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
Prof. Allegra De Filippo

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

1.2 Introduction

In recent years, we have witnessed a dramatic improvement in the performance of Artificial Intelligence (AI) technologies. While AI still falls short of human ability in some complex cognitive tasks, as of 2023, it has surpassed human capabilities in various domains, including image classification, basic reading comprehension, visual reasoning, and natural language inference [16]. Moreover, Generative AI has achieved astonishing results in areas such as image and video generation, further demonstrating the rapid advancements in the field [16].

These remarkable gains in AI performance have been primarily driven by the large-scale expansion of model sizes and computational resources (often referred to as ”compute”) dedicated to training state-of-the-art AI models. Research indicates that, for frontier AI models—those that ranked among the top 10 in terms of training compute at the time of their release—the amount of compute used for training has been increasing at an exponential rate, growing by a factor of 4-5x per year since 2010 [20].

However, this surge in compute requirements has led to a corresponding increase in energy consumption, which, in turn, has amplified the environmental impact of AI due to CO_2 emissions. For instance, the training of Meta AI's LLaMA models was estimated to take approximately five months on a cluster of 2048 A100 80GB GPUs, consuming a total of $2,638$ MWh of energy and producing approximately $1,015$ tCO_2eq in emissions [23]. Given the growing adoption of AI technologies, the rapid increase in model size and complexity, and the escalating energy demands of AI applications, the carbon footprint of AI has become an increasingly pressing concern, particularly in the context of the ongoing climate emergency.

In this work, we explore a novel approach to addressing the sustainability challenges of AI by extending the HADA (HArdware Dimensioning for AI Algorithms) framework. HADA is a framework that leverages Machine Learning (ML) to learn the relationship between an algorithm's configuration and its performance metrics—such as total runtime, solution cost, and memory usage—and then employs optimization techniques to identify the optimal hardware architecture and configuration that meet predefined performance and budget constraints. This process, known as Hardware Dimensioning, has been studied as a means to efficiently allocate computational resources while maintaining performance requirements [5].

Our contribution extends HADA by incorporating energy consumption and carbon emissions as additional performance metrics. By doing so, we aim to identify the best combination of algorithm and hardware configuration that minimizes the environmental impact of computation. To validate our approach, we will conduct experiments on small-scale algorithms that can be executed in a timely manner on local machines and high-performance computing (HPC) clusters. Ultimately, we seek to demonstrate how AI workloads can be optimized not only for efficiency and cost but also for sustainability, thus contributing to the broader goal of reducing AI's carbon footprint.

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 has been underscored by recent empirical assessments. Strubell et al. (2019) quantified the CO_2 emissions of several Natural Language Processing (NLP) models and discovered that training a large transformer with extensive hyperparameter tuning, including neural architecture search, emitted approximately 626,000 pounds (around 284 metric tons) of CO_2—comparable to the lifetime emissions of five cars [21]. Another study found out that GPT-3 released 552 metric tons of CO_2 into the atmosphere during training, the equivalent of 123 gasoline-powered passenger vehicles driven for one year [17]. These findings highlight that improvements in AI accuracy often come at a significant energy and carbon cost.

In response to these concerns, researchers have begun systematically reporting energy usage and the resulting CO_2 emissions associated with model training to raise awareness [6, 17]. Transparent reporting is essential for understanding and ultimately mitigating AI's climate impacts. For instance, Henderson et al. (2020) introduced a framework for tracking real-time energy consumption and carbon emissions during Machine Learning (ML) experiments, encouraging researchers to include these metrics in their publications

[10].

Broader studies have examined AI's total energy and environmental footprint across the industry. Gupta et al. (2021) 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. As data centers adopt cleaner power, the emissions embedded in hardware supply chains—such as chip fabrication and server manufacturing—become a dominant concern. [9]

Similarly, Wu et al. (2022) conducted a comprehensive study of AI at a large technology company (Meta/Facebook), examining the entire AI model lifecycle—from data processing and training to inference and hardware lifecycle. They reported super-linear growth in AI workloads and infrastructure; for instance, daily inference operations doubled over a recent three-year span, necessitating a 2.5x expansion in server capacity. Crucially, Wu et al. also highlighted 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. [24]

A recurring theme in these studies is the diminishing return on energy investment for AI model improvements. As models become larger and more complex, incremental accuracy gains often require disproportionately more compute power, and thus energy. Schwartz et al. (2020) termed this phenomenon "Red AI," where researchers prioritize accuracy regardless of computational cost, and noted this trend is unsustainable both environmentally and economically [19]. Thompson et al. (2021) similarly observed that progress in benchmarks was accompanied by exponentially increasing computing costs,

warning of diminishing returns and calling the situation unsustainable. [22]

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 demands not only harm the planet but also exacerbate inequalities in who can afford to conduct 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.

In summary, the environmental sustainability of AI has become a critical area of concern. As the field advances, it is imperative to balance the pursuit of performance improvements with the need to minimize energy consumption and carbon emissions. This balance is essential not only for mitigating climate change but also for ensuring equitable access to AI research and development.

2.2 Tools for Tracking Carbon Emissions

The increasing awareness of AI's carbon footprint has motivated the development of dedicated tools and methodologies for monitoring the environmental impact of AI workloads. A number of open-source tools and frameworks have been introduced to help practitioners measure the energy consumption and CO_2 emissions of their code, enabling more informed decisions about sustainability in AI research and deployment.

One of the early tools for estimating emissions from model training was **ML CO2 Impact** (Machine Learning Emissions Calculator), introduced by Lacoste et al. (2019) [13]. This web-based calculator allows users to input key information about their training runs—such as hardware type (CPU/GPU), runtime, cloud provider, and location—and computes the corresponding energy consumption and carbon emissions. The tool relies on known power draw profiles of hardware components and regional carbon intensity factors

to generate estimates. According to Bannour et al. (2021) [1], the latest iteration of this tool has evolved under the umbrella of the **CodeCarbon** initiative, integrating its functionality into the open-source CodeCarbon package.

CodeCarbon [2], which is the tool we employ in this work to enhance HADA, is an open-source Python package designed to track the carbon footprint of computing projects. It integrates seamlessly into ML workflows, logging resource usage (CPU, GPU, and other components) and estimating the CO_2 emissions produced by the workload. A key feature of CodeCarbon is its ability to account for the geographic location of the computation—leveraging region-specific electricity carbon intensity data to provide location-dependent emission estimates. **Carbon Intensity** refers to the amount of CO_2 emitted per unit of electricity generated, typically measured in grams of CO_2 per kilowatt-hour (gCO_2/kWh). This metric varies significantly across regions depending on the energy mix used for electricity generation. For example, a region relying primarily on coal-fired power plants will have a much higher carbon intensity compared to one powered predominantly by renewable sources such as hydropower or wind energy. By incorporating this factor, CodeCarbon enables users to assess and compare the emissions impact of running their workloads in different locations. This means that shifting computations to regions with lower-carbon grids can lead to substantial reductions in total CO_2 emissions. This feature allows researchers and engineers to assess the impact of their computational choices; for example, running the same training job on a hydro-powered grid (e.g., in Montreal) results in significantly lower emissions than executing it on a coal-reliant grid. By making these comparisons explicit, CodeCarbon aims to inform and incentivize practitioners to optimize or relocate their workloads in a manner that reduces emissions.

Considerable efforts have also been made to quantify the carbon footprint of **Large Language Models (LLMs)**, as these models have experienced the most dramatic growth in both complexity and size. While much attention has been given to the energy demands of training these models, recent studies

have also highlighted the substantial energy costs incurred during inference—the phase where models generate responses based on input data. Husom et al. (2024) introduced MELODI (Monitoring Energy Levels and Optimization for Data-driven Inference), a comprehensive framework designed to monitor and analyze energy consumption during LLM inference processes [11]. MELODI enables detailed observations of power consumption dynamics and facilitates the creation of a dataset reflective of energy efficiency across varied deployment scenarios. Their findings indicate substantial disparities in energy efficiency, suggesting ample scope for optimization and the adoption of sustainable measures in LLM deployment. Similarly, Samsi et al. (2023) conducted experiments to study the computational and energy utilization of LLM inference, focusing on different sizes of LLaMA—a state-of-the-art LLM developed by Meta AI—on two generations of popular GPUs (NVIDIA V100 and A100) [18]. They benchmarked inference performance and energy costs across diverse tasks and presented results of multi-node, multi-GPU inference using model sharding across up to 32 GPUs. Their work provides valuable insights into compute performance and energy utilization characteristics of LLM inference, highlighting the need for cost-saving measures, efficient hardware usage, and optimal deployment strategies. Faiz et al. (2024) [8], introduces a model to estimate the total emissions of an LLM, accounting for both its *Operational* Carbon Footprint (energy consumption during training and inference) and its *Embodied* Carbon Footprint (emissions from hardware production and infrastructure). This study also highlights the importance of **Data Center Energy Efficiency**, measured by the *Power Usage Effectiveness (PUE)*, and the **Carbon Intensity of Electricity**—factors that reinforce the findings of CodeCarbon. Specifically, the study emphasizes that performing AI computations in regions where energy sources are cleaner (e.g., renewable-powered grids) can significantly reduce CO_2 emissions.

The development and adoption of these tracking tools play a crucial role in promoting sustainable AI research and deployment. By quantifying emissions

and identifying optimization strategies, these tools provide the foundation for a more environmentally responsible approach to AI development.

Chapter 3

Methodology

3.1 Empirical Model Learning in HADA

As introduced earlier, the core focus of this work is **Hardware Dimensioning**, which refers to the challenge of identifying the most suitable hardware configuration for executing a given algorithm while meeting predefined performance and budget constraints. Specifically, given a target algorithm, our goal is to determine the optimal combination of algorithmic hyperparameters and hardware architecture that ensures the algorithm operates efficiently under specific performance requirements. However, this is a non-trivial optimization problem, primarily due to the difficulty of predicting an AI algorithm's performance across different hardware architectures and evaluating the impact of varying algorithmic and hardware configurations.

The key idea behind HADA is to integrate expert domain knowledge, such as execution time constraints, required solution quality, and budget limitations, with data-driven models that can infer performance relationships. At the foundation of HADA lies the **Empirical Model Learning (EML)** framework [15], which employs Machine Learning (ML) techniques to construct a model of an optimization problem. Instead of explicitly formulating complex relationships between algorithm performance and hardware resources,

EML leverages data-driven approximations, learning these dependencies empirically. These learned models are then embedded directly within the optimization process.

This approach is particularly advantageous in scenarios where deriving an exact mathematical formulation for the problem is infeasible due to its complexity. Broadly, EML facilitates the resolution of declarative optimization models that involve an intricate component, denoted as h, which captures the relationship between *decision variables* (also called *decidables*) x and *observables* y—i.e., measurable performance metrics of the system. This relationship is represented by the function $h(x) = y$. Since h is often highly complex and not directly optimizable, an alternative approach is to learn an approximate model, denoted as h_θ, where θ represents the set of learned parameters.

3.1.1 Mathematical Formulation

The general mathematical formulation of the EML framework can be expressed as follows:

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

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

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

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

where:

- x represents the vector of decision variables, each x_i belonging to a domain D_i;
- y denotes the vector of observed variables;
- The objective function $f(x, y)$, which depends on both decision and observed variables, is to be minimized;

- Constraints $g_j(x, y)$ enforce feasibility conditions on the variables, which can include classical mathematical programming inequalities and combinatorial constraints from Constraint Programming;
- The function $h_\theta(x)$ serves as an approximation of the complex relationship between decision and observed variables, and it is instantiated as a trained ML model.

The surrogate model $h_\theta(x)$ is learned through a standard supervised learning procedure. Given a training set $\mathcal{S} = \{(x_i, h(x_i))\}_{i=1}^m$, the goal is to determine the parameter vector θ that minimizes a loss function L:

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

where y_i^* are the ground-truth values (i.e., target outputs for regression tasks) and L is a predefined loss function, such as the L_1 or L_2 loss, depending on the nature of y.

3.1.2 Workflow of HADA

The HADA framework follows a structured three-phase approach:

1. **Data Collection (Benchmarking)**: In this initial phase, multiple runs of the target algorithm are conducted across different configurations, varying both hyperparameters and hardware settings. The purpose is to gather empirical performance data that will serve as the basis for training surrogate models.
2. **Surrogate Model Training**: Once the dataset $\mathcal{S}$ is constructed, ML models are trained on the collected data to learn the performance patterns. These models approximate the function $h_\theta(x)$ and are subsequently encoded into a structured optimization problem following the EML paradigm.

3. **Optimization**: The final phase involves solving the optimization problem, where user-defined constraints and an objective function are applied on top of the learned surrogate models and expert-defined constraints. The solver searches for the best combination of algorithmic and hardware configurations that meet the specified constraints while minimizing the objective function (e.g., energy consumption or execution time).

3.1.3 Extending HADA for Energy and Carbon Footprint Optimization

The core functionalities of *Surrogate Model Training* and *Optimization* are already implemented in the existing HADA prototype, which will be detailed in Chapter 5. Our primary contribution in this work is to extend HADA by incorporating additional metrics—**Resource Utilization, Energy Consumption, and Carbon Emissions**—into the empirical modeling process.

To achieve this, we systematically execute a variety of algorithms under diverse configurations (both in terms of hyperparameters and hardware platforms) while recording key performance metrics. These metrics serve as the target variables for the surrogate models, allowing us to model the energy footprint and environmental impact of computational workloads. Before implementing this extension, however, it is necessary to establish a framework for accurately measuring the Carbon Footprint of an AI algorithm. This is the focus of the next section.

3.2 Carbon Footprint of Computation

The carbon footprint of an algorithm is determined by two main factors: (1) the energy required to execute it and (2) the emissions produced in generating that energy. The first factor depends on computing resource usage—such as

the number of processing cores, execution time, and data center efficiency—while the second, known as **Carbon Intensity**, is influenced by the energy production methods and geographic location.

Carbon Intensity quantifies the greenhouse gas (GHG) emissions associated with electricity production, expressed in terms of carbon dioxide equivalent (CO_2e). This metric accounts for the global warming potential of various GHGs emitted over a given timeframe [14]. As described in [2], the total carbon footprint of a computational task can be estimated using the following equation:

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

where:

- E represents the total energy consumed by the computational infrastructure, measured in kilowatt-hours (kWh).
- CI represents the carbon intensity of the electricity consumed, measured in grams of CO_2e per kilowatt-hour (gCO_2e/kWh).

3.2.1 Measuring Carbon Footprint with CodeCarbon

CodeCarbon is an open-source tool designed to estimate the carbon footprint of computational workloads. It achieves this by directly measuring energy consumption from key hardware components—CPU, GPU, and RAM—at regular intervals (default: every 15 seconds). Additionally, it tracks execution time to compute the total electricity usage of the system.

A distinguishing feature of CodeCarbon is its ability to incorporate region-specific **Carbon Intensity** data to provide location-dependent emission estimates. Carbon intensity is computed as a weighted average of the emissions from various energy sources used to generate electricity. These sources can be categorized into:

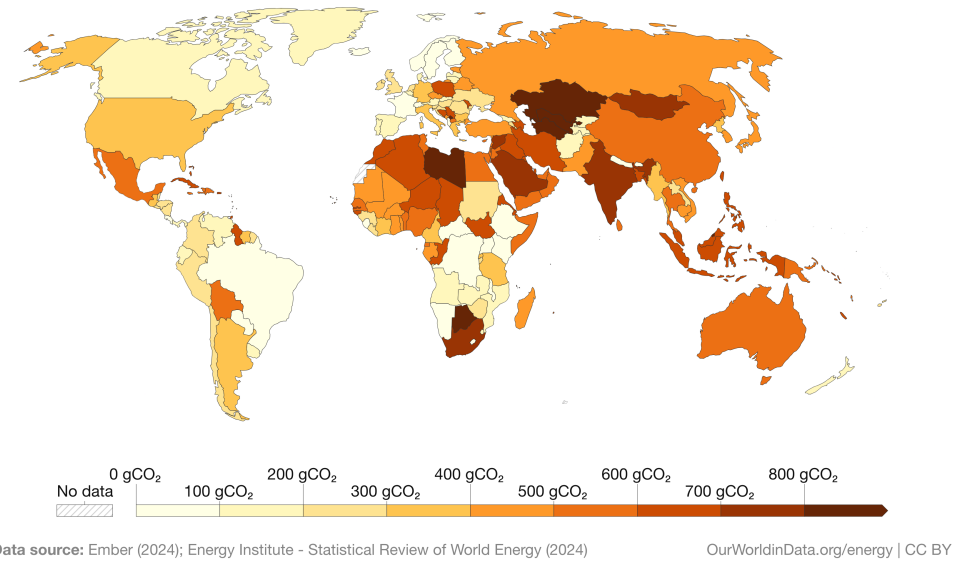

Figure 3.1: Carbon intensity of electricity generation, 2023. Source: [7]

- **Fossil Fuels**: Coal, petroleum, and natural gas—each associated with specific carbon intensities, meaning a known quantity of carbon dioxide is emitted per kilowatt-hour of electricity generated.
- **Renewable and Low-Carbon Energy**: Solar, wind, hydroelectricity, biomass, geothermal, and nuclear power—sources with significantly lower or near-zero carbon emissions.

Depending on the platform and data availability, CodeCarbon retrieves carbon intensity information from various sources. For private infrastructures, it relies on datasets such as those provided by Our World In Data [7]. Figure 3.1 illustrates the carbon intensity of electricity generation by country as of 2023.

By integrating real-time hardware usage monitoring with location-based carbon intensity data, CodeCarbon enables researchers and practitioners to quantify the environmental impact of their computations. This information can help guide decision-making, such as optimizing code efficiency, choosing lower-carbon data centers, or scheduling workloads during periods of higher renewable energy availability.

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, which is the sum of `cpu_energy` and `ram_energy`. When present it also includes the energy consumed by the GPU;
- `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) [12], 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 *Pseud-oflow* 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.

To generate a benchmark dataset for the Min-Cut/Max-Flow algorithms, we relied on the dataset provided by [Insert source]. This is an extensive collection of instances of Min-Cut/Max-Flow problems in the Computer Vision field. The authors also provided the implementations of the algorithms that were used to generate our benchmark data. Since they are implemented in C++ (BK and EIBFS) and C (HPF), we faced the problem of how to track the energy consumption and thus the emissions of such code. Apparently, there are not much tool available to monitor the emissions of C code. We opted for using a python wrapper script, that runs the C++/C scripts implementing the algorithms and uses CodeCarbon to track the energy consumption of the script execution. For reasons due to lacking support on MacOS, i was not able to compile the BK and EIBFS implementation on mbp19. Instead, i have managed to restore an old PC i had at home with a light distribution of Ubuntu to run some of these scripts. It is important to point out that we did not executed the selected algorithms on the whole dataset, since some of the problem instances were very big, sometimes exceeding the available memory, we just ran a subset of them - even if it was possible to create a swap partition big enough to accomodate bigger instances, the monitoring tools available just record the used RAM, so the estimate of the memory usage would be downsized in such

cases. Also, due to the large number of instances, running a benchmark phase on all of them would have been way too time consuming.

4.1.1 ANTICIPATE and CONTINGENCY

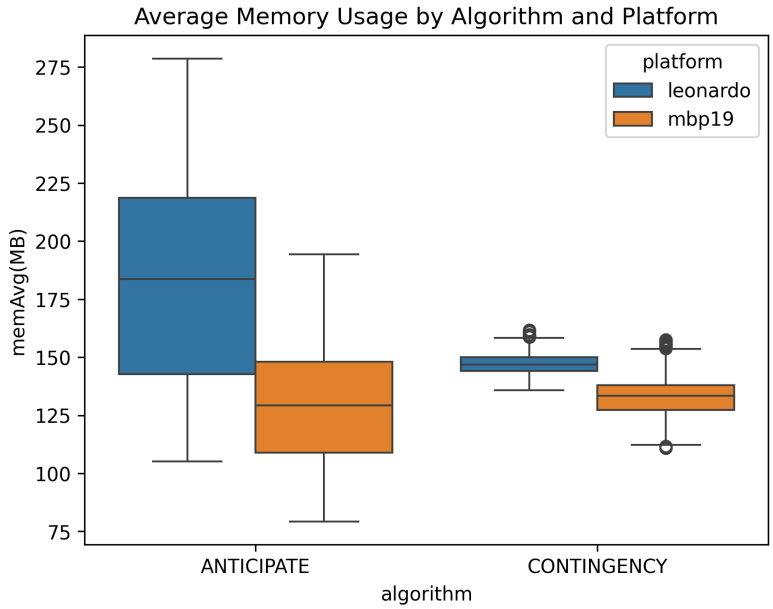

Figure 4.1: Average memory usage of ANTICIPATE and CONTINGENCY divided by platform

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.

As we can see in 4.5, `totEnergy` has a correlation of 1 with CO2e, meaning 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 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

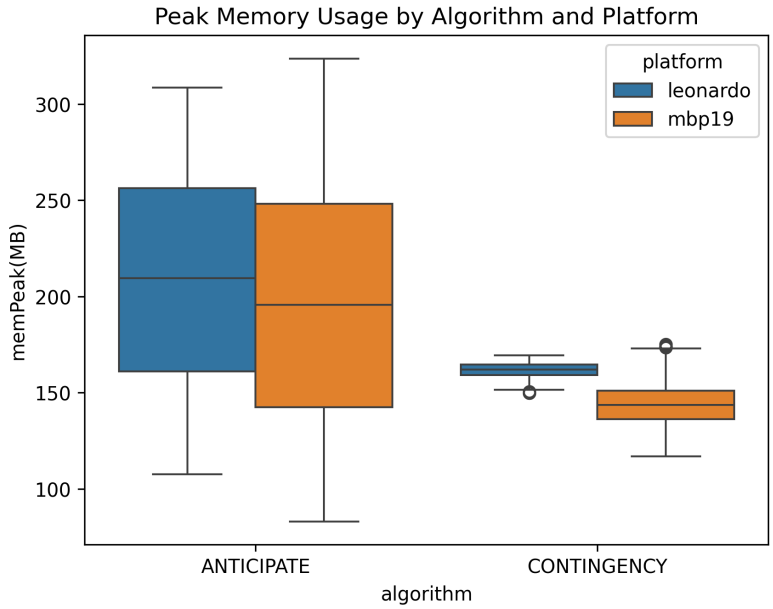

Figure 4.2: Peak memory usage of ANTICIPATE and CONTINGENCY divided by platform

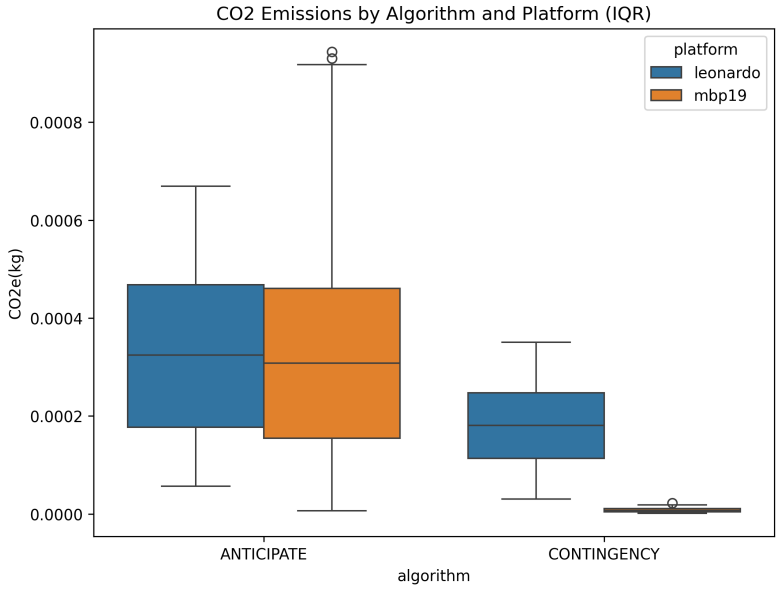

Figure 4.3: CO2 emissions for ANTICIPATE and CONGINGENCY ordered by platform

emissions is the energy consumed, while the Carbon Intensity is just a scaling factor. Indeed, `totEnergy` and `CO2e` both the same correlation values with the other variables. Also, as we could expect, the correlation of `CO2e` with

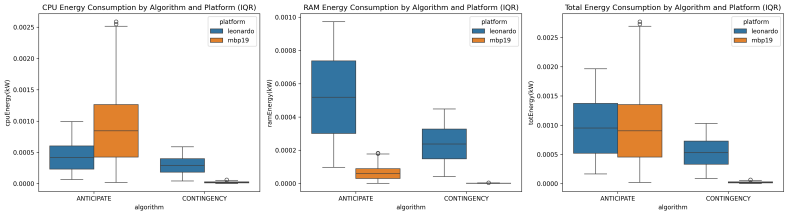

Figure 4.4: Energy Consumption data for ANTICIPATE and CONTINGENCY

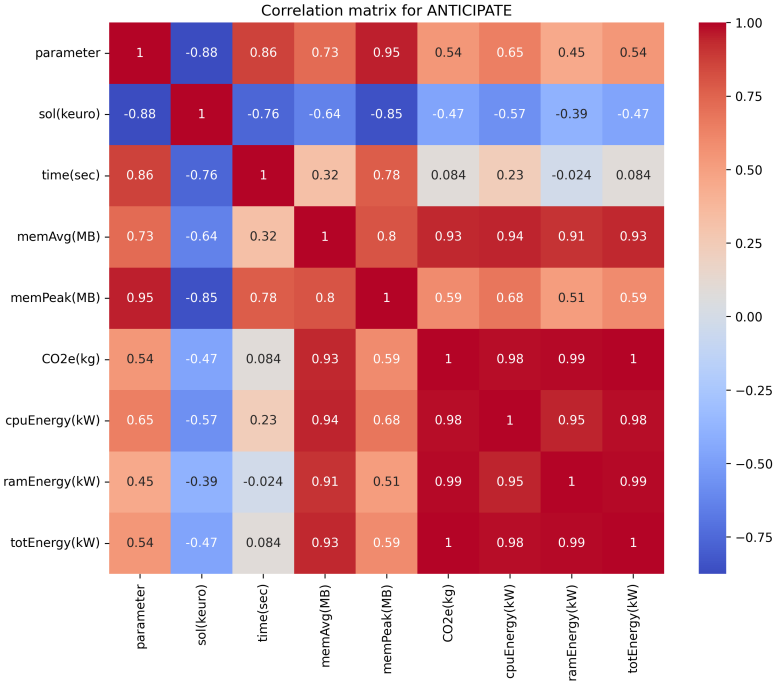

Figure 4.5: Correlation matrix for ANTICIPATE

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

`cpuEnergy` and `ramEnergy` is also pretty high, being of 0.98 and 0.99 respectively, and that reflects the fact that actually the total energy consumption is nothing but the sum of `cpuEnergy` and `ramEnergy`. Then, as one may point

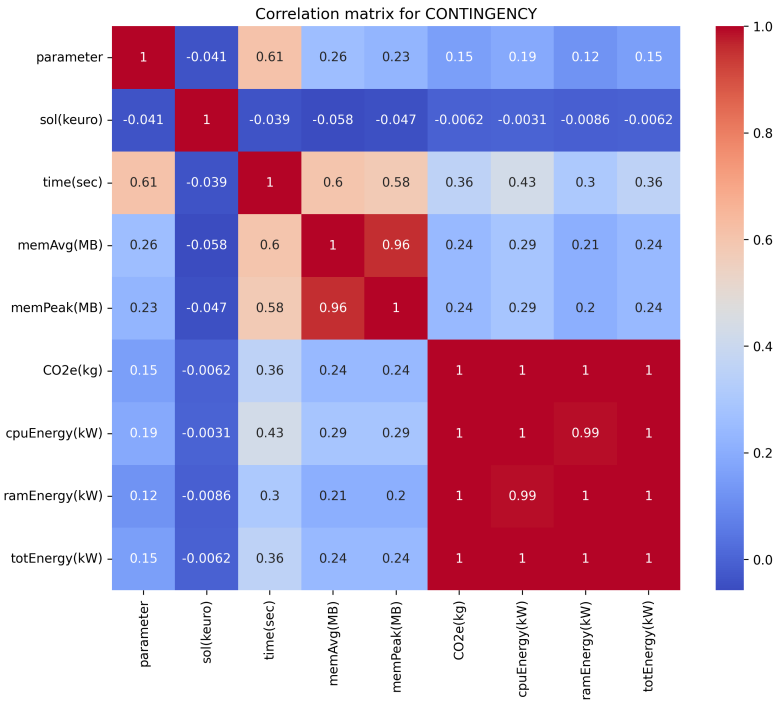

Figure 4.6: Correlation matrix for CONTINGENCY

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

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. Then, we can see that the average memory consumption is the variable with the highest correlation with the carbon emissions except for energy consumption, with a value of 0.93. Then, we can see that for ANTICIPATE the emissions are also influenced a bit by the value of the parameter (0.65)

and the Solution Cost (-0.57). We could then perform some tests on ANTICIPATE with HADA to asses the impact of constraints on memory and solution quality.

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	$100k$	none	none	none	none	none
100	$300k$	none	none	none	none	none
150	$300k$	64	mbp19	3.75×10^{-4}	141.53	$299.46k$
170	$270k$	97	mbp19	5.70×10^{-4}	169.85	$267.39k$
170	$280k$	87	mbp19	5.10×10^{-4}	160.12	$276.86k$
300	$500k$	1	mbp19	7.23×10^{-6}	81.47	$368.73k$

Table 4.3: Experimental results for the ANTICIPATE algorithm with bounds Memory usage and Solution cost

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.

For CONTINGENCY, 4.6 shows how the variables show very little amount of linear correlation between one another. The strongest relation is between the runtime and the amount of memory consumed (with a score of 0.60 for `memAvg` and 0.58 for `memPeak`). As for the Carbon Footprint, the value that influence it the most is the runtime, with a postivie correlation of 0.47. So we will try to test HADA on CONTINGENCY with constraints on the Runtime (expressed in seconds), and the Solution cost.

4.1.2 Min-Cut/Max-Flow Algorithms

Next, we take a look at the benchmark dataset of the Min-cut/Max-flow algorithms. By looking at the data, the first thing we may notice is that points are mostly concentrated in a small area. This is due to the fact that many of

Bounds		Solution				
Time	Cost	n^P	HW	CO2e	Time	Cost
no	no	2	mbp19	1.29×10^{-6}	-	-
no	$350k$	4	mbp19	1.53×10^{6}	-	$338.23k$
10	no	2	mbp19	1.29×10^{-6}	6.01	-
10	$320k$	none	none	none	none	none
60	$350k$	4	mbp19	1.29×10^{-6}	7.68	338.23
120	$320k$	90	mbp19	1.24×10^{-5}	119.47	$314.11k$

Table 4.4: Experimental results for the CONTINGENCY algorithm with bounds on Runtime and Memory usage

the instances were similar problems with standard graph sizes (nodes, arcs), which are quite small, and in proportion, bigger problems are way less.

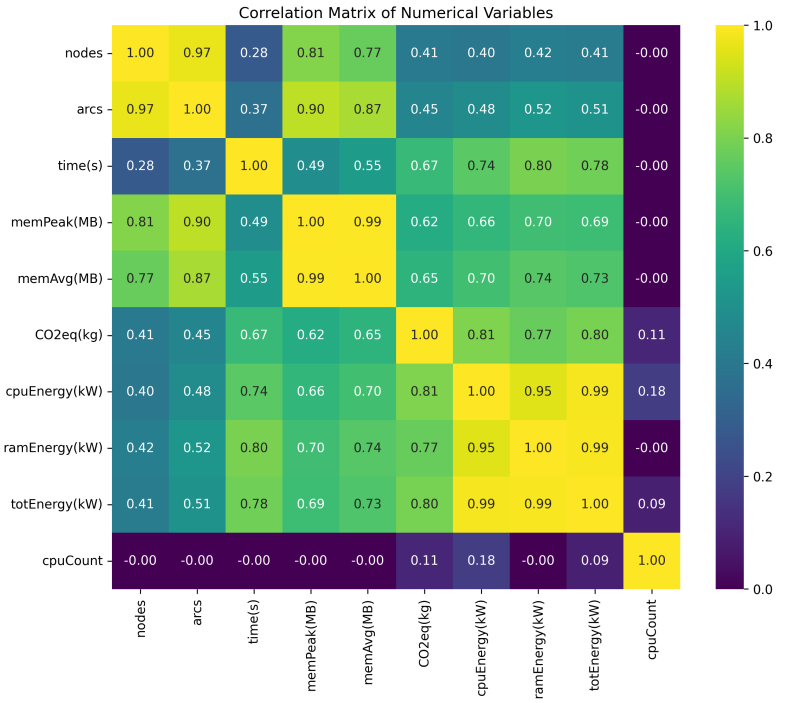

Figure 4.7: Correlation matrix for Min-Cut/Max-Flow Algorithms BK, EIBFS and HPF

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] E. J. Husom et al. *The Price of Prompting: Profiling Energy Use in Large Language Models Inference*. 2024. arXiv: 2407.16893 [cs.CY]. URL: https://arxiv.org/abs/2407.16893.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] S. Samsi et al. *From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference*. 2023. arXiv: 2310.03003 [cs.CL]. URL: https://arxiv.org/abs/2310.03003.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.