Accuracy of Code Carbon - A2

Ahnaf Tahmid Ul Huq
2853701
VU Amsterdam
a.t.u.huq@student.vu.nl

Mhreteabe Tesfaye Dula
2853688
VU Amsterdam
m.t.dula@student.vu.nl

Nawshin Ulfat
2853706
VU Amsterdam
n.ulfat@student.vu.nl

Nimrah Yousuf
2853698
VU Amsterdam
n.yousuf@student.vu.nl

Kiran Singh Saud
2853721
VU Amsterdam
k.s.saud@student.vu.nl

1 INTRODUCTION

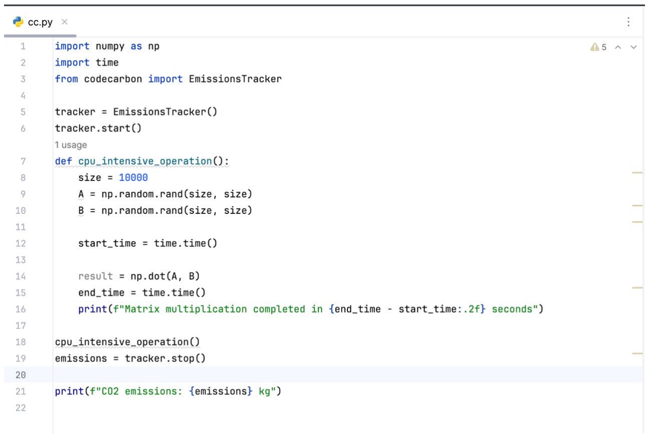
In recent years sustainability has become a very significant area in Information and Communication Technology (ICT), particularly with regard to the rise in energy-intensive operations [1], [2], [3]. It is predicted that by 2030, the ICT sector could consume up to 51% of the world's electricity and it can account for up to 23% of global greenhouse emissions [4]. As we are shifting towards sustainability, it is really important to record and report the carbon footprint of software systems and applications and to know their environmental impact.
Although hardware-based tools provide more precise energy and emissions measurements, the high price and constrained scalability make these tools less practical for general application. To mitigate this challenge, numerous software-based power meters such as Code Carbon (CC)[1] were developed to help software practitioners have a more accessible alternative. These tools assist in evaluating the environmental impact of computations utilizing power models and internal interfaces, making the process more precise for software practitioners in calculating the carbon footprint of their Python scripts and apps [5].
While these software-based solutions can help practitioners with sustainability practices, it's essential to verify the accuracy and reliability of the tools. Code Carbon [6] is comparatively new in the field of software-based power profilers and has not been addressed enough in the literature. Thus, the purpose of this paper is to evaluate how accurate Code Carbon is in estimating the carbon footprint of Python scripts. We will conduct our experiment by taking into account various factors such as workload types (machine learning problems like classification, regression,and algorithm types like supervised or unsupervised learning), CC configurations (sampling intervals and logging verbosity levels), and different types of hardware and software configurations used. Our experiment will help software practitioners select a proper power profiler for their work and allow them to contribute to sustainable computing.

2 RELATED WORK

One of the most critical issues that software practitioners face is the accurate measurement of energy consumption and the ways to

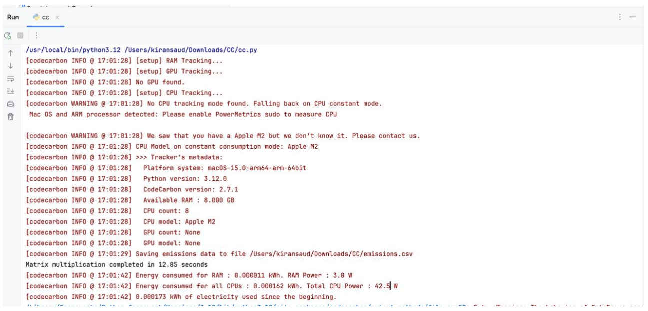
[1]https://codecarbon.io/

,
.



```python
import numpy as np
import time
from codecarbon import EmissionsTracker

tracker = EmissionsTracker()
tracker.start()
1 usage
def cpu_intensive_operation():
    size = 10000
    A = np.random.rand(size, size)
    B = np.random.rand(size, size)

    start_time = time.time()

    result = np.dot(A, B)
    end_time = time.time()
    print(f"Matrix multiplication completed in {end_time - start_time:.2f} seconds")

cpu_intensive_operation()
emissions = tracker.stop()

print(f"CO2 emissions: {emissions} kg")
```

Figure 1: Sample of CC running a simple code.



```
/usr/local/bin/python3.12 /Users/kiransaud/Downloads/CC/cc.py
[codecarbon INFO @ 17:01:28] [setup] RAM Tracking...
[codecarbon INFO @ 17:01:28] [setup] GPU Tracking...
[codecarbon INFO @ 17:01:28] No GPU found.
[codecarbon INFO @ 17:01:28] [setup] CPU Tracking...
[codecarbon WARNING @ 17:01:28] No CPU tracking mode found. Falling back on CPU constant mode.
 Mac OS and ARM processor detected: Please enable PowerMetrics sudo to measure CPU

[codecarbon WARNING @ 17:01:28] We saw that you have a Apple M2 but we don't know it. Please contact us.
[codecarbon INFO @ 17:01:28] CPU Model on constant consumption mode: Apple M2
[codecarbon INFO @ 17:01:28] >>> Tracker's metadata:
[codecarbon INFO @ 17:01:28]   Platform system: macOS-15.0-arm64-arm-64bit
[codecarbon INFO @ 17:01:28]   Python version: 3.12.0
[codecarbon INFO @ 17:01:28]   CodeCarbon version: 2.7.1
[codecarbon INFO @ 17:01:28]   Available RAM : 8.000 GB
[codecarbon INFO @ 17:01:28]   CPU count: 8
[codecarbon INFO @ 17:01:28]   CPU model: Apple M2
[codecarbon INFO @ 17:01:28]   GPU count: None
[codecarbon INFO @ 17:01:28]   GPU model: None
[codecarbon INFO @ 17:01:29] Saving emissions data to file /Users/kiransaud/Downloads/CC/emissions.csv
Matrix multiplication completed in 12.85 seconds
[codecarbon INFO @ 17:01:42] Energy consumed for RAM : 0.000011 kWh. RAM Power : 3.0 W
[codecarbon INFO @ 17:01:42] Energy consumed for all CPUs : 0.000162 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 17:01:42] 0.000173 kWh of electricity used since the beginning.
```

Figure 2: Sample of CC results after running a simple code.

reduce it. There are multiple approaches to address this issue including hardware-based tools and software-based tools. On one hand, hardware-based tools offer high accuracy in measuring energy consumption. However, they are costly, challenging to implement on a large scale, and incapable of providing measurements at the service level. On another hand, software-based tools are prone to be doubted for their reliability. Numerous tools, known as software-based power meters have been developed utilizing power models and internal interfaces to calculate energy consumption and CO2 emission focused on different implementations [7], [8], [9], [10], [11]. The development of software-based tools like Code Carbon lets software practitioners estimate the carbon dioxide (CO2) emissions generated by the cloud or personal computing resources used to run the code.

There have been comparative research experiments done on power consumption and different software-based power meters have been used till now like ML CO2 impact, Carbon Tracker, Code Carbon, and PowerAPI [12],[5]. The lack of research on Code Carbon as an individual tool makes it less likely to be chosen researchers and developers compared to mainstream tools like ML CO2 by the NLP research community. Till now the only research work by M. Jay et. al. [5] has conducted experiment on Code Carbon. This is an important point since it begs the question of the legitimacy and reliability of Code Carbon. There is a clear gap in research papers concerned with the measurement of power consumption, and carbon emissions with respect to the use of Code Carbon. M. Jay et. al. [5] did a comparative experiment on different software-based power meters focusing on CPU and GPU. Whereas our experiment will focus on Code Carbon alone considering different factors and usage scenarios.

As clearly mentioned in our goal, we aim to evaluate the accuracy of CC in the context of Python scripts. This would play a significant role in bridging the gap in research, help to know more about CC itself, and the level of acceptance that can be expected from the results produced by it. A running example of CC workflow is shown in Figure 1 and Figure 2.

3 EXPERIMENT DEFINITION

We have used the GQM framework to define our experiment and it can be seen in the following figure

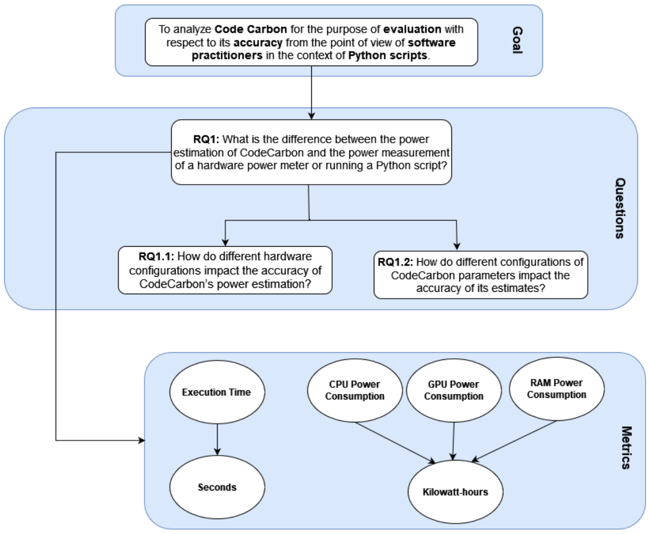

To analyze **Code Carbon** for the purpose of **evaluation** with respect to its **accuracy** from the point of view of **software practitioners** in the context of **Python scripts**.
Goal
RQ1: What is the difference between the power estimation of CodeCarbon and the power measurement of a hardware power meter or running a Python script?
Questions
RQ1.1: How do different hardware configurations impact the accuracy of CodeCarbon's power estimation?
RQ1.2: How do different configurations of CodeCarbon parameters impact the accuracy of its estimates?
Execution Time
CPU Power Consumption
GPU Power Consumption
RAM Power Consumption
Metrics
Seconds
Kilowatt-hours

Figure 3: The GQM tree followed for our experiment

3.1 Goal

The goal of this experiment is to evaluate how accurate CC is in calculating the energy consumption and carbon emissions of Python scripts. This will be achieved by running a set of Python functions for several minutes while measuring their power consumption using both Code Carbon and a hardware power meter at the same time. The experiment aims to quantify the differences between CC's estimations and the actual measurements obtained in real life from the hardware meter across various configurations, workloads, and hardware platforms.

3.2 Research Questions

[RQ1]: What is the difference between the power estimation of CodeCarbon and the power measurement of a hardware power meter for a running Python script?

This primary research question seeks to determine the overall accuracy of CodeCarbon in estimating the energy consumption associated with running Python scripts on various hardware configurations. In addition to this, we will also explore the configuration space of CC parameters and see if there are any relationship with it's measuring accuracy . By comparing CodeCarbon's energy estimates with precise measurements from a hardware power meter, we aim to assess how closely CodeCarbon's values align with actual power consumption data. This comparison will be made across different CPU types, GPU models, and RAM capacities as well as CC configurations as specified in CodeCarbon's support documentation.

Sub-questions

[RQ1.1]: How do different hardware configurations impact the accuracy of CodeCarbon's power estimation?

This sub-question explores the impact of varying hardware configurations on the accuracy of CodeCarbon's energy consumption estimates. We will focus on the supported hardware variations:

- **CPUs**: Different types (Intel, AMD, Apple Silicon) and their supported tracking methods (Intel Power Gadget for Intel CPUs on Windows and Mac, powermetrics for Apple Silicon on Mac, and Intel RAPL for Intel and AMD on Linux).
- **GPUs**: Variations in Nvidia GPU models as tracked using the pynvml library.
- **RAM**: Different capacities, evaluating how CodeCarbon's fixed estimation ratio of 3 Watts per 8 GB of RAM applies across these variations.

By conducting tests across these hardware variations with Python scripts, we will analyze if the hardware configuration leads to more accurate or less accurate estimations by CodeCarbon compared to the hardware power meter readings.

[RQ1.2] **How do different configurations of CodeCarbon parameters impact the accuracy of its estimates?**

This sub-question explores the effect of varying CodeCarbon's configuration parameters on the accuracy of its power consumption and carbon emission estimates. The parameters in focus are **measure_power_secs** (sampling interval), **log_level** (verbosity of logging), and **tracking_mode** (scope of power consumption tracking). The tracking_mode can be set to 'machine' to measure the power consumption of the entire machine, which is the default setting, or 'process' to try and isolate the power consumption of the tracked processes in isolation. By adjusting these parameters and observing the resulting estimations, we aim to identify how these configurations influence the accuracy of the measurements and whether they introduce any observational bias.

3.3 Metrics

To comprehensively address the research questions and achieve the experiment's objectives, the following metrics will be utilized: Component-Specific Energy Consumption:

CPU Energy Consumption: This will be the total energy consumed by the CPU as estimated by CC. It will be measured in kilowatt-hours.

GPU Energy Consumption: This will be the total energy consumed by GPU as estimated by CC. It will be measured in kilowatt-hours.

Memory Energy Consumption: This will be the total energy consumed by Memory as estimated by CC. It will be measured in kilowatt-hours.

Total Energy Consumption estimated: Total energy consumption will be the sum of the above three measures. It will be measured in kilowatt-hours.

Total Energy Consumption measured: Total energy consumption reported by the power meter measured in kilowatt-hours.

Execution Time: This will be the time taken for a given script to run to completion. It will be measured in seconds.

4 EXPERIMENT PLANNING

4.1 Subjects Selection

In this experiment, we aim to evaluate how accurate and reliable CC is as a tool for measuring energy consumption in Python-based data processing workflows. As CC's motivation clearly states that it focuses on energy used and the emissions produced by training and deploying AI models and tracking them across machine learning (ML) experiments as well, we have collected 20 Python scripts for ML having different properties (based on problem types and algorithms).
As for the dimensional context of our experiment, since we took existing Python scripts for ML from Github, it is considered as offline and using real objects. It is conducted by us, a group of students specifically focusing on Python scripts. Throughout the course of the experiment, consistency across runs and the accuracy of its observations under various computing demands will also be monitored.
To ensure the quality of the experiment, we have chosen the 20 Python scripts with a runtime not exceeding a maximum of 5 minutes. Since we will run each Python script multiple times to account for variability in energy consumption due to external factors such as hardware variability or background processes, this decision was made to ensure that it can be within a reasonable and achievable time frame. We chose ten different problem types from the several standard ones existing in the field of machine learning. After that, we collected the scripts and identified the algorithms and techniques that were used as well. The final collection of data can be seen in a tabular form to visualize the selection of the subjects for our experiment in the table number 1.

Problem Type	**Algorithm/ Technique**	**Selected Scripts**
Data Visualization	Matplotlib, Folium, Exploratory Analysis	05a Matplotlib Notebook.ipynb, 05b Exploring Indicator's Across Countries.ipynb, 05c Folium Notebook.ipynb
Statistical Analysis	Regression, Hypothesis Testing, Confidence Intervals	Quantile Regression.ipynb, nhanes hypothesis testing.ipynb, nhanes confidence intervals.ipynb
Natural Language Processing	LDA, Topic Modeling	LDA.ipynb, Topic_Modeling.ipynb, Using the Twitter API for Tweet Analysis.ipynb
Recommender Systems	Matrix Factorization, Bayesian Personalized Ranking	Recommender Systems Fundamentals.ipynb, mainbpr.ipynb
Regression Analysis	Regression Diagnostics	Regression Diagnostics.ipynb, G7 Countries Real Residential Property Prices.ipynb
Machine Learning Optimization	Softmax Function, Batch Processing	Softmax function.ipynb, c batched.ipynb
API and Database Interaction	Twitter API, Database Management	Using the Twitter API for Tweet Analysis.ipynb, Working with Databases.ipynb
Geospatial Analysis	Folium, Geographical Mapping	05c Folium Notebook.ipynb, FSA Matcher.ipynb
TensorFlow and Spark	TensorFlow Basics, Spark DataFrames	a tfstart.ipynb, Spark DataFrames Project Exercise Udemy.ipynb
Health and Epidemiology	Data Comparison, Vaccine Analysis	Compare Vaccines Effectiveness.ipynb

Table 1: Categorized Table of Machine Learning Scripts

By comparing the performance metrics, including execution time, CPU utilization, and memory usage, we can quantify any overhead introduced by CC. This comparison will help us determine if CC adds any significant performance cost when integrated into typical workflows.

4.2 Experimental Variables

In the following sections, we will define and explain the dependent and independent variables that we will be using for our experiment.

Independent Variables for RQ1.1

The independent variables considered in this part of experiment are:

- **CPU Types** ($c \in C$):
  - Treatments will include a range of CPU models and kinds from manufacturers such as Apple Silicon, NVIDIA, and Intel. Since CC relies on various low level power profilers to get CPU power consumption this will enable us to see how different models affect it's accuracy.
- **RAM Sizes** ($r \in R$):
  - Treatments include various RAM capacities (e.g., 8 GB, 16 GB, and 32 GB) which are explored to see how the linear relationship between energy consumption and memory size that's made by CC will affect it's accuracy of estimation.
- **GPU Models** ($g \in G$):
  - Since CC reports to works with NVIDIA GPU models, we will explore how it's accuracy changes when moving executions from CPU to GPU and vice-versa.
- **Operating Systems** ($o \in O$):
  - Operating systems can differ in how they handle hardware resources, as well as in how well they offer tools for tracking and power management. .
  - **Treatments**:
    * **Windows**:Well-known for being compatible with a large variety of devices and software, particularly in contexts involving machine learning.
    * **macOS**: Often used with Apple Silicon, which has integrated GPU capabilities and unique power tracking tools like powermetrics.
    * **Linux**: Preferred for its scalability and efficiency in managing resources, especially in server environments and extensive machine learning tasks.

Independent Variables for RQ1.2

- **Sampling Intervals** ($s \in S$):
  - Dictates how often CC will collect and record data on energy consumption.
  - **Treatments**:
    * $s = 5$ seconds: A more frequent interval that could capture rapid changes in energy usage more accurately.
    * $s = 15$ seconds (default): The standard interval set by CC, used as the baseline for comparisons.
    * $s = 30$ seconds: A less frequent interval, which might miss shorter bursts of energy usage but could reduce the overhead of data collection.
- **Logging Verbosity Levels** ($l \in L$):
  - Determines the amount of detail provided in the logs generated by CC, which could influence computational overhead and potentially affect energy consumption readings.
  - **Treatments**:
    * ld = debug: Provides the most detailed logs, potentially increasing computational load and affecting the accuracy of energy consumption estimates.
    * li = info (default): Standard level of detail, offering a balance between detail and performance.
    * lw = warning: Logs significant events only, possibly lowering overhead compared to more verbose settings.
    * le = error: Logs only serious errors, further reducing computational demand.
    * lc = critical: The least verbose setting, minimizing logging overhead.
- **Tracking Modes** ($m \in M$):
  - Specifies the scope of energy consumption tracking in CC.
  - **Treatments**:
    * m = machine (default): Measures the energy consumption of the entire machine.
    * m = process: Attempts to isolate the energy consumption of the tracked Python processes only.

Dependent Variables for RQ1.1 and RQ1.2

The primary dependent variable for both RQ1.1 and RQ1.2 is **energy consumption** (E), measured in two ways:

(1) **Estimated Energy Consumption** (E_{cc}):
  - Calculated using CC by summing the energy used by the CPU, GPU, and RAM.
  - Expressed in kilowatt-hours (kWh), these estimates will allow us to assess how well CC predicts energy usage under different configurations.

(2) **Measured Energy Consumption** (E_{pm}):
  - Directly measured using a hardware power meter to provide a real-world baseline for comparison.
  - Provides empirical data on the actual energy consumption during the experiments to evaluate the accuracy of CC's estimates.

4.3 Experimental Hypotheses

In this section, we have formulated the appropriate hypothesis that corresponds to our research questions. The variables used in our hypothesis are defined in the sections preceding the hypotheses.

For the research question **RQ1.1** , we constructed the following hypotheses:

Variable definitions:

- $E_{\text{cc}}(c, r, g, o)$: Estimated energy consumption by CC under configuration with:
  - $c \in C$: CPU types (e.g., Intel, NVIDIA, Apple Silicon)
  - $r \in R$: RAM sizes (e.g., 8 GB, 16 GB)
  - $g \in G$: GPU models (e.g., various NVIDIA GPUs)
  - $o \in O$: Operating Systems (e.g., Windows, MacOS, Linux)
- $E_{\text{pm}}(c, r, g, o)$: Measured energy consumption using a power meter under the same configuration.

Null Hypothesis((H_0):

$\text{H}_0 : \forall c \in C, \forall r \in R, \forall g \in G, \forall o \in O, \quad E_{\text{cc}}(c, r, g, o) = E_{\text{pm}}(c, r, g, o)(1)$

The null hypothesis assumes that the energy consumption estimated by CodeCarbon is equal to the actual energy consumption measured by power meters across all hardware configurations.

Alternative Hypothesis (H_A):
$\text{H}_A : \exists(c, r, g, o) \in C \times R \times G \times O \quad \text{such that} \quad E_{\text{cc}}(c, r, g, o) \neq E_{\text{pm}}(c, r, g, o)$(2)
The alternative hypothesis suggests that there is at least one hardware configuration where CodeCarbon's estimated energy consumption does not match the actual energy consumption measured by power meters.

For the research question **RQ1.2** , we constructed the following hypotheses :

Variable Definitions:

- $E_{\text{cc}}(s, l, m)$: Estimated energy consumption by CodeCarbon with:
  - $s \in S$: Sampling intervals $\{5\,\text{sec},\ 15\,\text{sec},\ 30\,\text{sec}\}$
  - $l \in L$: Logging verbosity levels $\{\text{debug},\ \text{info}$ $,\ \text{warning},\ \text{error},\ \text{critical}\}$
  - $m \in M$: Tracking modes $\{\text{machine},\ \text{process}\}$
- $E_{\text{pm}}(s, l, m)$: Measured energy consumption using a power meter under the same conditions.

Null Hypothesis (H_0):

$$H_0 : \forall s \in S,\ \forall l \in L,\ \forall m \in M, \quad E_{\text{cc}}(s, l, m) = E_{\text{pm}}(s, l, m) \tag{3}$$

This hypothesis proposes that variations in the configuration parameters of CodeCarbon (sampling intervals, logging levels, tracking modes) do not affect the accuracy of its estimated energy consumption when measured against actual measurements on power meter.

Alternative Hypothesis (H_A):

$$H_A : \exists(s, l, m) \in S \times L \times M \quad \text{such that} \quad E_{\text{cc}}(s, l, m) \neq E_{\text{pm}}(s, l, m) \tag{4}$$

The alternative hypothesis states that certain configurations of CodeCarbon's parameters may lead to discrepancies between estimated and measured energy consumption.

4.4 Experiment Design

We have designed our experiment as a one factor- multiple treatments (1F-MT) case, where the main factor is the hardware resources and the treatments are CPU type, RAM size, and GPU model. Regarding our experiment, we have taken into account various types of operating systems (OS). Because CC is unable to directly access hardware resources, it relies on the OS to measure energy usage, which is the rationale behind choosing it as a cofactor rather than a blocking factor. Here, we have considered three of the most popular OS, Windows, MacOS, and Linux.
To get the most out of the experiment, we have applied the principles of randomization, blocking, and balancing as well. As for randomization, we have applied partial randomization where we selected 20 Python scripts out of 60 scripts based on criteria. Additionally, the treatments will be executed in a random order.
We have chosen the hardware device as the blocking factor for our experiment. It will divide the data into two groups; laptops and microprocessors. These blocks will help us understand how various hardware manage energy usage under comparable computational demands. We will be using Windows laptop, Mac laptop, and Jetson Nano development kit as the hardware device platform.
To ensure the balancing aspect, we are going to use the same set of Python scripts for all treatments. This guarantees that the same computational tasks are applied to every treatment, resulting in a balanced comparison of energy consumption among various platforms.
As for the trial repetition, we have set the value 15 upon calculating the estimated runtime for the overall experiment. A maximum execution time of forty hours has been considered. We have presumptively determined that each script will run on average for three minutes. The cooling time between runs has been set at one minute, and the trials will be conducted using six distinct devices. With 6 distinct devices (5 laptops, 1 Jetson Nano development kit), 2 types of OS for any device, and 20 Python scripts, the total execution time:
$(6\ \text{device types} \times 2\ \text{OS} \times 20\ \text{scripts} \times 15\ \text{repetitions} \times 3\ \text{min})/6\ \text{devices} = (2160\ \text{min}/60) = 36\ \text{hours}$

4.5 Data Analysis

5 EXPERIMENT EXECUTION

In this section, we will present all the information that is necessary to replicate our experiment. This includes the specification of the devices that will be used to run the experiments as well as the measurement tools that will be used to perform the measurements. In addition, we will specify the specific CC parameters that we will use in our experiments. The structure of the run table which contains all the columns that are used to record our results is also linked.

5.1 Devices Setup

We will utilize six distinct devices in our experiment with their varying specifications ranging from Intel(R) Core(TM) i5 6200U CPU with Ubuntu and Windows to more powerful devices like MacOS with an 8-core Apple M2 chip. The table 2 lists the available devices with their specifications.

5.2 Measurement Tools

For this experiment, we will be utilizing two key tools to measure the total energy consumption during each run: **CodeCarbon** and the **Monsoon High Voltage Power Monitor**(Model No. AA10F). The Monsoon High Voltage Power Monitor is a specialized hardware device designed to accurately measure power consumption in electronic devices, particularly useful in scenarios where precision and real-time energy tracking are required. To facilitate communication with the power meter, we will employ a software application called **PowerTool**, which is provided by Monsoon. PowerTool is a software that provides a graphical user interface (GUI) that offers a comprehensive visual representation of the energy consumption data as measured by the Power Monitor. This interface will allow us to monitor and analyze the energy usage of our device in real time during the experimental runs, ensuring accurate and detailed energy profiling.

Device ID	CPU	RAM	GPU	Operating System
1	Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz	8GB	N/A	Ubuntu, Windows 10 Pro
2	Intel(R) Core(TM) i7-13700H CPU @ 2.40GHz	16GB	N/A	Ubuntu, Windows 10 Home
3	12th Gen Intel(R) Core(TM) i7-1255U CPU @ 1.70 GHz	16GB	7.8 GB	Ubuntu, Windows 11 Pro
4	macOS 15.0 8-core CPU Apple M2 CHIP	8GB	8-core	macOS
5	Intel(R) Core(TM) i7-13650HX @ 2.6 GHz	16GB	7.8 GB	Ubuntu, Windows 11 Home
6	CPU Quad-core ARM Cortex-A57 MPCore processor	4GB	128-cores	Linux based OS

Table 2: Specifications of available devices

5.3 Experimental Procedure

In this section, we will meticulously go through the actions that we will perform when making our measurements.

5.3.1 ***Initialization.*** We begin by selecting each device from the device_table. Since each devices comes installed with various operating systems we will run our experiments on each OS for that specific device.
Before running any of the scripts we will perform a calibration test by running an empty script and measuring its energy consumption. After that, we will run every script on each machine while varying the configurations of CC.

To remind our readers here is the list of sampling intervals, logging levels, and tracking modes that will be used in our experiments:

- **Sampling Intervals**: [5 seconds, 15 seconds (default), 30 seconds]
- **Logging Levels**: [debug, info (default), warning, error, critical]
- **Tracking Modes**: [machine (default), process]

5.3.2 ***Script Execution.*** For each combination of device and operating system, all scripts from the script_table are executed. Each script is run through 15 execution cycles to capture a comprehensive dataset on performance and energy usage.

5.3.3 ***Measurement of Energy Metrics.*** During each script run, the following two sets of measurements are taken to assess the device's performance:

- **Measured Energy Consumption**: Directly recorded using a Power Meter. Each measurement, referred to as pm_result, provides real-time data on energy consumption.
- **Estimated Energy Consumptions**: Calculated using the CodeCarbon tool. This advanced tool not only estimates the total energy consumption but also delineates the energy use specifics of the CPU, GPU, and RAM.

5.3.4 ***Data collection and Export.*** The measurements from both the Power Meter and CodeCarbon are meticulously recorded and appended to our run table after each script run. The run order in the run table has been randomized to minimize bias and lessen the impact of uncontrollable variables. This strategy distributes all possible confounding influences evenly, improving the validity of statistical analyses and ensuring the reliability of the experiment result.
The structure and detailed format of our run table can be viewed at this link.

6 TIME LOG

Time log report: Link

REFERENCES

[1] M. E. Bildirici, R. A. Castanho, F. Kayıkçı, and S. Y. Genç, "Ict, energy intensity, and co2 emission nexus," *Energies*, vol. 15, no. 13, p. 4567, 2022.
[2] K. Dong, J. Wang, and F. Taghizadeh-Hesary, "Assessing the embodied co2 emissions of ict industry and its mitigation pathways under sustainable development: A global case," *Applied Soft Computing*, vol. 131, p. 109760, 2022.
[3] N. Jones *et al.*, "How to stop data centres from gobbling up the world's electricity," *nature*, vol. 561, no. 7722, pp. 163–166, 2018.
[4] A. S. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.
[5] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: focus on cpu and gpu," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023, pp. 106–118.
[6] "CodeCarbon.io — codecarbon.io," https://codecarbon.io/, [Accessed 13-09-2024].
[7] L. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: quantifying the carbon footprint of computation," *Advanced science*, vol. 8, no. 12, p. 2100707, 2021.
[8] "GitHub - GreenAlgorithms/GreenAlgorithms4HPC — github.com," https://github.com/GreenAlgorithms/GreenAlgorithms4HPC, [Accessed 13-09-2024].
[9] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," *arXiv preprint arXiv:2007.03051*, 2020.
[10] T. Trébaol, "Cumulator—a tool to quantify and report the carbon footprint of machine learning computations and communication in academia and healthcare," 2020.
[11] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *arXiv preprint arXiv:1910.09700*, 2019.
[12] N. Bannour, S. Ghannay, A. Névéol, and A.-L. Ligozat, "Evaluating the carbon footprint of nlp methods: a survey and analysis of existing tools," in *Proceedings of the second workshop on simple and efficient natural language processing*, 2021, pp. 11–21.