UNIVERSITÀ
DEGLI STUDI
FIRENZE

Energy, Emissions and Performance: Cross-Language and Cross-Algorithm Analysis in Machine Learning

Leonardo Pampaloni, Niccolò Marini, Filippo Di Martino

Course: SOFTWARE ARCHITECTURES AND METHODOLOGIES

UNIVERSITÀ DEGLI STUDI FIRENZE

Introduction

Impact of AI

- Growth of AI - *"HealthCare, Finance, Education, Entertainment, Transportation, ..."*
- Climate Crisis - *"AI already uses as much energy as a small country. It's only the beginning"* *[1]*
- Project Aim - "*The green language*"

Project purpose

The purpose of the project is to identify the most environmentally sustainable programming language by comparing various languages based on their CO2 emissions, energy consumptions, and execution time while performing identical tasks.

Those metrics can provide insights into which language is the "greenest" in terms of environmental impact.

UNIVERSITÀ DEGLI STUDI FIRENZE

Approach

Languages

The selected programming languages were chosen for their widespread usage and their ability to implement algorithms in a similar manner, ensuring a fair comparison of their performance across the different tasks.

- Python
- R
- MatLab
- C++
- Java

Algorithms

The selected algorithms were primarily chosen due to their widespread use in the field of machine learning, which ensured their implementation across all the programming languages used in the project. This variety also provides a comprehensive evaluation of model performance across different tasks, making the results applicable to diverse real-world scenarios and industries.

- Decision Tree
- Random Forest
- K nearest neighbors
- Logistic Regression
- Naive Bayes
- AdaBoost
- Support Vector Classifier

Datasets

Datasets

	target	target Type	# rows	# features
Breast Cancer	diagnosis	binary	~570	31 + 1
Wine Quality	quality	binary (>6)	~6500	12 + 1
Iris	species	nominal	~150	4 + 1

The selected datasets were sourced from the UC Irvine Machine Learning Repository. These three datasets were chosen for their diversity in key characteristics, including size, target variable, and number of features.

CodeCarbon: What, Why and How

- **CodeCarbon** is an open-source tool that tracks the carbon footprint of computational tasks, particularly useful in machine learning. It monitors energy consumption and estimates emissions to help understand the environmental impact of their code.
- Accurate Emission Tracking, User-Friendly Integration
- Focused Measurement, Data Collection

Evaluation Metrics

CodeCarbon provides many misurations, there were selected a bunch of them based on what was important for the project:

- Energy_consumed: sum of cpu_energy, gpu_energy and ram_energy (kWh)
- Duration: Duration of the compute, in seconds
- Emissions: Emissions as CO2, in Kg
- Emissions_rate: emissions divided by duration, in Kg/s
- Cpu_power: CPU power in W
- Cpu_energy: Energy used per CPU (kWh)

General Implementation

- The final CSV file contains 30 rows, representing every possible combination of language, algorithm, dataset, and phase (train/test)
- The datasets were split using an 80/20 ratio, with 80% of the data used for training and 20% for testing.
- The implementation of the emission tracker was standardized across all languages. Each language calls the same Python function to start and stop the tracker, isolating only the specific lines of code responsible for the training or testing phase. This ensures consistency in measuring emissions during the key stages of model execution.
- This implementation ensures that all runs are measured with extreme accuracy, avoiding any bias introduced by other operations such as loading and splitting datasets. By isolating the emission tracking to only the lines of code responsible for training and testing, the results reflect the actual computational cost of these phases without interference from preparatory tasks

UNIVERSITÀ DEGLI STUDI FIRENZE

Methodology

Workflow

Global Emissions Tracking:

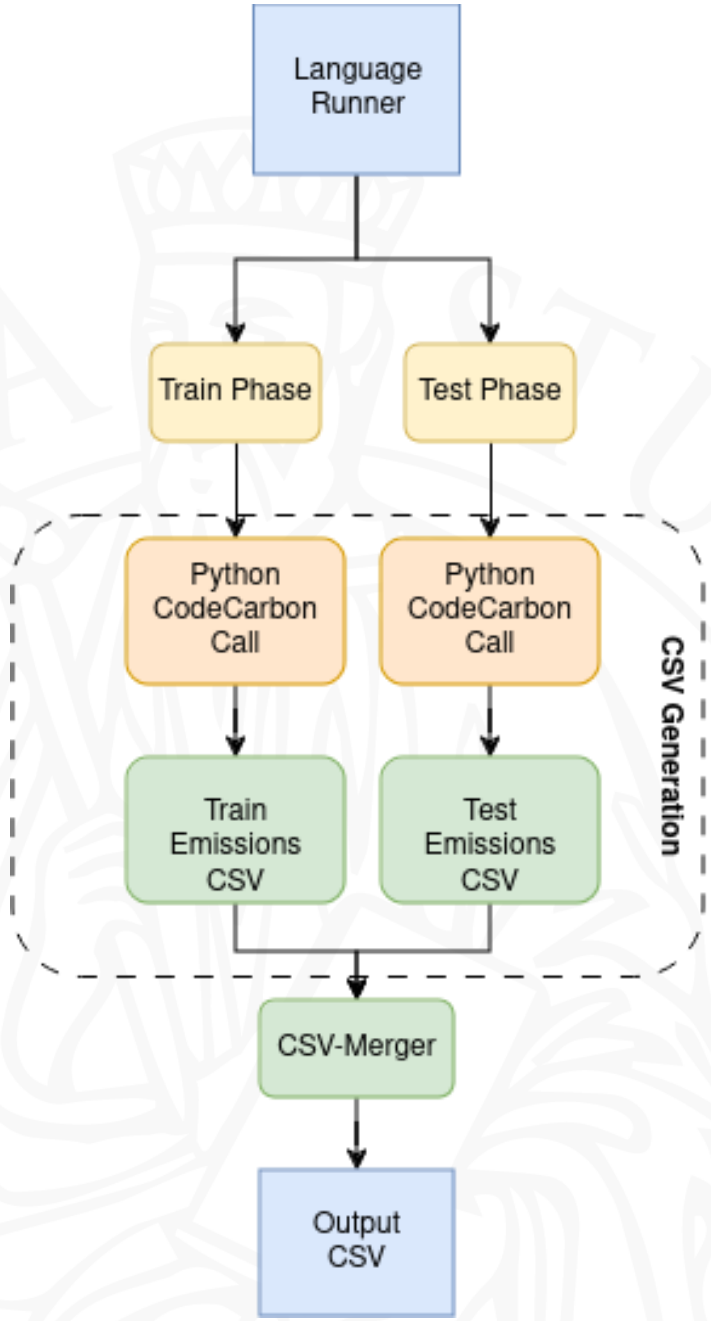
- A custom **Python function** was developed to **integrate CodeCarbon** into the relevant parts of the workflow.

Centralized Emissions Monitoring:

- This function was invoked **directly within each programming language**, enabling **consistent emissions monitoring** during both **Phases**,

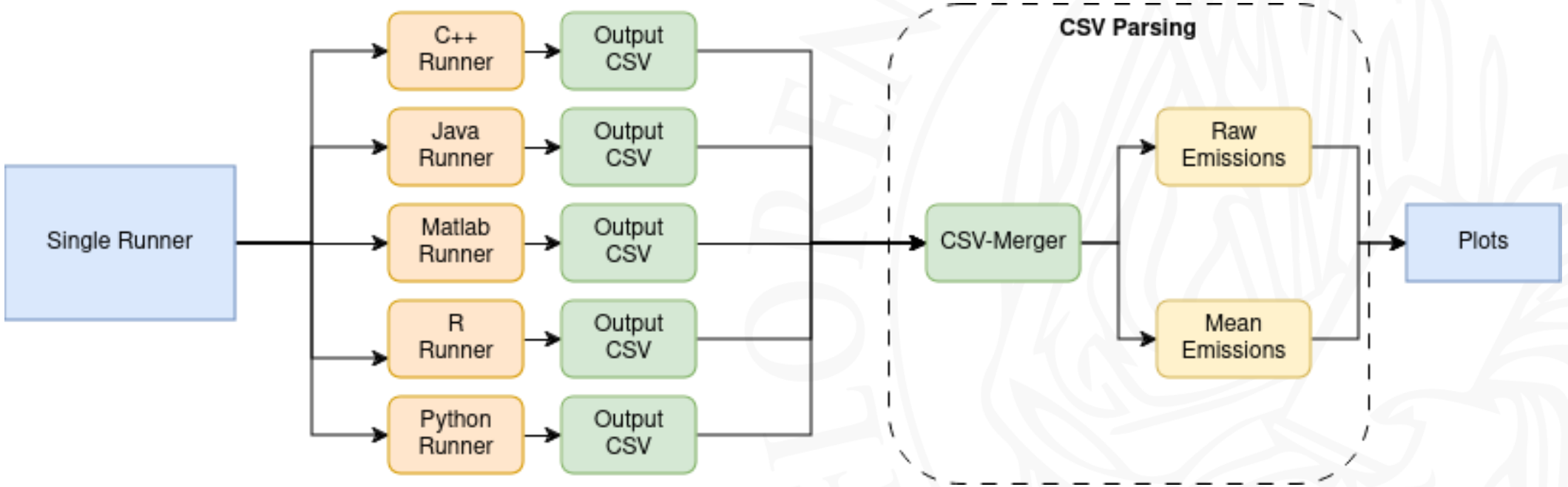
Precise Carbon Footprint Measurement:

- By targeting only the **core phases** (training and testing), the workflow ensures that the data **accurately reflects the true carbon footprint** of the models.



Workflow

We implemented a **single `.sh` runner script** that streamlines the execution of algorithms across various programming languages throught their runners.
Each runner generates an output CSV file with the results, which are subsequently merged into a single dataset, this dataset is then divided into two CSV files.



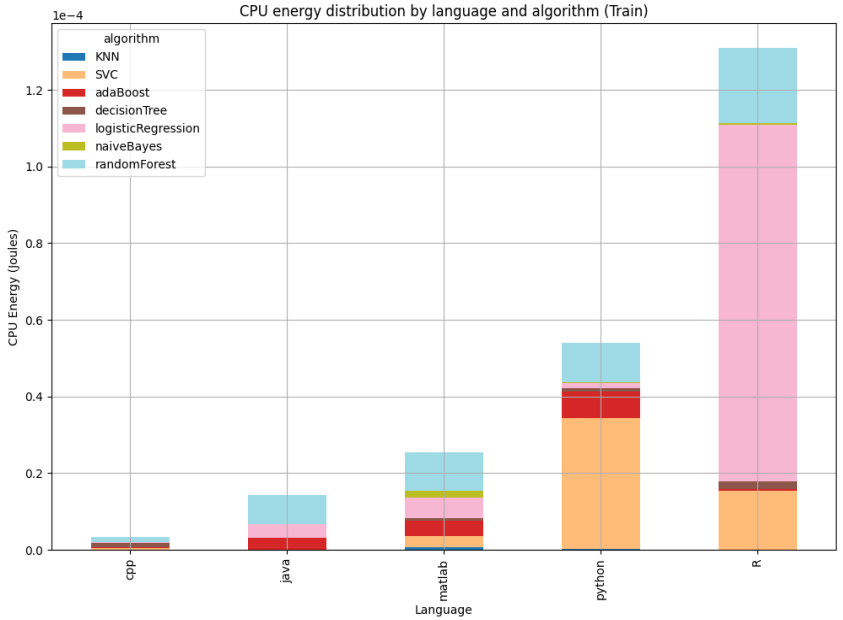
UNIVERSITÀ DEGLI STUDI FIRENZE

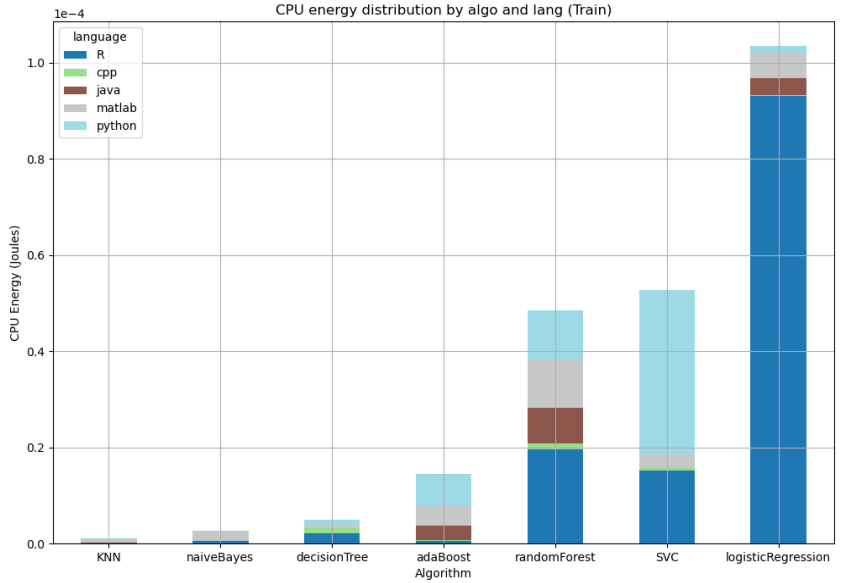
Results

Cpu Energy (Train)

For each language, the relative CPU energy consumption during the training phase can be observed. Notably:

- C++ demonstrates significantly faster performance, while R is considerably slower.
- The prolonged training time for R is mainly due to the implementation of logistic regression.
- Python shows particularly slow performance for the SVC (Support Vector Classifier) algorithm."



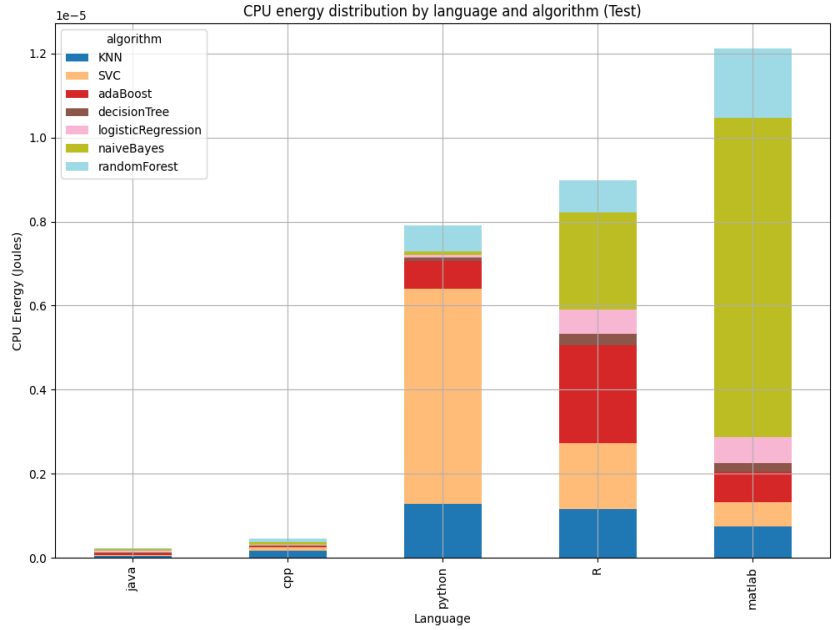


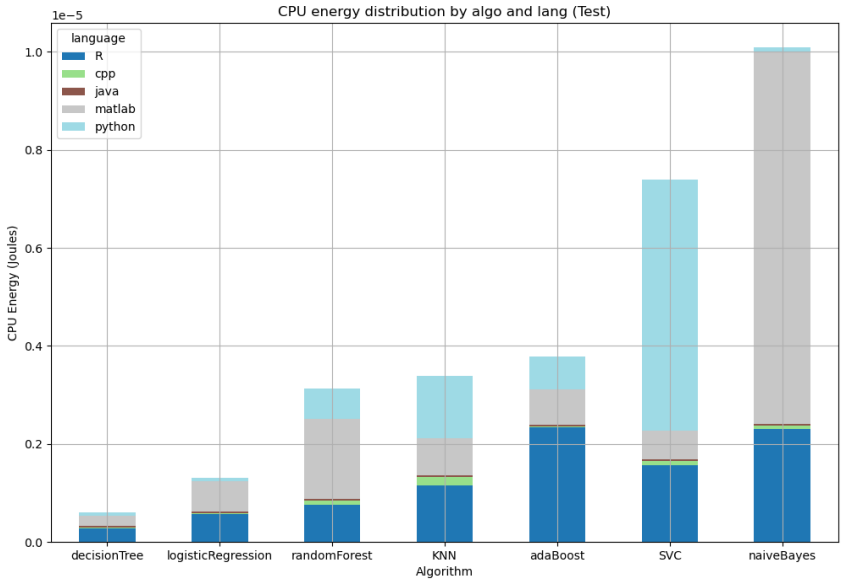
Cpu Energy (Test)

This section emphasizes the importance of inference, which plays a crucial role in real-world applications:

- C++ and Java outperform significantly, being 10 to 20 times faster than the other languages.
- Python, R, and MATLAB show relatively comparable performance.

Each language highlights differences in the implementation of algorithms.

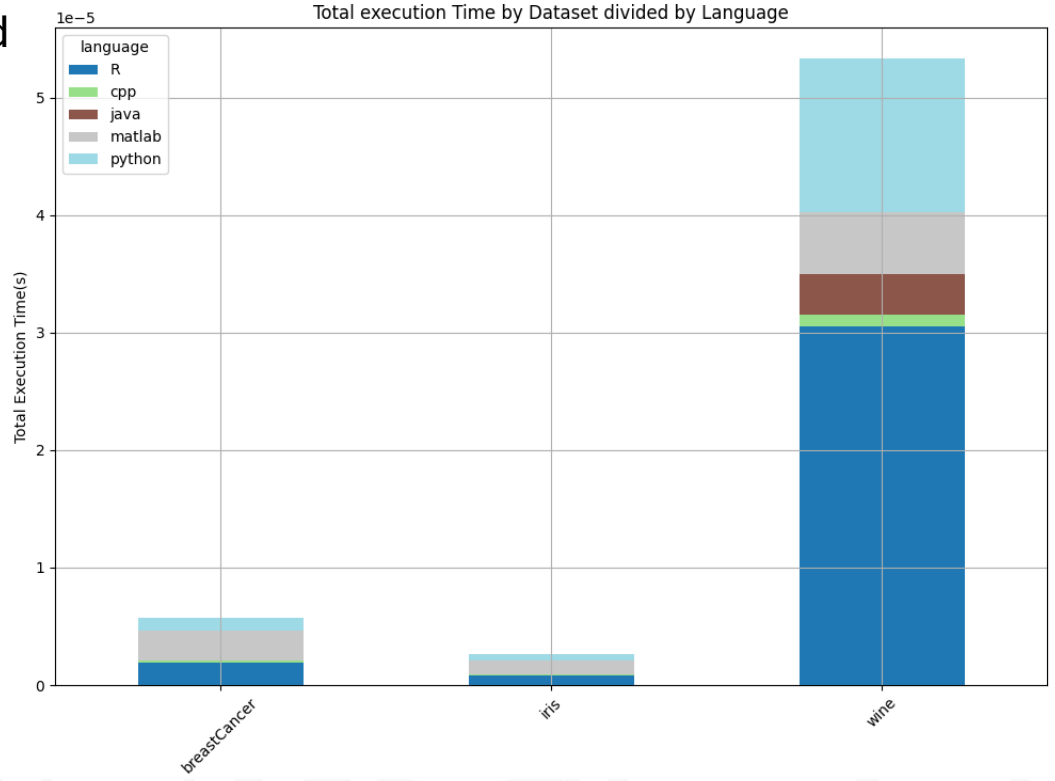




Dataset Execution Time (trest & train)

The Wine dataset is significantly larger than the others, and the execution time demonstrates a near-linear increase as the dataset size grows.

However, MATLAB does not follow this pattern, as smaller datasets require a similar amount of execution time compared to larger ones.

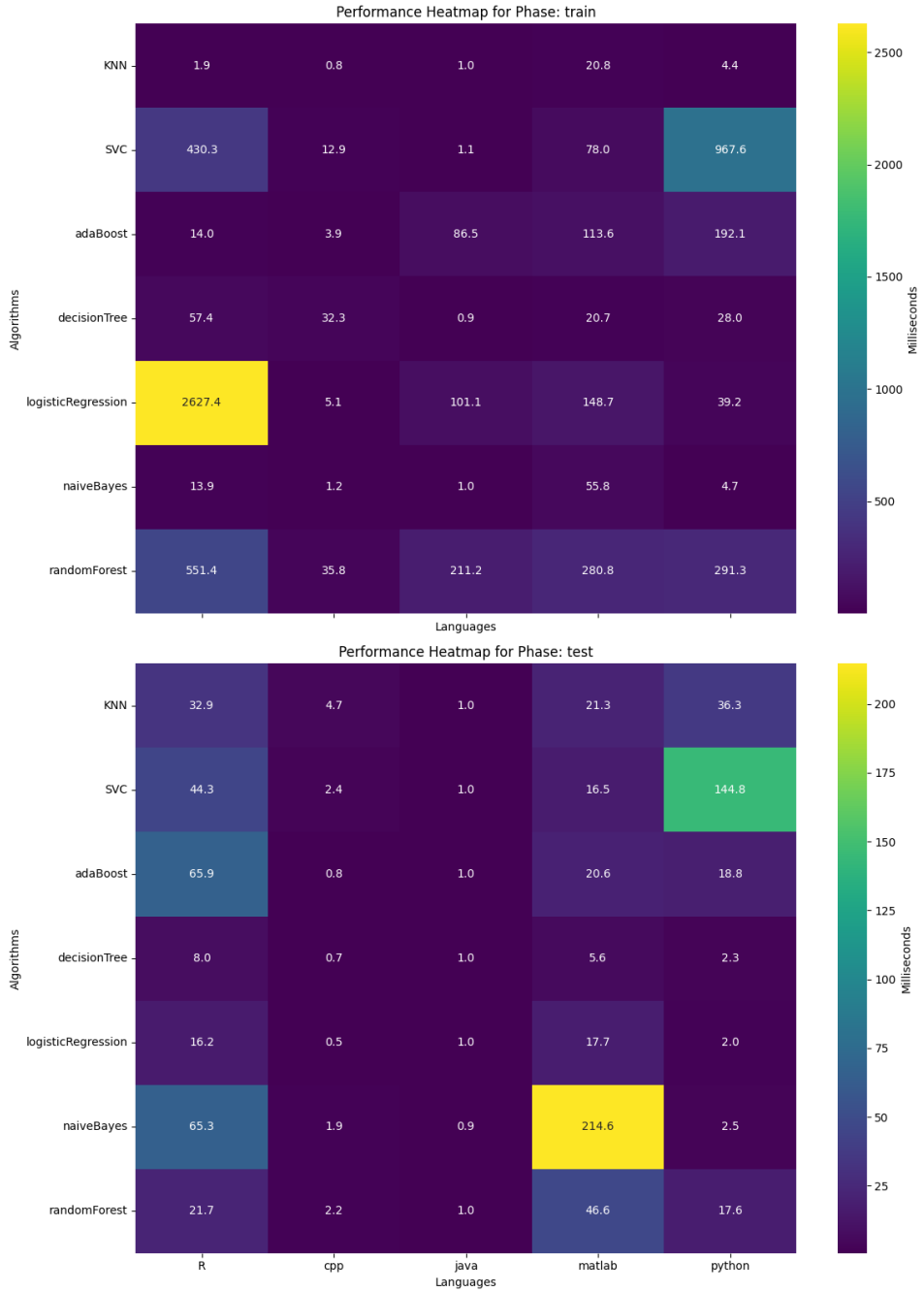


UNIVERSITÀ DEGLI STUDI FIRENZE

Heatmap for execution time

The heatmap is used to provide a visual representation that highlights the optimal combinations of algorithms and languages.

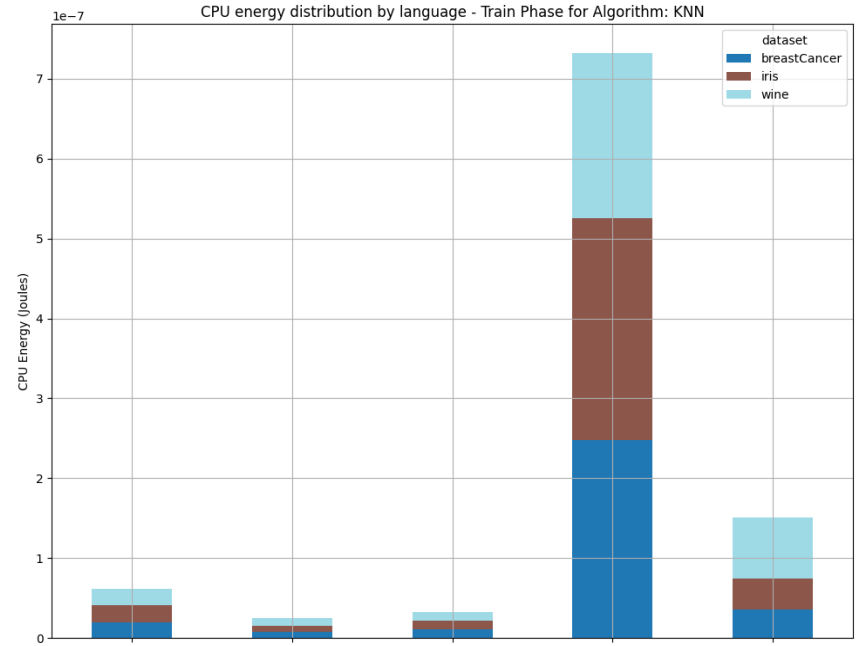
- C++ and Java emerge as the fastest across both the training and testing phases.
- Among the algorithms, KNN, Logistic Regression, and Decision Tree show the best performance."

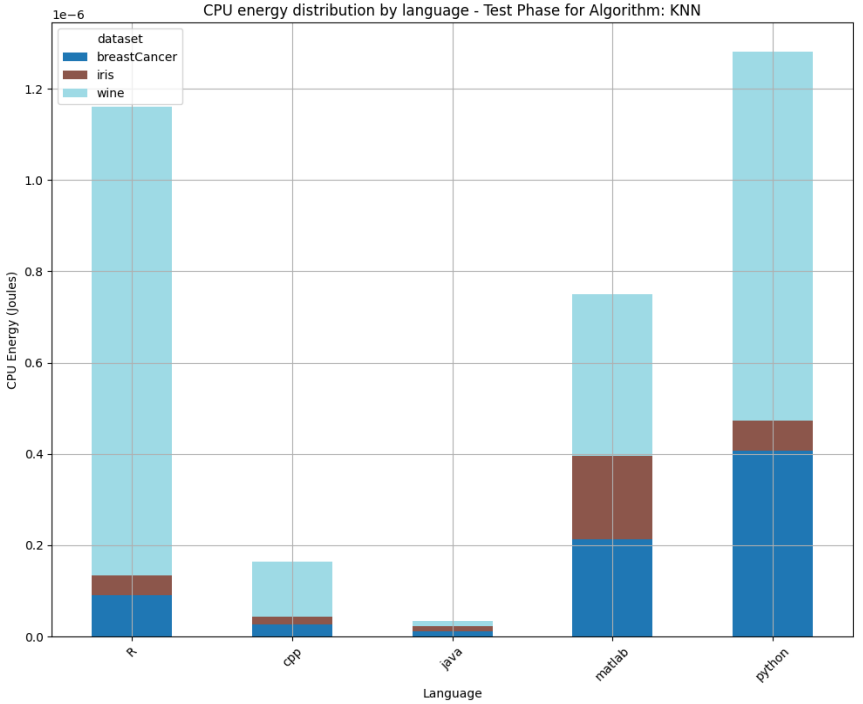


UNIVERSITÀ DEGLI STUDI FIRENZE

Cpu energy for KNN

- **Train Phase:** Except for MATLAB, all languages show similar performance.
- **Test Phase:** Even with faster algorithms, C++ and Java continue to be the fastest.



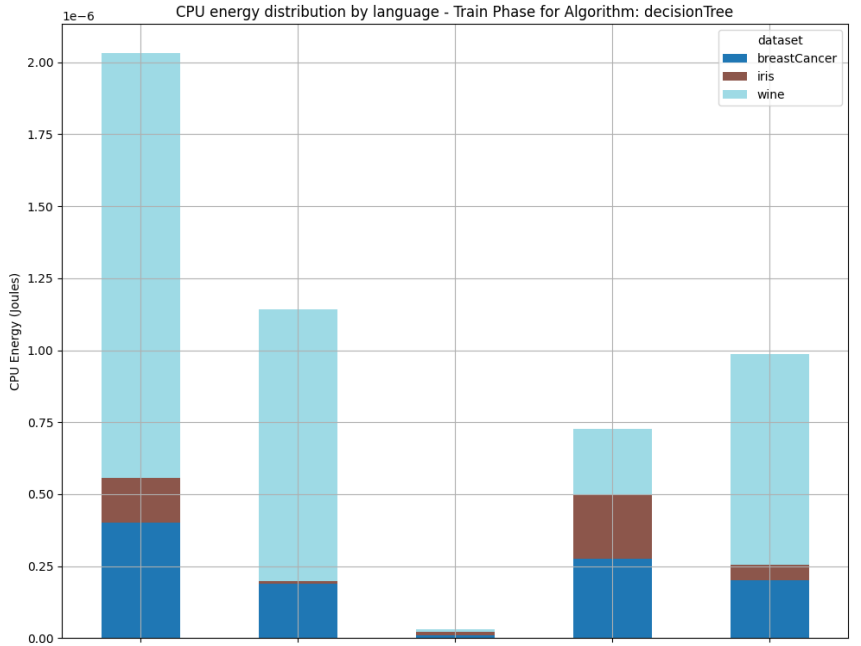


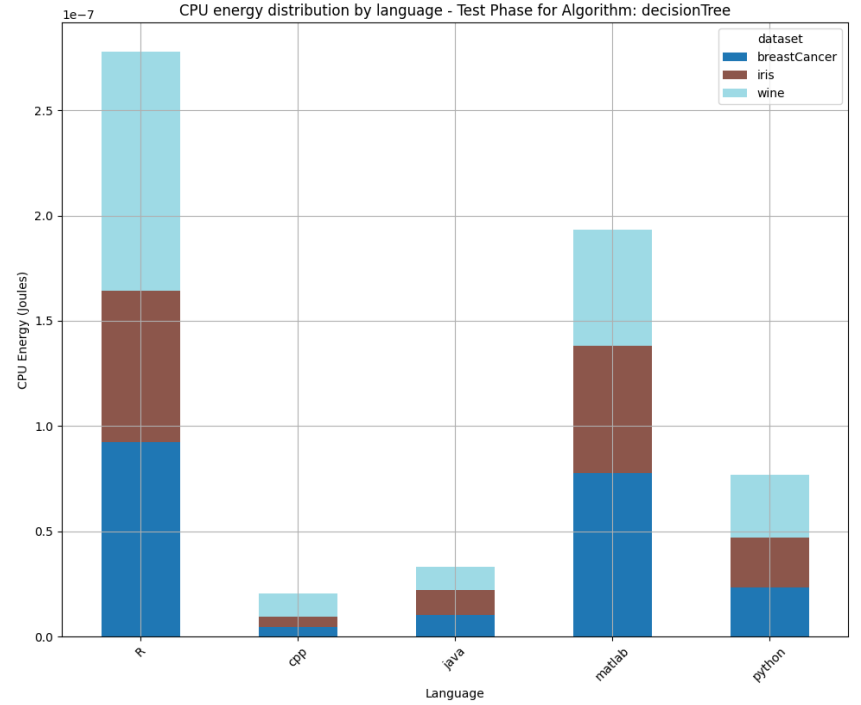
UNIVERSITÀ DEGLI STUDI FIRENZE

Cpu energy for Decision tree

- **Train Phase:** Java is the fastest among the languages evaluated.

- **Test Phase:** In this configuration, Java and C++ exhibit similar performance levels when compared to Python. This suggests an interesting opportunity to optimize operations using Python.

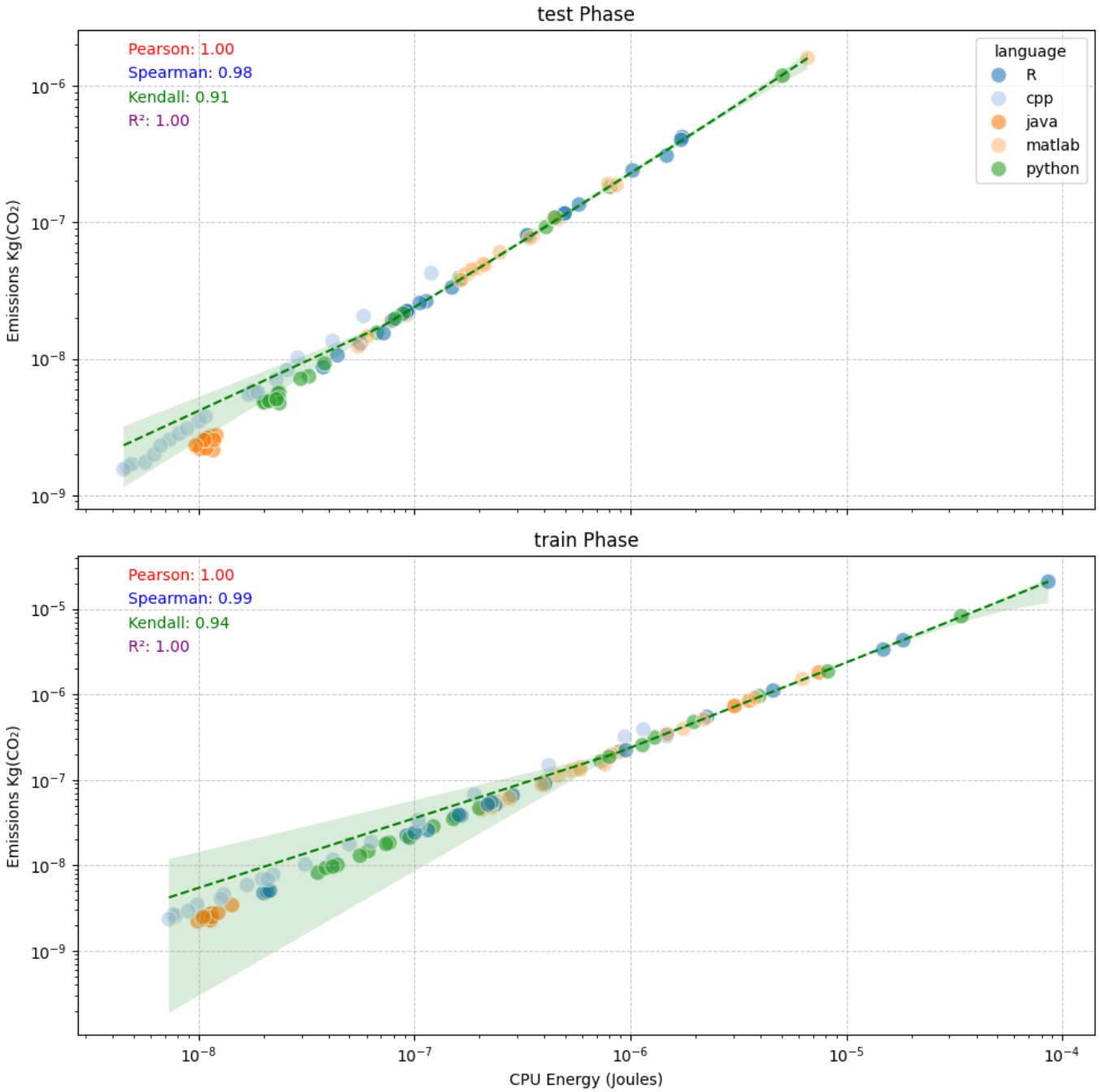




UNIVERSITÀ DEGLI STUDI FIRENZE

Correlations

CPU energy consumption and emissions exhibit an almost linear correlation. This suggests that all languages utilize similar resources across the different phases.



Real case scenario

Using our data, we create a hypothetical scenario involving an AI model running continuously for inference throughout the day, illustrating the number of requests that can be processed.

With a logistic regression implemented in C++, we can achieve a 265% increase in the number of requests processed.

Number of Requests (C++ Logistic Regression):

$$\mathrm{NR}_{\mathrm{C++}} = \frac{24 \times 60 \times 60}{0.000603} \approx 143{,}454{,}125$$

Number of Requests (Python Logistic Regression):

$$\mathrm{NR}_{\mathrm{Python}} = \frac{24 \times 60 \times 60}{0.002174} \approx 39{,}358{,}681$$

Number of Requests (C++ Decision Tree):

$$\mathrm{NR}_{\mathrm{C++}} = \frac{24 \times 60 \times 60}{0.000995} \approx 86{,}406{,}251$$

Number of Requests (Python Decision Tree):

$$\mathrm{NR}_{\mathrm{Python}} = \frac{24 \times 60 \times 60}{0.00269} \approx 31{,}958{,}577$$

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusions

Conclusions

- **C++ emerges as the clear winner** in terms of performance and efficiency, particularly for high-demand tasks requiring low-latency inference. Its superior execution speed and resource management make it the ideal choice for large-scale, time-critical applications.

- **Python remains unmatched** for ease of implementation and rapid prototyping, making it an essential tool for research and development, though it falls short in terms of raw performance compared to C++.

- **Disclosure**: It's important to recognize that while languages like C++ offer high levels of optimization and performance, they often come with a **higher cost of development**. Implementing complex algorithms in C++ generally requires more time, expertise, and resources compared to languages like Python, which offer faster development cycles at the expense of runtime performance.

Reproducibility

- **Python:** Ideal for reproducibility.
- **Matlab:** Great reproducibility tools.
- **R:** Good reproducibility.
- **Java:** Decent reproducibility via Maven, but harder to set up.
- **C++:** Difficult reproducibility due to manual configuration needs.

During our research, we encountered challenges with specific libraries in certain programming languages, which necessitated the creation of separate environments for C++ and the other languages. These issues arose primarily due to differences in library compatibility and dependencies, making it essential to isolate the C++ implementation from the others.

Future Directions

Supplementary research can focus on **deep learning tasks**, exploring how C++ and Python perform under more complex neural networks.

Testing across **other datasets** would help validate these findings, providing insights into generalization across various machine learning scenarios.

A **broader energy efficiency study** could investigate different hardware configurations and optimizations to minimize environmental impact across all languages.

Our current experiments were conducted on a **Mac ARM** platform; future tests could extend to **x64 and the new Qualcomm ARMs** to compare performance and efficiency across a wider range of architectures, ensuring broader applicability and optimization strategies.

Future studies could also examine the **trade-off between development time and runtime efficiency**, analyzing whether the additional time spent on implementation in a more optimized language like C++ justifies the performance gains, especially in production environments where time-to-market is critical.

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Thanks for the Attention!

UNIVERSITÀ DEGLI STUDI FIRENZE

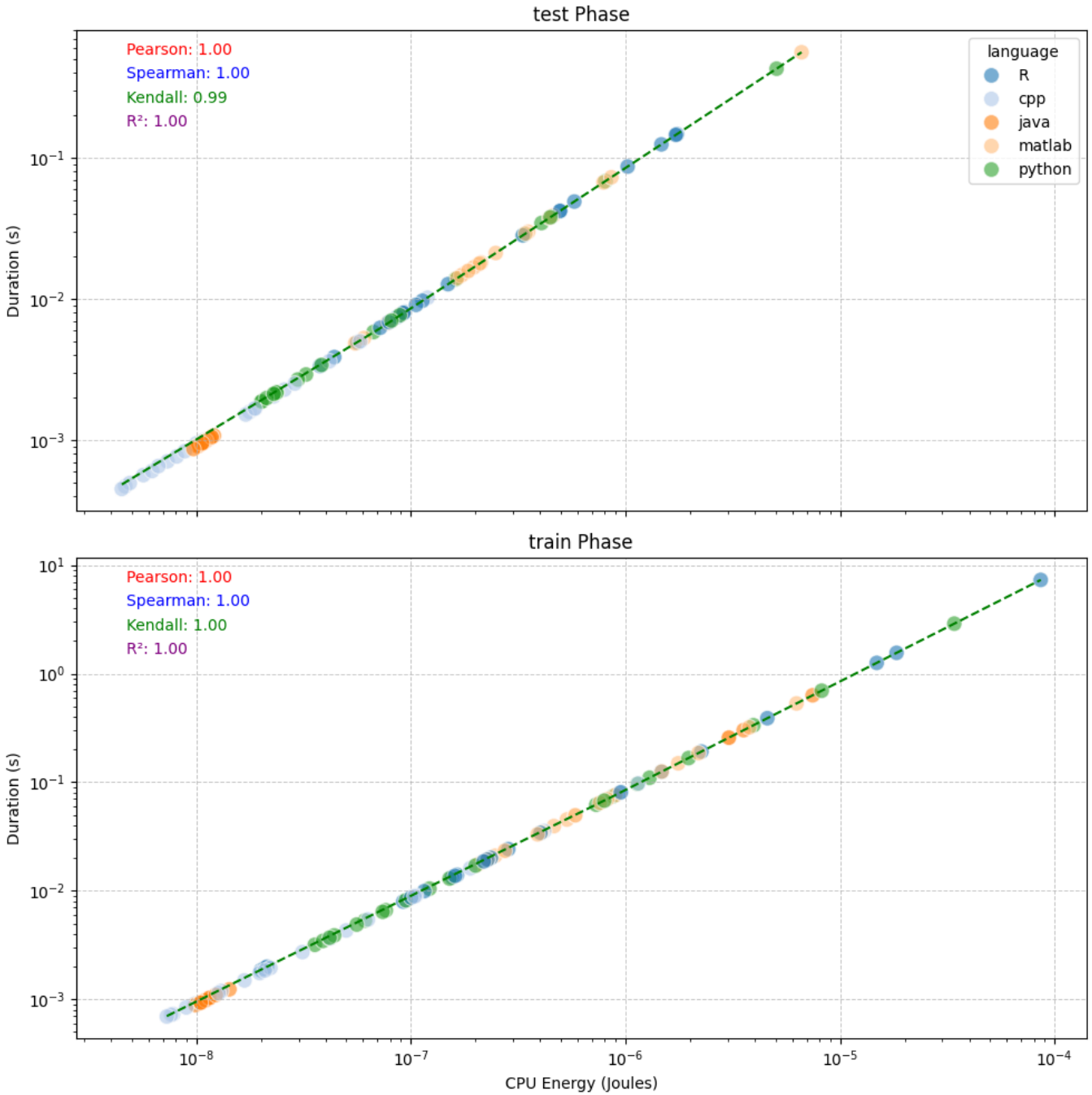
Supplementary material

UNIVERSITÀ DEGLI STUDI FIRENZE

Usefull insights

Our additional analysis focused on the correlation between energy consumption and execution time across different languages. From this, we conclude:

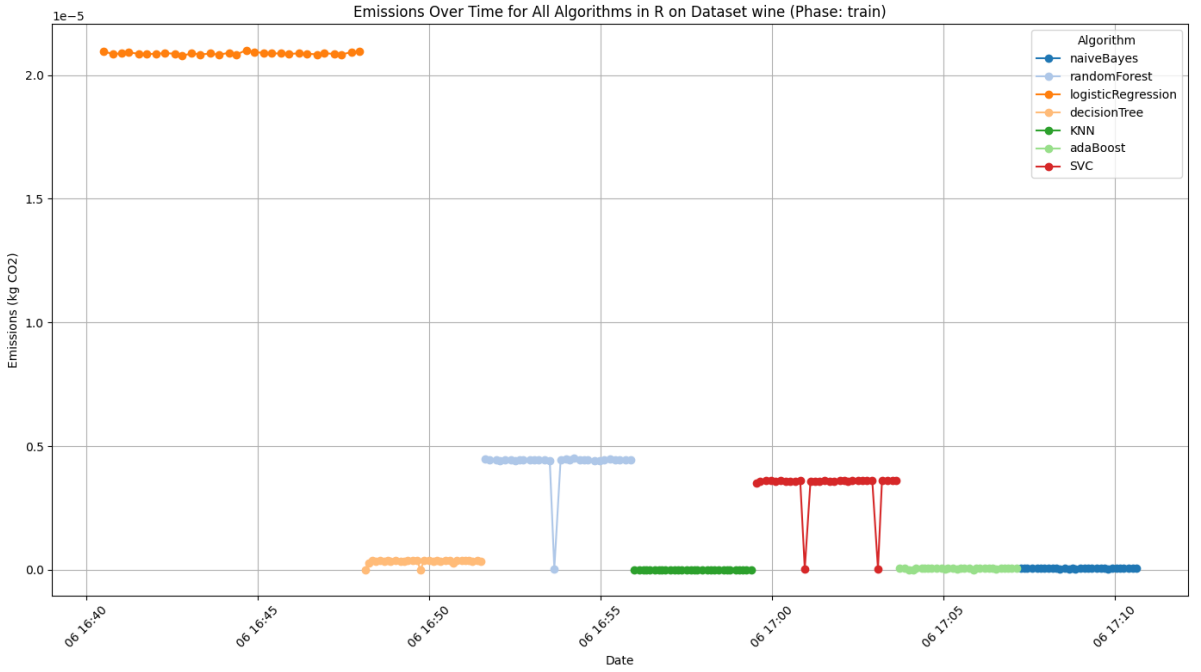
- Duration, energy consumption, and execution time exhibit a linear relationship.
- All languages are almost perfectly correlated.

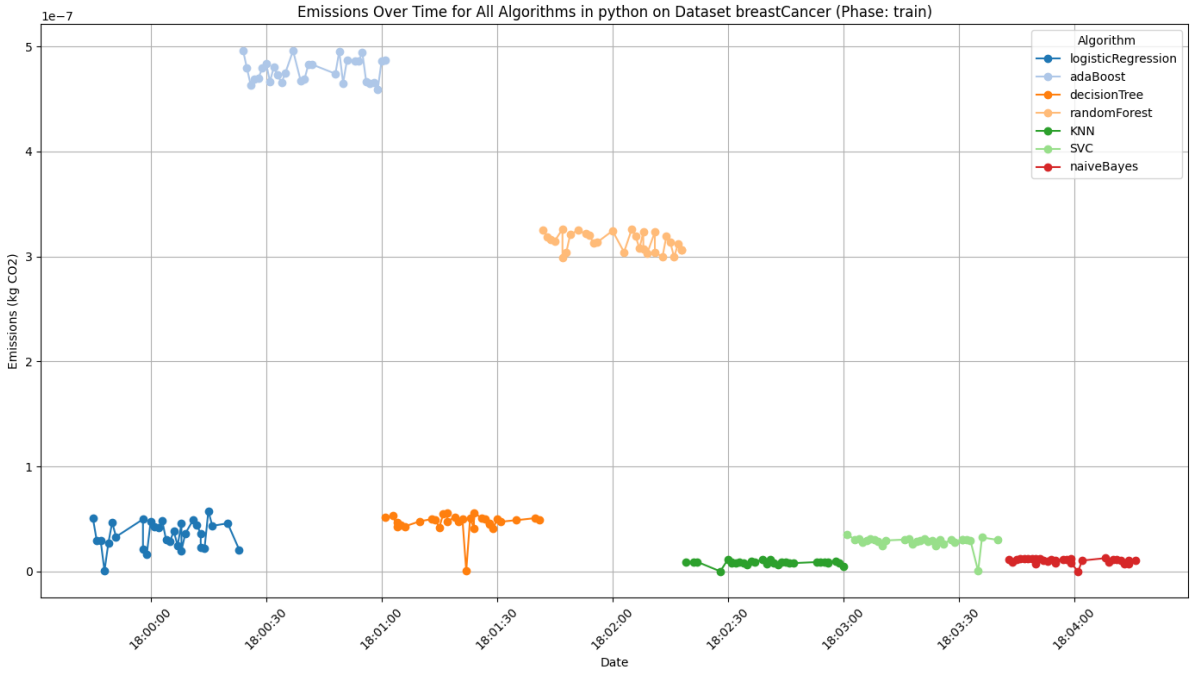


UNIVERSITÀ DEGLI STUDI FIRENZE

Timeseries

Emissions remain stable during both the training and testing phases, indicating that the machine operates consistently without throttling, which confirms that performance is not affected by hardware limitations during the simulation.





Performance tables

Language	Phase	Duration (s)	Energy Consumed (J)	Emissions (kg CO_2)	Emissions Rate	CPU Energy (J)
R	Test	0.0363	4.57×10^{-7}	9.90×10^{-8}	2.73×10^{-6}	4.27×10^{-7}
R	Train	0.5281	6.67×10^{-6}	1.49×10^{-6}	2.73×10^{-6}	6.23×10^{-6}
C++	Test	**0.0019**	**2.26** $\times 10^{-8}$	**7.19** $\times 10^{-9}$	3.67×10^{-6}	**2.13** $\times 10^{-8}$
C++	Train	0.0131	1.65×10^{-7}	5.22×10^{-8}	3.83×10^{-6}	1.54×10^{-7}
Java	Test	**0.0010**	**1.15** $\times 10^{-8}$	**2.45** $\times 10^{-9}$	2.57×10^{-6}	**1.09** $\times 10^{-8}$
Java	Train	0.0575	7.26×10^{-7}	1.63×10^{-7}	2.61×10^{-6}	6.79×10^{-7}
Matlab	Test	0.0490	6.17×10^{-7}	1.37×10^{-7}	2.74×10^{-6}	5.77×10^{-7}
Matlab	Train	0.1026	1.30×10^{-6}	2.87×10^{-7}	2.73×10^{-6}	1.21×10^{-6}
Python	Test	0.0320	4.03×10^{-7}	8.79×10^{-8}	2.64×10^{-6}	3.77×10^{-7}
Python	Train	0.2182	2.76×10^{-6}	6.15×10^{-7}	2.74×10^{-6}	2.57×10^{-6}

Algorithm	Phase	Duration (s)	Energy Consumed (J)	Emissions (kg CO_2)	Emissions Rate	CPU Energy (J)
KNN	Test	0.0192	2.41×10^{-7}	5.32×10^{-8}	2.94×10^{-6}	2.26×10^{-7}
KNN	Train	**0.0058**	**7.13** $\times 10^{-8}$	1.52×10^{-8}	2.81×10^{-6}	**6.68** $\times 10^{-8}$
SVC	Test	0.0418	5.27×10^{-7}	1.13×10^{-7}	2.83×10^{-6}	4.92×10^{-7}
SVC	Train	0.2980	3.77×10^{-6}	8.37×10^{-7}	2.99×10^{-6}	3.52×10^{-6}
adaBoost	Test	0.0214	2.69×10^{-7}	5.99×10^{-8}	2.90×10^{-6}	2.52×10^{-7}
adaBoost	Train	0.0820	1.04×10^{-6}	2.34×10^{-7}	2.98×10^{-6}	9.67×10^{-7}
decisionTree	Test	**0.0035**	**4.26** $\times 10^{-8}$	**9.48** $\times 10^{-9}$	2.81×10^{-6}	**4.01** $\times 10^{-8}$
decisionTree	Train	0.0279	3.51×10^{-7}	8.30×10^{-8}	2.91×10^{-6}	3.28×10^{-7}
logReg	Test	**0.0075**	**9.29** $\times 10^{-8}$	2.06×10^{-8}	2.79×10^{-6}	**8.70** $\times 10^{-8}$
logReg	Train	0.5843	7.38×10^{-6}	1.67×10^{-6}	2.93×10^{-6}	6.90×10^{-6}
naiveBayes	Test	0.0570	7.20×10^{-7}	1.62×10^{-7}	2.89×10^{-6}	6.72×10^{-7}
naiveBayes	Train	0.0153	1.92×10^{-7}	4.11×10^{-8}	2.88×10^{-6}	1.80×10^{-7}
randomForest	Test	0.0178	2.23×10^{-7}	4.89×10^{-8}	2.93×10^{-6}	2.09×10^{-7}
randomForest	Train	0.2741	3.46×10^{-6}	7.70×10^{-7}	2.99×10^{-6}	3.24×10^{-6}