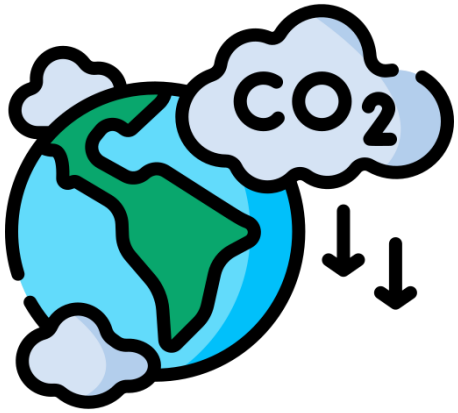


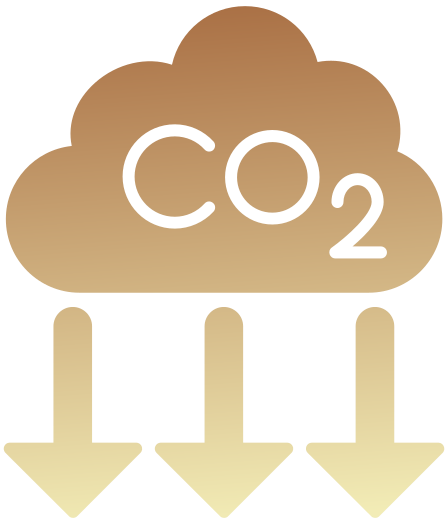
Estimation & Simulation of the CO_2 Impact of LLM Queries

Yasmine MAARBANI, Aya BOUANANE,
Yannis BRIK, Sami CHELLIA

Table of contents

1. Context

The project consists of designing a complete solution that combines a user interface and an artificial intelligence model capable of estimating the carbon footprint (CO_2e) of a request sent to a large language model (LLM). It includes **an analysis of energy consumption to determine the energy cost per token or per request**. Based on these data, our model estimates the energy consumed according to various parameters such as the query length, its complexity, and the type of model used. This consumption is then converted into CO_2 equivalent ($kgCO_2e$) based on the energy mix of a specific region. Finally, a web interface allows users to visualize, compare, and simulate the environmental impact of their queries, thus providing a concrete understanding of the carbon footprint of artificial intelligence.



2. Problem definition

Our objective is to estimate and simulate the environmental impact of LLM inference at request time. To achieve this, several steps are required:

- Develop a predictive model for energy consumption per request (kWh/request) and another for energy consumption per token (kWh/token).
- Convert energy usage to CO_2e emissions (kg) using a configurable grid carbon intensity.
- Create an interactive web interface to visualize, simulate, and compare impacts for individual queries, multi-turn conversations, and batch requests. 

The following key questions guide our study and help structure the analysis:

- How does energy depend on the number of tokens and other operational features?
- How do predictions change under different grid carbon intensities?
- How to provide fast, reproducible estimates for practical decision-making (e.g., limiting response length)?

2.1 Dataset definition

The dataset used for this study, llm_inference_energy_consumption_final.csv, contains detailed measurements of LLM inference processes. The following summarizes the core columns used (or derived) in the modeling, as well as the excluded variables to prevent data leakage.Core columns used (or derived) in modeling: 

- Tokens and lengths:
  - prompt_token_length, response_token_length, total_tokens (= prompt + response)
- Timings:
  - total_duration_s, prompt_duration_s, response_duration_s, load_duration_s
- System/context:
  - model_name, type (categorical)
- Target(s):
  - energy_consumption_llm_total (kWh/request)
  - Derived: energy_per_token_kWh = energy_consumption_llm_total /total_tokens (when total_tokens > 0)

Leakage-prone columns intentionally excluded from features:

- energy_* (sub-components), power_draw_* columns, and any direct energy measurements.

2.2 Tools

- Python 3.12, numpy, pandas, scikit-learn, joblib
- Streamlit + Plotly for the interactive UI
- tiktoken (or heuristic fallback) for token estimation
- Simple conformal residual calibration and permutation importance (sklearn) for uncertainty and explainability
- Watsonx for data visualizations

3. Literature review and state of the art

- Environmental impact of ML:

Strubell et al. (2019) drew attention to the carbon cost of large NLP models; Patterson et al. (2021) outlined datacenter/hardware efficiencies and decarbonization strategies.
Lacoste et al. (2019) and Green Algorithms (Lannelongue et al., 2021) popularized practical emission accounting tools.

- LLM inference behavior:

Inference cost scales roughly linearly with output tokens for fixed model/hardware, plus a fixed overhead (initialization, scheduling, KV-cache). Telemetry trackers (e.g., CodeCarbon) provide direct measurements; this work learns predictive surrogates from logs to enable fast request-time estimation.

- Gap addressed:

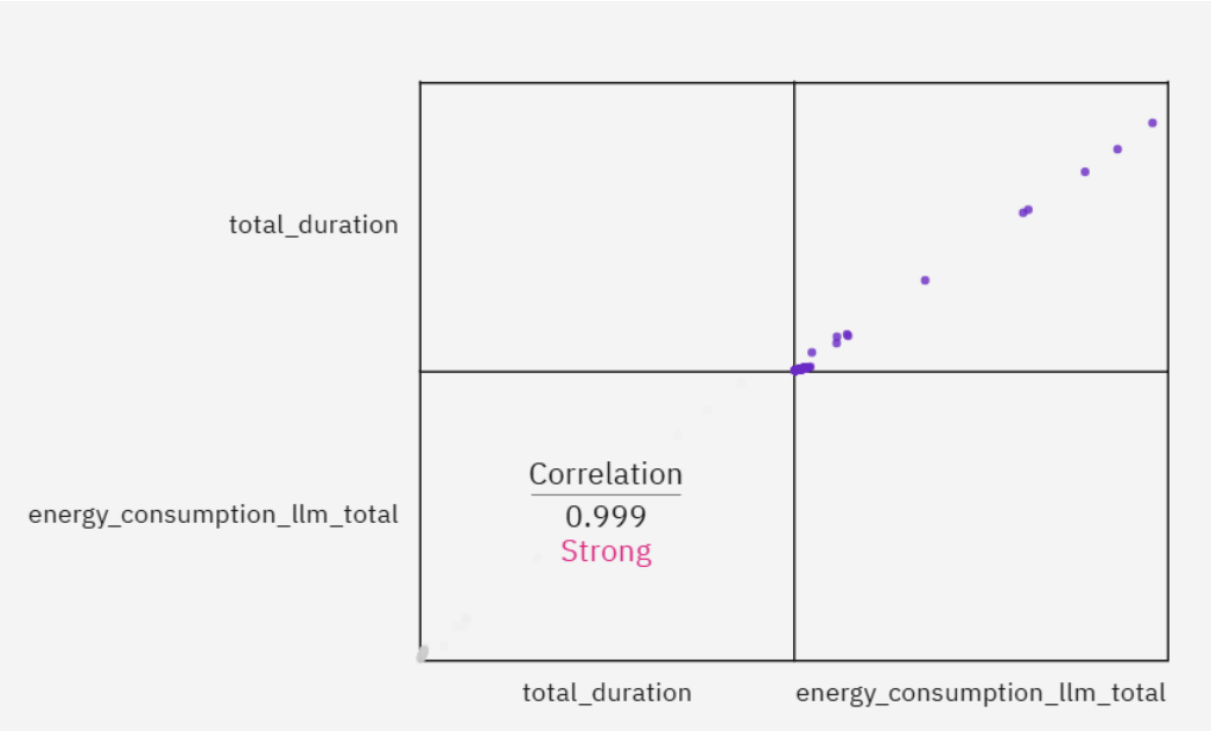
A lightweight, data-driven predictor using production-available features (tokens, durations, model type) with an interface for scenario analysis and uncertainty display.

References :

- Strubell, Ganesh, McCallum (2019)
- Patterson et al. (2021)
- Lacoste et al. (2019)
- Lannelongue et al. (2021)

4. Data Exploration and Feature Engineering

Before deciding which columns to keep or remove, we conducted an exploratory data analysis using WatsonX Studio's visualization tools. We examined correlations, scatter plots, and other descriptive graphs to better understand the relationships between variables. For instance, the figure below shows a very strong correlation (0.999) between total_duration and energy_consumption_llm_total, indicating an almost linear relationship between inference time and the model's total energy consumption.

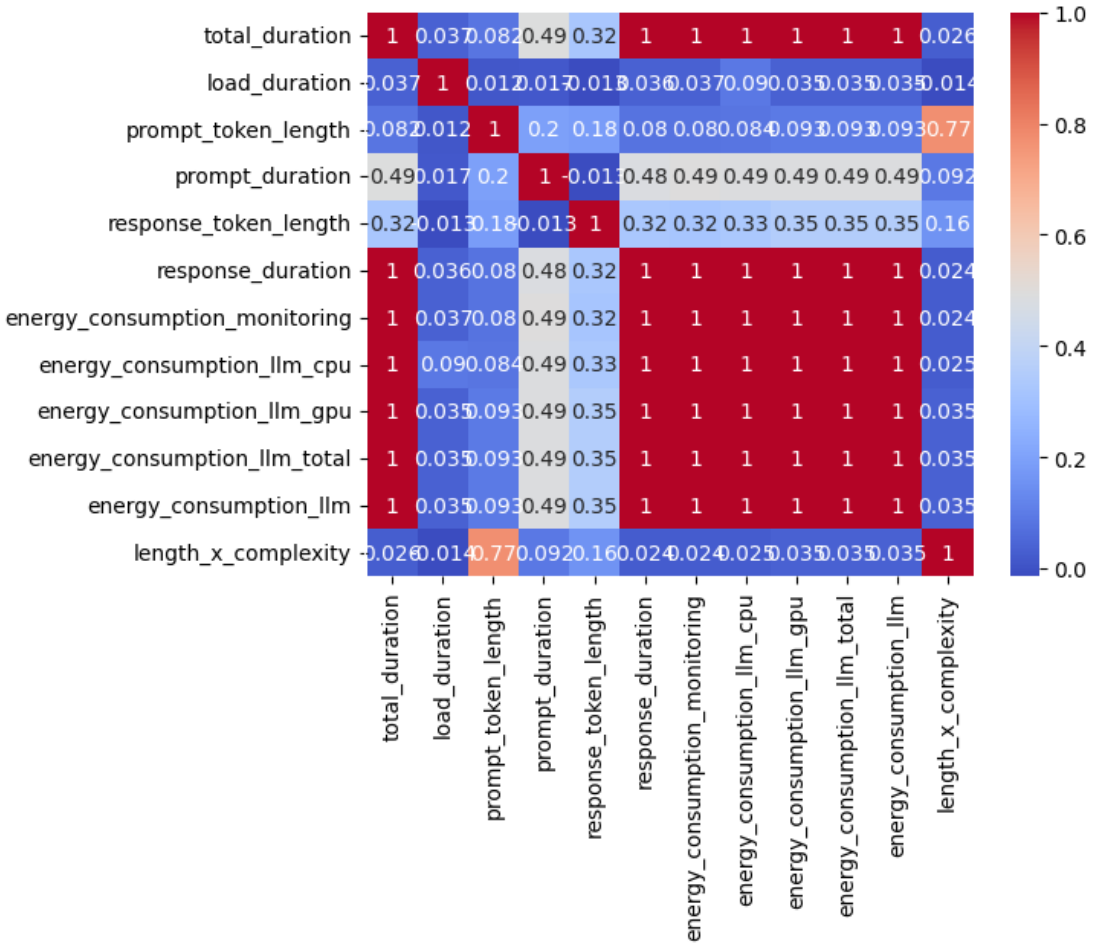


Based on our analysis, we decided to drop some columns that were not useful for prediction, such as szigriszt_pazos, gutierrez_polini, and syllable_count. We also performed several conversions and computations to create new relevant features, such as the average power draw of the GPU, CPU as well as token-related indicators like total_tokens and prompt_response_ratio.

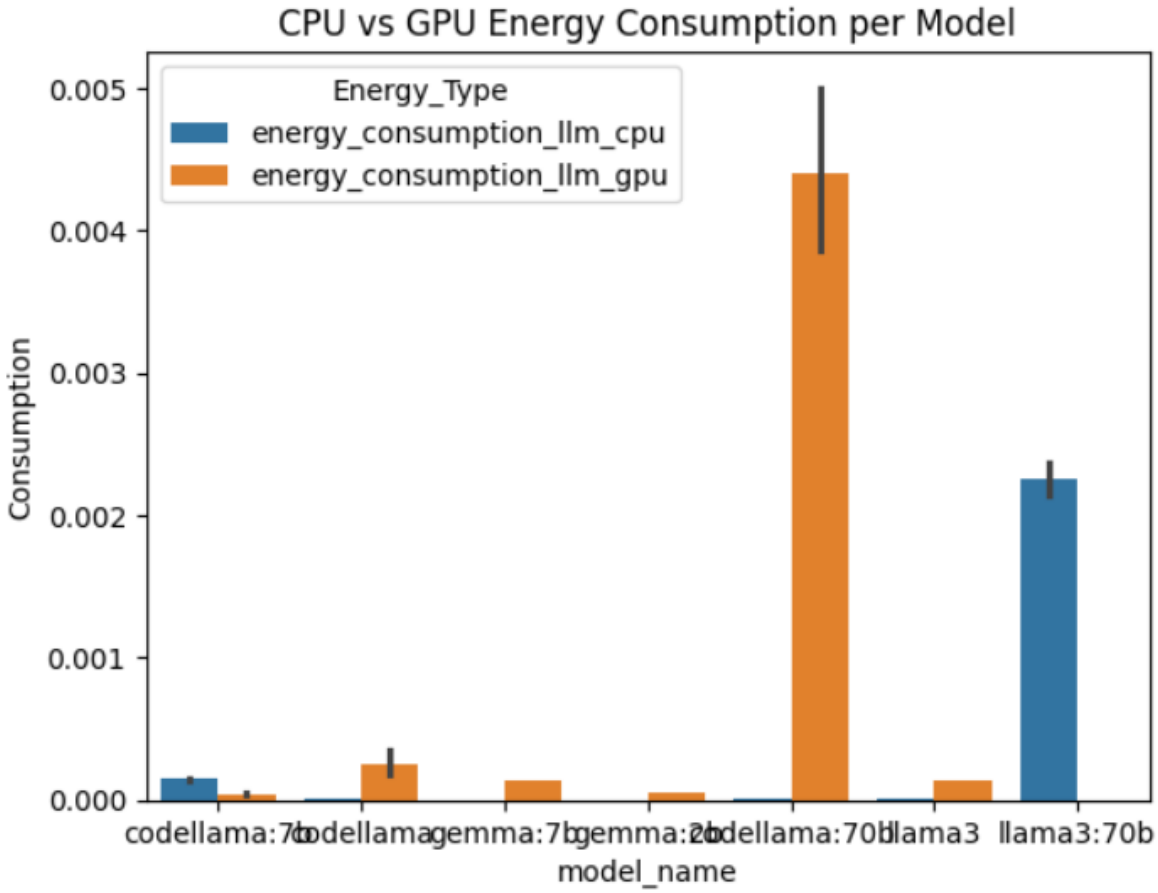
```
# Power draw in watts
df["power_draw_gpu"] = (df["energy_consumption_llm_gpu"] * 1000) / (df["inference_time"] / 3600)
df["power_draw_cpu"] = (df["energy_consumption_llm_cpu"] * 1000) / (df["inference_time"] / 3600)
df["power_draw_total"] = (df["energy_consumption_llm_total"] * 1000) / (df["inference_time"] / 3600)

# Token features
df["total_tokens"] = df["prompt_token_length"] + df["response_token_length"]
df["prompt_response_ratio"] = df["prompt_token_length"] / df["response_token_length"].replace(0,1)
```

In addition to scatter plots, we also generated a correlation heatmap to identify relationships between numerical features. As shown in the figure below, several variables related to energy consumption—such as energy_consumption_llm_total, energy_consumption_llm_gpu, and energy_consumption_llm_cpu—are highly correlated with each other (correlation ≈ 1.0). This strong multicollinearity indicates that these variables convey almost identical information. Similarly, total_duration and response_duration are strongly correlated, suggesting that total inference time is largely driven by the response generation phase. On the other hand, features such as prompt_token_length, response_token_length, and length_x_complexity show much weaker correlations with energy-related variables, meaning they may provide additional, non-redundant information for prediction.

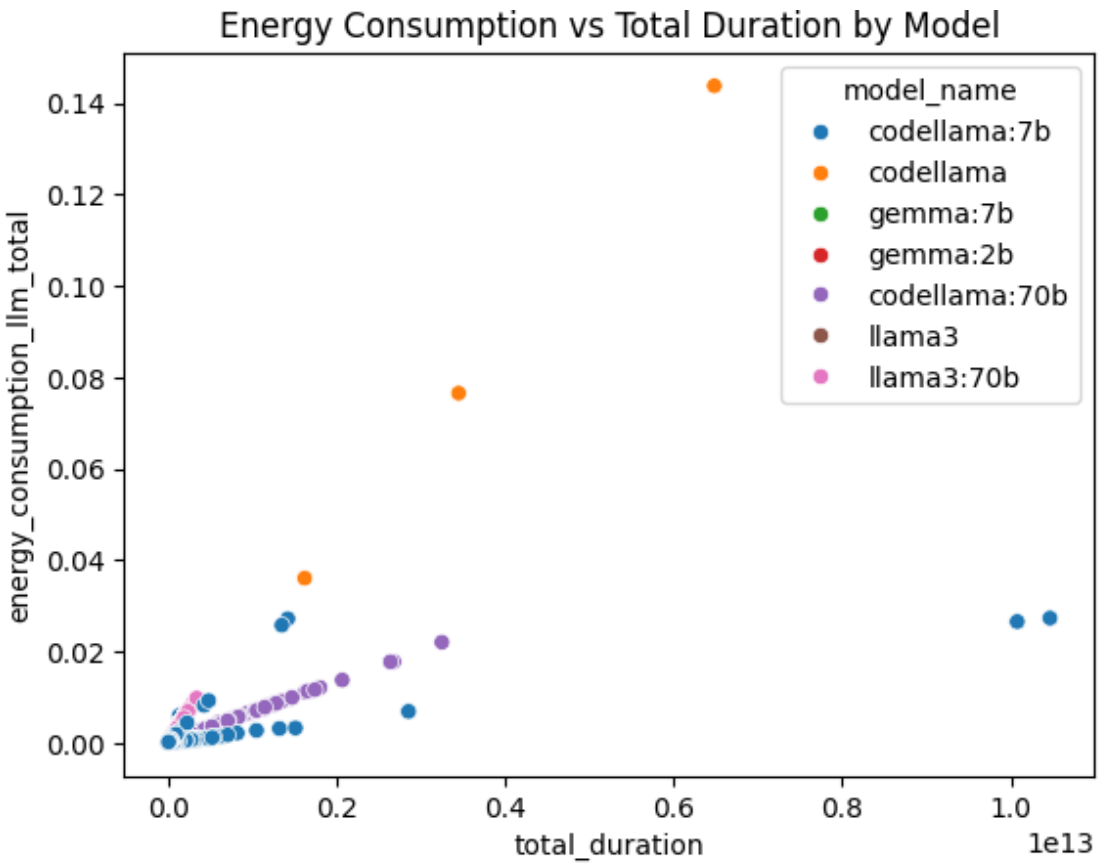
Based on these observations, we removed redundant columns (e.g., keeping only energy_consumption_llm_total as a representative feature) and retained variables that capture distinct aspects of the inference process.



This graph compares energy consumption between the CPU and GPU for different language models. Overall, GPU consumption is significantly higher than CPU consumption, especially for large models like *codellama:70b*, which shows the highest GPU usage. In contrast, *llama3:70b* stands out with relatively higher CPU consumption but lower GPU usage.



This graph shows the relationship between total execution time and total energy consumption for each model. A positive correlation is observed: longer execution times generally correspond to higher energy consumption. Some models, such as *codellama* and *gemma*, consume more energy overall, indicating lower energy efficiency. Conversely, models like *llama3* appear to achieve a better balance between duration and energy use, suggesting more effective hardware optimization.


Energy Consumption vs Total Duration by Model
energy_consumption_llm_total
total_duration
1e13
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.0
0.2
0.4
0.6
0.8
1.0
model_name
codellama:7b
codellama
gemma:7b
gemma:2b
codellama:70b
llama3
llama3:70b

Design of the approach (model choice, evaluation metrics)

5.1 Two complementary targets

- Per-request regression (direct): predict total energy in kWh for a whole inference call.

$$\hat{E}_{request} = f(X)$$

- Per-token regression: predict kWh/token, then scale by total tokens at inference time.

$$\hat{E}_{token} = g(X)$$
$$\hat{E}_{request\text{-}from\text{-}token} = \hat{E}_{token} \; x \; total\_tokens$$

- CO_2e conversion: Convert predicted energy used into CO_2-equivalent emissions (in kg) :

$$CO_{2e} \text{ (kg)} = \hat{E}_{request} \; kWh) \; x \; grid\_intensity \; (kg/kWh)$$

This pair gives:

- an end-to-end estimate (per-request), often capturing fixed overheads; and
- a controllable, "what-if" axis via response length (per-token).

4.2 Features (no leakage)

Numeric	`prompt_token_length`, `response_token_length`, `total_tokens`, `*_duration_s` (total/prompt/response/load), `length_x_complexity` (if present).
Categorical	`model_name`, `type`.
Excluded to avoid leakage	direct or component energy columns (e.g., `energy_consumption_llm_*`), and `power_draw_*`.

5.3 Models and preprocessing

- Preprocessor (ColumnTransformer):

  - Numeric: `StandardScaler(with_mean=False)`
  - Categorical: `OneHotEncoder(handle_unknown="ignore")`
- Candidate regressors:
  - `LinearRegression` (baseline, compact)
  - `RandomForestRegressor` (non-linear baseline)
- Pipelines: `Pipeline([("pre", pre), ("model", regressor)])`

LinearRegression provides a compact, interpretable baseline and tiny artifacts suitable for deployment. RandomForest captures nonlinearity and feature interactions without heavy tuning.

5.4 Metrics

- MAE (Mean Absolute Error): average absolute error in the target's units.
  - Per-request: kWh per call
  - Per-token: kWh per token
  - Easy to interpret; robust to outliers relative to MSE/RMSE.

- MAPE (Mean Absolute Percentage Error): relative error (% of true). Sensitive near zero (percentages can explode). We clip denominators (1e-9 / 1e-12) to stabilize, but very small targets can still inflate MAPE.

- RMSE: square-root MSE; penalizes larger errors; reported for completeness.

- R^2: variance explained. High R^2 means predictions follow target variance; can be lower even with small MAE if the model shrinks variability (e.g., averages like forests).

- Auxiliary classification sanity check (terciles): convert continuous y into three quantile bins and compute accuracy and recall to assess ranking consistency.

5. First experiments and preliminary tests

6.1 Experimental setup

- Split: 80/20 train/validation (`random_state=42`).
- Models per target: `LinearRegression` and `RandomForestRegressor(n_estimators=300, n_jobs=-1, random_state=42)`.
- Preprocessing: `StandardScaler(with_mean=False)` for numeric features + `OneHotEncoder(ignore_unknown)` for categoricals.
- Targets variables:
  - Per-request: `energy_consumption_llm_total` (kWh/request).

- Per-token: `energy_per_token_kWh` (kWh/token), filtered to rows with `total_tokens > 0`.

5.2 Key observations (preliminary)

Per-token × tokens scales linearly with response length as expected, enabling intuitive what-if simulations. Per-request captures fixed overhead; at very small token counts, unconstrained regressors can produce tiny negatives (handled in UI by clamping to 0). RandomForest generally improves MAE/MAPE vs LinearRegression, while LinReg can retain a higher R^2 when it better tracks variance.

6. Development and tuning (parameters, architectures)

7.1 Pre-processing

- Duration normalization: converted raw durations to seconds (`*_duration_s`) by auto-detecting nanosecond-like magnitudes.
- `total_tokens = prompt_token_length + response_token_length`.
- Numeric NaNs → median imputation; categoricals → one-hot with unknowns ignored.
- Leakage guard: all energy/power columns excluded from features.

7.2 Parameters

- ColumnTransformer:
  - `("num", StandardScaler(with_mean=False), num_features)`
  - `("cat", OneHotEncoder(handle_unknown="ignore"), cat_features)`
- RandomForest:
  - `n_estimators=300, random_state=42, n_jobs=-1`
  - Light grid checks (200/300/500 trees; optional `max_depth`) showed diminishing returns beyond ~300 trees relative to model size and latency.
- LinearRegression:
  - No hyperparameters; relies on preprocessing; chosen as a compact "light" baseline.

7.3 Architectures and model choices

Per-request:

- Trained: LinReg and RF pipelines.

- Deployment choices:
  - "Light" LinReg for minimal artifact size and fast load.
  - RF favored when slightly larger artifacts are acceptable due to lower MAE/MAPE.

Per-token:

- Trained: LinReg and RF pipelines.
- Deployment choice:
  - RF is preferred (dominates on MAE/MAPE/R^2); LinReg kept as a tiny fallback.

Artifacts persisted (`.joblib`) along with a metadata bundle (feature lists, medians/modes, median response tokens).

7.4 Interface integration

- Token estimation via `tiktoken` (fallback heuristic ≈1 token/4 chars).
- Response length slider seeded from dataset median and capped at the 95th percentile (max 4096).
- Non-negativity clamp in presentation layer: `max(0, prediction)`.
- Optional conformal residual intervals (calibrated from dataset) and permutation importance for explainability.
- Repo-relative paths with pinned environment versions; Streamlit launched via `python -m streamlit` to guarantee interpreter consistency.

8. Intermediate validation, refinement, improvement, and documentation

8.1 Results (validation split)

The following metrics come from your "clean" train/test evaluation:

Model	MAE (kWh)	RMSE (kWh)	MAPE	R^2	Acc (terciles)	Recall macro	Recall weighted
LinearReg	0.00006012	0.00025289	109.53%	0.643	0.642	0.642	0.642
RandomForest	0.00000292	0.00004863	1.90%	0.987	0.993	0.993	0.993

Per-request (kWh/request)

Model	MAE (kWh/token)	RMSE (kWh/token)	MAPE	R^2	Acc (terciles)	Recall macro	Recall weighted
LinearReg	0.0000000513	0.0000002280	12.45%	0.944	0.879	0.879	0.879
RandomForest	0.0000000159	0.0000001048	3.54%	0.988	0.962	0.962	0.962

Per-token (kWh/token)

RandomForest is now the best model for both targets by every metric that matters.The per-request RF shows a very strong fit (R^2=0.987) with tiny absolute and relative errors.The per-token RF remains excellent (R^2=0.988, MAPE≈3.5%) and is ideal for "what-if" token control in the UI. LinearRegression remains a compact fallback but is clearly outperformed, especially for per-request.

Per-request (kWh per request)

- LinearRegression
    - MAE 6.01e-05 kWh (0.060 Wh), RMSE 2.53e-04 kWh, MAPE 109.53%, R^2 0.643.
    - Interpretation: The model tracks variance moderately (R^2 ~0.64) but exhibits large percentage error because many true request energies are very small. Even tiny absolute deviations inflate MAPE near zero.
- RandomForest
    - MAE 2.92e-06 kWh (0.00292 Wh), RMSE 4.86e-05 kWh, MAPE 1.90%, R^2 0.987, Tercile Acc/Recall ≈ 0.993.
    - Interpretation:
        - Absolute accuracy: outstanding. 2.92e-06 kWh ≈ 0.00292 Wh ≈ 10.5 J.
        - Relative accuracy: MAPE ~1.9% is very low, even with near-zero targets.
        - Variance tracking: R^2 ~0.987 indicates the model closely follows target variability.
        - Ranking quality: near-perfect tercile classification confirms consistent ordering of low/medium/high energy cases.

- RMSE > MAE indicates a few outliers exist, but overall errors are still tiny.

Per-token (kWh per token)

- LinearRegression
  - MAE 5.13e-08 kWh/token, RMSE 2.28e-07 kWh/token, MAPE 12.45%, R^2 0.944, terciles ≈ 0.879.
  - Interpretation: Good fit, but higher error than RF.
- RandomForest
  - MAE 1.59e-08 kWh/token, RMSE 1.05e-07 kWh/token, MAPE 3.54%, R^2 0.988, terciles ≈ 0.962.
  - Interpretation:
    - Absolute error: 1.59e-08 kWh/token ≈ 1.59e-05 Wh/token ≈ 0.057 J/token.
    - For a 300-token request, the MAE translates to ~4.8e-06 kWh ≈ 0.0048 Wh ≈ 17 J — consistent with the per-request RF error magnitude, which is reassuring.
    - Strong variance fit (R^2 ~0.988) and low relative error make it ideal for scenario analysis.

Why these metrics (and how to read them)

- MAE (primary): directly in energy units; best for judging typical absolute error you'll show in the app.
- MAPE: good for relative error but unstable near zero (explains LinReg's >100% on per-request).
- RMSE: highlights outliers; here it's modest and consistent with small overall errors.
- R^2: checks how well variance is captured; your per-request RF improved dramatically (0.987), resolving the earlier "low R^2 despite small MAE" pattern.
- Tercile accuracy/recall: a ranking sanity check (does the model at least order low/medium/high correctly?). RF is excellent.

9. Web interface

Technologies used : Streamlit + Plotly.

Capabilities

Single Request

- Token estimation for prompt (tiktoken or heuristic).
- Response token control (auto: dataset median; manual slider with data-derived bounds).
- Two estimates: per-request (direct kWh) and per-token×tokens (kWh/token scaled by total_tokens).
- CO_2e conversion via grid presets (Global, EU, US, India, Renewable) or custom intensity.
- Optional conformal intervals and a CO_2 budget optimizer (suggested response token cap).
- Heatmap of CO_2e vs response_tokens × grid intensity.
- Feature row preview showing exactly what the pipelines consume.

Conversation

- Multi-turn simulation; optional context carry so prompts grow with history.
- Per-turn and cumulative energy/CO_2 curves.

Batch Simulate

- CSV upload: `prompt, response_tokens, [model_name, type]`.
- Produces per-row energy/CO_2; downloadable results.

Compare Pipelines

- Side-by-side comparison of per-request vs per-token×tokens.

Explain & Validate

- Validation metrics (MAE, R^2) on a dataset sample.
- Permutation feature importance for both pipelines.

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
python -m streamlit run source\web-interface.py
```