Green Computing Final Project Report

Binary Classification: Optimization Strategies and Language Efficiency for Digital Decarbonization

Lorenzo Raia 894535

July, 2026

1 Introduction

The increasing adoption of Artificial Intelligence comes with a substantial environmental cost due to the high energy consumption of model training and inference. This project focuses on **Green AI**, aiming to optimize the resources utilized during the software's life cycle.

A binary classifier was implemented using the **Random Forest** algorithm. Because ensemble methods can be highly energy-intensive and computationally greedy, Random Forest serves as an ideal candidate to demonstrate the effectiveness of green optimization strategies across different programming paradigms (Python vs. Rust).

2 Theoretical Background

To accurately measure the sustainability of the software, execution time alone is insufficient. We must evaluate the underlying Energy Consumption (measured in Watt-hours) and the resulting Carbon Footprint (CO_2eq).

Performance is evaluated using the **Matthews Correlation Coefficient (MCC)**, which provides a reliable metric even in the presence of class imbalances by utilizing all four quadrants of the confusion matrix.

3 Methodology and Optimization Strategies

The evaluation framework relies on a Repeated Hold-out validation technique (100 iterations) for robustness. The dataset undergoes a two-step split: initially 60/40 (Train/Temp), and subsequently the Temp set is divided 50/50 into Validation and Test sets.

To achieve lighter and more sustainable models, the following techniques were explicitly implemented in the codebases:

- **Data Trimming:** In the optimized scripts, the training data size is drastically reduced, retaining only 20% of the original training split (`train_size=0.20`). This heavily reduces the number of operations required during the `.fit()` phase.
- **Model Pruning & Architecture Simplification:** The standard Random Forest relies on a large number of redundant trees. The baseline model's 100 trees were pruned down to 15 (`n_estimators=15`), and the complexity of each tree was capped by setting a maximum depth of 5 (`max_depth=5`).
- **Hardware Efficiency:** The `n_jobs=-1` parameter was introduced in the Python Green implementation to exploit multicore parallelism, reducing the overall active computing time.

- **Data Quantization:** The Python Green code and Rust Green code allow for casting data to lower precision (`float32`), reducing the memory footprint and accelerating matrix multiplications.

4 Implementation

4.1 Python Implementations (Baseline vs. Green)

Two Python scripts utilizing 'scikit-learn' were developed. The **Baseline** approach simulates an "unaware" developer using default parameters. The **Green** implementation injects the aforementioned optimization strategies. Energy consumption and emissions for both scripts were tracked in real-time using the `CodeCarbon` library.

4.2 Rust Implementation (Compiled Efficiency)

To address the overhead caused by Python's interpreted nature and dynamic typing (the Von Neumann bottleneck), a third implementation was written in **Rust** using the `smartcore` machine learning library. The Rust version strictly replicates the data trimming and model pruning strategies (15 trees, max depth 5). Furthermore, a custom MCC calculation algorithm was implemented from scratch to handle dynamic class detection in Rust safely.

To measure Rust's consumption accurately, a Python Wrapper using `subprocess` was implemented. This allowed **CodeCarbon** to monitor the hardware sensors while the compiled binary executed the heavy math operations, ensuring a fair comparison between the high-level and low-level implementations.

5 Results and Analysis

Table 1 summarizes the average metrics obtained across the datasets.

Implementation	**Avg. MCC**	**Time (s)**	**Energy (Wh)**	**CO2 (g)**
Python Baseline	0.5971	7.0951	0.0647	0.0214
Python Green	0.4719	3.6872	0.0355	0.0117
Rust Green	0.4193	0.5812	0.0092	0.0030

Table 1: Average performance and environmental impact comparison.

5.1 Visualizations

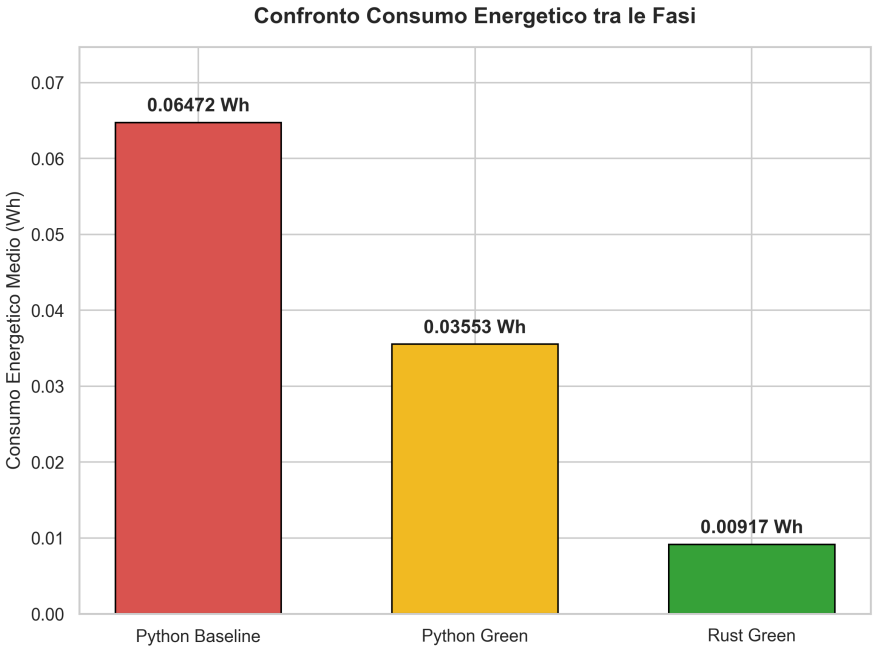

Figure 1: Energy consumption across the three implementations.

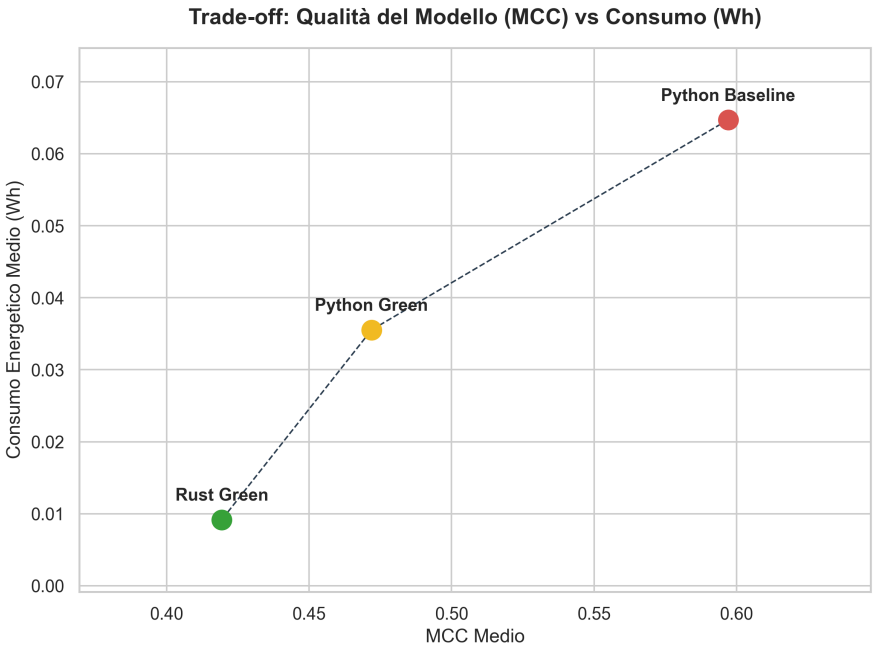

Figure 2: The Accuracy-Sustainability Trade-off: Comparing MCC degradation against energy savings.

6 Discussion

The results validate the hypothesis that standard machine learning workflows contain a high degree of computational redundancy, while also clearly exposing the "Accuracy-Sustainability

Trade-off".

- **Algorithmic Optimization (Python Baseline vs. Python Green):** The transition from the standard approach to the Green version demonstrates the direct impact of Model Pruning and Data Trimming. By reducing the forest to 15 trees, capping depth to 5, and trimming 80% of the training data, we achieved a **45.1% reduction in energy consumption** (from 0.0647 Wh to 0.0355 Wh) and cut execution time by roughly 48%. However, this aggressive pruning resulted in an MCC penalty of roughly 0.125 (a 21% drop from the baseline). This confirms that while the baseline is "energy-greedy", a portion of its complexity actively contributes to its predictive power.
- **The Rust Advantage (Language Efficiency):** Moving the optimized logic to a compiled language yielded extraordinary hardware efficiency. The Rust Green implementation achieved an **85.8% overall energy reduction** and a **91.8% speed-up** compared to the Python Baseline. Even more remarkably, when comparing Rust to the already-optimized Python Green, Rust consumed **74% less energy** (0.0092 Wh vs. 0.0355 Wh) to execute the exact same pruning and trimming logic. This stark contrast directly exposes the severe energy tax caused by the Von Neumann Bottleneck and Python's interpreter overhead.
- **Framework Discrepancies:** Despite using the exact same hyperparameters (15 trees, max depth 5, 20% training data), Rust's MCC (0.4193) was slightly lower than Python Green's (0.4719). This minor degradation is likely attributable to the internal architectural differences between the mature `scikit-learn` library and the newer Rust `smartcore` crate regarding how tree nodes are split and random seeds are handled.
- **The Trade-off Conclusion:** The project confirms that sustainable AI requires conscious choices. If a deployment scenario (e.g., edge computing) can tolerate an MCC degradation of $\sim$0.17, developers can leverage architectural simplification and compiled languages to abate carbon emissions (CO_2**eq**) by over 85% (from 0.0214g to 0.0030g).

7 Conclusion

This project demonstrates that sustainable AI is highly achievable through deliberate software engineering choices. By applying Model Pruning, Data Trimming, and utilizing Compiled Languages, the carbon footprint of classification tasks can be severely reduced. Moving forward, integrating these optimizations should be a standard practice rather than an afterthought, ensuring that energy efficiency is treated as a primary metric alongside model accuracy.

8 Software Repository

The complete source code, datasets, and execution instructions can be found at:
`https://codeberg.org/lorenzoraia/green-computing-project`