2 Input Parameters

Table 1 summarises all user-configurable inputs. Energy consumption E is optional. If omitted it is estimated from server hardware profiles (Section 3.4).

Table 1: GreenestRoute input parameters.

Param.	Unit	Description
δ	h	Job execution duration
E	kWh	Energy consumption (optional)
d	timestamp	Job deadline
$\mathcal{R}$	—	Candidate server locations
p_r	$/h	Server cost per location
π	$/tCO$_2$e	Carbon price

3.3 Carbon Intensity via the Carbon Aware SDK

Real-time and forecast grid intensity data are retrieved through the Carbon Aware SDK [1], which processes data from `Electricity Maps`. For each location–time pair the SDK returns:

$$\hat{c}(r,t) \in \mathbb{R}_{\geq 0} \qquad [\text{gCO}_2\text{eq/kWh}]$$

Forecast resolution is provider-dependent (typically one hour).

3.4 Dynamic Energy Estimation

A key challenge in carbon-aware scheduling is accurately estimating a job's energy consumption (E). While a static model can provide an initial guess, it fails to capture the diversity of real-world workloads. GreenestRoute addresses this through a dynamic feedback loop that refines the energy estimate over time.

3.4.1 Initial Estimate. If no historical data or estimate by the user is available, energy consumption is estimated with a static model based on server hardware profiles and expected CPU utilisation. We assume power draw increases linearly with CPU utilisation from idle power draw, and estimate the amount of energy this results in using the job time. The Power Usage Effectiveness (PUE) incorporates the energy overhead of running a data center. This gives the equations:

$$P_{\text{active}} = P_{\text{idle}} + U \cdot (P_{\text{max}} - P_{\text{idle}}) \tag{1}$$

$$E_{\text{initial}} = P_{\text{active}} \cdot \delta \cdot \text{PUE} \cdot 10^{-3} \tag{2}$$

where P_{idle} and P_{max} represent the server's idle and maximum power draw in Watts, $U \in [0,1]$ is the estimated CPU utilisation, δ is job duration in hours, and PUE is the data centre's Power Usage Effectiveness (e.g., 1.10 for Google Cloud [10]).

To accurately model U, we apply a keyword-based heuristic to the job name. Low-intensity tasks (e.g., `update`, `dependabot`, `stale`, `tweet`, `winget`, `leaderboard`, `vex`) are assigned $U = 0.15$. High-intensity tasks (e.g., `build`, `install`, `pnnx`, `esp32`) are assigned $U = 0.85$. All other jobs fall back to a default $U = 0.30$.

3.4.2 Measurement and Feedback Loop. This initial estimate is improved upon with each job execution. Our proof-of-concept, implemented as a GitHub Actions workflow, integrates the `eco-ci` tool [11] to measure the actual energy consumed by each job run. After a job completes, its measured energy consumption, E_{measured}, is captured.

This new measurement is used to update the stored energy estimate for future scheduling decisions using an Exponential Moving Average (EMA). This smooths out variations and adapts to changes

in the job's behaviour over time:

$$E_n = \alpha \cdot E_{\text{measured}} + (1 - \alpha) \cdot E_{n-1} \quad (3)$$

where E_n is the new estimate, E_{n-1} is the previous estimate, and α is a smoothing factor that controls the trade-off between responsiveness and stability. We selected $\alpha = 0.3$, which gives more weight to the historical trend (70%) than to a single new data point (30%). This is desirable in our context, as a single job's energy consumption can be noisy due to transient system factors, and we want the estimate to converge smoothly rather than overreacting to outliers. This value is a common choice for EMA filters and proved effective for our purposes.

The dynamic feedback mechanism is demonstrated in our live proof-of-concept (Section 4.2). For the large-scale historical simulation where live measurement is not possible, we rely on the static model.

3.5 Carbon Pricing

By default, GreenestRoute converts projected emissions into a monetary cost using the Social Cost of Carbon (SCC), set conservatively to $185 per metric ton [12]. This ensures scheduling decisions reflect the true economic damage of emitted carbon. However, the system is fully configurable; users operating under specific regulatory regimes or internal budgets can override this default with market-driven carbon prices, such as historical EU ETS data sourced from the ICAP Allowance Price Explorer [2].

3.6 Cost Formulation

The CO_2 mass produced by running a job at (r, t) is:

$$m(r, t) = E \cdot \hat{c}(r, t) \cdot 10^{-6} \quad [\text{tCO}_2\text{e}] \quad (4)$$

The monetary carbon cost, using the selected carbon price π, is:

$$C_{\text{carbon}}(r, t) = \pi \cdot m_{\text{CO}_2}(r, t) \quad [\$] \quad (5)$$

Total cost combines server cost and carbon cost:

$$C_{\text{total}}(r, t) = p_r \cdot \delta + C_{\text{carbon}}(r, t) \quad (6)$$

GreenestRoute selects the (location, start time) pair that minimises total cost subject to the deadline:

$$(r^*, t^*) = \arg \min_{\substack{r \in \mathcal{R} \\ t \le d - \delta}} C_{\text{total}}(r, t) \quad (7)$$

The search space is practically small (tens of locations, up to 24 hourly slots), so brute-forcing is feasible.

4 Methodology

Our evaluation is two-fold. First, we conduct a large-scale simulation on a historical dataset of CI/CD jobs to assess the potential carbon and cost savings of our scheduling algorithm at scale. Second, we implement a live proof-of-concept (PoC) within GitHub Actions to demonstrate the end-to-end feasibility of the system, including its dynamic energy feedback loop.

4.1 Historical Workload Simulation

Workload. We evaluate the scheduler on a historical corpus of 65,145 GitHub Actions CI/CD jobs. This provides a realistic and reproducible benchmark based on real-world development activity. The dataset was downloaded via the GitHub API and spans five active open-source repositories: `apache/spark`, `kubernetes/minikube`, `k3s-io/k3s`, `microsoft/vscode`, `Tencent/ncnn`. The extracted jobs encompass a wide range of workflow types (e.g., standard builds, dependabot updates, stale issue management) with execution durations (δ) extracted directly from the API ranging from under one minute to over two hours. To ensure comparable server energy measurements across GCP and AWS regions, the corpus was filtered to include only jobs executed on Linux hardware.

Methodology. Each historical job is replayed through the GreenestRoute scheduler. The job's original timestamp is treated as the earliest start time. The scheduler then uses Equation 7 to determine the optimal location and start time within a given deadline window. Since live energy measurement is not possible for historical data, the energy consumption E is modeled using the static formula in Section 3.4.

Baseline. The baseline for comparison is the "as-is" scenario: executing the job immediately upon its submission time. For the simulation baseline, all jobs were assumed to run in the `eastus` region. This location was selected as it is a standard default for GitHub's Azure-hosted runners, possesses an average grid carbon intensity, and represents a relatively low-cost computing tier. Baseline emissions and costs are calculated using the historical grid intensity for that location and time, retrieved from the Carbon Aware SDK. The baseline energy calculation utilizes a PUE of 1.10 [10] and the power profile of a standard 2vCPU Ubuntu runner, defined as 4.5W at idle and 12.0W at maximum load.

4.2 Live Proof-of-Concept

Workload. To validate the system in a live environment, we created a recurring GitHub Actions workflow. This workflow runs a synthetic task: three minutes of random forest training using data from the scikit-learn package [13]. Using a synthetic workload ensures a reproducible energy profile, which is ideal for demonstrating the convergence of the dynamic energy estimation model.

Methodology. The PoC is implemented as a set of interconnected GitHub Actions workflows. However, due to the observed unreliability of GitHub's native 'schedule' cron trigger for frequent, time-sensitive tasks, the scheduling workflow was triggered externally. A local script executed 'gh workflow run' on a reliable cron schedule, ensuring consistent initiation of the scheduling logic. The system operates as follows:

(1) **Scheduler (`scheduler.yml`):** Triggered by the external cron, this workflow fetches and caches carbon intensity forecasts for the candidate locations. It then executes the core logic from Equation 7 to calculate the optimal start time. Finally, it uses the third-party `austenstone/schedule` action to queue the worker job for that specific time.
(2) **Poller (`poller.yml`):** Triggered by the external cron, this workflow fetches the schedule and triggers any workers on the schedule that should execute now.
(3) **Worker (`energy_test_job.yml`):** This is the actual job that runsthe synthetic task. It integrates the `eco-ci` tool [11] to measure the energy consumed during its execution. Upon completion, the workflow takes the new energy measurement, applies EMA smoothing against the previously stored

value (Equation 3), and updates a GitHub repository variable with the new, refined estimate. This updated value is then automatically used by the scheduler in its next run.

This setup demonstrates the feasibility of deploying GreenestRoute in a real-world CI/CD platform and validates the effectiveness of the adaptive energy estimation, while also highlighting practical platform limitations.

4.3 Metrics

For both the simulation and the PoC, we report per-job and aggregate values for: average grid intensity (gCO_2eq/kWh), total CO_2 emitted (kg), server cost ($), carbon cost ($), and total cost ($), together with percentage reductions vs. the baseline.

5 Results

5.1 Historical Simulation Results

To isolate the impact of the allowed delay window, Table **??** compares the relative savings achieved across varying maximum deadlines. The data reveals a stark contrast in how spatial and temporal flexibility scale. In the multi-region scenario, the vast majority of savings are unlocked almost immediately. Even with a highly restrictive 1-hour delay window, the scheduler aggressively routes workloads to cheaper, greener regions, instantly capturing an 11.26% reduction in compute cost and an 88.21% reduction in carbon emissions. Expanding the delay window to 24 hours yields diminishing but meaningful marginal returns, pushing the emissions reduction up to 91.57%, while maintaining the same compute cost savings. In contrast, the single-region scenario relies entirely on temporal shifting and is heavily dependent on longer deadlines. With a 1-hour window, the grid mix rarely changes enough to warrant a delay, yielding effectively zero savings. As the window expands to 12 and 24 hours, the scheduler can make use of the daily cycle of the local grid, eventually reducing emissions by about 6%.

Max Delay	Single-Region	Multi-Region (8 Locations)	
	Emissions Saved	Emissions Saved	Compute Saved
1 hour	0.00%	88.21%	11.26%
2 hours	0.90%	88.48%	11.26%
6 hours	3.06%	89.30%	11.26%
12 hours	4.60%	90.69%	11.26%
24 hours	6.08%	91.57%	11.26%

Table 2: Impact of the maximum delay window on relative emissions and compute cost reductions (evaluated against an immediate-execution baseline). SCC modeled at $185/t.

Figure 2 illustrates this both the emissions and compute cost benefits via a Green Premium Waterfall. A baseline compute-only cost of $3896.71 is reduced by $438.70 (11.3%) by the spatial routing of jobs to regions with cheaper server pricing. A further $42.41 (1.09%) is saved through the active minimization of the monetized carbon cost, yielding a final, optimized system cost of $3415.60.

Figure 3 visualizes the distribution of actual wait times for jobs with a 24-hour maximum delay policy. A substantial majority of jobs execute almost immediately (0–1 hours), indicating that the grid is frequently already at a local carbon optimum at the time of submission, or that no significantly cleaner window exists within the permitted deadline. However, the pronounced long tail and the notable spike at the 22–24 hour mark validate the algorithm's efficacy: it actively utilizes the full extent of the delay window to bypass highly carbon-intensive grid periods, executing at the very last responsible moment to secure the lowest possible emissions footprint.

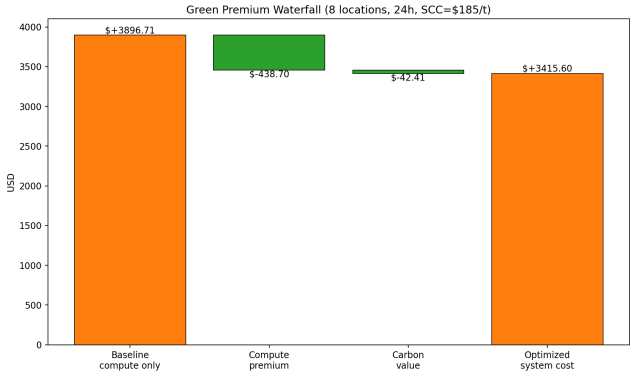

Figure 2: Green Premium Waterfall under a 24-hour scheduling policy and a $185/t Social Cost of Carbon (SCC). The methodology successfully reduces both the raw compute value and the internalized carbon value, resulting in a cheaper, greener overall system cost.

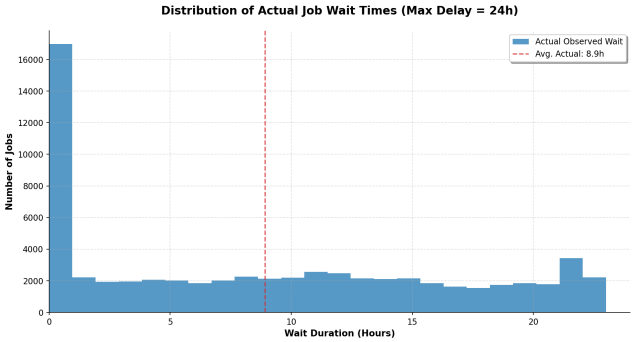

Figure 3: Histogram of expected versus actual wait times for jobs evaluated under a 24-hour delay policy. The scheduler capitalizes on immediate clean energy when available but actively exploits the full delay window to avoid carbon-intensive generation periods.

5.2 Live Proof-of-Concept Validation

The live PoC served two primary goals: validating the dynamic energy feedback loop and observing the scheduler's behavior in a real-world environment.

Energy Estimation Convergence. The dynamic feedback loop proved highly effective. As shown in Figure 4, the absolute error between the scheduler's energy estimate and the energy measured by `eco-ci` decreased exponentially over successive runs. The initial heuristic-based estimate was significantly different from the measured reality, but the EMA smoothing algorithm (Equation 3)

quickly corrected the estimate, which then converged towards a stable value that accurately reflected the workload's true energy consumption. This validates the system's ability to self-correct and adapt to specific, recurring jobs.

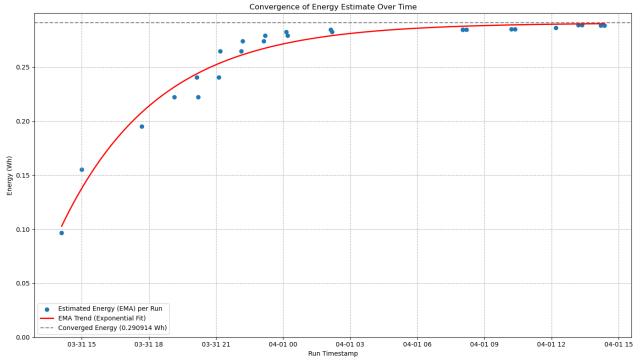

Figure 4: Convergence between the estimated and measured energy consumption for the recurring PoC job. The EMA feedback loop rapidly reduces the initial error, converging on an accurate estimate.

Scheduling Decisions. In the live experiment, the scheduler consistently determined the optimal action was to run the job immediately, with no delay. This outcome is a direct consequence of the cost model (Equation 6) and the specific workload parameters. The synthetic job had a short duration (3 minutes) and therefore very low absolute energy consumption. Even with a high carbon price (π = \$185), the resulting monetary carbon cost (C_{carbon}) was a fraction of a cent. This value was several orders of magnitude smaller than the compute cost ($p_r \cdot \delta$) and the fluctuations in carbon cost over the scheduling window were financially negligible. As a result, the optimizer found no economic benefit in delaying the job. This finding corroborates the "Apparent Negligibility" limitation discussed in section 7 and highlights that for temporal shifting to be triggered for short jobs, the carbon price π would need to be exceptionally high or the job's energy consumption significantly larger.

6 Discussion

The evaluation on historical CI/CD workloads reveals a compelling synergy between environmental sustainability and financial optimization. By dynamically navigating the spatio-temporal search space of cloud compute provisioning, the methodology demonstrates that aggressive carbon reduction does not inherently require a "green premium."

A critical finding from our large-scale simulation is that raw server costs overwhelmingly dominate the total cost equation (C_{total}). The hourly cost disparity between cloud regions is substantial—for example, the `e2-standard-2` instance costs \$0.0670/hour in US West (Oregon) compared to \$0.1064/hour in Central Brazil. This baseline financial delta far exceeds the monetary penalty of the associated carbon emissions, even when aggressively modeling the Social Cost of Carbon at \$185 per tonne [12], and even when accounting for regional PUE differences (e.g., Google Cloud's 1.10 fleet average versus AWS's 1.135) and varying hardware power profiles.

Because of this severe financial imbalance, the scheduler fundamentally acts as a cost-minimization engine that primarily selects locations based on the lowest compute price. The carbon pricing mechanism, therefore, serves a secondary role: it primarily optimizes the *execution time* within that pre-selected, cheapest region. This can be seen by the identical compute savings in Table **??**, indicating the same region is always chosen.

Furthermore, our sensitivity analysis reveals that artificially increasing the cost of carbon (π) does not meaningfully alter the selected locations. The variance in compute cost is simply too large for the carbon penalty to override. From a practical standpoint, cheaper locations will consistently be preferred by the optimization algorithm. Consequently, for spatial carbon reduction to be truly transformative, organizations cannot rely solely on the scheduler's financial optimizer; they must intentionally and strategically provision environments in regions that are natively green.

Importantly, the relationship between low-cost and low-emission regions are not necessarily conflicting. In some cases, both can coincide: regions with abundant renewable energy can offer both low electricity prices and low carbon intensity. However, many low-cost regions get their pricing advantage from fossil fuel-based generation or favorable market dynamics unrelated to carbon intensity, leading to higher emissions despite the lower cost. Conversely, cleaner regions may have higher compute costs due to infrastructure constraints, or supply and demand. This partial but inconsistent overlap explains why carbon pricing alone cannot make the optimizer a sustainable solution. Instead, effectively lowering emissions requires a curated candidate pool in which economic and environmental incentives are better aligned. Within this pool, GreenestRoute can further minimize cost and emissions. In this sense, the scheduler is not necessarily a mechanism to be more sustainable in arbitrary markets, but an amplifier of sustainability when it's already structurally accessible.

7 Limitations and Threats to Validity

Static, single-location model. GreenestRoute assumes a job runs entirely at one location. Workloads that could be split, migrated mid-execution, or parallelised across regions cannot currently be optimised. This might lead to an underestimation of cost savings.

Carbon intensity estimation. The carbon intensity in different regions is forecasted by the data source, in this case Electricity Maps. In the evaluation we use historical data, not predictions, giving an upper bound on savings. This weakens the validation in the context of real-time use.

Energy estimation uncertainty. The historical simulation relies exclusively on the static heuristic model(Equations 1–2) to estimate each job's energy consumption E, since live measurement via `eco-ci` is not possible for historical data. This heuristic assigns CPU utilisation U based on keyword matching against the job name and assumes a fixed hardware power profile, neither of which is empirically validated against real runner measurements at scale. Because E enters the carbon cost linearly(Equation 4), a systematic under- or over-estimate in Upropagates directly into C_{carbon} and therefore into the absolute carbon figures reported in Section 5. If the static model underestimates E by an order of magnitude, which could happen for bursty build workloads, the true carbon

cost would be correspondingly larger, potentially altering the scheduler's location preference in regions where price and compute are priced similarly. The dynamic feedback loop(Section 3.4) mitigates this for recurring live jobs,but offers no retrospective correction for the simulation results.

Apparent negligibility of carbon cost. The carbon cost component C_{carbon} appears almost negligible relative to compute cost$p_r \cdot \delta$ throughout the historical evaluation. This was directly confirmed by our live PoC (§5), where the scheduler opted for immediate execution because the monetized carbon savings were too small to be financially meaningful for a short workload. Several factors compound to produce this outcome. First, many CI/CD jobs are short-lived, so absolute energy consumption per job is small. Second, the SCC of$185/t$CO_2$e, while high relative to market carbon prices,still yields a monetary carbon penalty on the order of fractions of a cent per job. Together, these factors mean that the simulation is best interpreted as a lower bound on carbon cost sensitivity: a more accurate energy model applied to longer or more compute-intensive workloads would yield a larger C_{carbon} signal.

Platform Constraints of the PoC. The live proof-of-concept was subject to limitations of the GitHub Actions platform. The native 'schedule' cron was found to be unreliable for frequent triggers, necessitating a workaround using a local scheduler. More significantly, GitHub-hosted runners do not allow for the selection of a specific geographic region. Consequently, our PoC could only validate *temporal shifting* within a single, default region and could not test the *spatial routing* aspect of the algorithm in a live environment. The validation of spatio-temporal optimisation is therefore confined to the historical simulation.

Evaluation scope. Results are based on GitHub Actions workloads under specific location and hardware assumptions; generalisability to other contexts is not yet established. The PoC in particular used a single, synthetic workload, and its results may not transfer to other workload types without further study.

8 Future Work

Multi-region migration would allow dispatching to an alternative region if grid conditions change favorably elsewhere before the scheduled execution begins.

Package as a Reusable Action. The existing GitHub Actions workflows could be packaged into a single, reusable action or workflow to drastically lower the adoption friction for other repositories.

Broader workload analysis on code builds, scientific computing, and data engineering jobs would characterise savings across different job types and strengthen generalisability. More compute-intensive workloads may also allow carbon cost to play a bigger role in the decision making process.

Live carbon pricing integration through a commercial data provider would provide a real-world component in the cost signal, tying scheduling decisions directly to real-time market dynamics. Alternatively, web scraping approaches could be explored for a free option at the cost of complexity.

Advanced scheduling algorithms. While brute-force search is sufficient for now, exploring more advanced optimisation techniques could support scheduling a large number of interdependent jobs or incorporating more complex constraints.

9 Conclusion

Reducing ICT emissions does not strictly require new hardware or redesigned infrastructure; it requires scheduling existing workloads intelligently. GreenestRoute demonstrates that jointly optimising over location and time—and expressing the result as a tangible financial metric—makes carbon-aware scheduling both technically tractable and economically compelling. While prior literature establishes that temporal shifting alone can cut emissions by 31–70% [4, 5] and spatial routing adds up to 66% in further gains [6], combining them unlocks unprecedented efficiency.

However, the findings also suggest an important limitation. The optimisation is mostly driven by regional price differences, with carbon cost playing a smaller role. As a result, GreenestRoute is most effective when organisations already operate in regions where price and carbon cost align well. In this context, the scheduler acts as a way to amplify existing sustainability options rather than a standalone solution.

Our evaluation of 65,145 real-world GitHub Actions workloads validates this hybrid approach at scale. Given a globally distributed candidate pool and a 24-hour delay window, GreenestRoute reduced CO_2 emissions by over 91% and simultaneously cut total compute costs by over 11% compared to immediate execution. Even in highly constrained, single-region scenarios, purely temporal shifting secured up to a 6% emission reduction.

Running a job in the right place at the right time is systematically cheaper and greener. By making this automatic, GreenestRoute provides a practical, drop-in solution to sustainably scale the next generation of cloud computing.

References

[1] Green Software Foundation. Carbon aware SDK. https://github.com/Green-Software-Foundation/carbon-aware-sdk, 2022. Accessed: April 2, 2026.

[2] International Carbon Action Partnership. Allowance Price Explorer, 2026.

[3] International Telecommunication Union (ITU) and World Bank. Measuring the emissions & energy footprint of the ICT sector: Implications for climate action. Technical report, ITU and World Bank, 2024. Geneva and Washington, D.C.

[4] Henrik Claßen, Jonas Thierfeldt, Julian Tochman-Szewc, Philipp Wiesner, and Odej Kao. Carbon-awareness in CI/CD. In *Proceedings of the 1st International Workshop on Sustainable Service-Oriented Computing (SSCOPE) at ICSOC '23*. Springer, 2023. arXiv preprint: https://arxiv.org/abs/2310.18718.

[5] A. Souza et al. CASPER: Carbon-aware scheduling and provisioning for distributed web services, 2024.

[6] Carbon-aware temporal data transfer scheduling across cloud datacenters, 2025.

[7] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In *Proceedings of the 22nd International Middleware Conference*, pages 260–272. ACM, 2021.

[8] Green Software Foundation. Carbon-aware computing whitepaper - how UBS succeeded in measuring and reducing carbon emissions of their core risk platform. https://greensoftware.foundation/articles/carbon-aware-computing-whitepaper-how-ubs-succeeded-in-measuring-and-reducing-car/, 2023. Accessed: April 2, 2026.

[9] J. You et al. Zeus: Understanding and optimizing GPU energy consumption of deep learning. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[10] Google Data Centers. Efficiency: Power usage effectiveness (pue), 2026. Accessed: 2026-03-30.

[11] green-coding-solutions. Eco ci: Energy estimation for github actions, gitlab and jenkins. https://github.com/green-coding-solutions/eco-ci-energy-estimation, 2026.

[12] Kevin Rennert, Frank Errickson, Brian C. Prest, Richard G. Newell, William Pizer, Cora Kingdon, Jordan Wingenroth, Roger Cooke, Bryan Parthum, David Smith, et al. Comprehensive evidence implies a higher social cost of CO2. *Nature*, 610(7933):687–692, 2022.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron

Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.