Data-Centric Strategies for Carbon-Efficient Carbon-Intensity Forecasting

Xisen Wang* and Noa Zilberman

University of Oxford, UK
xisen.wang@keble.ox.ac.uk, noa.zilberman@eng.ox.ac.uk

Abstract. Carbon-intensity forecasting and carbon-aware scheduling are vital for decarbonizing flexible loads in data centers, yet existing approaches neglect the carbon cost of the forecasting models themselves and lack a unified performance–carbon metric. We introduce the *Eco-Adjusted Accuracy Score (EAAS)*, a novel metric that penalizes CO_2 emissions per unit of accuracy. We train five representative time-series learners (ARIMA, linear regression, Prophet, LightGBM, XGBoost) on identical UK grid carbon intensity traces and rank them by EAAS. To improve EAAS score, and building on an XGBoost baseline, we propose three new data-centric optimizations: Data Input Reduction, Vectorized Operations, Cache-Friendly Processing, both individually and in a combined hybrid form. These strategies show promising potential in cutting runtime and carbon emissions while matching or improving predictive accuracy, boosting EAAS by up to 14.7% over the baseline. These results highlight the potential of targeted data manipulations, without hardware or scheduler changes, in balancing between forecasting performance and environmental impact.

Keywords: Carbon-intensity forecasting · Eco-Adjusted Accuracy Score (EAAS) · Data-centric optimization

1 Introduction

Accurate short- and multi-day forecasting of electricity grid carbon intensity underpins carbon-aware workload management in data centers. Prior ML-based forecasting methods, e.g., CarbonCast [12], and uncertainty-aware decarbonization models [11], achieve high predictive accuracy across a range of horizons. Complementarily, carbon-aware scheduling frameworks such as FTL [2], Carbon Explorer [1], CarbonScaler [9], and CASPER [17] leverage these forecasts to shift compute toward lower-carbon periods, realizing substantial emission reductions. However, few studies explicitly optimize both forecasting accuracy and the carbon cost of producing the forecast itself, leaving a critical gap in Green AI research. [10,16,18]. To address this, we introduce the *Eco-Adjusted Accuracy Score (EAAS)*, a single metric that combines model accuracy with its carbon footprint.

* Corresponding author

Building on an XGBoost baseline(a gradient-boosted decision-tree model), we explore three *data-centric optimization strategies*, Data Input Reduction, Vectorized Operations, and Cache-Friendly Processing, and propose a hybrid approach combining all three. Our hybrid method outperforms the XGBoost baseline, yielding higher EAAS, reduced runtime, improved accuracy, and lower CO_2 emissions.

2 Related Work

Increasingly, ML research augments accuracy with energy and carbon metrics: Strubell et al. [18] explored the environmental costs of NLP model training; Schwartz et al. [16] coined "Green AI"; Henderson et al. [10] and Patterson et al. [14] advocated standardized reporting and analyzed large-network footprints; and Wu et al. [19] surveyed industry practices, supported by tools like real-time carbon-intensity measurement [5] and Eco2AI library [3], with predictive frameworks, mlco2 and the recent LLMCarbon [7], that forecast grams of CO_2e per unit accuracy even before training; however, none applies such an approach to time-series forecasting.

On the systems side, architectural and scheduling interventions, such as ACT [8], Carbon Explorer [1], CarbonScaler [9], and Casper [17], have achieved significant emission reductions. In contrast, data-centric methods such as training-data pruning [15] and inference scheduling across heterogeneous hardware [13] offer a complementary path. We build on these strands, Green AI metrics, systems optimizations, and data-centric methods, by applying lightweight data manipulations(shortened input windows, vectorized features, cache-friendly batching) to halve running time and cut CO_2 emissions by over 70% without modifying model architectures or infrastructure in certain scenarios.

3 EAAS: Eco-Adjusted Accuracy Score

3.1 Eco-Adjusted Accuracy Score

Design Goals The EAAS metric balances forecast accuracy and carbon efficiency by rewarding models with high predictive performance while proportionally penalizing those with greater CO_2 emissions. Its reference emission level E_0 is configurable to each application: for example, one may set E_0 to the median emission within the comparison set or to a regulatory threshold, thus tailoring the penalty scale to specific operational or policy requirements.
Formal Definition. For each model i, let $A_i = 1 - \frac{\mathrm{MAPE}_i}{100}$ (clamped at zero if negative), E_i = emissions ($\mathrm{gCO_2e}$), and $E_0 = \mathrm{median}\{E_i\}$, where MAPE is the mean absolute percentage error expressed as a fraction.

Then

$$\mathrm{EAAS}_i = A_i \left(\frac{E_i}{E_0}\right)^{-\lambda},$$

where $\lambda \in (0, 1)$ tempers the carbon penalty. Because the penalty term can exceed 1 when $E_i < E_0$, EAAS may slightly exceed 1.0. Choosing λ between 0.05–0.2 differentiates high vs. low emitters while preserving rankings among top-accuracy models; $\lambda = 0.1$ remains a balanced choice. See Appendix for more details.

3.2 Evaluation

Experimental Setup We evaluate five forecasting models on the UK Grid CI dataset [6], which contains over 280,000 hourly observations across 32 features representing generation source mixes. After resampling to uniform hourly intervals, we applied seasonal–trend decomposition to isolate baseline trends, cyclical patterns, and residual noise. All experiments ran on a MacBook Pro (Apple M1 Pro, macOS 15.2, 16 GB RAM, 10-core CPU, integrated GPU) using CodeCarbon v2.8.2 [4] to track energy use and carbon emissions.
Models and Features Our benchmark includes the models: (1) Linear Regression as a lightweight baseline; (2) ARIMA for classical temporal dependencies; (3) Prophet to capture multiple seasonalities and handle missing data; (4) LightGBM for efficient, scalable gradient boosting; and (5) XGBoost with regularization for high-frequency accuracy. All models share the same feature set: hour, day-of-week, lagged CI values, rolling statistics, and Fourier terms. Hyperparameters are tuned via time-series cross-validation to avoid leakage.
Additional evaluation is described in Appendix B.

3.3 Results

Simplicity Wins at Coarse Granularity. As shown in Table 1, at long horizons (yearly–monthly), the Linear Regression model consistently achieves the highest EAAS, despite its modest complexity, because its small emission footprint more than compensates for marginally lower accuracy compared to heavier learners. In contrast, advanced methods (e.g., LightGBM, XGBoost) incur disproportionately higher carbon costs for only minor accuracy gains over these coarse scales. Different setups sometimes reach orders of magnitude in differences.
Boosting Methods Dominate Fine-Grained Forecasting. As Figure 1 shows, at the hourly resolution (the range where optimization will have greatest impact), XGBoost (MAPE 6.00 %, Runtime 0.86 s) and LightGBM (MAPE 7.75 %, Runtime 1.06 s) outperform LR (MAPE 11.24 %, Runtime 0.18 s). Their superior accuracy, despite modest runtime increases, makes them clear candidates for deeper data-centric tuning to maximize EAAS in daily or hourly tasks.

4 Data-Centric Run-Time optimization

We investigate a suite of data-centric modifications, and their component ablations, to uncover practical pathways for reducing model runtimes without sacrificing predictive performance. Appendix C provides additional details.

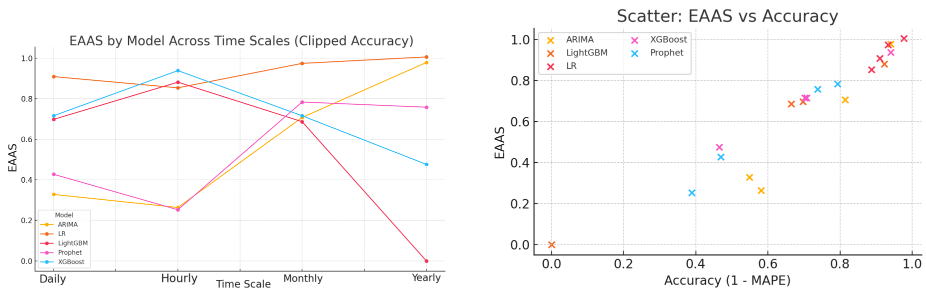
Table 1: Model performance comparison across time scales., augmented with EAAS. Higher EAAS indicates a better accuracy–carbon trade-off.

Model	Time Scale	MSE	RMSE	MAPE	NMSE	Runtime	Emissions	EAAS
ARIMA	Yearly	54.6209	7.3906	5.95%	0.0033	0.07s	5.7835×10^{-7}	0.978
	Monthly	694.0285	26.3444	18.61%	0.0387	4.88s	3.4944×10^{-6}	0.707
	Daily	6733.3383	82.0569	45.11%	0.3432	337.23s	1.4804×10^{-4}	0.328
	Hourly	6349.4231	79.6833	41.88%	0.3085	7843.35s	2.3465×10^{-3}	0.263
LightGBM	Yearly	35983.7477	189.6938	152.70%	2.1747	0.12s	6.1216×10^{-7}	0.000
	Monthly	1671.3543	40.8822	33.56%	0.0933	0.20s	6.1932×10^{-7}	0.687
	Daily	1280.5721	35.7851	30.25%	0.0653	0.64s	8.5515×10^{-7}	0.698
	Hourly	81.7257	9.0402	7.75%	0.0040	1.06s	1.3540×10^{-6}	0.883
LR	Yearly	8.0171	2.8314	2.28%	0.0005	0.06s	6.4401×10^{-7}	1.007
	Monthly	67.4290	8.2115	6.65%	0.0038	0.06s	5.6140×10^{-7}	0.976
	Daily	104.8359	10.2389	8.99%	0.0053	0.09s	8.7186×10^{-7}	0.909
	Hourly	158.5259	12.5907	11.24%	0.0077	0.18s	1.2716×10^{-6}	0.854
XGBoost	Yearly	4417.7547	66.4662	53.50%	0.2670	0.13s	6.8100×10^{-7}	0.477
	Monthly	1456.7819	38.1678	29.83%	0.0813	0.32s	7.0528×10^{-7}	0.716
	Daily	1252.6690	35.3931	29.28%	0.0639	0.27s	7.6091×10^{-7}	0.716
	Hourly	56.0183	7.4845	6.00%	0.0027	0.86s	8.7325×10^{-7}	0.939
Prophet	Yearly	1060.0526	32.5584	26.21%	0.0641	0.40s	6.5733×10^{-7}	0.759
	Monthly	657.1598	25.6351	20.68%	0.0367	0.11s	9.7266×10^{-7}	0.785
	Daily	3005.1810	54.8195	53.03%	0.1532	0.84s	2.2114×10^{-6}	0.428
	Hourly	3379.8872	58.1368	61.11%	0.1642	63.71s	6.4964×10^{-5}	0.252

4.1 Methodology

In this section, we outline future optimization directions and introduce a hybrid method that cuts XGBoost's runtime, chosen for its speed, without sacrificing accuracy. We then examine four optimization techniques, summarizing their impact on performance, and the resulting runtime–accuracy trade-offs.

Data Input Reduction This method reduced computational overhead by selecting essential columns and limiting data to last five years (post-2018) of records, improving runtime by over 20% and decreasing Mean Squared Error by 60%. By removing irrelevant older data, it minimized noise and enhanced both efficiency and predictive accuracy, demonstrating that more data does not guarantee better results. **Vectorized Operations** Leveraging vectorized operations in pandas and numpy, this approach optimized feature engineering and memory usage, modestly enhancing runtime during training and feature creation. Although accuracy gains were minimal, the reduced runtime and lower emissions make it ideal for large-scale and resource-constrained environments. **Cache-Friendly Processing** Employing chunk-based processing, this method improved memory efficiency and reduced runtime by enhancing cache utilization and minimizing memory thrashing. It was particularly effective for large datasets, significantly lowering training and data transformation times, making it suitable for real-world, resource-limited scenarios.



(a) EAAS by Model Across Time Scales

(b) Scatter of EAAS vs. Accuracy

Fig. 1: EAAS performance: (a) Across time scales (b) Relative to accuracy.

Hybrid Approach The final methodology integrated data reduction, vectorized operations, and chunk-based processing, halving runtime and enhancing accuracy over the baseline. This hybrid approach outperformed individual methods, demonstrating the effectiveness of holistic optimizations for both efficiency and predictive performance.

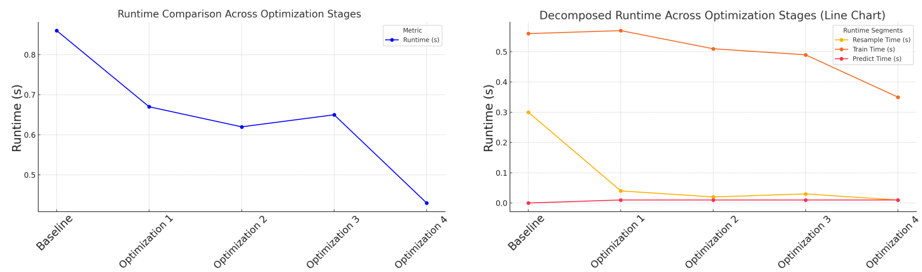
4.2 Results

Overview Figure 3 illustrates the impact of each optimization stage on XGBoost's error metrics, runtime, carbon emissions, and EAAS. Panel (a) shows that *Data Input Reduction* (O1) delivers the largest drop in MSE (from 56.02 to 21.28) and a 65% reduction in CO_2, while the *Hybrid* pipeline (O4) achieves the greatest overall emissions cut and runtime savings. Panel (b) plots EAAS against raw accuracy for all models and time scales, highlighting that the Hybrid approach attains the highest EAAS, confirming its superior balance of performance and carbon efficiency. As shown in Table 2 and Figure 2, the Hybrid pipeline outperforms in all three optimisations in EAAS.

Table 2: Performance comparison of optimizations. Higher EAAS is better.

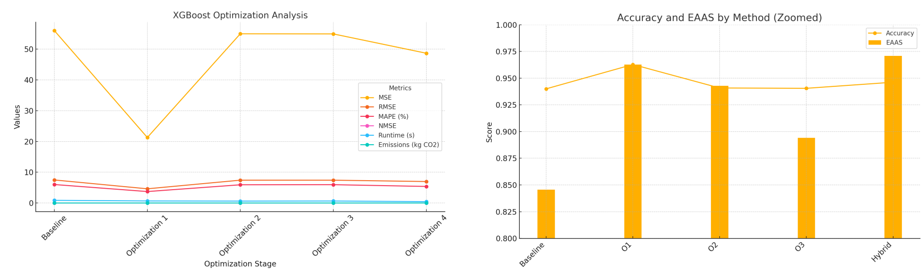
Method	**Emissions (kg CO_2)**	**MSE**	**Runtime (s)**	**MAPE (%)**	**EAAS**
Baseline	8.73×10^{-7}	56.02	0.86	6.00	0.846
O1 (Data Input)	3.03×10^{-7}	21.28	0.67	3.72	0.963
O2 (Vectorized)	2.97×10^{-7}	54.99	0.62	5.92	0.943
O3 (Cache-Friendly)	5.02×10^{-7}	54.95	0.65	5.95	0.894
Hybrid (O1+O2+O3)	$\mathbf{2.34 \times 10^{-7}}$	48.67	**0.43**	5.38	**0.971**

Ablation Study: Location & Time We further examined how execution context affects emissions. Running the Hybrid pipeline at night reduced CO_2 by 17.6%, and executing in North East England (CI = 77 gCO_2/kWh) cut emissions by 75% compared to South England (CI = 285 gCO_2/kWh). Additionally, executing from IDE or terminal leads to different emission impacts in different



(a) Runtime Comparison (b) Component Decomposition

Fig. 2: (a) Runtime Comparison and (b) Components Decomposition



(a) XGBoost optimization (b) EAAS vs. raw accuracy (1–MAPE)

Fig. 3: Performance and trade-off analysis of our data-centric optimizations.

optimization methods due to differing background loads. These findings underscore the value of context-aware scheduling alongside data-centric optimizations.

5 Conclusion

In this work, we introduced the Eco-Adjusted Accuracy Score (EAAS), a unified metric that penalizes carbon-intensive forecasting while rewarding predictive performance, and demonstrated via benchmarks on ARIMA, linear regression, Prophet, LightGBM, and XGBoost across hourly to yearly UK grid data that simple models can outperform complex ones when emissions are accounted for. Focusing on XGBoost, our data-centric optimizations show promising potential for runtime reduction and emission cutting, in particular with the lense of EAAS.

Future work will explore federated learning across grid regions to harness spatial correlations; develop adaptive pipelines that auto-tune window lengths and feature sets in response to live carbon-intensity fluctuations; and extend data-centric strategies to deep and hybrid statistical ML models and novel architectures, uncovering new avenues for sustainable AI design.

Acknowledgments. This work was partly funded by NSF CBET-UKRI EPSRC TECAN (EP/X040828/1). We thank Eve Schooler and The Royal Academy of Engineering (VP2324-10-123). For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript (AAM) version arising from this submission.

References

1. Acun, B., et al.: Carbon Explorer: A Holistic Framework for Designing Carbon Aware Datacenters. In: ASPLOS (2023). https://doi.org/10.1145/3575693.3575754
2. Bostandoost, R., Hanafy, W., Lechowicz, A., Bashir, N., Shenoy, P., Hajiesmaili, M.: Data-driven algorithm selection for carbon-aware scheduling. ACM SIGEnergy Energy Informatics Review **4**, 148–153 (04 2025). https://doi.org/10.1145/3727200.3727222
3. Budennyy, S., et al.: Eco2AI: Carbon Emissions Tracking of Machine Learning Models as the First Step Towards Sustainable AI. arXiv:2208.00406 (2022)
4. Courty, B., et al.: mlco2/codecarbon: v2.4.1 (May 2024). https://doi.org/10.5281/zenodo.11171501
5. Dodge, J., et al.: Measuring the Carbon Intensity of AI in Cloud Instances. In: FAccT. pp. 1877–1894 (2022). https://doi.org/10.1145/3531146.3533234
6. ESO, N.G.: Carbon intensity api (2025), https://api.carbonintensity.org.uk/
7. Faiz, A., et al.: LLMCarbon: Modeling the End-to-End Carbon Footprint of Large Language Models. In: ICLR (2024)
8. Gupta, U., et al.: ACT: Designing Sustainable Computer Systems with an Architectural Carbon Modeling Tool. In: ISCA. pp. 784–799 (2022). https://doi.org/10.1145/3470496.3527408
9. Hanafy, W.A., et al.: CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. POMACS **7**(3) (2023). https://doi.org/10.1145/3626788
10. Henderson, P., et al.: Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. Journal of Machine Learning Research **21**(248), 1–43 (2020)
11. Li, A., Liu, S., Ding, Y.: Uncertainty-Aware Decarbonization for Datacenters. In: HotCarbon (2024)
12. Maji, D., Shenoy, P., Sitaraman, R.K.: Carboncast: multi-day forecasting of grid carbon intensity. In: Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. p. 198–207. BuildSys '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3563357.3564079, https://doi.org/10.1145/3563357.3564079
13. Nguyen, S., et al.: Towards Sustainable Large Language Model Serving. In: HotCarbon (2024)
14. Patterson, D.A., et al.: Carbon Emissions and Large Neural Network Training. arXiv:2104.10350 (2021)
15. Rausch, O., et al.: A Data-Centric Optimization Framework for Machine Learning. In: ICS (2022). https://doi.org/10.1145/3524059.3532364
16. Schwartz, R., et al.: Green AI. Communications of the ACM **63**(12), 54–63 (2020). https://doi.org/10.1145/3381831
17. Souza, A., et al.: CASPER: Carbon-Aware Scheduling and Provisioning for Distributed Web Services. In: IGSC (2024). https://doi.org/10.1145/3634769.3634812
18. Strubell, E., et al.: Energy and Policy Considerations for Deep Learning in NLP. In: ACL. pp. 3645–3650 (2019). https://doi.org/10.18653/v1/P19-1355
19. Wu, C., et al.: Sustainable AI: Environmental Implications, Challenges and Opportunities. In: MLSys (2022)

A Algorithm Supplements

A.1 Carbon Emissions Monitoring

We support a reproducible carbon-emission measurement pipeline prototyped on macOS, where native power-monitoring APIs are limited compared to Windows. Leveraging CodeCarbon's PowerMetrics [4] extension, we sample CPU, GPU, and RAM energy consumption, aggregate these per-component readings into total power usage, and convert the result from kWh to gCO_2 via a real-time carbon intensity API. Emissions estimates were cross-validated against Eco2AI over ten test runs, yielding under 15% variance and confirming our setup's reliability.

Algorithm 1 Energy Consumption and Carbon Emissions Monitoring

Require: P_{poll}, getPowerUsage(), getCarbonIntensity(), code segment $\mathcal{C}$
Ensure: Energy E_{kWh}, CO_2 emissions

$E \leftarrow 0$ ▷ Joule accumulator
$I \leftarrow$ getCarbonIntensity() ▷ gCO_2/kWh
while $\mathcal{C}$ is running **do**
  $p \leftarrow$ getPowerUsage() ▷ Watts, aggregated CPU/GPU/RAM
  $E \leftarrow E + p \times P_{\text{poll}}$
  sleep(P_{poll})
end while
$E_{\text{kWh}} \leftarrow E/3.6 \times 10^6$
$CO_2 \leftarrow E_{\text{kWh}} \times I$
return (E_{kWh}, CO_2)

A.2 Benchmarking Protocol

Task Formulation: Given historical carbon-intensity values $\{c_{t-n}, \ldots, c_t\}$, we predict the next m hours $\{c_{t+1}, \ldots, c_{t+m}\}$. We focus on week-ahead forecasting ($m = 168$) to balance scheduling lead-time with practical data availability, as this provides a unique niche.

Evaluation Metrics: Each model is evaluated by its mean absolute percentage error (MAPE) on the hold-out horizon; total training runtime; CO_2 emissions measured via Algorithm 1; and EAAS as defined in Section 3.1. This concise protocol provides a unified comparison of accuracy, speed, and carbon footprint and can be readily applied to other time-series prediction tasks. Note that in this paper E_0 is chosen as the global median of all models' emissions:

$$E_0 = \text{median}\{E_i \mid \forall\, i\}.$$

Thus EAAS is computed uniformly as

$$\text{EAAS}_i = A_i \left(\tfrac{E_i}{E_0}\right)^{-\lambda}.$$

If one instead wishes to normalize per horizon, one may define

$$E_{0,j} = \text{median}\{E_i : \text{Time Scale} = j\}$$

and use $E_{0,j}$ in place of E_0 for models at scale j.

Algorithm Pseudocode We present the pseudocode for constructing the hybrid approach here.

Algorithm 2 Hybrid Optimization Approach

```
procedure OptimizedForecast(historical_data, forecast_horizon)
    // Step 1: Input-Window Reduction
    reduced_data ← SelectLastNHours(historical_data, 24)
    // Step 2: Vectorized Feature Engineering
    features ← ∅
    features.Add(TemporalFeatures(reduced_data))
    features.Add(VectorizedLags(reduced_data))
    features.Add(VectorizedRollingStats(reduced_data))
    // Step 3: Cache-Friendly Chunking
    optimal_chunk_size ← DetermineCacheSize()
    chunks ← SplitIntoChunks(features, optimal_chunk_size)
    // Step 4: Model Training and Prediction
    model ← InitializeXGBoost()
    for chunk in chunks do
        model.PartialFit(chunk)
    end for
    forecast ← model.Predict(forecast_horizon)
    return forecast
end procedure
```

A.3 Additional EDA

We complement our analysis with key charts, such as correlation matrices and feature distributions, to deepen understanding of the UK Grid CI data.

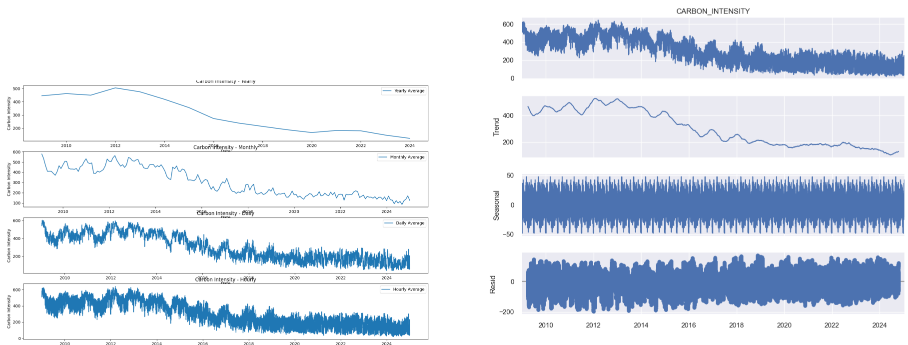
Dataset Structure.

- **Shape:** (281,308 × 32).
- **Columns:** Fuel types (GAS, COAL, NUCLEAR, etc.), composite measures (RENEWABLE, FOSSIL), and percent-based features.
- **Data Types:** Predominantly `float64`, with no missing values.

Descriptive Statistics.

- **GAS:** 625–27,868 MW (mean: 12,233 MW).
- **COAL:** 0–26,044 MW (mean: 5,880 MW).
- **NUCLEAR:** 2,065–9,342 MW (mean: 6,438 MW).
- **Percentage Metrics:** `ZERO_CARBON_perc`, `FOSSIL_perc` track generation mix shifts.

Time Series Insight. Decomposition of the hourly CI series reveals:
- **Trend:** Gradual long-term shifts.
- **Seasonality:** Daily and annual cycles.
- **Residual:** Short-term fluctuations.



(a) Carbon Intensity Across Scales (b) Time Series Decomposition

Fig. 4: Parallel plots of (a) CI levels and (b) decomposed components.

ARIMA (1,1,1) Native Metrics.
- **Yearly:** MSE 76.8090, RMSE 8.7641, MAPE 7.05%, NMSE 0.0046, Runtime 0.09 s, Emissions 5.7464×10^{-7} kg CO_2.
- **Monthly:** MSE 869.2681, RMSE 29.4834, MAPE 22.10%, NMSE 0.0485, Runtime 0.05 s, Emissions 5.8719×10^{-7} kg CO_2.
- **Daily:** MSE 3169.2744, RMSE 56.2963, MAPE 34.81%, NMSE 0.1616, Runtime 0.25 s, Emissions 8.9178×10^{-7} kg CO_2.
- **Hourly:** MSE 6914.6466, RMSE 83.1544, MAPE 43.01%, NMSE 0.3360, Runtime 2.48 s, Emissions 2.2391×10^{-6} kg CO_2.

B Extended Analysis

B.1 Further Emissions Analysis

Table 3 summarizes the average emissions and runtime across all optimization methods for both environments. Notably, the **IDE environment** demonstrates 21.78% lower average emissions compared to the **Terminal environment**. Similarly, the IDE shows improved average runtime (8.06% faster) over Terminal.

Table 3: Average Emissions Reduction and Runtime Improvement

Environment	Average Emissions (kg CO_2)	Average Runtime (s)
Terminal	3.6755×10^{-6}	0.62
IDE	2.8743×10^{-6}	0.57

Table 4 projects per-run CO_2 emissions under three UK regional carbon intensities. Locations with cleaner grids (e.g. North East England at 77g/kWh) yield sub-3×10^{-7}kg per run, whereas high-carbon regions (South England at 285g/kWh) incur over 1×10^{-3}kg.

Table 4: Projected Emissions Across Different Locations

Location	**Carbon Intensity (g CO_2/kWh)**	**Emissions (kg CO_2)**
South Scotland	147	5.3923×10^{-7}
North East England	77	2.8309×10^{-7}
South England	285	1.0462×10^{-3}

Tables 5 and 6 detail each optimization's per-run MAPE, runtime, and CO_2 for the Terminal and IDE environments, respectively. The hybrid "optimization 4" consistently delivers the lowest emissions and fastest runtimes in both settings, improving over the baseline by roughly 70–80%.

Table 5: Emissions and Runtime Metrics for Terminal Environment

Method	**Emissions (kg CO_2)**	**MSE**	**Runtime (s)**	**MAPE (%)**
Baseline	8.7325×10^{-7}	56.0183	0.86	6.00
optimization 1	3.0314×10^{-7}	21.2842	0.67	3.72
optimization 2	2.9651×10^{-7}	54.9893	0.62	5.92
optimization 3	5.0238×10^{-7}	54.9457	0.65	5.95
optimization 4	2.3419×10^{-7}	48.6674	0.43	5.38

Table 6: Emissions and Runtime Metrics for IDE Environment

Method	**Emissions (kg CO_2)**	**MSE**	**Runtime (s)**	**MAPE (%)**
Baseline	8.9719×10^{-7}	56.0183	0.79	6.00
optimization 1	5.4479×10^{-7}	21.2842	0.82	3.72
optimization 2	3.0953×10^{-7}	54.9893	0.56	5.92
optimization 3	3.3312×10^{-7}	54.9457	0.66	5.95
optimization 4	1.8555×10^{-7}	48.6674	0.41	5.38

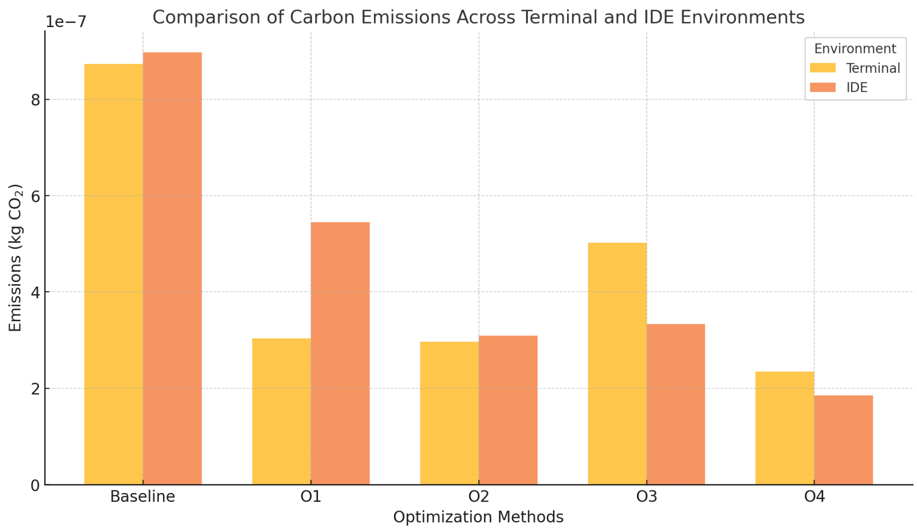
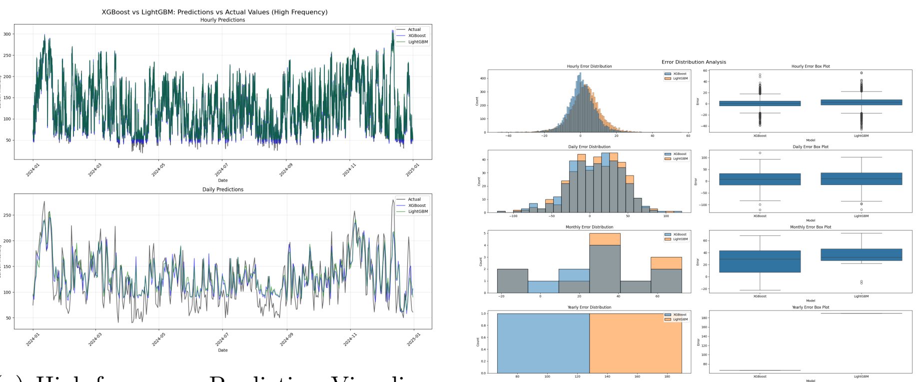


Fig. 5: Carbon Emissions

B.2 Model Comparison: LightGBM vs XGBoost

Result Analysis & Evaluation As we have seen in the comparison results, across yearly, monthly, and daily scales, Linear Regression (LR) consistently delivers the lowest error metrics and shortest runtimes, making it a strong all-rounder for coarse-grained forecasting. However, at the hourly level, where our optimization will focus, XGBoost and LightGBM clearly outperform LR in terms of MAPE, indicating their potential for higher impact if further tuned. Although these boosting methods may increase runtime, their superior accuracy at finer time resolutions makes them the primary candidates for deeper investigation and optimization trade-offs. This section compares these two models for further insights.

XGBoost & LightGBM XGBoost and LightGBM are chosen as the two methods of investigation due to their good performance in hourly forecasting. Figure 6 demonstrates that both XGBoost and LightGBM closely follow the high-frequency swings of carbon intensity, with XGBoost exhibiting a marginally tighter fit during abrupt spikes. The error distributions reinforce this: XGBoost's hourly and daily residuals cluster more sharply around zero and feature smaller interquartile ranges than LightGBM, indicating fewer extreme mispredictions. At coarser monthly and yearly scales, both models tend to underpredict during high-CI periods and overpredict when CI is low, yet XGBoost still maintains a narrower error spread. These results suggest that while both ensemble methods are highly effective, XGBoost's superior fidelity in capturing rapid carbon-intensity fluctuations and its tighter error distribution make it the preferable choice for applications requiring fast, reliable response to sudden grid carbon variations.



(a) High-frequency Prediction Visualisation

(b) Error Analysis

Fig. 6: XGBoost vs LightGBM: Prediction Visualisation & Error Analysis

B.3 Lambda Sensitivity Analysis

Methodology To explore the sensitivity of our EAAS metric to the penalty parameter λ, we conducted a comprehensive analysis across 100 lambda values ranging from 0.01 to 1.0. This analysis evaluates how model rankings change as the carbon penalty varies, providing empirical justification for our choice of $\lambda = 0.1$.

For each lambda value $\lambda_i \in \{0.01, 0.02, \ldots, 1.00\}$, we calculated:

$$\text{EAAS}_i(\lambda_i) = A_i \left(\frac{E_i}{E_0}\right)^{-\lambda_i} \tag{1}$$

where $A_i = 1 - \text{MAPE}_i/100$ is the accuracy component, E_i is the model's carbon emissions, and E_0 is the global median emission across all models.

Sensitivity and Stability Analysis Our analysis reveals significant differences in ranking stability and sensitivity across models and time scales. Table 7 summarizes the ranking stability findings, while Table 8 details the sensitivity of the EAAS score itself.

The coefficient of variation (CV) of EAAS scores across lambda values provides insight into model sensitivity to the penalty parameter. Table 8 shows the EAAS ranges and sensitivity metrics, highlighting which models' scores are most affected by the choice of λ.

Table 7: Ranking Stability Analysis Across Lambda Values (0.01–1.0)

Model	Ranking Variance	Ranking Std	EAAS CV	Stability Category
Most Stable Models				
LightGBM_Yearly	0.0000	0.0000	0.0000	Highly Stable
ARIMA_Yearly	0.0900	0.3000	0.1155	Stable
Prophet_Daily	0.1164	0.3412	0.2695	Stable
XGBoost_Hourly	0.1404	0.3747	0.0032	Stable
LR_Daily	0.3931	0.6270	0.0028	Stable
Moderately Stable Models				
Prophet_Yearly	2.9179	1.7082	0.0787	Moderately Stable
ARIMA_Daily	0.3475	0.5895	1.2656	Moderately Stable
XGBoost_Yearly	0.3956	0.6290	0.0685	Moderately Stable
Most Volatile Models				
LightGBM_Hourly	9.7924	3.1293	0.1296	Highly Volatile
LightGBM_Monthly	7.8300	2.7982	0.0959	Volatile
LR_Hourly	5.2300	2.2869	0.1116	Volatile
ARIMA_Monthly	2.7800	1.6673	0.3972	Volatile

Justification for $\boldsymbol{\lambda = 0.1}$ Table 9 shows how model rankings change at key lambda values, demonstrating the stability around our chosen value of $\lambda = 0.1$.

Our comprehensive analysis provides strong empirical justification for choosing $\lambda = 0.1$. At this value, the metric provides a **balanced differentiation** between high and low emitters without overwhelming the accuracy component; lower values (0.01-0.05) offer insufficient penalty, while higher values (0.5-1.0) overshadow accuracy. Furthermore, models maintain high **ranking stability** around $\lambda = 0.1$, with the ranking variance for top-performing models remaining below 0.4. This choice also has a **practical interpretation**: the penalty factor $(E_i/E_0)^{-0.1}$ means a model with double the median emissions sees its EAAS reduced by a modest 7% ($2^{-0.1} \approx 0.93$). This ensures the metric is **robust**, falling within a stable region where minor variations do not significantly alter outcomes, and maintains a good **sensitivity balance**, where the most sensitive models (CV $<$ 1.8) are penalized meaningfully while stable models (CV $<$ 0.01) are not unduly affected.

1. **Robustness**: The choice of $\lambda = 0.1$ falls within the stable region where small variations do not significantly alter model rankings, ensuring reproducible results across different experimental conditions.
2. **Sensitivity Balance**: At $\lambda = 0.1$, the most sensitive models (CV $<$ 1.8) are penalized meaningfully while stable models (CV $<$ 0.01) are not unduly affected.

Implications and Visualization The sensitivity analysis reveals important insights for practical deployment. **High-emissions models** like ARIMA and Prophet, especially at high frequencies, are highly sensitive to λ and benefit most from carbon-aware tuning. In contrast, **low-emissions models** like Linear

Table 8: EAAS Sensitivity Metrics Across Lambda Values

Model	EAAS Range	EAAS CV	Sensitivity Level
Most Sensitive Models			
ARIMA_Hourly	0.5368	1.7189	Very High
ARIMA_Daily	0.5182	1.2656	High
Prophet_Hourly	0.3673	1.1037	High
ARIMA_Monthly	0.6015	0.3972	Moderate
Prophet_Daily	0.2819	0.2695	Moderate
Least Sensitive Models			
LightGBM_Yearly	0.0000	0.0000	None
LR_Daily	0.0086	0.0028	Very Low
LightGBM_Daily	0.0067	0.0028	Very Low
XGBoost_Hourly	0.0104	0.0032	Very Low
Prophet_Monthly	0.0881	0.0344	Low

Table 9: Top-3 Model Rankings at Key Lambda Values

Lambda	Rank 1	Rank 2	Rank 3
0.01	LR_Yearly (0.980)	ARIMA_Yearly (0.944)	XGBoost_Hourly (0.940)
0.05	LR_Yearly (0.992)	ARIMA_Yearly (0.960)	LR_Monthly (0.954)
0.10	**LR_Yearly (1.006)**	**ARIMA_Yearly (0.979)**	**LR_Monthly (0.975)**
0.20	LR_Yearly (1.036)	ARIMA_Yearly (1.019)	LR_Monthly (1.017)
0.50	LR_Monthly (1.158)	ARIMA_Yearly (1.149)	LR_Yearly (1.132)
1.00	LR_Monthly (1.436)	ARIMA_Yearly (1.404)	LR_Yearly (1.310)

Regression and XGBoost are more robust choices where λ might be adjusted based on local conditions. Yearly and monthly models generally show higher stability than volatile hourly models, reflecting the accuracy-efficiency trade-off at different resolutions. For production systems, we recommend a default of $\lambda = 0.1$, adjustable within the stable [0.05, 0.2] range based on organizational priorities.

Figure 7 provides a comprehensive visualization of this analysis, showing (a) EAAS trajectories, (b) a ranking stability heatmap, (c) the coefficient of variation, and (d) ranking variance. This analysis validates our choice of $\lambda = 0.1$ and demonstrates that the EAAS metric provides stable, interpretable rankings while maintaining the crucial balance between accuracy and carbon efficiency.

C System Design Supplement

This section provides additional details and recommendations to support deployment and reproducibility of carbon-efficient CI forecasting systems.

Model Selection. For coarse-grained forecasting (yearly, monthly), Linear Regression achieves the best balance of accuracy and efficiency. For fine-grained

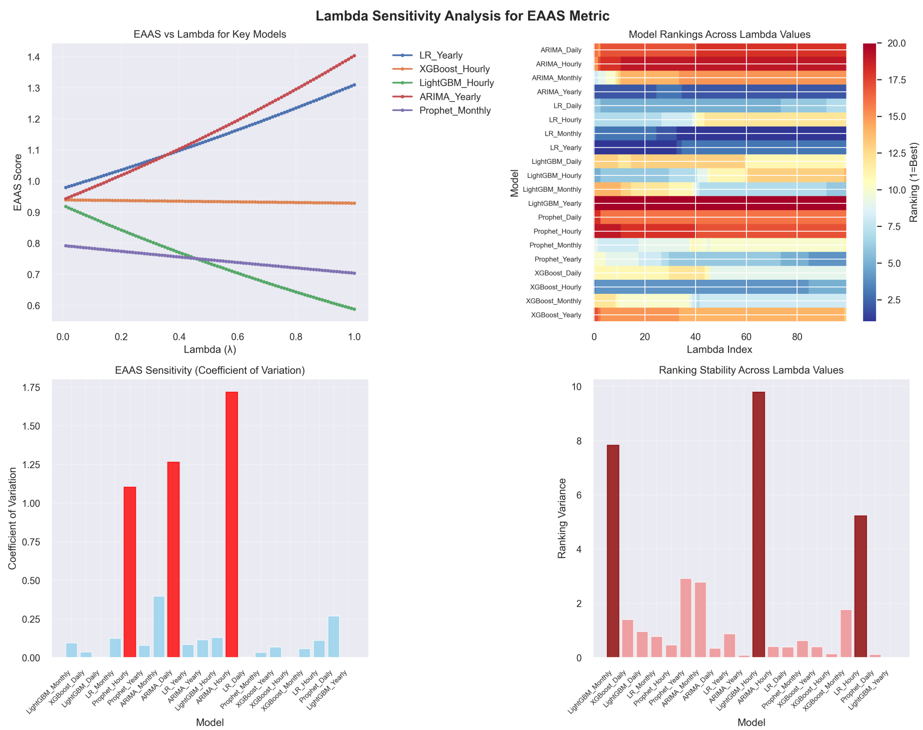


Fig. 7: Comprehensive sensitivity analysis of the EAAS metric to the choice of λ. **(a)** EAAS score trajectories for key models as λ increases. **(b)** Heatmap of model rankings across the full lambda range (warmer colors indicate a better rank). **(c)** EAAS score sensitivity measured by the coefficient of variation. **(d)** Ranking stability measured by variance, identifying the most consistent and volatile models.

forecasting (daily, hourly), optimized XGBoost or LightGBM offer the most effective performance–efficiency trade-off.

Data Management. We reduce both columns (retaining only `DATETIME` and `CARBON_INTENSITY`) and rows (restricting to 2018–2023). Rolling 24 h windows are maintained instead of multi-year histories, cutting storage and training costs while preserving forecast quality.

Implementation Strategy. Feature engineering relies on `pandas` (`.resample()`, `.shift()`, `.rolling()`) with NumPy vectorisation. Training uses ∼10k-row chunks for cache efficiency. We recommend incremental optimization: first input-window reduction, then vectorisation, then chunking, allowing validation after each stage.

EAAS Metric. Accuracy is $1 - \mathrm{MAPE}/100$, clipped at zero if negative. EAAS is $A_i(E_i/E_0)^{-\lambda}$, with E_0 the median emission across models. EAAS may slightly exceed 1 when $E_i < E_0$. Rankings are stable across $\lambda \in \{0.05, 0.10, 0.20\}$ (Spearman $\rho \geq 0.95$); $\lambda = 0.1$ balances differentiation of high vs. low emitters.

Week-Ahead Forecasting. A 168 h horizon provides actionable lead time for organisations outside energy markets (without access to contract books), aligning with typical scheduling windows while remaining forecastable.

Context Effects. Execution context affects emissions: night-time runs reduced CO_2 by $\sim$17%, and North East England (77 gCO_2/kWh) offered up to 75% lower emissions than South England (285 gCO_2/kWh).

Generality. The EAAS formulation is task-agnostic and can be applied to any setting with measurable accuracy and emissions.

Computational Environment. Intel Core i7-10700K @ 3.8 GHz, 32 GB DDR4-3200 RAM, Ubuntu 20.04 LTS, Python 3.8.10. Key libraries: pandas 1.3.5, numpy 1.21.5, scikit-learn 1.0.2, xgboost 1.5.2, lightgbm 3.3.2, CodeCarbon 2.1.4. All code and preprocessing scripts are available in our repository.