ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Intelligent Systems

HARDWARE DIMENSIONING FOR ENVIRONMENTAL SUSTAINABILITY: BENCHMARK OF AI ALGORITHMS AND ENVIRONMENTAL IMPACT

CANDIDATE
Enrico Morselli

SUPERVISOR
Prof. Andrea Borghesi

CO-SUPERVISOR
Allegra De Filippo, PhD.

Academic year 2024-2025

Session 5th

dedicated(X) :- friend(X).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Rationale

In recent years we have witnessed a dramatic increase in the performance of Artificial Intelligence technologies. Even if AI still fails to exeed human ability in some complex cognitive tasks, as of 2023 it has surpassed human capabilites in a range of tasks, such as image classification, basic reading comprehension, visual reasoning and natural language inference [15]. Not to mention the astonishing results achieved by Generative AI in tasks as Image and Video Generation [15]. This great advances in performance were made possible by a massive upscale of model sizes and computational resources ("compute" in short) dedicated to training state-of-the-art AI models. Research shows that for frontier AI models (i.e. those that were in the top 10 of training compute when they were released), the training compute has grown by a factor of 4-5x/year since 2010 [18]. This surge in required compute has driven a corresponding spike in energy consumption for AI, and consequently, an higher environmental impact due to CO_2 emissions. For instance, for trainining their LLaMA models, Meta AI researchers have estimated a period of approximately 5 months of on 2048 $\mathrm{A100}$ $80\mathrm{GB}$ GPUs, resulting in a total of $2,638\mathrm{MWh}$ of energy and a total emission of $1,015$ $\mathrm{tCO_2eq}$ [21]. Given the widespread application, the steep increase in model size and complexity and

the crescent energy requirements for AI applications, the Carbon Footprint of AI has become a growing concern in the context of the current climate emergency.

In this work, we will explore an approach for addressing the issue of AI sustainability, by means of HADA (HArdware Dimensioning for AI Algorithms), which is a framework that uses ML to learn the relationship between an algorithm configuration and performance metrics, like total runtime, solution cost and memory usagem and then uses Optimization to find the best Hardware architecture and its configuration to run an algorithm under required performances and budget limits which is the problem known as Hardware Dimensioning [5]. What we will do is to extend this framework in order to consider also the performance of the algorithms in terms of Energy consumption and Carbon Emissions, so that, ideally, we could find the best Algorithm and Hardware configuration that reduces the Carbon Footprint of computation. We will then proceed to test this approach on some small-scale algorithms that we could easily execute in a timely manner on local machines and HPC clusters.

The rest of the work is structured as follows:

- **Chapter 2** Introduces Related works that addressed the issue of AI's carbon footprint, and how Carbon Footprint is determined
- **Chapter 3** Introduces some theoretical background about HADA, and explains the integration of the new metrics.
- **Chapter 4** Presents the experimental setup and the results of the experiments
- **Chapter 5** Presents the HADA framework, providing an overview of the tool
- **Chapter 6** Presents the conclusions

Chapter 2

Related Works

2.1 Sustainability in AI

The environmental impact of training large AI models is illustrated by recent empirical assessments. Strubell et al. (2019) quantified the CO2 emissions of several NLP models and found that training a big transformer with extensive hyperparameter tuning (including neural architecture search) emitted roughly 626,000 pounds of CO_2 - about the same as the lifetime emissions of five cars [19] [6]. These findings brought attention to the fact that accuracy gains in AI often come at a steep energy and carbon price. In response, researchers have begun to sistematically reporting energy use and consequent CO_2 emissions of model training to raise awareness [6] [16]. Beyond individual models, broader studies have examined AI's total energy and environmental footprint across the industry. Henderson et al. (2020) [10] introduced a framework for tracking real-time energy consumption and carbon emissions during ML experiments, encouraging researchers to include these metrics in publications. Their work underscored that transparent reporting is essential for understanding and ultimately reducing AI's climate impacts. Industry-scale analyses also reveal sobering trends. Gupta et al. (2021) [9] analyzed the end-to-end footprint of computing and found that while operational emissions (from running hardware) have been partly curbed by efficiency improvements, the overall

carbon footprint of computing continues to grow due to increased scale. Notably, they showed that for modern data centers and mobile devices, manufacturing and infrastructure (embodied carbon) now account for the majority of emissions. In other words, as data centers adopt cleaner power, the emissions "hidden" in hardware supply chains (chip fabrication, server manufacturing, etc.) become a dominant concern. Similarly, Wu et al. (2022) [22] present a holistic study of AI at a large tech company (Meta/Facebook), examining the entire AI model lifecycle - from data processing and training to inference and hardware lifecycle. They report super-linear growth in AI workloads and infrastructure: for instance, daily inference operations doubled over a recent 3-year span, forcing a 2.5x expansion in server capacity. Crucially, Wu et al. also highlight that embodied carbon is an increasing fraction of AI's total footprint, echoing that improvements in hardware efficiency alone cannot eliminate AI's impact. Their analysis argues for looking beyond training alone - considering data center construction, supply chains, and the frequency of model retraining - to truly grasp AI's environmental impact.

A recurring theme in these studies is the diminishing return on energy investment for AI model improvements. As models get larger and more complex, the incremental accuracy gains often require disproportionately more compute (and thus energy). Schwartz et al. (2020) [17] dub the status quo "Red AI," where researchers prioritize accuracy at almost any computational cost, and they note this trend is unsustainable both environmentally and even economically. Thompson et al. (2021) [20] similarly observed that progress in benchmarks was coming with exponentially increasing computing cost, warning of diminishing returns and calling the situation unsustainable. Another challenge is equitable access: massive energy requirements make cutting-edge AI research expensive, potentially concentrating it in wealthy institutions and regions with robust infrastructure. This raises concerns that AI's growing energy hunger not only harms the planet but also exacerbates inequalities in who can afford to do top-tier research. These concerns have prompted calls

for a paradigm shift toward “Green AI”, where efficiency and sustainability are treated as primary goals in model development. Several studies have addressed the issue of AI’s carbon footprint.

2.2 Tools for tracking Carbon Emissions

The growing awareness about AI’s Carbon Footprint also motivated the development of dedicated tools and methodologies to monitor the carbon footprint of AI workloads. A number of open-source tools and frameworks have been created to help practitioners measure the energy consumption and CO_2 emissions of their code. ML CO_2 Impact (Machine Learning Emissions Calculator), introduced by Lacoste et al. (2019), [12] was one of the early tools for estimating emissions from model training. It is a web-based calculator where users input information about their training run - such as hardware type (CPU/GPU), runtime, cloud provider, and location - and it computes the energy consumed and corresponding carbon emissions. The calculator leverages known power draw profiles of hardware and regional carbon intensity factors. According to [1] the latest incarnation of this tool has evolved under the umbrella of the CodeCarbon initiative, integrating its functionality into CodeCarbon’s open-source package.

CodeCarbon [2], which is the tool we used to expand HADA in this work, is an open-source Python package for tracking the carbon footprint of computing projects. It integrates into ML code to log the resources used (CPU, GPU, etc.) and estimates the CO_2 emissions produced by the workload. Uniquely, CodeCarbon accounts for the location of the computation - using region-specific electricity carbon intensity data - to provide location-dependent emission estimates. This allows developers to see, for instance, that running the same training job on a low-carbon grid (e.g. hydro-powered Montreal) results in a much smaller footprint than running it on a coal-heavy

grid. The goal is to inform and incentivize researchers and engineers to optimize or relocate their workloads to reduce emissions.

Considerable work has been done in the field of Large Language Models (LLMs), which are the models that have had the most dramatic growth in complexity and size.

There exists also other works that tried to address not only the emissions resulting from model's training and inference, but also those emissions resulting from the manifacturing of the Hardware components, which are *embodied* in the total Carbon Footprint. A work in this direction is [8], that provides a model to estimate the total emissions of an LLM both in terms of *Operational* Carbon Footprint and *Embodied* Carbon Footprint. This work also deploys considerations about the Data Center Energy Efficency - expressed by the Power Usage Efficency (PUE) - and the Carbon Intensity of the Electricity used, reinforcing the observations by CodeCarbon, i.e. that performing the computations in places where the energy is cleaner can significantly reduce CO_2 emissions.

Chapter 3

Metodology

3.1 Empirical Model Learning in HADA

As we mentioned in the introduction, the main focus of this work is Hardware Dimensioning, i.e. the problem of finding the right Hardware configuration to run the desired algorithm under required performance and budget limits. More specifically, we are given a target algorithm, and we would like to know what is the optimal settings of Algorithm Hyperparameters and Hardware Architecture to run the algorithm under a set of constraints in terms of performance requirements. This is not a trivial problem to model, owing to the complexity of knowing beforehand the performance of an AI algorithm on different HW architectures and evaluating the effect of different HW and algorithm configurations. The idea behind HADA is to integrate the domain knowledge held by experts (time constraints, required solution quality, budget limits) with data-driven models. The basis of HADA is the **Empirical Model Learning (EML)**, [14] framework, which uses Machine Learning (ML) techniques to learn the model of an Optimisation problem. The idea is to learn, rather than directly express, complex relationships between algorithm performance and HW resources via ML models, using *empirical* knowledge. These models are then embedded in the optimisation problem. In general, this approach is useful

whenever the problem to solve using optimisation is very complex and therefore not trivial to derive a model to describe it. Broadly speaking, EML deals with solving declarative optimisation models with a complex component, h, which represents the relation between variables which can be acted upon (the decision variables, also called *decidables*) x and the *observables* y related to the system considered; the function $h(x) = y$ describes this relationships. Since the h component is complex and we cannot optimise directly over it, we exploit empirical knowledge to build a surrogate model h_θ learned from data, where θ is the parameter vector.

We present here the general mathematical formulation of EML:

$$\min \quad f(x, y) \tag{3.1}$$

$$\text{s.t.} \quad h_\theta(x) = y \tag{3.2}$$

$$g_j(x, y) \quad \forall j \in J \tag{3.3}$$

$$x_i \in D_i \quad \forall x_i \in x \tag{3.4}$$

where x is the decision variables vector (each x_i has its own domain D_i) and y is the vector of observed variables. The goal is to minimize the objective function $f(x, y)$, which might depend on both decision and observed variables. Both decision and observed variables can be subject to a set of constraints $g_j(x, y)$, such as classical inequalities from Mathematical Programming and combinatorial predicates from Constraint Programming (e.g., global constraints). The function $h_\theta(x)$ represents the approximate behaviour of the complex relation between decision and observed variables, and it is, in practice, the encoding of a ML model.

The surrogate model $h_\theta(x)$ is trained as in the typical Supervised Learning setting; EML requires a training set $\mathcal{S} = (x_i, h(x_i))_{i=1}^m$ which is then used to find the parameter vector θ minimizing a loss function:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} L(h_\theta(x_i), y_i^*) \tag{3.5}$$

where y_i^* are the ground-truth labels (or targets, in case of regression) of each data point in the training set $\mathcal{S}$ and L is the loss function (e.g., L_1 or L_2 loss for scalar y's).

The way in which HADA works can then be summarized in three main phases:

1. *Data Set Collection* - an initial phase to collect the data set $\mathcal{S}$ by running multiple times the target algorithms, under different configuration, which we call the *Benchmark* phase. This implies running the algorithms with different configurations on different Hardware platforms;

2. *Surrogate Model Creation* - once a training set is available, a set of ML models is then trained on such data. These are the h_θ surrogate models. Then, these models are encoded as a set of variables and constraints following the EML paradigm;

3. *Optimization* - post the user-defined constraints and objective function on top of the combinatorial structure formed by the encoded ML models and the domain-knowledge constraints, and finally solve the optimization model (either until an optimal solution or a time limit is reached).

The creation of the *Surrogate Model* and the *Optimization* phase are already implemented by the HADA prototype that will be presented in chapter 5. Our main goal is then to run a set of algorithms under different configurations, in terms of hyper-parameters and Hardware platform, recording a series of metrics that describe Resource Usage, Energy Consumption and Carbon Emissions (i.e. the Carbon Footprint), which will be our targets for the surrogate models. First, we need to understand how the Carbon Footprint of an algorithm is measured.

3.2 Carbon Footprint of Computation

The Carbon Footprint of an algorithm depends on two factors: the energy needed to run it and the pollutants emitted when producing such energy. The former depends on the computing resources used (e.g. number of cores, running time, data centre efficiency) while the latter, called carbon intensity, depends on the location and production methods used (e.g. nuclear, gas or coal). The Carbon Intensity is expressed in terms of carbon dioxide equivalent (CO_2e), and summarises the global warming effect of the Greenhouse Gases (GHG) emitted in the determined timeframe [13]. According to [2], the carbon footprint can then be computed with the following formula:

$$C = E \times CI \tag{3.6}$$

Where:

- E is the energy consumed by the computational infrastructure: quantified as kilowatt-hours (kWh).
- CI is the Carbon Intensity of the electricity consumed for computation: quantified as g of CO_2e emitted per kilowatt-hour of electricity.

Codecarbon directly measures the energy consumption of the CPU, GPU and RAM on which the code is executed at intervals of 15 sec by default. It also monitor the duration of code execution to compute the total electricity consumption. Then it retrieves information about the carbon intensity of the electricity in your geographic area based on the location. Carbon Intensity of the consumed electricity is calculated as a weighted average of the emissions from the different energy sources that are used to generate electricity, including fossil fuels and renewables. In CodeCarbon, the fossil fuels coal, petroleum, and natural gas are associated with specific carbon intensities: a known amount of carbon dioxide is emitted for each kilowatt-hour of electricity generated. Renewable or low-carbon fuels include e.g. solar power,

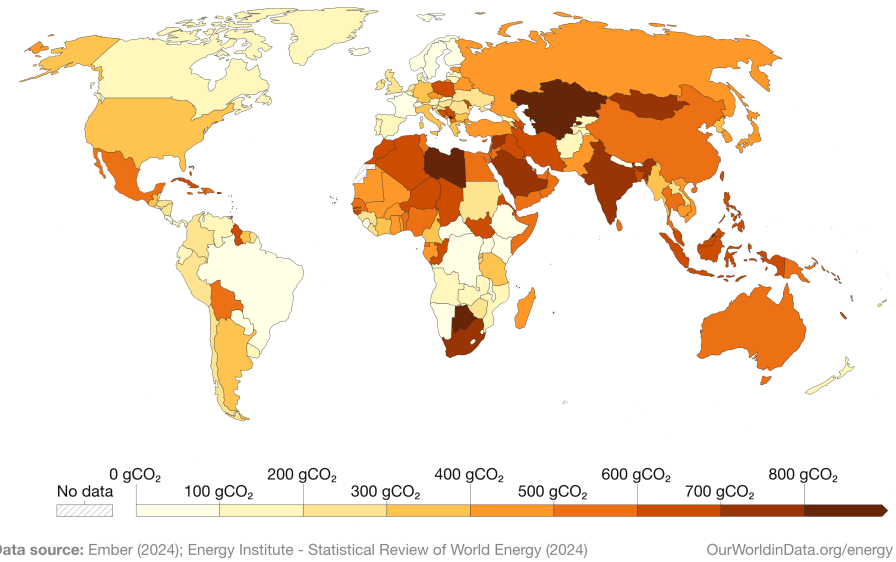

Figure 3.1: Carbon intensity of electricity generation, 2023. Source [7]

hydroelectricity, biomass, geothermal. Based on the mix of energy sources in the local grid, this package calculates the Carbon Intensity of the electricity consumed. The sources for Carbon Intensity data can vary depending on the platform and availability. For example, on private infrastructures, CodeCarbon uses data from Our World In Data [7]. Figure 3.1 visualizes data about the Carbon Intensity of each country as of 2023.

3.3 Extending HADA with CodeCarbon

3.3.1 Energy Management Algorithms

The first step to extend HADA with Carfon Intensity is therefore to gather a dataset which includes data about carbon footprint of the algorithms execution, integrating CodeCarbon in code for running our algorithms. The first two algorithms that were analysed are ANTICIPATE and CONTINGENCY, introduced in [3] [4] and used in the HADA paper to present the framework [5]. These are two Stochastic Algorithms from the energy management system domain, which calculate the amount of energy that must be produced by the

energy system to meet the required load, minimising the total energy cost over the daily time horizon and by taking into account the uncertainty. In particular, ANTICIPATE is an online (scenario-based) anticipatory algorithm, while CONTINGENCY is an integrated offline/online algorithm that tackles the uncertainty by building (offline) a pool of solutions to guide an efficient online method. Both these methods are characterized by a parameter that can be changed based on the required constraints in terms of time constraint and required solution quality (resp., the number of scenarios for ANTICIPATE, and the number of traces for CONTINGENCY). The goal of HADA is to learn the relationship between this configuration parameter and a series of performance metrics which describes the Resource Usage of the Algorithm and the quality of the solution. The dataset used originally in HADA then includes the following attributes:

- `nParam`: Value of the algorithm configuration parameter, i.e. `nScenarios` For ANTICIPATE and `nTraces` for CONTINGENCY;
- `time(s)`: Time required to find a solution;
- `sol(keuro)`: Cost of the solution found (expresses solution quality);
- `memAvg(MB)`: Average memory used by the algorithm.

As already mentioned, we would like to extend this work by considering also the algorithm performance in terms of Energy Consumption and Carbon Footprint. We then added the following metrics taken from CodeCarbon:

- `emissions`: the total emissions of CO_2e (kg)
- `emission_rate`: the amount of CO_2e emissions per second (kg/s);
- `cpu_energy`: the energy consumed by the cpu;
- `ram_energy`: the energy consumed by the ram;

- `tot_energy`: the total energy consumed;
- `country` and `region`: the country and region where the computation took place;
- `cpu_count`: the number of cores.

In addition to these, we also implemented the tracking of the peak memory usage (which we called `memPeak(MB)` in the dataset).

3.3.2 Min-Cut/Max-Flow Algorithms

After ANTICIPATE and CONTINGENCY, that were already included in HADA, we decided that it would be interesting to further extend HADA with new algorithms. We opted for a set of algorithms used to solve Minimum Cut/Maximum Flow problems on Graphs, that are used for a4paper number of Computer Vision tasks. We reffered to the work of Jensen et al. (2023) [11], that offers a review of Min-Cut/Max-Flow state-of-the-art algorithms, evaluated on a large Dataset of Computer Vision problems. In particular, we focused on three algorithms:

- Boykov-Kolmogorov (BK)
- Excess Incremental Breadth First Search (EIBFS)
- Hochbaum's Pseudo Flow (HPF)

Accoring to Jensen et al. there are different families of Min-Cut/Max-Flow algorithms. BK belongs to the so called Augmenting Paths (AP) family, which is the oldest one, introduced with the Ford-Fulkerson Algorithm [insert citation]. Instead, EIBFS and HPF are from the family of the so called *Pseudoflow* algorithms [insert citation]. The main differences are the order in which they scan through nodes when looking for an arc connecting two trees in the forest, and how they push flow along the paths.

Differently from ANTICIPATE and CONTINGENCY, these algorithms does not have a configuration parameter. Since these three algorithms can be implemented in different variants, we decided to treat the algorithm's implementation type as a configuration parameter. The variants considered are the following:

- For Boykov-Kolmogorov we considered:
  - The reference implementation (BK);
  - The implementation by Jensen et al. (MBK), with several optimisations, such as using indices instead of pointers to reduce memory footprint;
  - A second implementation by Jensen et al. (MBK-R) which reorders arcs, so that all outgoing arcs from a node are adjacent in memory.
- For Excess Incremental Breadth First Search we considered:
  - A slightly modified version of original algorithm (EIBFS);
  - A second version replacing pointers with indices (EIBFS-I);
  - A third version without arc reordering (EIBFS-I-NR).
- For Hochbaum's Pseudo Flow we considered:
  - Highest label variant, with FIFO buckets (HPF-H-F);
  - Highest label variant, with LIFO buckets (HPF-H-L);
  - Lowest label variant, with FIFO buckets (HPF-L-F);
  - Lowest label variant, with LIFO buckets (HPF-L-L).

Jensen et al. provided a series of scripts to evaluate the runtime of those algorithms - considering both the initialization time and the time required to come up with a solution. In order to monitor our target metrics, we wrapped

the algorithms scripts - which are implemented in C++ code - with a Python script that implements memory usage tracking and uses CodeCarbon to monitor Energy Usage and Carbon Emissions. Differently from ANTICIPATE and CONTINGENCY here we do not take the solution quality into consideration.

Chapter 4

Experimental Analysis

4.1 Benchmarking on Different Hardware Platforms

For the benchmark phase, we opted for running the algorithms on different Hardware Platforms, in order to better capture the differences in performance when changing the machine on which the algorithm is executed:

Experiments were conducted on:

- `mbp19`: A Laptopr running MacOS Ventura 13.6.7 with:
  - 1,4 GHz Quad-Core Intel Core i5;
  - 8 GB 2133 MHz LPDDR3;
- `PC`: A PC with Ubuntu 22.0.4 LTS
- `leonardo` An HPC infrastructure, i.e. Leonardo hosted by CINECA. It has a total of 4992 computing nodes organized in two partitions, the Booster partition and the Data Centric General Purpose (DCGP) partition. For running the benchmark phase we used the booster partition, which has the following specifics:
  - 8×64 GB DDR4 3200 MHz (512 GB)

– single socket 32-core Intel Xeon Platinum 8358 CPU, 2.60GHz (Ice Lake) (110592 cores).

– 4x NVIDIA custom Ampere A100 GPU 64GB HBM2e, NVLink 3.0 (200GB/s)

ANTICIPATE and CONTINGENCY were executed on 30 instances which were sampled with a statistical model with a Gaussian distribution [5]. On each instance, each algorithm was executed with parameter (i.e. number of traces or scenarios) values ranging from 1 to 100. For ANTICIPATE and CONTINGENCY we used only `mbp19` and `leonardo`. Thus in the end we obtained a dataset with a total of $30 \times 100 \times 2 \times 2 = 12,000$ entries.

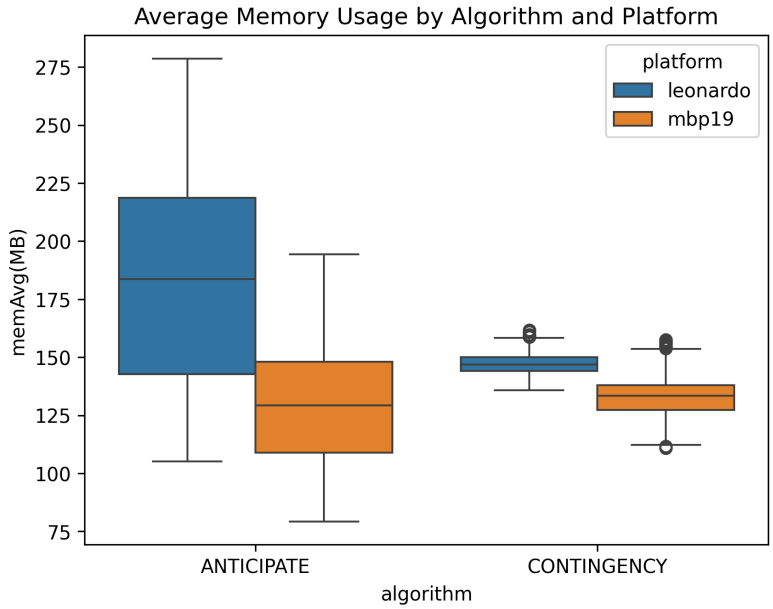

Figure 4.1: Average memory usage of ANTICIPATE and CONTINGENCY divided by platform

As we can see in [old matrix], the average and peak memory usage has the highest correlation with the Carbon Emissions of the algorithms. In particular memAvg has a positive correlation of 0.87, and memPeak of 0.69. As we could expect, these two metrics are highly correlated with one another, with a score of $.80$. Then, `totEnergy` has a correlation of 1 with CO2e, meaning

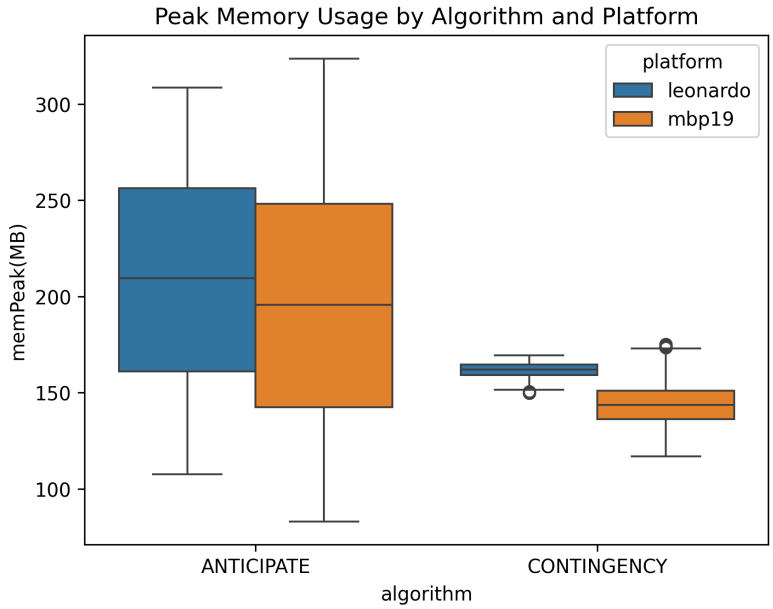

Figure 4.2: Peak memory usage of ANTICIPATE and CONTINGENCY divided by platform

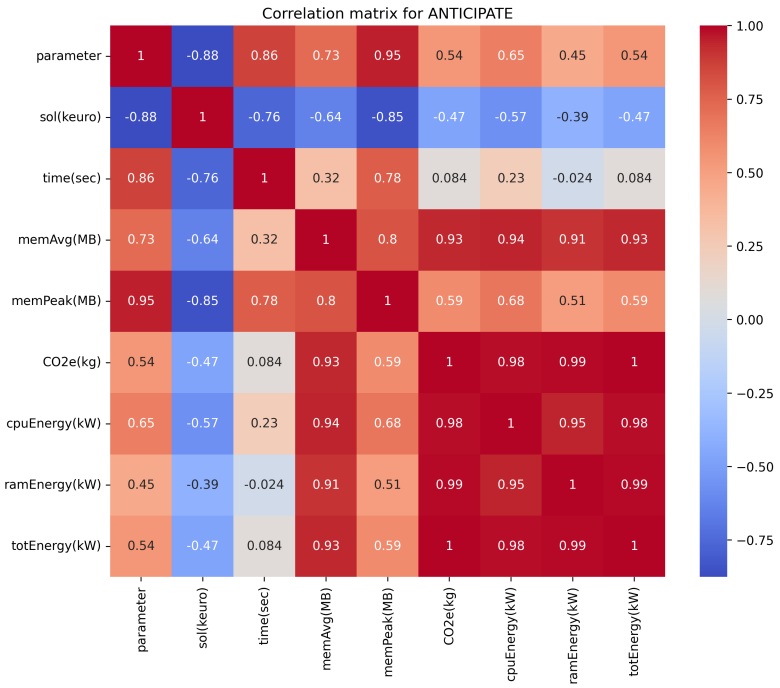

Figure 4.3: Correlation matrix for ANTICIPATE

they are perfectly proportional to each other. This reflects the fact that Carbon Emissions, as measured by CodeCarbon, depends by two factors, i.e. the energy consumed by running the algorithm and the Carbon Intensity. Since

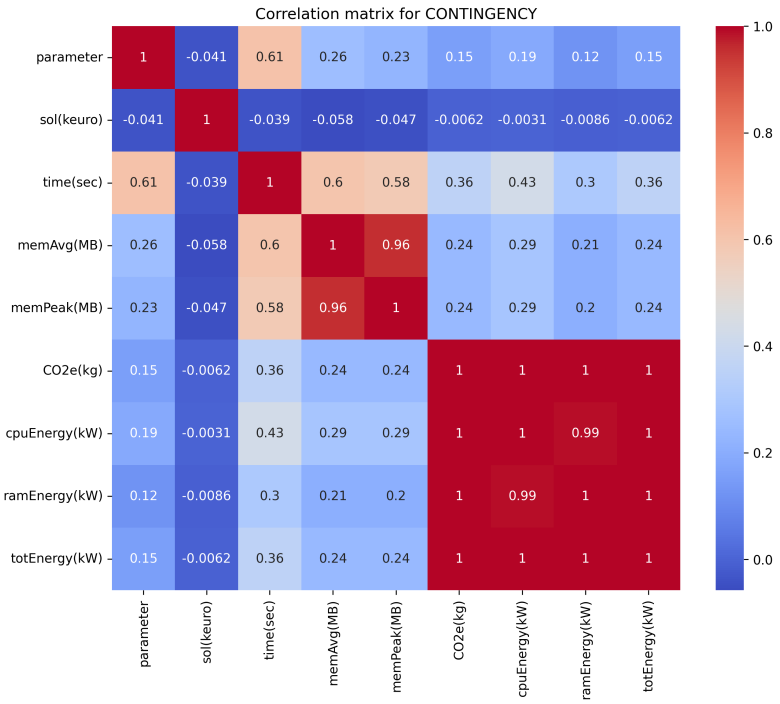

Figure 4.4: Correlation matrix for CONTINGENCY

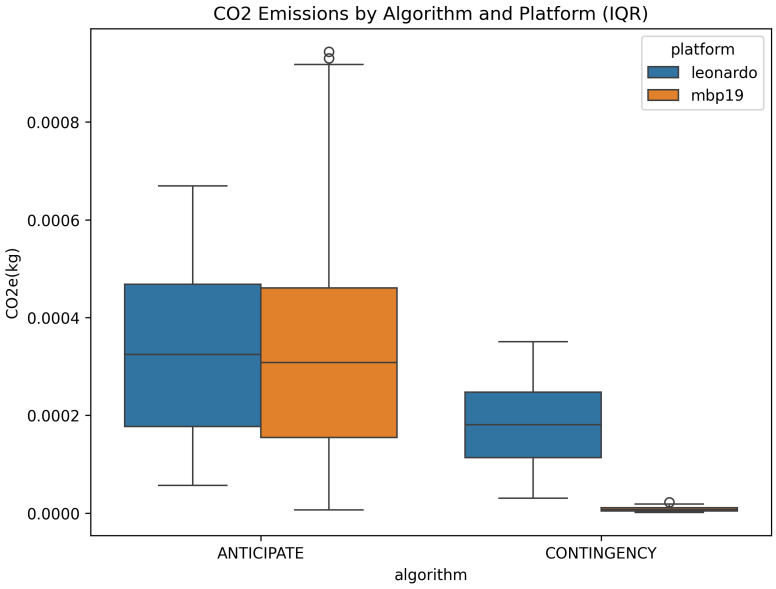

Figure 4.5: CO2 emissions for ANTICIPATE and CONGINGENCY ordered by platform

we executed all the experiments in the same country, the Carbon Intensity over the whole dataset is fixed, meaning that the only factor influencing the CO2e emissions is the energy consumed, while the Carbon Intensity is just a

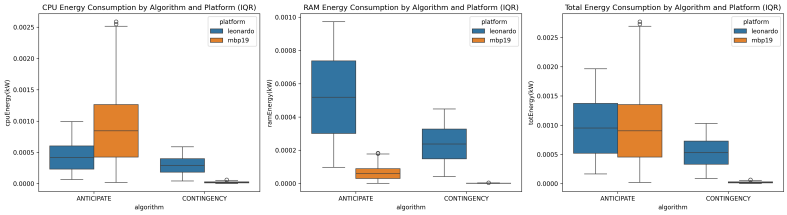

Figure 4.6: Energy Consumption data for ANTICIPATE and CONTINGENCY

scaling factor. Indeed, `totEnergy` and `CO2e` both the same correlation values with the other variables. Then, as one may point out, we could just remove the energy consumption values from the dataset, and just keep the emissions. However, the energy consumption values can be scaled by different Carbon Intensity values, giving the possibility of evaluating the impact of changing the location of the computations, as we will see later on. Lastly, regarding the solution cost (which is inversely proportional to the solution quality), we can point out that the factor which has the most impact on it is the average memory, having a negative correlation of -0.47. Between the solution cost and the emissions, the correlation is a bit weaker, being -0.36. We could then perform some tests with HADA to asses the impact of constraints on memory and solution quality on the carbon emissions and viceversa.

Here we can see some results given by the HADA framework tested on the updated dataset for ANTICIPATE and CONTINGENCY. 4.3 presents the results of optimising any target without any constraints for ANTICIPATE. We can notice how, for most metrics keeping the number of scenarios as low as possible is the best choice, while increasing them brings the solution cost down.

In 4.3 we can see some the results of some combination of constraints on Solution cost (which is expressed €), and Average Memory consumption (expressed in MB), with the objective of minimizing the CO2e emissions (in kg). The rows filled with ”none” indicates that HADA didn’t found a solution satisfying those constraints for the given algorithm.

Objective	Solution		
Target	n^P	HW	Value
CO2e (kg)	1	mbp19	7.23×10^{-6}
Cost (€)	97	mbp19	$267.39k$
Time (s)	1	mbp19	1.24
Avg. Memory (MB)	1	mbp19	81.47
Peak Memory (MB)	2	mpb19	92.19
Energy (kW)	1	mbp19	2.12

Table 4.1: Minimum values for each target for the ANTICIPATE algorithm

Objective	Solution		
Target	n^P	HW	Value
CO2e (kg)	2	mbp19	1.28×10^{6}
Cost (€)	90	mbp19	$314.11k$
Time (s)	1	mbp19	5.16
Avg. Memory (MB)	1	mbp19	117.66
Peak Memory (MB)	2	mbp19	130.58
Energy (kW)	2	mbp19	3.77×10^{-6}

Table 4.2: Minimum values for each target for the CONTINGENCY algorithm

Bounds		Solution				
Memory	Cost	n^P	HW	CO2e	Memory	Cost
no	no	1	mbp19	7.23×10^{-6}	-	-
80	no	none	none	none	none	none
no	100	none	none	none	none	none
100	300	none	none	none	none	none
150	300	64	mbp19	3.75×10^{-4}	141.53	$299.46k$
170	270	97	mbp19	5.70×10^{-4}	169.85	267.39k
170	280	87	mbp19	5.10×10^{-4}	160.12	276.86k
300	500	1	mbp19	7.23×10^{-6}	81.47	368.73k

Table 4.3: Experimental results for the ANTICIPATE algorithm with bounds on the runtime

We'll now see some experiments on CONTINGENCY.

Next, we take a look at the correlation values for the Min-cut/Max-flow algorithms

Bounds		Solution				
Memory	Cost	n^P	HW	CO2e	Memory	Cost
no	no	2	mbp19	1.29×10^{-6}	-	-
100	no	none	none	none	none	none
no	100	none	none	none	none	none
150	400	2	mbp19	1.29×10^{-6}	119.80	380.10

Table 4.4: Experimental results for the CONTINGENCY algorithm with bounds on the runtime

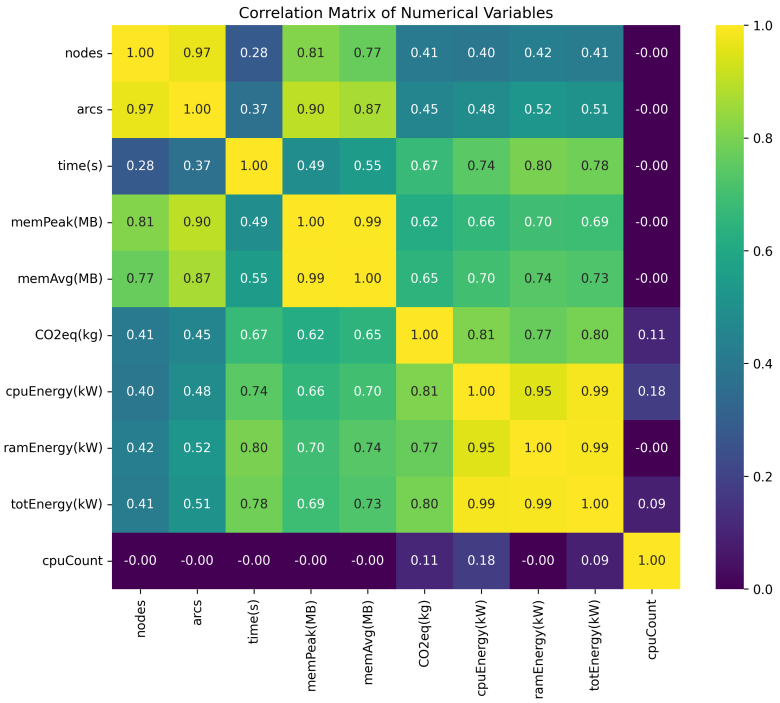

Figure 4.7: Correlation matrix for Min-Cut/Max-Flow Algorithms BK, EIBFS and HPF

Chapter 5

HADA-as-a-Service

5.1 HADA Web Application

Benchmark data was integrated into the HADA web application, requiring:

- Creation of JSON configuration files for each algorithm-hardware combination.
- Specification of hyperparameters and performance targets.

Example JSON structure:

```
{
"name": "anticipate",
"HW_ID": "macbook",
"hyperparams": [
{"ID": "num_scenarios", "type": "int", "LB": 1, "UB": 100}
],
"targets": [
{"ID": "time", "LB": null, "UB": null},
{"ID": "memory", "LB": null, "UB": null},
{"ID": "emissions", "LB": null, "UB": null}
]
```

```
}
```

Chapter 6

Conclusions

This work extends HADA by integrating carbon emission constraints, enhancing its applicability for sustainable AI hardware selection. Through experimental benchmarks on laptops and HPC systems, we validated the framework's ability to balance performance and environmental impact. The web-based prototype enables users to make informed decisions when configuring AI workloads under sustainability constraints.

Bibliography

[1] N. Bannour et al. “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*. Virtual: Association for Computational Linguistics, 2021, pp. 11–21. DOI: 10.18653/v1/2021.sustainlp-1.2. URL: https://aclanthology.org/2021.sustainlp-1.2/.

[2] B. Courty et al. *mlco2/codecarbon: v2.4.1*. Version v2.4.1. 2024. DOI: 10.5281/zenodo.11171501. URL: https://doi.org/10.5281/zenodo.11171501.

[3] A. De Filippo, M. Lombardi, and M. Milano. “How to Tame Your Anticipatory Algorithm”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 1071–1077. DOI: 10.24963/ijcai.2019/150. URL: https://doi.org/10.24963/ijcai.2019/150.

[4] A. De Filippo, M. Lombardi, and M. Milano. “Off-Line and On-Line Optimization Under Uncertainty: A Case Study on Energy Management”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings*. Delft,

The Netherlands: Springer-Verlag, 2018, pp. 100–116. ISBN: 978-3-319-93030-5. DOI: 10.1007/978-3-319-93031-2_8. URL: https://doi.org/10.1007/978-3-319-93031-2_8.

[5] A. De Filippo et al. "HADA: An automated tool for hardware dimensioning of AI applications". In: *Knowledge-Based Systems* 251 (2022), p. 109199. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2022.109199. URL: https://www.sciencedirect.com/science/article/pii/S0950705122005974.

[6] J. Dodge et al. "Measuring the Carbon Intensity of AI in Cloud Instances". In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '22. Seoul, Republic of Korea: Association for Computing Machinery, 2022, pp. 1877–1894. ISBN: 9781450393522. URL: https://doi.org/10.1145/3531146.3533234.

[7] Ember, E. Institute, and O. W. in Data. *Carbon Intensity of Electricity Generation – Ember and Energy Institute*. Dataset. Major processing by Our World in Data. Based on Ember's "Yearly Electricity Data" and Energy Institute's "Statistical Review of World Energy". 2024. URL: https://ourworldindata.org/grapher/carbon-intensity-electricity (visited on 03/06/2025).

[8] A. Faiz et al. *LLMCarbon: Modeling the end-to-end Carbon Footprint of Large Language Models*. 2024. arXiv: 2309.14393 [cs.CL]. URL: https://arxiv.org/abs/2309.14393.

[9] U. Gupta et al. "Chasing Carbon: The Elusive Environmental Footprint of Computing". In: *CoRR* abs/2011.02839 (2020). arXiv: 2011.02839. URL: https://arxiv.org/abs/2011.02839.

[10] P. Henderson et al. "Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning". In: *Journal of Machine*

Learning Research 21.248 (2020), pp. 1–43. URL: http://jmlr.org/papers/v21/20-312.html.

[11] P. M. Jensen et al. "Review of Serial and Parallel Min-Cut/Max-Flow Algorithms for Computer Vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.2 (2023), pp. 2310–2329. DOI: 10.1109/TPAMI.2022.3170096.

[12] A. Lacoste et al. "Quantifying the Carbon Emissions of Machine Learning". In: *arXiv preprint* (2019). arXiv: 1910.09700. URL: https://arxiv.org/abs/1910.09700.

[13] L. Lannelongue, J. Grealey, and M. Inouye. "Green Algorithms: Quantifying the Carbon Footprint of Computation". In: *Advanced Science* 8.12 (2021), p. 2100707. DOI: https://doi.org/10.1002/advs.202100707. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/advs.202100707. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/advs.202100707.

[14] M. Lombardi, M. Milano, and A. Bartolini. "Empirical decision model learning". In: *Artificial Intelligence* 244 (2017). Combining Constraint Solving with Mining and Learning, pp. 343–367. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2016.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0004370216000126.

[15] N. Maslej et al. *The AI Index 2024 Annual Report*. Tech. rep. AI Index Steering Committee, Stanford University, 2024.

[16] D. A. Patterson et al. "Carbon Emissions and Large Neural Network Training". In: *CoRR* abs/2104.10350 (2021). arXiv: 2104.10350. URL: https://arxiv.org/abs/2104.10350.

[17] R. Schwartz et al. "Green AI". In: *CoRR* abs/1907.10597 (2019). arXiv: 1907.10597. URL: http://arxiv.org/abs/1907.10597.

[18] J. Sevilla and E. Roldán. *Training Compute of Frontier AI Models Grows by 4-5x per Year*. Accessed: 2025-03-03. 2024. URL: https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year.

[19] E. Strubell, A. Ganesh, and A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243. URL: http://arxiv.org/abs/1906.02243.

[20] N. Thompson et al. "Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable". In: *IEEE Spectrum* 58 (2021), pp. 50–55. ISSN: 0018-9235.

[21] H. Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[22] C. Wu et al. "Sustainable AI: Environmental Implications, Challenges and Opportunities". In: *CoRR* abs/2111.00364 (2021). arXiv: 2111.00364. URL: https://arxiv.org/abs/2111.00364.

Acknowledgements

I’m very grateful to the inventor of the Prolog language, without whom this thesis couldn’t exist. I’d also like to acknowledge my advisor Prof. Mario Rossi by tail-recursively acknowledging my advisor.