



Optimizing performance and energy consumption for machine learning inference

Tiago da Silva Barros

► To cite this version:

Tiago da Silva Barros. Optimizing performance and energy consumption for machine learning inference. Artificial Intelligence [cs.AI]. Université Côte d'Azur, 2025. English. ⟨NNT : 2025COAZ4020⟩. ⟨tel-05507211⟩

HAL Id: tel-05507211

<https://theses.hal.science/tel-05507211v1>

Submitted on 12 Feb 2026

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

THÈSE DE DOCTORAT

Optimisation des performances et de la consommation énergétique pour l'inférence dans le domaine de l'apprentissage automatique

Tiago DA SILVA BARROS

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 Univ. Côte d'Azur CNRS

**Présentée en vue de l'obtention
du grade de docteur en Informatique
d'Université Côte d'Azur**

Dirigée par : Frederic GIROIRE, Directeur de
recherche, CNRS, I3S
Soutenue le : 03/11/2025

Devant le jury, composé de :

Patricia STOLF, Professeure des Universités, Université Toulouse 2 Jean-Jaurès, IRIT
Romain ROUVOY, Professeur des Universités, Université de Lille
Anne-Cécile ORGERIE, Directrice de recherche, CNRS, IRISA
Guillaume URVOY-KELLER, Professeur des Universités, Université Côte d'Azur, I3S
Ramon APARICIO-PARDO, Maître des Conférences, Université Côte d'Azur, I3S

**OPTIMISATION DES PERFORMANCES ET DE LA CONSOMMATION
ÉNERGÉTIQUE POUR L'INFÉRENCE DANS LE DOMAINE DE
L'APPRENTISSAGE AUTOMATIQUE**

***Optimizing Performance and Energy Consumption for Machine
Learning Inference***

Tiago DA SILVA BARROS



Jury :

Rapporteurs

Patricia STOLF, Professeure des Universités, Université Toulouse 2 Jean-Jaurès, IRIT
Romain ROUVOY, Professeur des Universités, Université de Lille

Examineurs

Anne-Cécile ORGERIE, Directrice de recherche, CNRS, IRISA
Guillaume URVOY-KELLER, Professeur des Universités, Université Côte d'Azur, I3S

Directeur de thèse

Frederic GIROIRE, Directeur de recherche, CNRS, I3S

Co-encadrant de thèse

Ramon APARICIO-PARDO, Maître des Conférences, Université Côte d'Azur, I3S

Tiago DA SILVA BARROS

*Optimisation des performances et de la consommation énergétique pour l'infé-
rence dans le domaine de l'apprentissage automatique*

xiv+209 p.

For all who supported and inspired me in this journey

Optimisation des performances et de la consommation énergétique pour l'inférence dans le domaine de l'apprentissage automatique

Résumé

Les progrès rapides de l'utilisation de l'IA ont conduit à une augmentation significative des besoins en calcul. L'émergence de nouveaux paradigmes de déploiement, tels que le MLaaS (Machine Learning as a Service), et la popularité croissante de Large Language Models (LLM) notables, tels que GPT, Gemini et Deepseek, ont intensifié cette demande. En conséquence, la consommation d'énergie associée aux modèles d'IA a considérablement augmenté, soulevant des inquiétudes quant à leur empreinte environnementale.

Dans cette thèse, nous abordons la demande croissante en ressources et proposons des stratégies d'optimisation pour améliorer l'efficacité énergétique lors de l'inférence IA. Pour répondre à ces questions, nous abordons deux problèmes. Dans le premier, nous considérons un système comportant plusieurs tâches d'inférence avec des délais stricts qui doivent être planifiées sur un ensemble de machines. De plus, les utilitaires de compression de modèles appliqués pendant le temps d'inférence permettent à chaque tâche d'être exécutée avec une certaine compression, ce qui présente un compromis entre son niveau de compression (et son temps de traitement) et son utilité obtenue. L'objectif est de maximiser l'utilité de la tâche. Nous proposons donc des algorithmes d'approximation avec des garanties éprouvées pour résoudre le problème et étendre le modèle afin d'intégrer les contraintes budgétaires énergétiques.

Nous avons ensuite étudié les impacts environnementaux croissants associés à l'utilisation de l'IA. Récemment, de nouveaux paradigmes en matière d'IA verte ont émergé, passant du principe « plus c'est gros, mieux c'est », qui privilégie les grands modèles, à celui « petit suffit », qui met l'accent sur la sobriété énergétique grâce à des modèles plus petits et plus efficaces. Nous étudions comment la communauté de l'IA peut adopter la sobriété énergétique aujourd'hui en se concentrant sur la sélection des modèles pendant l'inférence. Nous avons estimé les économies d'énergie réalisées en sélectionnant le modèle pour exécuter les demandes d'inférence. Enfin, afin d'analyser de manière globale les implications environnementales de l'IA, nous avons réalisé une analyse du cycle de vie (ACV) de Pl@ntNet, une application basée sur l'IA. Celle-ci comprend une évaluation multicritères et multi-étapes afin d'obtenir une image complète de son empreinte écologique.

Mots-clés : Intelligence Artificielle, Efficacité Énergétique, optimisation, Ordonnancement, Compression de Modèles, Sélection de Modèles

Optimizing Performance and Energy Consumption for Machine Learning Inference

Abstract

The rapid advancement of AI usage has led to a significant increase in computing demands. The emergence of new deployment paradigms, such as MLaaS (Machine Learning as a Service), and the growing popularity of notable Large Language Models (LLMs), such as *GPT*, *Gemini* and *Deepseek*, has intensified this demand. Consequently, the energy consumption associated with AI models has increased considerably, raising concerns about their environmental footprint.

In this thesis, we address the AI increasing resource consumption demand and propose optimization strategies for improving energy efficiency during AI inference. To address these issues, we tackle two problems. In the first one, we consider a system with several inference tasks with hard deadlines that must be scheduled onto a set of machines. Furthermore, model compression utilities applied during inference time allow that each task can be executed with a certain compression, presenting a trade-off between its compression level (and, its processing time) and its obtained utility. The objective is to maximize the tasks utilities. We propose, thus, an approximation algorithm with proven guarantees to solve the problem and extend the model to incorporate energy budget constraints.

We then investigated the increasing environmental impacts associated with AI usage. Recently, new paradigms in green AI emerged, shifting from “*bigger is better*”, which prioritizes large models, to “*small is sufficient*” emphasizing energy sobriety through smaller and more efficient models. We investigate how the AI community can adopt energy sobriety today by focusing on model selection during inference. We estimated the energy savings by selecting the model for executing inference requests. Finally, to broadly analyze the environmental implications of AI, we conduct a Life Cycle Assessment (LCA) of Pl@ntNet, an AI-based application. This includes a multi-criteria, multi-stage evaluation to capture a comprehensive picture of its ecological footprint.

Keywords: Artificial Intelligence, Energy Efficiency, Optimization, Scheduling, Model Compression, Model Selection

Acknowledgements

First, I would like to express my deepest gratitude to my advisors, Prof. Frédéric Giroire and Prof. Ramon Aparicio-Pardo, for their invaluable guidance, support, patience and encouragement during all these years. Their insights and mentorship have been fundamental to the completion of this work.

I also thank the members of the thesis jury — Prof. Patricia Stolf, Prof. Romain Rouvoy, Prof. Anne-Cécile Orgerie, and Prof. Guillaume Urvoy-Keller — for dedicating their time to reviewing this thesis and for providing thoughtful and constructive feedback.

I am deeply thankful to my family. Their unwavering support, encouragement, and love were vital throughout this journey. Even from a distance, they were always present, and their belief in me made a profound difference.

I am also grateful to the many friends I have met along this path. You were fundamental to this journey. Each of you contributed meaningfully to this experience, and I have learned a great deal from you. A special thank you to my friends from the UBINET, COATI, and SIGNET teams. Your support during this period was important and inspiring to me.

Lastly, I would like to thank my colleagues from the research laboratory in Lyon and the collaborators involved in the Pl@ntNet assessment. A special thanks to Prof. Laurent for all the support during the research visit. I also thank the Pl@ntNet for providing us with relevant information on the architecture and usage of the Pl@ntNet service. It was a truly enriching experience, and I am grateful for the opportunity to learn and collaborate with such an inspiring group.

Table of Contents

1	Introduction	1
1.1	Machine Learning as a Service	1
1.2	MLaaS energy impact	3
1.2.1	Model size impact	5
1.3	Improving AI energy efficiency	6
1.3.1	Scheduling compressible tasks	6
1.3.2	Model Selection	7
1.4	Other environmental Impacts	8
1.5	Research objectives	9
1.6	Contributions	9
1.7	Thesis outline	10
	Notations	13
	Joint Scheduling algorithms for ML Inference tasks with compression techniques	
2	Background	17
2.1	Optimization Problem	19
2.1.1	Linear Programming	19
2.1.2	Mixed Integer Linear Programming	20
2.1.3	Convex Problems	21
2.2	Duality	23
2.2.1	Lagrangian	23
2.2.2	Dual problem	23
2.2.3	Karush-Kuhn-Tucker (KKT) conditions	24
2.3	Approximation algorithms	25
2.4	Solvers	26
3	Related works for scheduling compressible tasks	27
3.1	Neural Network Compression	29
3.2	Scheduling	30
3.2.1	Compressible tasks scheduling	31
3.2.2	Cloud-to-edge Continuum Scheduling	32
3.2.3	ML inference tasks scheduling	32
3.3	Energy-Aware scheduling	33
3.3.1	Scheduling compressible tasks	33

4	Scheduling with Fully Compressible Tasks: Application to Deep Learning Inference with Neural Network Compression	35
4.1	Introduction	37
4.2	Experiments and latency-accuracy tradeoff analysis	38
4.3	Modeling	39
4.3.1	Problem formulation	39
4.3.2	Accuracy function	41
4.4	Problem on a single machine	42
4.4.1	Formulation as a Convex Program	42
4.4.2	Optimal algorithm	44
4.5	Problem on several machines	49
4.5.1	Formulation as a Mixed Integer Program	49
4.5.2	NP-Hardness complexity	50
4.5.3	Approximation Algorithm	51
4.6	Experimental evaluation	57
4.7	Conclusion and Discussion	61
5	Scheduling Machine Learning Compressible Inference Tasks with Limited Energy Budget	65
5.1	Introduction	67
5.2	Problem Formulation	68
5.2.1	Accuracy function	70
5.2.2	Fractional Relaxation	70
5.3	Optimal algorithm for DSCT-EA-FR	73
5.3.1	Algorithm for DSCT-EA-FR, one machine	73
5.3.2	Algorithm for DSCT-EA-FR, multiple machines	74
5.4	Approximation algorithm for DSCT-EA	78
5.5	Results	80
5.6	Conclusion	84
	Evaluating AI Environmental Impacts for improving efficiency	
6	Small is Sufficient: Reducing the World AI Energy Consumption Through Model Selection	87
6.1	Introduction	89
6.2	Analysis of AI tasks and models	89
6.2.1	Identifying the more popular AI tasks	90
6.2.2	AI benchmarks methodology	93
6.2.3	Utility dependency on model size	95
6.2.4	Model adoption	98
6.3	Estimating the savings of AI model selection	100
6.3.1	AI inference energy consumption measurement methodology	101
6.3.2	Savings by model selection	106
6.4	Conclusion	109

7	Evaluating an AI-based application environmental impacts through a Life Cycle Assessment	113
7.1	Introduction	115
7.2	PI@ntNet Service	116
7.2.1	Identification Request	117
7.2.2	PI@ntNet Infrastructure	117
7.3	Related Works	119
7.4	Methodology	121
7.4.1	Study scope and perimeter considered	121
7.4.2	LCA modeling	123
7.4.3	Parameters and sources	129
7.5	Results	130
7.5.1	Global environmental impacts	131
7.5.2	Detailed assessment of PI@ntNet impacts	132
7.6	Conclusion	139
8	Conclusion and Perspectives	141
8.1	Future perspectives	143
	References	145
	List of figures	161

Annexes

A	PRISMA	171
A.1	Simulation Framework	173
A.1.1	Introduction	173
A.1.2	Related works	174
A.1.3	Background	174
A.1.4	PRISMA description	176
A.1.5	Usage	178
A.1.6	Illustrative example	179
A.1.7	prisma-v2: Extension to Overlay Cloud Networks	182
A.1.8	Conclusion	185
A.2	Impact Evaluation of Control Signalling onto Distributed Learning-based Packet Routing	186
A.2.1	Introduction	186
A.2.2	Related works	187
A.2.3	RL for Distributed Packet Routing: Formulation and Signalling	188
A.2.4	Simulation Experiments	193
A.2.5	Discussion	198
A.2.6	Conclusion	199

B	<i>Small is Sufficient: AI tasks and benchmarks description</i>	201
B.1	Text generation	203
B.2	Mathematical reasoning	203
B.3	Code generation	203
B.4	Translation	203
B.5	Text classification	204
B.6	Text Clustering	204
B.7	Image classification	204
B.8	Image segmentation	205
B.9	Text to image	205
B.10	Automatic speech recognition	205
B.11	Audio classification	206
B.12	Image-text to text	206
B.13	Time series forecasting	206

CHAPTER 1

Introduction

1.1 Machine Learning as a Service

Machine Learning (ML) is one of the most important areas in Artificial Intelligence (AI). It denotes the techniques that enable computer systems to learn behaviors from data and improve their performance over time without being explicitly programmed. ML models are designed to recognize patterns, make decisions, and adapt their behavior based on experience (Mitchell, 1997). These capabilities make ML a key tool for developing AI systems that can perform tasks traditionally carried out by humans.

The ML models have emerged as an important way for deploying intelligent applications. They are used across a wide range of fields such as robotics, healthcare, finance, and education, powering systems that perform tasks like scientific modeling, demand forecasting, consumer behavior analysis, etc.

The process of building a ML system often follows a series of steps. Some of the ML models phases are illustrated in Fig. 1.1. During the *data collection*, the data points used for training the model are collected or generated in order to build a dataset. In *training*, the model parameters are adjusted to minimize some loss or error. In *inference*, the weights are fixed and are used from input data to produce an output. Finally, in *deployment*, the models are usually integrated into an API or a web service. This thesis is primarily focused on the inference stage.

In recent years, the ML models popularity has grown rapidly. According to the AI Index Report (Perrault & Clark, 2024), organized by Stanford University, the number of AI-related publications tripled between 2010 and 2022, achieving more than 240,000 publications in 2022 (see Fig. 1.2a). The report also indicates a significant increase in AI-related number of patents granted, reaching 62,260, with a growth rate of 62.7% between 2021 and 2022 (see Fig. 1.2b).

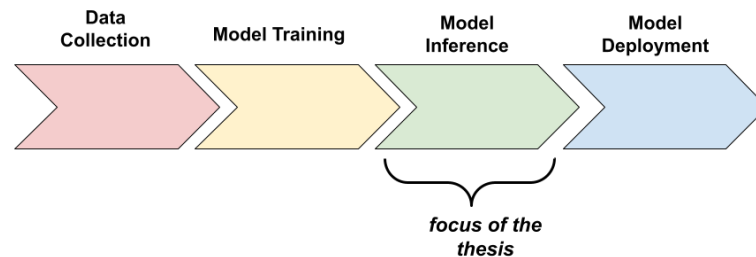


Figure 1.1 – ML models phases during its life cycle

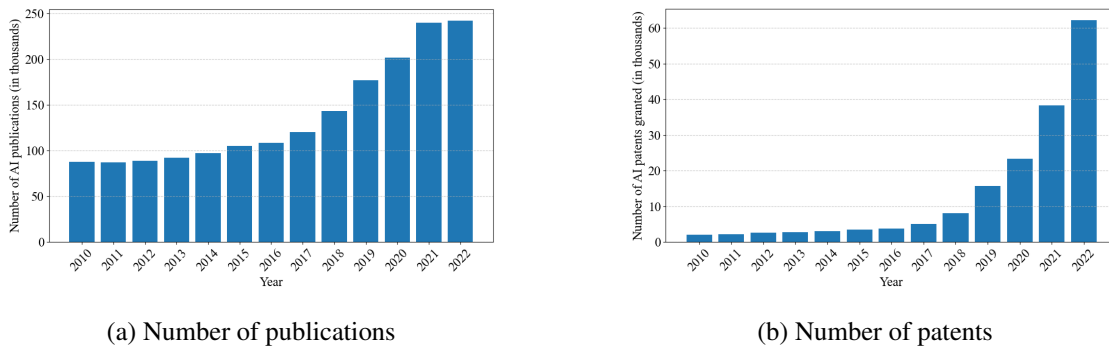


Figure 1.2 – Evolution of AI (a) number of publications and (b) number of patents over time indicating a significant increase of AI popularity.

A significant part of the AI advancement comes from some particular models, named as *foundation models*, which have attracted the attention of researchers and users due to their learning potential (Epoch AI, 2024). These models have been trained on large datasets and demonstrate the ability to solve several tasks in various applications. Some notable examples of foundation models, such as *GPT-4*, *Claude 3*, *Llama 2*, *Gemini-1.5*, *deepseek-R1* (language); *DiNO-V2*, *ViT* (vision); *HuBert* (audio); and *AlphaGo-Zero* (games) have been widely adopted by many users, skyrocketing ML models popularity.

For instance, the *ChatGPT*, a chatbot based on *GPT* family models, took five days to reach one million users, demonstrating significant growth. According to similarWeb (SimilarWeb, 2025), nowadays ChatGPT has more than 5.1 billions visits monthly. In the same way, Meta reports that *Llama* usage by token volume grew 10 folds from January until July 2024 (Meta AI, 2024). These statistics demonstrate the significant increase in AI usage across datacenters, creating new paradigms for computing systems.

With the rising demand for ML-based services, many cloud providers offer Machine Learning as a Service (MLaaS) (Ribeiro, Grolinger, & Capretz, 2015). This paradigm allows users to access ML models from anywhere, at any time, without needing to manage the underlying infrastructure.

MLaaS Frameworks Some examples of this new approach have emerged recently, providing scalable environments for managing ML models. For instance, PredictionIO (Chan, Stone, Szeto, & Chan, 2013), developed in 2013, proposes an open-source platform for interacting with multiple-machine learning processes. Another example is OpenCPU (Ooms, 2014), an open-source platform designed for hosting ML models through a public API.

Furthermore, some big tech companies invested in developing frameworks allowing users to deploy their models. For instance, in 2014, Microsoft launched Machine Learning Azure and in 2015, Amazon launched AWS Machine Learning. These platforms offer flexible and cloud-based environments for deploying machine learning (ML) models, allowing users to adopt artificial intelligence (AI) on a massive scale.

Cloud-to-Edge Continuum The paradigms of computation have experienced a profound transformation recently, driven by advances in both hardware and software technologies. The increasing computational capabilities of microcontrollers and embedded systems have opened new opportunities for deploying ML models, thereby enhancing edge computing.

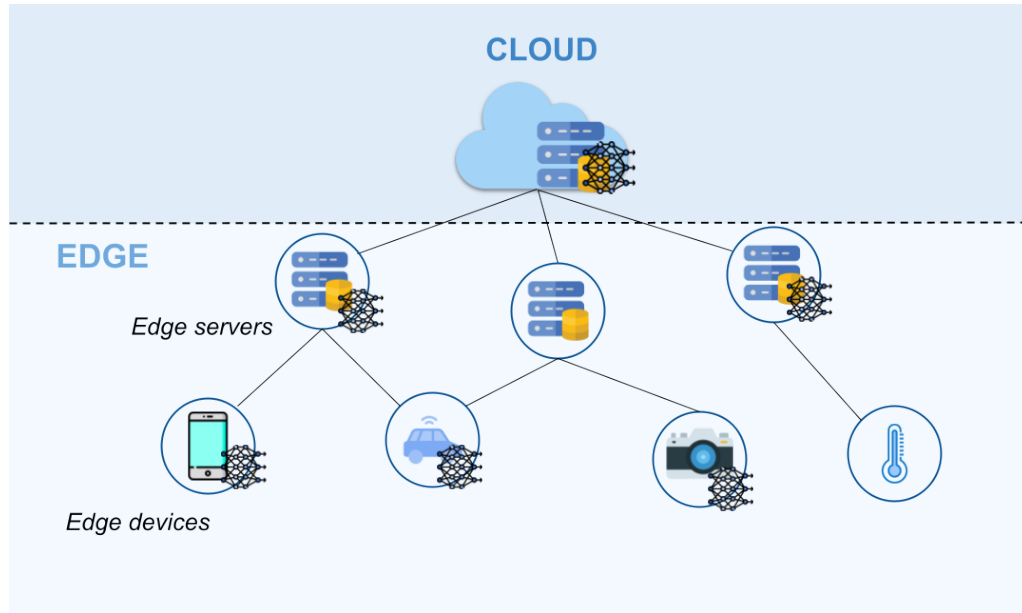


Figure 1.3 – ML models are deployed along the cloud-to-edge continuum

Furthermore, recent advances in network technology offer new possibilities for deploying computational loads. Software Defined Networks (SDN) and Network Function Virtualization (NFV) provide a flexible infrastructure that facilitates dynamic resource allocation and service orchestration (Svorobej, Bendeckache, Griesinger, & Domaschka, 2020). Moreover, research advances in 5G/6G allowed scaling the services deployment, increasing the network capacity. Thus, a new paradigm arises, where the ML models are deployed along the cloud-to-edge continuum.

According to this novel paradigm, ML models are not restricted to cloud infrastructures. Notably, it allows models deployment on a wide range of devices, from high-performance cloud servers to low-power microcontrollers, depending on latency, energy and bandwidth constraints. Fig. 1.3 illustrates the ML models deployed along cloud-to-edge continuum, such as gateways, microcontrollers, antennas, and mobile phones (Drainakis, Pantazopoulos, Katsaros, Sourlas, & Amditis, 2021). In this way, the models can provide accurate inferences close to the users, reducing latency and transmission overhead (Zeng, Ansari, Montpetit, Schooler, & Tarchi, 2021).

The approach is important in some time-sensitive applications, such as self-driving cars, unmanned aerial vehicles (UAVs) (Raj, Gupta, & Simmhan, 2023) and Mobile Edge Computing (MEC) (Shakarami, Ghobaei-Arani, & Shahidinejad, 2020). However, edge devices typically present strong constraints in terms of disk space, computational power, and memory. These limitations impact models latency and accuracy (W. Gao et al., 2022), posing challenges for optimizing resources consumption and model performance.

1.2 MLaaS energy impact

As the use of Machine Learning expands, the associated societal and environmental impacts rise, posing new challenges for researchers and companies. The rapid progress of ML models capabilities, coupled with the growth in data volumes and the infrastructure required to run them,

with even more powerful machines, has led to a significant gain in model performance. However, the energy consumption increases, rising economic, societal and environmental costs (Schwartz, Dodge, Smith, & Etzioni, 2020; C.-J. Wu et al., 2022).

According to the United States energy report (Shehabi et al., 2024), US data centers are projected to consume between 325 and 580 TWh in 2028, an equivalent of 12.0% of the country total energy consumption. Furthermore, according to the report, only ML-related operations account for approximately 22% of the datacenter consumption, representing a significant share of the energy impact. The International Energy Agency (IEA) also reports that, by 2030, the AI energy demand is expected to quadruple, scaling up the energy consumption (International Energy Agency, 2025).

The rising energy demand of AI systems, coupled with the associated greenhouse gas emissions (GHGE), is a concern for different actors, including tech companies, governments, the scientific community and society. These trends pose a significant risk to the efforts for mitigating the climate change impacts, such as those established in the Paris Agreement (Paris, 2015) and in the Intergovernmental Panel on Climate Change (IPCC) report (H. Lee et al., 2023).

Extensive work has been carried out to evaluate the energy consumption of infrastructures in which ML models are run. For instance, Ahvar, Orgerie, et Lebre (2019) provides a large study, where they investigate the energy implications of cloud-edge system architectures with different levels of distribution. Indeed, in such systems, the energy consumption due to communication between end-users and data centers (or within a data center) may pose significant challenges. Furthermore, the energy required for keeping the infrastructure, e.g., for cooling, accounts for a significant portion of the total footprint.

Several works have been proposed to optimize energy efficiency in data center servers and network links. For instance, Huin et al. (2018) aimed to optimize network energy consumption by turning off underutilized routers and links in a hybrid SDN-legacy environment. For computing, Li, Samsi, Gadepally, et Tiwari (2023) proposed a framework for partitioning GPU resources in cloud infrastructures to optimize jointly the overall accuracy and the carbon reduction. Mo et Xu (2021) proposed to jointly optimize communication and computation in federated learning to reduce energy usage. Furthermore, Alzahrani et al. (2016) investigated dynamic CPU frequency scaling as a method for reducing power consumption, Jacquet, Ledoux, et Rouvoy (2024) introduced a dynamic approach to VM oversubscription that adjusts resource allocation per physical machine to enhance efficiency without degrading performance, and Haddad et al. (2021) and Gnibga, Blavette, et Orgerie (2023) examined the integration of data centers and renewable power sources. All these works highlight the importance of addressing the energy consumption of ML workloads.

In particular, the energy associated with the ML models inference phase is significant. Google estimates that 60% of the datacenters energy consumption comes from inference (Patterson et al., 2022) phase. Although a single inference request is low-resource intensive, continuous usage of ML models may result in thousands of requests, increasing the energy impact.

Understanding, thus, the inference energy consumption patterns is essential for reducing the environmental impacts associated with ML. Researchers have begun to analyze the energy consumption of ML models during inference.

S. Luccioni, Jernite, et Strubell (2024) studied several vision and language models for evaluating the energy consumption during the inference phase. Their findings point out that generative models are more energy-intensive when compared to classification models, reflecting the architecture complexity. Moreover, general-purpose foundation models, such as *GPT* and *Gemini* model families, consume more energy than task-specific models, such as *BERT*. They advocate, then, that a paradigm change in users behavior when using models is vital to achieve energy sufficiency.

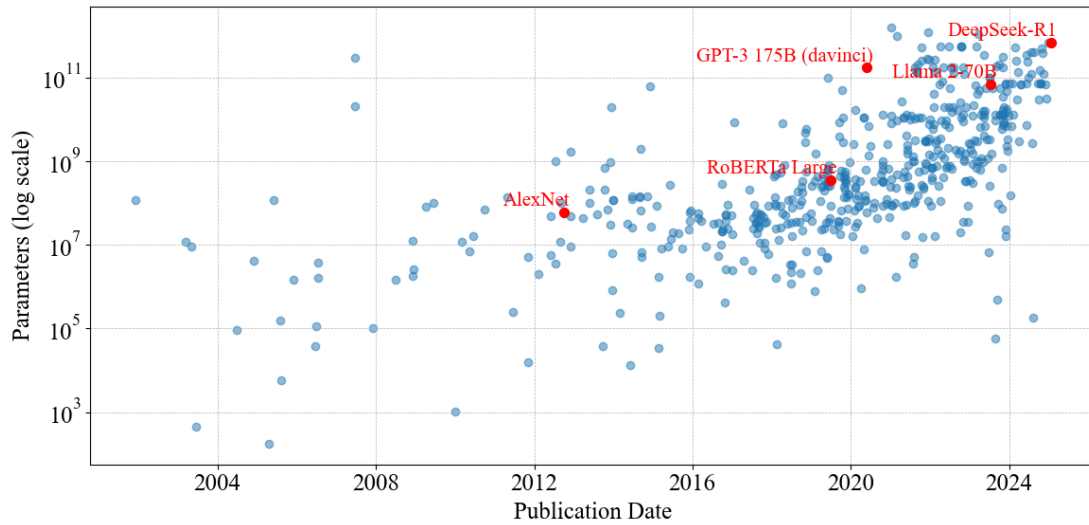


Figure 1.4 – Number of parameters of foundation models over time

[J. Yang et al. \(2024\)](#) investigated several Large Language Models (LLMs) performance across multiple language tasks, comparing task-specific models and multi-purpose LLMs. They identified some tasks where task-specific models can be advantageous, such as text classification, sentiment analysis, and toxicity detection.

The above understanding is followed by [Dodge et al. \(2022\)](#). They proposed studying the energy consumption of Natural Language Processing (NLP) and vision models in cloud instances. Moreover, they investigated various techniques for improving energy efficiency, including scheduling tasks across different regions and times of day.

1.2.1 Model size impact

Machine Learning has evolved significantly during the years. In 1960s, the early ML models, e.g. single-layer *Perceptron* ([Rosenblatt, 1958](#)), were proposed by Frank Rosenblatt with pattern recognition goals. During this period, the lack of supporting hardware capable of performing large quantities of operations has limited model development and constrained computing scalability.

However, with the emergence of Graphical Processing Units (GPU), the computational capabilities improved, leading researchers to develop and design deeper (and more complex) models, improving the models performance. A turning moment was the release of the *AlexNet* model, in 2012, which introduced the deep learning paradigm, by significantly increasing the number of layers and outperforming prior models.

As technology has progressed, we have seen a significant growth in ML models dimensions. [Fig. 1.4](#) illustrates the number of parameters of foundation models as a function of the year. The model size presents an exponential increase (figure on logarithmic scale), indicating that ML models have become larger and more complex.

The model size increase has led to models utility improvements. One of the fundamentals behind this progress is the *overparameterization* ([Thompson, Greenewald, Lee, & Manso, 2020](#)), when the number of parameters is larger than the number of data points used for training. An

analysis on Epoch AI benchmarks points that 47% of the notable models have more parameters than data points (Jay, 2024). One example of this application is for *image classification* task. The CoCa model, for instance, has 2.1B parameters, while the traditional dataset ImageNet contains 1.2M of images.

According to Thompson et al. (Thompson et al., 2020), the *overparameterization* brings performance benefits, increasing the models accuracy. However, the computing cost is also high. The authors estimate that the computation scales at least with a fourth-order polynomial law, meaning that even small increases in size can lead to a significant increase in computation.

This trend has serious implications for energy consumption. As deep learning models get larger and more computationally demanding, more energy is required, posing significant challenges for improving energy efficiency.

1.3 Improving AI energy efficiency

Given the increasing demand for running complex and large ML models and the constraints limiting very large models, such as limited hardware, real-time applications, energy budget, it is vital to find alternative ways to optimise the resources consumption and leverage energy efficiency.

One possible solution involves the usage of renewable power sources, such as wind and solar. These energy generation modalities provide the advantage of being a low carbon source, which mitigates the AI associated environmental impacts. However, some challenges for renewable power sources emerge: (i) it is not derived from carbon neutral sources; and (ii) the renewable energy source is usually associated only to the computing machine where the models are deployed on (Strubell, Ganesh, & McCallum, 2020).

Beyond renewable energy, several techniques have been proposed to optimize ML energy efficiency, targeting different stages in ML life cycle: (i) *training* (e.g. hardware optimization (Tao, Ma, Shyu, & Chen, 2020), model compression (B. Zhang, Davoodi, & Hu, 2018), architecture designing (Mehta, Rastegari, Shapiro, & Hajishirzi, 2019), model sparsification (Frantar & Alishtarh, 2023), quantization (W. Shao et al., 2023)); (ii) and *inference* (e.g. model selection (Asperti, Evangelista, & Loli Piccolomini, 2021; J.-R. Yu et al., 2022) and model compression (Cai, Gan, Wang, Zhang, & Han, 2020)). Following, some of these techniques are described.

1.3.1 Scheduling compressible tasks

One promising way to improve resource efficiency in ML inference is through the scheduling of compressible tasks. In general, *scheduling* refers to the optimal allocation of resources (e.g., time, hardware) to tasks while respecting constraints such as deadlines, resource availability, and task dependencies (Karger, Stein, & Wein, 1999).

In computing systems, scheduling ensures that tasks are assigned to machines without overlaps and that no task is executed simultaneously in more than one machine (B. Chen, Potts, & Woeginger, 1998). Fig. 1.5 exhibits a scheduling system example. The clients submit ML inference requests to cloud platforms. A scheduler determines *when* and *where* each task is processed, ensuring deadlines are met while optimizing resource use.

Furthermore, model compression plays an important role in achieving efficiency in resource allocation (Cheng, Wang, Zhou, & Zhang, 2018). Compressing ML models allows reducing the computational load, leading to memory and communication bandwidth savings and improving energy efficiency.

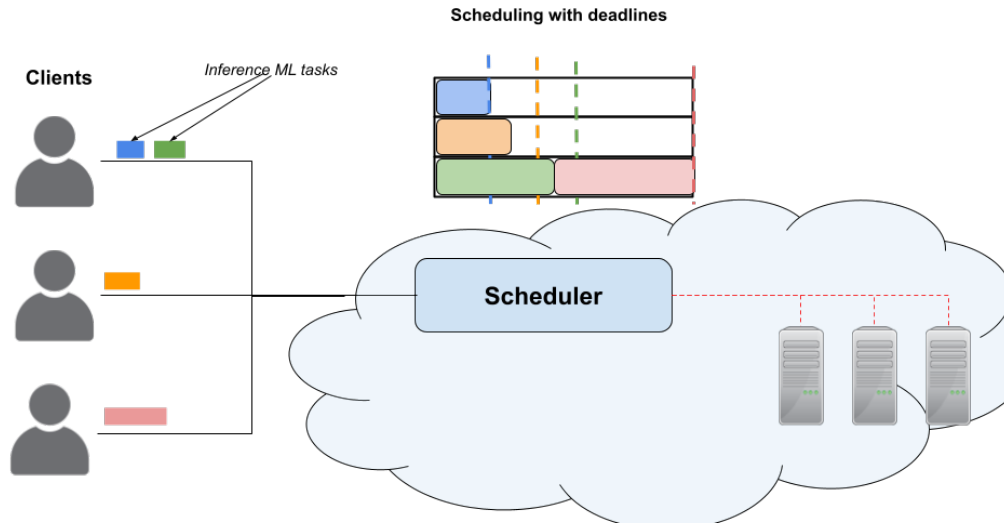


Figure 1.5 – Scheduling scheme system example.

Several methods were proposed for compressing ML models, such as pruning ([J. Shao & Zhang, 2020](#)), quantization ([W. Shao et al., 2023](#)), and distillation ([Bhardwaj, Lin, Sartor, & Marculescu, 2019](#)). These approaches significantly reduced the models computation, leading to faster and more efficient models.

However, these traditional compression techniques require weight retraining and cannot be applied at inference time due to their prohibitive time, energy, and cost for updating the weights. Recently, some approaches ([J. Yu & Huang, 2019a](#); [Cai et al., 2020](#)) have been proposed for compressing Neural Networks (NNs) during inference. They propose, then, decoupling the training and architecture search, which is done in the inference phase after the weights are computed.

These methods are particularly valuable in time and energy-sensitive applications, such as autonomous vehicles and edge computing. In such cases, using "*slimmable models*" jointly with task scheduling can significantly improve energy efficiency with a minimal performance drop.

1.3.2 Model Selection

Energy-efficient *model selection* refers to selecting a model that is smaller than the state-of-the-art model to solve an AI inference task, but presenting a comparable performance. Indeed, a large number of models exist to solve any AI task, and they are widely available at platforms such as *Hugging Face* ([Hugging Face, s. d.](#)) and *Papers with Code* ([Papers with Code: The latest in Machine Learning, s. d.](#)). They facilitate access to diverse AI models, allowing practical implementation of model selection strategies. Model selection is thus a very simple method that can be immediately applied by any user who needs to solve an AI task. There is no need to retrain models, use specific hardware, implement complex architecture, etc.

If we take the *Image Classification* as an example, several thousand models have been introduced to solve it (more than 5,000 models are available in *Hugging Face* and 4,000 models in *Papers with Codes*), each having a different trade-off between performance, model size and energy consumption. Fig. 1.6 illustrates the release of bigger and more efficient models over time,

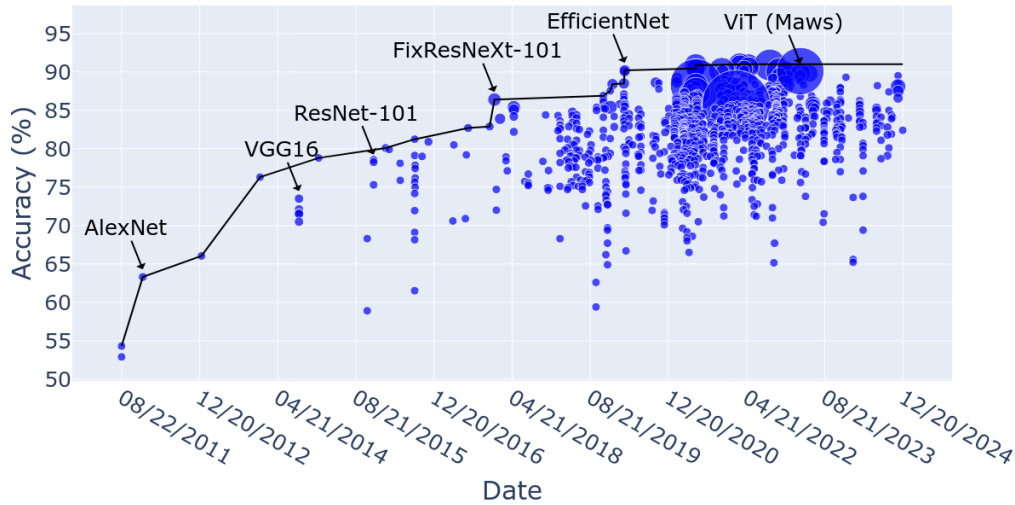


Figure 1.6 – Evolution of *Image Classification* models over time. Accuracy (%) of image classification models on the *ImageNet* dataset plotted against their publication date. Each point corresponds to one model. Each data point represents a model, with the marker size proportional to the number of parameters. The plot shows an accuracy "plateau" post-2020, with the continued release of models across various sizes. Data extracted from the *Papers With Code* platform.

from the first convolutional neural network, *AlexNet* (Krizhevsky, Sutskever, & Hinton, 2012), with 65M parameters to vision transformers with more than 1B parameters.

We observe two main phases over time: a rapid increase in performance followed by a plateau from 2021 onwards. This is related to a very general phenomena of every economical activity, the law of diminishing return, also known as the law of diminishing marginal productivity, which can be traced back to the 18th century, in the work of Jacques Turgot (Faccarello, 2016). As the task matures, researchers have focused in developing efficient models reducing the model size, while keeping close performance, using different compression methods. Thus, the impact of model selection for an AI task depends on its stage of development. For instance, for *image classification* tasks, a very mature task, optimization techniques have allowed the development of lightweight models, such as *MobileNet* and *ResNet*, thereby reducing the energy impact. On the other hand, some recent tasks, such as *reasoning*, require very large models, which increases the carbon footprint.

Selecting energy efficient models for running ML inference tasks supports a paradigm shift in AI: From **"bigger-is-better"**, in which everybody uses the most powerful hardware (e.g., from NVIDIA) to run the best-performing (and usually the largest) models, to **"small-is-sufficient"**, in which users prioritize much smaller models with similar performance than the state-of-the art models.

1.4 Other environmental Impacts

The Artificial Intelligence is responsible for a wide range of environmental impacts, beyond the energy consumption. Other important environmental impacts of AI usage include *water consumption*, *human toxicity*, and other abiotic natural resources, such as *aluminum*, *cobalt*, *cop-*

per, glass, gold, tin, lithium, zinc, and plastic. (Gupta et al., 2021 ; Bannour, Ghannay, Név  ol, & Ligozat, 2021).

The above impacts are associated with different stages of the digital product’s life cycle, making this an important attention point. To account for environmental costs at different stages, such as manufacturing and the end of product life, several researchers have proposed the Life Cycle Assessment (LCA) concept (Day, 1981). The idea of this research field is to investigate the societal and environmental impacts of computing systems at different stages of the product life cycle.

1.5 Research objectives

Given the increasing popularity of AI and the widespread adoption of ML models, the environmental impacts associated with resource consumption have significantly increased. In this context, this thesis aims to *optimize the performance and energy consumption for Machine Learning inference*.

To achieve the above-mentioned goal, we first *explore the tradeoff between ML utility and resources consumption* to improve model efficiency during inference. Then, we *assess the environmental impacts of ML inference* to identify opportunities for optimizing resources.

1.6 Contributions

For achieving the above objective, we provide the following contributions.

[C1] Scheduling ML compressible inference tasks (Barros, Giroire, Aparicio-Pardo, Perennes, & Natale, 2024)

We investigate the ML inference tasks scheduling problem in conjunction with model compression (Cai et al., 2020), which offers a significant decrease in processing time while keeping a good performance. We examine problem complexity and we provide an (i) exact algorithm for a single-machine scenario and (ii) an approximation algorithm with proved guarantees for a multiple-machines scenario, to maximize the accuracy while meeting the deadlines.

[C2] Scheduling ML compressible inference tasks with energy budget (da Silva Barros, Ferre, Giroire, Aparicio-Pardo, & Perennes, 2024)

We extended the scheduling problem by adding that the energy consumption must adhere to an energy budget. We (i) proposed an optimal algorithm for single-machine and (ii) extended our approximation algorithm for the multiple-machines scenario.

[C3] Energy consumption savings through model selection (da Silva Barros, Giroire, Aparicio-Pardo, & Moul  ierac, 2025) We investigate the energy savings when applying model selection. We examine the tradeoff between utility and model size for several tasks to identify optimization opportunities to improve the energy efficiency during inference. We, then, estimate how much energy can be saved by selecting efficient models during inference.

[C4] Environmental impacts of an AI-based application life cycle Finally, we conduct a life cycle assessment for evaluating the environmental impacts associated with an AI-based application on a large scale. We examine the *Pl@ntNet*, a vision based application dedicated to recognizing plant species. We then estimate the embodied and operational impacts associated with application usage.

Additional contribution This thesis primarily focuses on optimizing resource allocation for Machine Learning (ML) inference. In addition to this main area of research, I collaborated as part of my Master’s thesis and during the first year of my PhD with Redha A. Alliche on his PhD thesis within the framework of the Artificial Intelligence-based Cloud Network Control (ARTIC) project ([Agence Nationale de la Recherche, 2019](#)). In this additional contribution, we explored ML-based approaches to improve resource allocation in networking environments. In particular, we concentrated on Distributed Packet Routing (DPR), where multiple nodes exchange packets to deliver data to designated destinations. In DPR systems, routers usually lack access to complete network topology and traffic information, which makes efficient routing difficult. To address this, we investigated AI-based solutions employing Multi-Agent Reinforcement Learning (MARL), with a focus on their integration into realistic network simulators. The main contribution of this work is described below:

[C5] Network simulation framework for integrating with Multi-Agent Deep Reinforcement Learning Agent ([Alliche, Da Silva Barros, Aparicio-Pardo, & Sassatelli, 2022](#); [Alliche, Barros, Aparicio-Pardo, & Sassatelli, 2022, 2023](#)) We develop a network simulation framework for cloud overlay networks to be integrated with Multi-Agent Deep Reinforcement Learning agents. The goal of the agent is to find effective policies for routing packets to minimize end-to-end delay and save network resources. We built a realistic simulation framework utilizing the `ns3` and `gym` tools, capable of reproducing the packets exchanged between the several agents for learning their corresponding policies. The PRISMA contribution is described in Appendix A.

1.7 Thesis outline

Our thesis is structured as follows.

In the first part, we address the *joint scheduling problem for ML inference tasks with compression techniques*. In Chapter 2, we provide a theoretical background for optimization problems. We first provide the main classes of optimization problems addressed in this thesis. Then, we discuss convex optimization and present the Karush-Kuhn-Tucker (KKT) conditions for achieving optimality. Then, we present the concept of approximation algorithms as a possible solution for NP-hard problems. Finally, we provide the main solvers used in academia and industry for solving optimization problems.

Chapter 3 provides a revision of the literature and the state of the art for scheduling algorithms and ML models compression techniques. We first review the literature for ML model compression, presenting the main approaches. Then, we review the algorithms addressing scheduling tasks with deadline constraints. Finally, we provide an overview of the state-of-the-art in energy-aware scheduling algorithms.

In Chapter 4, we discuss the scheduling algorithms for ML inference tasks along with model compression (**contribution C1**). We first carried out some experiments on the Once-For-All ap-

proach (Cai et al., 2020) to study the accuracy-processing time trade-off in a wide range. Then, we modeled the accuracy-latency tradeoff as concave functions, and then, we provide the algorithmic solutions. Finally, we validate our approach by comparing it against the state-of-the-art.

In Chapter 5, we address the compressible tasks scheduling problem with the energy budget (**contribution C2**). We discuss the problem, its conditions, and then, we provide (i) a new algorithm for achieving the optimal value; and (ii) an extension of our approximation algorithm for the multiple machines' scenario.

In the second part, we *evaluate the environmental impacts of AI for improving efficiency*. Chapter 6 investigates the environmental benefits of choosing energy-efficient models for ML inference (**contribution C3**). We first identify the most commonly addressed tasks in datacenters, based on statistics from *Hugging Face* and *Papers With Code*. Then, we investigate the tradeoff between utility and model size across a wide range of tasks, analyzing different task maturity and model adoption levels to identify the main opportunities for improving AI energy efficiency. Finally, we quantify the energy savings obtained from model selection.

In Chapter 7.1, we present our Life Cycle Assessment (LCA) on the Pl@ntNet application (**contribution C4**). We first provide the methodology applied for quantifying the environmental impacts for different application sections. Then, we estimated the embodied and operational environmental impacts associated with the Pl@ntNet application.

Finally, in Chapter 8, we discuss the results obtained during the thesis and we draw the main perspectives for the future.

In Annexe A, we provide the simulation framework for distributed packet routing using Multi-Agent Reinforcement Learning (MARL) (**contribution C5**). Here, we first present the related works and necessary background for implementing the framework. Then, we describe the framework details about its usage. We then provide some illustrative examples of distributed packet routing, comparing the proposed solution using MARL with state-of-the-art techniques. Finally, we introduce the `prisma-v2`, an extension of the simulation framework for overlay networks.

Notations

ADP	Abiotic Depletion Potential
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DPR	Distributed Packet Routing
DQN	Deep Q-Network
DSCT	Deadline Scheduling with Compressible Tasks
DSCT-EA	Deadline Scheduling with Compressible Tasks - Energy Aware
EDF	Earliest Deadline First
EGM	Electricity Grid Mix
FLOPS	Floating-Point Operations Per Second
GPU	Graphics Processing Unit
GWP	Global Warming Potential
HPC	High Performance Computers
IoT	Internet of Things
LCA	Life Cycle Assessment
LLM	Large Language Model
LP	Linear Programming
MARL	Multi-Agent Reinforcement Learning
MEC	Mobile Edge Computing
MILP	Mixed-Integer Linear Programming
ML	Machine Learning
MLaaS	Machine Learning as a Service
PE	Primary Energy
PUE	Power Usage Effectiveness
RL	Reinforcement Learning
VM	Virtual Machine

**Joint Scheduling algorithms for ML
Inference tasks with compression
techniques**

CHAPTER 2

Background

In this chapter, we provide a theoretical background about optimization theory, in order to solve combinatorial problems addressed in this thesis. We identify several classes of optimization problems addressed in this thesis. Then, we discuss the main tools existing for addressing them, namely, duality theory and optimization algorithms.

2.1	Optimization Problem	19
2.1.1	Linear Programming	19
2.1.2	Mixed Integer Linear Programming	20
2.1.3	Convex Problems	21
2.2	Duality	23
2.2.1	Lagrangian	23
2.2.2	Dual problem	23
2.2.3	Karush-Kuhn-Tucker (KKT) conditions	24
2.3	Approximation algorithms	25
2.4	Solvers	26

2.1 Optimization Problem

Optimization plays a fundamental role in a wide range of combinatorial and decision-making problems. It involves selecting the best element according to a specific criterion. A formal definition is given in Definition 2.1.1, based on [Boyd et Vandenberghe \(2004\)](#).

Definition 2.1.1 (Optimization Problem). ([Boyd & Vandenberghe, 2004](#)) *Optimization problems* consist of finding the optimal value of a set of decision variables $x \in \mathcal{R}^n$, as formally described in Eq. (2.1). The referred problem contains a set of n variables, $x_1, \dots, x_n$, and a set of m inequality constraints and a set of p equality constraints. Depending on the context, the optimization problems aim to *maximize* or *minimize* an objective function.

$$\min \quad f_0(x) \quad (2.1)$$

$$\text{s.t.} \quad f_i(x) \leq 0, \quad \forall i \in [1..m] \quad (2.1a)$$

$$h_i(x) = 0, \quad \forall i \in [1..p] \quad (2.1b)$$

In the above formulation, we aim to find the optimal value for the variables $x = [x_1, \dots, x_n]$. The function $f_0(x)$ describes the objective function to be optimized, and the functions $f_1(x), \dots, f_m(x)$ and $h_1(x), \dots, h_p(x)$ describe, respectively, the inequality and the equality constraints.

Following, we describe some classes of optimization problems addressed in this thesis.

2.1.1 Linear Programming

Linear Programming (LP) refers to a class of problems in which the *objective function* and the *constraints* are linear. In history, LP theory has been applied to a wide range of combinatorial problems. Traditionally, the *simplex* algorithm ([Dantzig, 2016](#)) has been used to solve them. Although, on average, the method exhibits polynomial-time performance, in the worst case, it can take exponential time ([Klee & Minty, 1972](#)).

Eq. (2.2) describes an example of a Linear Programming problem formulation. Here, coefficients c_i denotes the linear objective function and coefficients a_{ij} and b_j refer to the constraints in the form of a linear inequality.

$$\min \quad \sum_{i \leq n} c_i \cdot x_i \quad (2.2)$$

$$\text{s.t.} \quad \sum_{i \leq n} a_{ij} x_{ij} - b_j \leq 0, \quad \forall j \in [1..m] \quad (2.2a)$$

$$x_i \geq 0, \quad \forall i \in [1..n] \quad (2.2b)$$

Furthermore, LP problems can also be described using a matrix notation. Eq. (2.3) describes an example of a maximization problem using matrix formulation. In the referred example, we aim to optimize a vector x , given the objective function described by the vector c and the set of constraints defined by matrix A and vector b .

$$\begin{array}{ll} \max & c^T x \end{array} \quad (2.3)$$

$$\text{s.t.} \quad Ax \geq b, \quad (2.3a)$$

$$x \geq 0 \quad (2.3b)$$

When designing solutions for LP problems, a graphical solution can facilitate the problem visualisation. Fig. (2.1) illustrates the graphical visualisation of an LP problem described in Eq. (2.4). In this example, two variables were used, and the set of constraints is represented as lines. The constraints delimit the feasible solution, which is marked as a shaded area. A red dashed line represents the objective function, and the optimal point is highlighted in red.

$$\begin{array}{ll} \max & 15x_1 + 9.3x_2 \end{array} \quad (2.4)$$

$$\text{s.t.} \quad 7x_1 + 3x_2 \leq 15, \quad (2.4a)$$

$$-x_1 + 2.2x_2 \leq 10, \quad (2.4b)$$

$$x_1, x_2 \geq 0 \quad (2.4c)$$

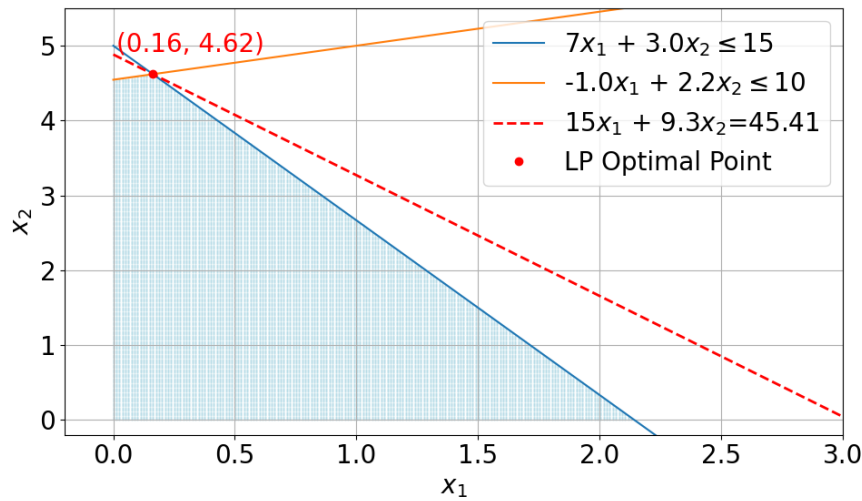


Figure 2.1 – LP problem example. The problem, described in Eq. (2.4) contains two variables and two constraints. The red dashed line represents the objective function. The feasible space is shaded and the optimal point is highlighted.

2.1.2 Mixed Integer Linear Programming

In Linear Programming, optimization problems may involve both *discrete* (integer) and *continuous* (fractional) variables. Such cases are referred as Mixed Integer Linear Problems (MILP).

Fig. 2.2 shows an example of an MILP problem. This problem is the same as described in Eq. (2.4), but considering the variables x_1 and x_2 are integer (discrete). The feasible set now is composed of discrete values and the optimal point is distinct from the one observed in Fig. 2.1, since only integer values are allowed to be assigned to the variables.

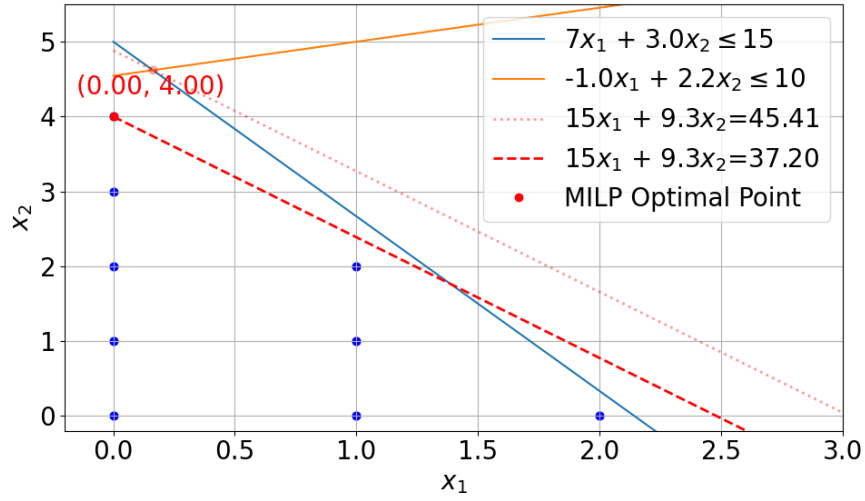


Figure 2.2 – MILP problem example. The variables are now integers. The points in blue compose the feasible solution set. The lines in blue and orange are the problem constraints. We display two values for the objective function: (i) the solution for the fractional problem, in red and dotted line, with value 45.41; and (ii) the solution for the integral problem (MILP), in red and dashed line, with value 37.20.

Optimization problems involving integer variables are usually very hard to solve and are often classified as *NP-hard*. As a result, usually, they do not support algorithms finding the optimal solution in polynomial time.

Researchers have proposed various approaches for addressing MILP problems. One widely used method is the *branch-and-bound* algorithm (Land & Doig, 2009), which breaks an optimization problem into smaller sub-problems and then, eliminates the ones with unfeasible solutions. Another algorithm family is the *approximation algorithms* (Williamson & Shmoys, 2011), which search for near-optimal results with proven guarantees. The *approximation algorithms* are discussed in more detail in Section 2.3.

2.1.3 Convex Problems

Another important category of optimization problems is the *convex optimization problems*. This refers to a class of optimization problems, in which the objective function and constraint functions are *convex*. A formal definition is given in Definition 2.1.2.

Definition 2.1.2 (Convex Problem). (Boyd & Vandenberghe, 2004) A *convex optimization problem* is an optimization problem which can be formulated in the following way:

$$\min \quad f_0(x) \quad (2.5)$$

$$\text{s.t.} \quad f_i(x) \leq 0, \forall i \in [1..m] \quad (2.5a)$$

$$h_i(x) = 0, \forall i \in [1..p] \quad (2.5b)$$

Where the domain is a convex set and $f_0(x), \dots, f_m(x), h_1(x), \dots, h_p(x)$ are convex functions.

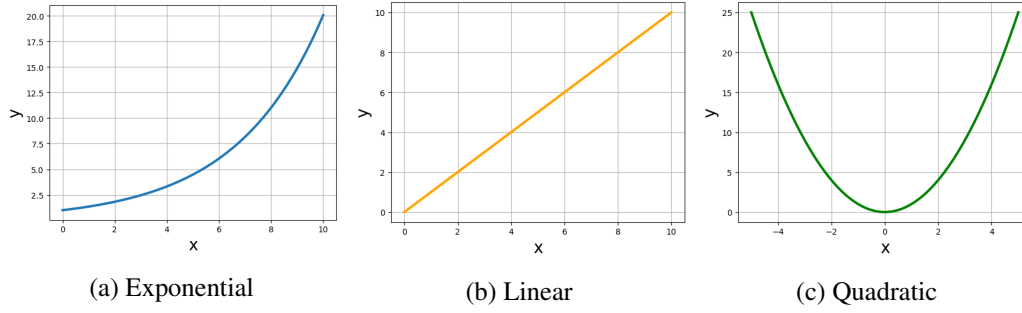


Figure 2.3 – Example of three convex functions: (a) exponential, (b) linear, and (c) quadratic

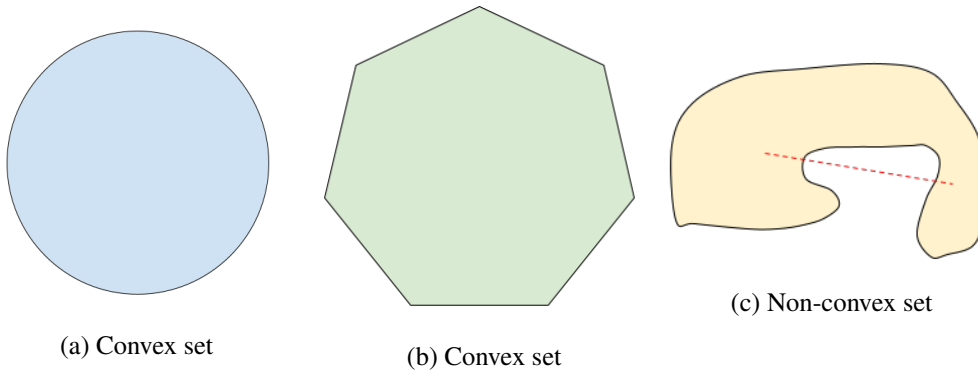


Figure 2.4 – Convex and non-convex sets.

Graphically, a *convex set* is defined if the line segment between any two points of the set also lies in the set. Fig. 2.4 illustrates some examples of convex and non-convex sets.

Formally, as discussed in [Boyd et Vandenberghe \(2004\)](#), a function $f : \mathcal{R}^n \rightarrow \mathcal{R}$ is defined as *convex* if its domain is a convex set and if, for every $x, x' \in \text{dom } f$ and for any $0 \leq \theta \leq 1$, the inequality described in Eq. (2.6) holds. Fig. 2.3 illustrates some examples of convex functions.

$$f(\theta x + (1 - \theta)x') \leq \theta f(x) + (1 - \theta)f(x') \quad (2.6)$$

Transforming a maximization to a minimization problem We can transform a maximization concave problem into a minimization convex problem. Eq. (2.7) refers to a maximization problem, where the function $f_0(x)$ is a concave function and the functions $f_1(x), \dots, f_m(x)$ are convex. If we consider a minimization problem with objective function $-f_0(x)$, we get a minimization convex problem.

$$\max \quad f_0(x) \quad (2.7)$$

$$\text{s.t.} \quad f_i(x) \leq 0, \forall i \in [1..m] \quad (2.7a)$$

$$a_i^T x = b, \forall i \in [1..p] \quad (2.7b)$$

2.2 Duality

2.2.1 Lagrangian

An optimization problem may be reformulated into its *dual Lagrangian form*. For this transformation, we extend the original objective function with a weighted sum of the constraints. For a minimization problem, as presented in Eq. (2.1), the Lagrangian dual function is denoted as follows:

$$L(x, \lambda, \mu) = f_0(x) + \sum_{i=1}^m \mu_i f_i(x) + \sum_{i=1}^p \lambda_i h_i(x) \quad (2.8)$$

In the above equation, the constants μ_i and λ_i are denoted as *Lagrangian multipliers* and weight the inequality and equality constraints, respectively. These multipliers play a fundamental role in the Lagrangian formulation.

Based on the Lagrangian formulation, we define the *dual function* $g : \mathcal{R}^m \times \mathcal{R}^p \rightarrow \mathcal{R}$ as the infimum of the Lagrangian over the problem domain $\mathcal{D}$. The dual function is described in Eq. (2.9).

$$g(\lambda, \mu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \mu) \quad (2.9)$$

2.2.2 Dual problem

We can derive an optimization problem, referred to as the *dual problem*, from maximizing the dual function. It provides a lower bound for the original problem, which is referred to as the primal. This bound is useful for algorithm design, especially in the context of convex optimization.

The dual optimization problem associated with the primal formulation (Eq. 2.1) is described in Eq. (2.10).

$$\max \quad g(\lambda, \mu) \quad (2.10)$$

$$\text{s.t.} \quad \mu_i \geq 0, \quad \forall i \in [1..m] \quad (2.10a)$$

In the dual problem, the variables are the Lagrangian multipliers $\mu = [\mu_1, \dots, \mu_m]$ and $\lambda = [\lambda_1, \dots, \lambda_p]$. Eq. (2.10) describes the maximization of the objective dual function. Eq. (2.10a) sets that the Lagrangian multipliers associated with the primal should assume non-negative values.

We denote as *duality gap* the difference between the optimal values for the primal and dual problems. In general, the duality gap may be non-zero. However, when the primal problem is convex, the duality gap becomes zero, i.e., the optimal values of the primal and dual problems coincide. For such cases, the problem is said to satisfy the *strong duality* condition as described in Theorem 2.2.1.

Theorem 2.2.1 (Strong Duality Condition). *Let the primal be a convex optimization problem. Then, the strong duality holds, i.e., the optimal values for the primal and the dual values are equal and the dual optimal value achieves its maximal value.*

The above theorem plays a fundamental role in convex optimization. In several cases, dual problems are easier to solve. Then, the dual formulation may be used for finding the optimal values or to validate the problem feasibility.

2.2.3 Karush-Kuhn-Tucker (KKT) conditions

In the context of optimization theory, certain conditions are necessary to achieve an optimal solution. The Karush-Kuhn-Tucker (KKT) conditions provide the necessary conditions for achieving optimality in optimization problems.

Let us assume an optimization problem with both equality and inequality constraints, such as described in Eq. (2.1). Moreover, the objective function $f_0(x)$ and the constraints functions, $f_1(x), \dots, f_m(x)$ and $h_1, \dots, h_p(x)$ are differentiable. We denote x^*, λ^*, μ^* any primal and dual variables attending the KKT conditions, not necessarily the optimal values.

According to [Boyd et Vandenberghe \(2004\)](#), the KKT conditions consist of the following four criteria:

Condition 1: Zero-gradient Lagrangian At the optimal point, the Lagrangian function $L(x^*, \lambda^*, \mu^*)$ must present zero-gradient at the point:

$$\nabla f_0(x^*) + \sum_{i=1}^m \mu_i^* \nabla f_i(x^*) + \sum_{i=1}^p \lambda_i^* \nabla h_i(x^*) = 0 \quad (2.11)$$

Condition 2: Primal feasibility The solution must satisfy all primal (original) problem constraints:

$$h_i(x^*) = 0, \forall i \in [1..p] \quad (2.12)$$

$$f_i(x^*) \leq 0, \forall i \in [1..m] \quad (2.13)$$

Condition 3: Dual feasibility The Lagrangian multipliers associated to inequality constraints μ must be feasible and non-negative:

$$\mu_i^* \geq 0, \forall i \in [1..m] \quad (2.14)$$

Condition 4: Complementary slackness For each inequality constraint, the product of the inequality constraint function and its corresponding inequality is zero as formalized in Eq. (2.15).

$$\mu_i^* f_i(x^*) = 0, \forall i \in [1..m] \quad (2.15)$$

Then, for each constraint, either:

- $f_i(x^*) = 0$, which means that the constraints is "*tight*". Then, the Lagrangian multiplier can assume positive values (It is possible that $\mu_i > 0$).
- $\mu_i = 0$, which means that the constraints may not be "*tight*". There is a "*slack*" value. (It is possible that $f_i(x) < 0$)

For *convex problems*, the KKT conditions (Eqs. 2.11 - 2.15) are *necessary and sufficient* for achieving the optimal point. This is because, in convex functions, any local optimal value is also a global optimal value. Verifying the KKT conditions is then sufficient for optimality.

In practice, different heuristics and algorithms for convex problems are proposed based on the KKT conditions. These methods aim to find optimal solutions with a low time complexity.

2.3 Approximation algorithms

Optimization problems may be classified into several classes (e.g., P , NP , NP -complete) based on their time complexity. Formally, as described in [Cormen, Leiserson, Rivest, et Stein \(2022\)](#) the class P comprises the optimization problems solvable in polynomial time, i.e., they can be solved in $O(n^k)$, where n is the input size and k is a constant. The class NP denotes the problems that are verifiable in polynomial time, i.e., verifying if a given input is optimal takes polynomial time in the input size. The class NP -complete comprises all the NP problems that are "as hard as" any other NP problem.

Many optimization problems studied by researchers are known to be NP -complete. This means that, unless $P = NP$, there is no algorithm capable of finding an optimal solution in polynomial time. Consequently, finding an exact solution for these problems is often infeasible in practice.

To address these problems, *approximation algorithms* have been proposed. Their idea is to provide near-optimal results in polynomial time with proven guarantees. Usually, near-optimal results are good enough ([Cormen et al., 2022](#)). Approximation algorithms can be useful for solving difficult problems. A formal definition can be found in Definition 2.3.1.

Definition 2.3.1 (Approximation algorithm). An approximation algorithm is an algorithm that produces a solution with proven guarantees in polynomial time. The theoretical bounds provided by the approximation algorithms may be classified as *relative* or *absolute*. A *relative* ρ -approximation algorithm, for every instance of a given optimization problem, provides a solution with value V , such that:

$$\max\left(\frac{V}{V^*}, \frac{V^*}{V}\right) \leq \rho \quad (2.16)$$

where V^* is the value of the optimal solution, and $\rho \geq 1$ is the approximation ratio of the algorithm. The closer the value of ρ is to 1, the better the solution generated by the algorithm.

On the other hand, an *absolute* approximation algorithm provides *additive* guarantees with respect to the optimal solution, according to the following:

$$V^* + c \leq V \leq V^* - c \quad (2.17)$$

where c is a constant term which refers to the additive approximation bounds. We remark that the approximation algorithm aims to find a solution with the smallest possible approximation bound.

One important remark about the previous definition applies to both maximization and minimization problems. For a relative approximation algorithm, described in Eq. (2.16), in a minimization problem, we have $V \leq \rho \cdot V^*$. On the other hand, in a maximization problem, $V^* \leq \rho \cdot V$. Similarly, for an absolute approximation algorithm, according to Eq. (2.17), in a minimization problem, the inequality $V - V^* \leq c$ stands. In a maximization problem, the guarantees are given by $V^* - V \leq c$.

One widely used strategy for designing approximation algorithms is through *LP-relaxation*. This approach involves relaxing (removing) integrality constraints, converting an MILP problem (NP -complete) into an LP problem, which is solvable in polynomial time. Then, the fractional solution can be converted back to a feasible (integer) solution (referred to as "rounding" phase), with a guaranteed approximation ratio. [Gonzalez \(2007\)](#) analyzes several approximation algorithms and divides them into two distinct classes:

- (i) *Non-deterministic (randomized)*: Involves some probability when rounding the LP solution.

- (ii) *Deterministic*: Methods that deterministically choose the value (with no stochasticity) for the integer variables when "rounding" the solution.

Approximation algorithms can be applied to various problems. Following, we provide a use-case example for *vertex cover*.

Example: Vertex Cover problem In a Vertex Cover, given a graph $G = (V, E)$, where V refers to a set of vertices and E refers to the set of edges. In the Vertex Cover problem, the objective is to find a set with the minimum number of vertices such that, for every edge in the graph, at least one of the endpoint vertices is present in the selected set. Eqs. (2.18) and (2.19) describes, respectively, the MILP and the LP (with fractional relaxation) formulations.

MILP:

$$\min \sum_{u \in V} x_u \quad (2.18)$$

$$\text{s.t. } x_u + x_v \geq 1, \forall (u, v) \in E \quad (2.18a)$$

$$x_u \in \{0, 1\}, \forall u \in V \quad (2.18b)$$

LP:

$$\min \sum_{u \in V} x_u \quad (2.19)$$

$$\text{s.t. } x_u + x_v \geq 1, \forall (u, v) \in E \quad (2.19a)$$

$$0 \leq x_u \leq 1, \forall u \in V \quad (2.19b)$$

[Gonzalez \(1995\)](#) addresses the above problem, for instance. He proposes solving the above problem by (i) first, solving the LP problem, described by Eq. (2.19); (ii) then, "rounding" the solution of fractional relaxation to an integral solution by rounding the variables $x_u \geq 0.5$ to 1 and $x_u < 0.5$ to zero. The above procedure can be demonstrated to give a 2-approximation algorithm.

2.4 Solvers

A range of mathematical solvers has been developed for addressing optimization problems with diverse characteristics. These solvers may differ in their capabilities and licensing models. Below, we describe some of the most widely used solvers in academic environments.

The *GNU Linear Programming Kit (GLPK)* ([Makhorin, s. d.](#)) is an open-source solver that addresses LP and MILP problems. It is lightweight and freely available. *CPLEX* ([Cplex, s. d.](#)) is a proprietary solver developed by IBM designed for solving LP and MILP problems. Moreover, it is also capable of solving quadratic optimization problems. It offers an academic version with no costs. *Gurobi* ([Gurobi Optimization, LLC, s. d.](#)) is another proprietary solver, widely used in industry, which addresses LP, MILP and quadratic optimization problems.

Some solvers also support optimization of particular convex problems, including the ones containing exponential functions, as seen in Chapter 4. For instance, *COPT* ([Ge, Huangfu, Wang, Wu, & Ye, s. d.](#)) is a proprietary solver capable of handling exponential functions. However, it does not support Mixed Integer Problems containing exponential functions. On the other hand, *Mosek* ([ApS, 2022](#)) is a widely used commercial solver designed for several optimization problems, such as LP, MILP, quadratic and exponential problems. It is freely available for academic purposes.

In this thesis, we utilize MOSEK due to its ability to address problems with general convex functions. Furthermore, the solver provides a large documentation and can be easily integrated through optimization frameworks, such as `cvxpy`.

Related works for scheduling compressible tasks

In this chapter, we present the state-of-the-art of the techniques for scheduling ML inference tasks jointly applying model compression. We first discuss model compression techniques, presenting the main approaches found in the literature, including compression techniques applied during inference. Then, we discuss scheduling. We present a literature review of the main algorithms used for scheduling machine learning (ML) inference tasks and compressible tasks. Finally, we provide a literature review of energy-aware scheduling algorithms.

3.1 Neural Network Compression	29
3.2 Scheduling	30
3.2.1 Compressible tasks scheduling	31
3.2.2 Cloud-to-edge Continuum Scheduling	32
3.2.3 ML inference tasks scheduling	32
3.3 Energy-Aware scheduling	33
3.3.1 Scheduling compressible tasks	33

In this chapter, we present the state-of-the-art techniques for scheduling ML inference tasks that jointly apply model compression (**contributions C1 and C2**). We first discuss model compression techniques, presenting the main approaches found in the literature, including compression techniques applied at inference time, and provide a comparison with traditional techniques.

Then, we discuss the state-of-the-art for scheduling approaches (**contribution C1**). We present a literature review of the main algorithms used for scheduling tasks allowing compressible processing times. Then, we review scheduling techniques for Machine Learning (ML) inference tasks.

Finally, we provide a literature review of scheduling algorithms that aim to minimize energy consumption or set an energy budget (**contribution C2**). We review works performing energy-aware scheduling of ML inference tasks and then, we review scheduling compressible tasks to minimize energy consumption.

3.1 Neural Network Compression

Model compression plays a crucial role in Edge Computing, as it enables the deployment of lightweight models, resulting in savings in computation, memory, and communication bandwidth. A large review (Cheng et al., 2018) investigated model compression techniques for Deep Learning (DL), such as *model pruning* and *knowledge distillation*.

Model pruning involves setting redundant or less important parameters, such as neurons, weights and layers, to zero. The main principle of this technique is that, in over-parameterized models, many parameters contribute minimally to the final output (LeCun, Denker, & Solla, 1989; Zhu & Gupta, 2017). The pruning process is described in Fig. 3.1. Then, removing them leads to the model size reduction and the efficiency improvements.

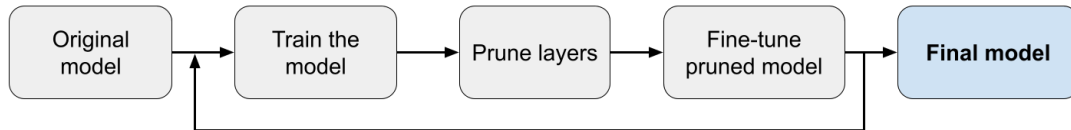


Figure 3.1 – Pruning method description diagram. At each round of pruning (shrinking the model), the weights need to be retrained.

Knowledge distillation consists of a compression technique, where the learning is transferred from a large model (teacher) to a small model (student). A seminal work was proposed by Hinton, Vinyals, et Dean (2015). In knowledge distillation, by trying to mimic the teacher’s performance, the students models can present comparable or even superior performance and are applied to several applications, such as object detection (G. Chen, Choi, Yu, Han, & Chandraker, 2017) and language processing (Sanh, Debut, Chaumond, & Wolf, 2019).

These methods are usually applied during the training phase and require a significant amount of time to update the weights. They cannot be applied for compressing models at inference time due to their prohibitive times, energy, and cost for updating the weights.

Model compression during inference To address this problem, recent approaches (J. Yu & Huang, 2019a; Cai et al., 2020) have been proposed for compressing Neural Networks (NNs)

during inference. They propose, then, decoupling the training and architecture search, which is done in the inference phase after the weights are computed, as described in Fig. 3.2.

Yu and Huang (J. Yu & Huang, 2019a) proposed the AutoSlim, an approach that performs NN model compression at inference time. According to their approach, when an ML inference request is executed, it is possible to set the number of channels in order to achieve the best accuracy under resource constraints.

Their approach consists of the following steps: First, a "*slimmable*" Convolutional Neural Network (CNN) is trained (J. Yu & Huang, 2019b) in a one-shot approach. During the training process, the model is sampled with several width configurations. Then, during inference, a sub-neural network may be sampled for executing the inference request using the corresponding weights. For selecting the sub-network, a greedy heuristic is proposed. In this method, the layers are greedily reduced, starting with those that have the smallest accuracy drop. The process is repeated until the resource constraints are met.

The authors validated their approach for image classification models: *MobileNet*, *MobileNetV2*, *ResNet*, and *MnasNet*. Their findings indicate that the approach achieves considerable accuracy, outperforming models with similar size that do not utilize the AutoSlim architecture.

Cai et al. (Cai et al., 2020) propose a similar strategy for model compression, referred to as Once-For-All (OFA). Their approach aims similarly compressing the model at inference time, with no need of retraining weights. They target CNN models used for image classification tasks. In their approach, the model can be shrunk in four dimensions: *width*, *depth*, *kernel size*, and *input resolution*.

The training process, denoted as progressive shrinking, initially trains the weights with the model in its largest configuration. Then, smaller sub-networks are progressively added to the sampling space during the training rounds. At each training round, one of the sub-networks is randomly picked from the sampling space for the forward and backward propagation. The idea of the above technique is that the models weights do not interfere across the multiple sub-networks.

During inference, a sub-network is sampled for executing the inference request. For this, the authors propose two structures to predict the model performance:

- *Neural-network-twin* for predicting a sub-network accuracy. The twin model was trained on a dataset comprising 16,000 sub-networks and 10,000 images from the validation set.
- *Lookup Table* for predicting the sub-network latency. For each possible layer configuration, the latency is measured and used to estimate the sub-network's total latency.

The authors propose using the two above structures in an evolutionary search that aims to find the sub-network with the maximum accuracy while meeting the latency constraints. The proposed methods, tested on devices with multiple hardware characteristics, outperformed the existing literature for image classification tasks.

3.2 Scheduling

Scheduling involves optimizing resource allocation for running jobs. In this problem, the systems should decide when and where each task should be processed in order to maximize some objective function, e.g., minimize the completion time. One classical constraint present in traditional scheduling problems is the deadline, which indicates the maximal end time of a task.

Scheduling problems may present different complexity levels. They can be classified as multi-stage (when the tasks are composed of multiple steps) or single-stage (when there is only one

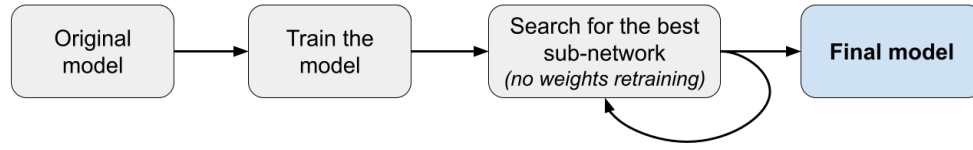


Figure 3.2 – Machine Learning compression during inference time (Cai et al., 2020; J. Yu & Huang, 2019a). The search for the best sub-network is done at inference time, without requiring retraining the model.

operation per task). For the single-stage problems, the scheduling problems are as well classified as: (i) single-machine; (ii) identical machines, when the machines have the same speed; (iii) parallel machines, when the machines may have different speeds; and (iv) unrelated, when the processing time may vary for different machines.

A large number of works have been proposed to address the scheduling problem. A comprehensive review of this topic was developed by B. Chen et al. (1998). Since scheduling is normally an NP-hard problem and does not support exact solutions in polynomial time, different methods have been proposed. Two main paradigms are used: (i) exact algorithms, which search for the optimal solution, using methods such as branch-and-bound and dynamic programming; and (ii) approximation algorithms, such as heuristics.

A seminal example of a heuristic is the work of Liu and Layland (C. L. Liu & Layland, 1973) in which they proposed the Earliest Deadline First (EDF) algorithm. Following this approach, the tasks are selected based on their deadlines in order to guarantee that the deadlines are met.

3.2.1 Compressible tasks scheduling

Some works addressed the problem of scheduling, enabling the compression of task processing time. In these systems, researchers argue that adding computational resources makes task execution faster, thus reducing processing time. However, the compression comes with some costs, usually associated with the utility drop. Vickson (Vickson, 1980) was the first to propose controlling the processing times in a single machine without deadlines.

Then, some works addressed the scheduling problem in a multi-machine scenario. Alidaee et al. (Alidaee et al., 1993) considered a scheduling problem with single-operation tasks and multiple parallel (non-identical) machines. The tasks may have their processing time compressed with a linear cost. Furthermore, in the evaluated problem, a single deadline poses a limit for all tasks. They analyzed two distinct objective functions: (i) to minimize the maximum completion time (or makespan), i.e., the maximum ending time of a task; and (ii) to minimize a weighted function of lateness and earliness. The authors transformed the scheduling problems into transportation problems in order to solve them.

R. Wu, Bao, et al. (2023) analyzes a scheduling problem with multiple tasks and multiple machines. Each task can be compressed on discrete levels. Furthermore, the tasks present multiple arrival times. The scheduler system, then, must assign the tasks to a machine (from a subset of all the machines), such that no machine processes more than one task simultaneously. The authors then developed a heuristic with guarantees for addressing the problem.

In [Shabtay et Kaspi \(2006\)](#), the authors address a scheduling system with multiple identical machines. The tasks may present multiple processing times and are preemptive, i.e., the tasks may be stopped and then, resumed by the system.

The above works were proposed for addressing scheduling problems with compressible tasks in a multiple-machine environment. However, the tasks in these works do not have deadlines or present a common deadline for all tasks.

3.2.2 Cloud-to-edge Continuum Scheduling

In the cloud-to-edge continuum, a large number of works have addressed the scheduling problems such as optimizing completion time, latency and energy ([Q. Liu & Fang, 2023](#); [Huang, Wang, Wang, & Liu, 2019](#); [Angelelli et al., 2023](#); [Giroire, Huin, Tomassilli, & Pérennes, 2019](#); [Aslanpour, Toosi, Cheema, & Gaire, 2022](#)). In this context, one of the first scenarios considering *compressible* tasks with several possible processing times is the transcoding of video streams ([D. Lee & Song, 2021](#)). In such cases, video chunks can be transcoded into different video quality levels, allowing users to choose based on their device and network conditions. This choice leads to a trade-off between the utility (the Quality of Experience, QoE) and the task processing time (the transcoding time). The authors model the optimization problem to maximize the QoE, ensuring that all deadline constraints are met. For this, they developed a heuristic that assigns tasks to processors and to a discrete set of quality levels (resolution), prioritizing tasks with earlier deadlines.

3.2.3 ML inference tasks scheduling

Recently, a collection of works, e.g. ([H. Zhang et al., 2017](#); [Salem, Castellano, Neglia, Pianese, & Araldo, 2021](#); [Jin et al., 2020](#); [Ogden & Guo, 2018](#)), studied how to efficiently allocate inference requests for Machine Learning (ML) models among network nodes with available computational resources. For example, [Salem et al. \(2021\)](#) introduce an approximation algorithm to allocate inference tasks in the cloud-to-edge continuum, while minimizing a cost function that combines latency and accuracy error. However, this work does not investigate the scheduling of the requests.

The closest work to ours are ([Jiang et al., 2020](#); [Fang, Zeng, & Zhang, 2018](#); [Nigade, Bauszat, Bal, & Wang, 2022](#)). They investigate the scheduling of DL inference tasks, while proposing creating a collection of DDN models using compression. [Jiang et al. \(2020\)](#) studied scheduling Deep Neural Network (DNN) mobile vision based tasks that need to be scheduled in a FPGA system. In the evaluated scenario, a task can be served by a pool of DNN models built using model quantization techniques. The authors propose methods to select the model configuration and the hardware profile to maximize the QoE (Quality of Experience). Similarly, [Fang et al. \(2018\)](#) proposes to maximize a utility function based on latency and accuracy ([Fang et al., 2018](#)). However, both works have different objective functions with respect to this thesis and do not consider deadline requirements as hard constraints for scheduling. [Nigade et al. \(2022\)](#) study scheduling DL inference requests. They aim to maximize the accuracy of a set of DL model inference requests while satisfying network and processing latency constraints. For this, the scheduler system decides, for each client, a worker (execution place, usually associated with a GPU card) where the task will be deployed and a DNN model from a model pool to perform the task.

However, all the above-mentioned works use a limited number of models, constrained by the storage constraints of devices. Such a system does not scale up if we want to arbitrarily increase the number of possible configurations to adapt to any real system. On the contrary, in this thesis, we consider fully compressible tasks using adjustable neural networks to any arbitrary size, without requiring to store a high number of models. Studying it allows us to be the first, to the best of our knowledge, to propose a *solution with proved guarantees* to choose any arbitrary compression level and the machine on which to execute each task, while maximizing the global accuracy and satisfying deadline constraints for a large family of accuracy functions.

3.3 Energy-Aware scheduling

Additionally, several papers have addressed the scheduling problem with the goal of minimizing energy consumption. [Xu et al. \(2020\)](#) investigated performing Deep Neural Network (DNN) inference requests in a Mobile Edge Computing (MEC) environment. In the studied context, they observe that (i) servers are too far from users, challenging delay requirements, and (ii) mobile devices present constraints in computation and energy resources. Then, the authors investigate offloading a portion of the inference requests to the server to minimize energy consumption and computational resources usage while meeting latency requirements.

[Ma, Li, Zhang, Zhou, et Chen \(2023\)](#) considers a scenario where, in IoT devices, due to the computational constraints, it is unfeasible to deploy large DNNs on these devices. Then, they propose an environment where IoT devices run a smaller model (which reduces energy consumption and accuracy for performing inference requests) and, in servers, large DNNs are deployed. Then, the authors proposed to maximize a weighted function balancing reliability and accuracy within an energy budget, optimizing task allocation.

[Wang, Karimi, Xiang, et Kim \(2021\)](#) analyzed GPU energy consumption with spatial multitasking (simultaneous execution of multiple kernels). They proposed a scheduling system to balance energy efficiency and performance.

[Carastan-Santos, Da Costa, Poquet, Stolf, et Trystram \(2024\)](#) proposed a light-weight method for predicting tasks energy consumption in High Performance Computers (HPC) based on the historical data. Then, these predictions are then integrated into an EASY backfilling algorithm, which uses a first-come first serve (FCFS) policy for ordering jobs. The scheduler sets power capping constraints to ensure that the power consumption of running jobs does not exceed a specified limit.

3.3.1 Scheduling compressible tasks

Some authors ([L. Yin, Li, Lu, & Gao, 2016](#); [Tsao, Thanh, & Hwang, 2020](#)) have investigated energy-aware scheduling tasks, allowing compressing processing time. Regarding scheduling considering energy challenges, these works addressed the single-machine scenario.

[L. Yin et al. \(2016\)](#) studied scheduling tasks with controllable processing times where the tasks have deadlines and are penalized for the earliness/tardiness when the tasks' processing ends before or after the deadlines. They modeled a multi-objective optimization problem to minimize (i) the earliness and tardiness, (ii) the compression costs; and (iii) the energy consumption on a single machine, considering discrete levels of compression.

[Tsao et al. \(2020\)](#) also investigated scheduling tasks with controllable processing times in a single-machine scenario. They address minimizing a weighted function, which considers the

energy consumption and the productivity. They propose a genetic algorithm to optimize task processing times.

On the other hand, [X. Wu et Che \(2019\)](#) addressed the problem of scheduling in multiple machines with controllable processing times. They aimed to minimize makespan and energy consumption, considering variables such as job processing time (discrete levels) and the machines where tasks are assigned. They proposed an evolutionary metaheuristic for solving the problem.

However, the above works never consider both deadlines and multiple machines, and they often employ discrete levels of task compression. In contrast, our solution utilizes the Once-For-All approach ([Cai et al., 2020](#)) to accommodate compressible tasks at any arbitrary level of compression. Additionally, we are the first to propose an approximation algorithm with proven guarantees for scheduling such compressible tasks under energy constraints.

Scheduling with Fully Compressible Tasks: Application to Deep Learning Inference with Neural Network Compression

With the advent and the growing usage of Machine Learning as a Service (MLaaS), cloud and network systems are now offering the possibility to deploy ML tasks on heterogeneous clusters. Then, network and cloud operators have to schedule these tasks, determining both when and on which devices to execute them. In parallel, several solutions, such as neural network compression, were proposed to build small models which can run on limited hardware. These solutions allow choosing the model size at inference time for any targeted processing time without having to re-train the network.

In this work, we consider the Deadline Scheduling with Compressible Tasks (DSCT) problem: a novel scheduling problem with task deadlines where the tasks can be compressed. Each task can be executed with a certain compression, presenting a trade-off between its compression level (and, its processing time) and its obtained utility. The objective is to maximize the tasks utilities. We propose an approximation algorithm with proved guarantees to solve the problem. We validate its efficiency with extensive simulation, obtaining near optimal results. As application scenario, we study the problem when the tasks are Deep Learning classification jobs, and the objective is to maximize their global accuracy, but we believe that this new framework and solutions apply to a wide range of application cases.

4.1	Introduction	37
4.2	Experiments and latency-accuracy tradeoff analysis	38
4.3	Modeling	39
4.3.1	Problem formulation	39
4.3.2	Accuracy function	41
4.4	Problem on a single machine	42
4.4.1	Formulation as a Convex Program	42
4.4.2	Optimal algorithm	44
4.4.2.1	Optimality conditions	44
4.4.2.2	Algorithm proposal	45
4.5	Problem on several machines	49
4.5.1	Formulation as a Mixed Integer Program	49
4.5.2	NP-Hardness complexity	50
4.5.3	Approximation Algorithm	51
4.6	Experimental evaluation	57
4.7	Conclusion and Discussion	61

4.1 Introduction

Nowadays, with the deployment of new networks, such as 5G and IoT (Internet of Things), the traditional paradigms for placing and deploying many cloud and network services have been changed. The new hardware capabilities of devices and the network requirements, such as latency and delay, create a new scenario in which the services and functions, in particular Machine Learning as a Service (MLaaS), are deployed along the cloud-to-edge continuum (Hua et al., 2023). This topic has attracted a lot of attention and can be applied in many applications, such as self-driving cars, unmanned aerial vehicles (UAVs) (Raj et al., 2023) and Mobile Edge Computing (MEC) (Shakarami et al., 2020). In this domain, the researchers study the deployment of Deep Learning (DL) models on devices in the network edge, such as gateways, microcontrollers, antennas, and mobile phones (Drainakis et al., 2021). In this way, the models may provide accurate inferences close to the users. However, along this continuum, the edge devices present strong constraints in terms of disk space, computational power, and memory, limiting the latency and accuracy of the models (W. Gao et al., 2022).

Thus, it is necessary to find ways for running complex neural networks in devices with constrained hardware (Lin, Chen, Cohn, Gan, & Han, 2020). A first solution is to design light neural networks to be used on specific devices, e.g. MobileNet (Howard et al., 2019) for cellphones. Another technique, suitable for a large set of devices, is to use model compression techniques. The principle is to reduce the size of an efficient neural networks in order to reduce its memory and storage usage while having a minimal accuracy drop (Cheng et al., 2018). Several techniques were proposed to compress Neural Networks (NN) during training, for example pruning (S. Liu et al., 2021), in which less relevant weights (e.g. the one with smallest values) are discarded, quantization (P. Liu et al., 2022), where input or weight sizes are decreased (e.g. by using fewer bits to store the numbers). Recent works, such as Yu and Huang (J. Yu & Huang, 2019a) and Cai et al. (Cai et al., 2020), propose to train an adaptable Deep Neural Network (DNN), which can be configured during the inference without requiring to re-train the network. For example, the Once-For-All (OFA) solution (Cai et al., 2020) allows to adjust the width and kernel size of each layer, the model depth, and the input image size to build faster models at inference time.

In this work, we study how to *use such adjustable neural networks* to deploy a set of DL models in a networked system. The inference requests are considered as tasks, which have to be scheduled along the cloud-edge continuum before their deadlines. Each task usually has a fixed duration and the scheduling problem can be solved by traditional techniques, such as Earliest Deadline First (EDF) and Shortest Job First (SJF) (Javaid et al., 2019). The requests can be executed on very heterogeneous devices using compressed networks. Recent works (Jiang et al., 2020; Fang et al., 2018; Nigade et al., 2022) have proposed to use a collection of pretrained models with different sizes (i.e., different number of parameters) and to select the best one to optimize latency and accuracy. Here, we propose to go a step further by allowing a machine to use *any* level of compression on the model size. Indeed, given any processing time target, adjustable networks, thanks to their very large number of potential configurations (e.g. for MobileNet, more than 10^{19} possible architectures), allow to produce a model with a size (and then, a processing time) very close to the targeted time. This leads to the following optimization problem.

In this chapter, we study the new problem of scheduling a set of *fully compressible tasks* with deadlines, in which *each task can be run by a compressed algorithm with an arbitrary number of parameters*, presenting a trade-off between its processing time and the obtained utility. The goal is to decide when, on which machine, and with which compression level, each task should

Parameters	Values
Resolution	[16, 32, 48, 64, 96, 128, 144, 160, 176, 192, 224]
Depth	[1, 2, 3, 4]
Width factor	[0.6, 0.8, 1.0]
Expansion Ratio	[0.2, 0.25, 0.35]

TABLE 4.1 – Expansion of OFA experiments range

be processed in order to maximize the global utility. We propose *convex optimization models* and an *approximation algorithm* to solve the problem for a large family of utility functions. We used as example image classification inference tasks solved using the OFA technique, but our framework is general, and can be applied to any task following the classic law of diminishing marginal utility (Cournot, 1838). The contributions can be summarized as follows:

- We study the scheduling problem in which tasks can be compressed. The tasks have a utility function expressing their trade-off between processing time and utility. We focus on classification tasks solved with compressible neural networks. In this case, the utility is given by the attained accuracy.
- We carry out experiments to study this trade-off for the OFA (Cai et al., 2020) solution, particularly for low processing times.
- We model the accuracy function of large families of compressible tasks, as concave differentiable functions, and we propose convex optimization models to solve the problem exactly.
- We discuss the problem complexity and propose (i) an exact algorithm to solve the problem on one machine, and (ii) an approximation algorithm with proved guarantees for several heterogeneous machines.
- Finally, the solutions are validated and compared to state-of-the-art solutions with extensive experimentation.

The rest of this chapter is organized as follows. In Section 4.2, we describe the experiments done to generalize the OFA solution and to get its accuracy function. In Section 4.3, we present the problem formulation. In Sections 4.4 and 4.5, we present details and analysis of our scheduling algorithms. In Section 4.6, we evaluate our proposed algorithms. In Section 4.7, we discuss our solution and conclusions are drawn, together with open questions for future work.

4.2 Experiments and latency-accuracy tradeoff analysis

In this section, we derive the latency-accuracy trade off of families of compressible neural networks such as (J. Yu & Huang, 2019a; Cai et al., 2020) experimentally. Indeed, in their works, the authors study a limited range of reached accuracy, mostly between 70% and 80%, as their goal is to obtain efficient models on devices with limited hardware. For our problem, we need to assess the latency-accuracy trade-off for the full accuracy range for processing times from 0 to the maximum one. For this, we conducted new experiments. We selected Once-For-All (OFA) (Cai et al., 2020), as it obtains better results than other similar approaches.

As discussed in Section 3.1, OFA approach reduces the model by changing the value of 4 different parameters: depth, width, kernel size, and resolution. We extended the possible configu-

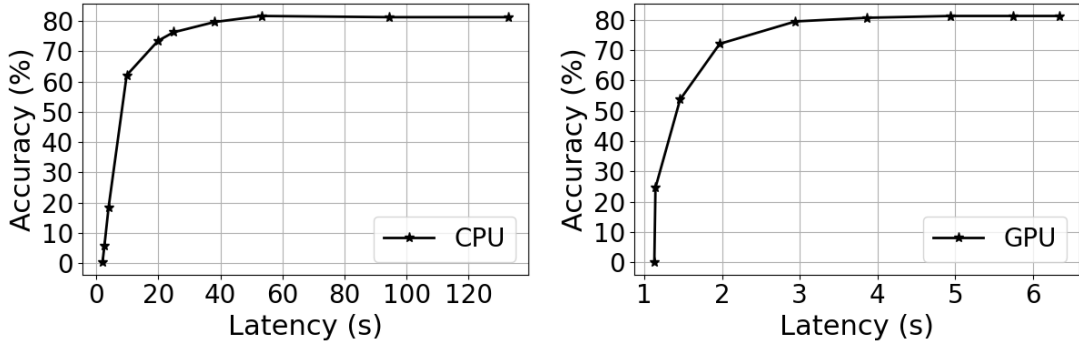


Figure 4.1 – Accuracy vs. Latency using the OFA framework for classifying a set of 1000 images from `imagenet-1k`. Experiments on a CPU (left) and GPU (right).

rations by adding new values for *resolution* and *depth*, as they are the parameters with the largest impact on the accuracy. Added values are given in bold in Table 4.1.

We used the classical `imagenet-1k` dataset (Deng et al., 2009), which provides 1000 images classes and we used the pre-trained `resnet` model (He, Zhang, Ren, & Sun, 2016) made available by the authors. Then, we performed fine-tuning to adapt the weights over the new configuration range over 20 epochs. OFA also uses an evolutionary search during the inference to choose the best model configuration for a maximal input latency. This approach is based on a lookup table to predict the latency, and on a neural network to predict the configuration accuracy.

We then evaluated the updated `OFA-resnet` in two different devices: one equipped with a CPU (Intel Core i9-12900H), and the other one with a GPU (NVIDIA RTX A2000). Results are shown in Fig. 4.1 (left) and (right), respectively. The latency and accuracy of the configurations selected by the evolutionary search algorithm are plotted. Each point represents the average accuracy reached when classifying a set with 1000 images for a given maximum latency. First, we observe that we obtained a full range for the trade-off, going from the minimal accuracy until the maximum accuracy obtained.

We also observe that in GPU the inference classification is a lot faster when compared to the CPU. The GPU and CPU experiments present a maximum latency times of 133s vs 6.33s, respectively, for the longest configuration. This difference is due the high computing capabilities present in GPU. Moreover, for both CPU and GPU cases, we see that the accuracy function is a non decreasing function presenting decreasing accuracy gains. We observe, then, on one hand, for small latency values, the curve presents very strong accuracy gains. On other hand, for large inference latency times, the model presents low accuracy gains, providing almost a flat curve.

4.3 Modeling

4.3.1 Problem formulation

Here, we define the DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS (DSCT) problem. The notation used in the chapter are summarized in Table 4.2.

Formally, we have a set $J = \{1, \dots, n\}$ of fully compressible tasks to be scheduled. Each task $j \in J$ is characterized by a deadline d_j , a maximum number of floating-point operations $F_j^{\max}$, and a utility function $u_j(f_j)$ giving its utility when processed with f_j floating-point operations.

TABLE 4.2 – Notation used in this chapter.

Params	Description	Type
n	number of tasks	Integer
J	set of tasks	$\{1, \dots, n\}$
m	number of machines	Integer
M	set of machines	$\{1, \dots, m\}$
d_j	task j deadline	Real
θ_j	task j time efficiency	Real
$F_j^{\max}$	task j max number of float operations	Real
$t_j^{\max}$	task j maximum processing time	Real
s_r	speed factor of machine r	Real
S	Sum of machines speed	Real
t_j	processing time of task j	Real
t_{jr}	processing time of task j on machine r	Real
x_{jr}	1 if task j is executed in machine r , 0 otherwise	Boolean
$a(t)$	accuracy function	Real $\rightarrow [0, 1]$
$a^{\max}$	maximum accuracy $a(t^{\max})$	Real
$a^{\min}$	minimum accuracy $a(0)$	Real
$a'_j(t_j)$	marginal gain of task j for an processing time t_j	Real
Δa	accuracy range	Real
v_j	cut processing time of task j	Real
v_j^c	cut processing time for tasks before j	Real
ρ	deadline tolerance value	Real
μ	task heterogeneity ratio	Real

Given that the number of floating-point operations f_j for task j is determined by its processing time t_j and the machine speed s (i.e., $f_j = s \cdot t_j$), we can derive the maximum processing time for task j , denoted $t_j^{\max}$, as: $t_j^{\max} = \frac{F_j^{\max}}{s}$.

The goal is to choose the processing time $t_j \in [0, t_j^{\max}]$ for each task $j \in J$ in order to maximize the global utility, i.e., $\sum_{j=1}^n u_j(s \cdot t_j)$, the sum of the utility of all processed tasks.

Moreover, we have a set $M = \{1, \dots, m\}$ with heterogeneous machines, where the tasks should be processed. Each machine $r \in M$ presents a speed factor s_r which indicates the processing rate of the machine. An illustrative example is shown in Fig. 4.2. Here, 4 independent tasks with deadlines are scheduled in a set of 3 machines.

In this chapter, we specifically focus on *image classification inference tasks* performed using compressible deep neural networks. Nevertheless, this framework is broadly applicable to other domains, such as *live video transcoding* as explored in (D. Lee & Song, 2021). In such a scenario, compression directly translates to reducing transcoding time by generating video streams with lower quality levels.

The utility function for a task $j \in J$ corresponds to the accuracy $a_j(s_r t_j)$ achieved by the neural network (or any computing process) for a given processing time, t_j , when scheduled on the

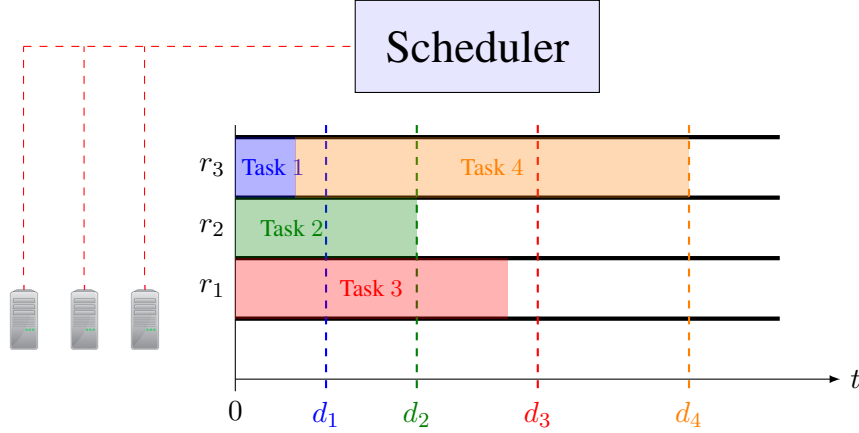


Figure 4.2 – Scheduling illustrative example with $n = 4$ tasks and $m = 3$ machines.

machine r . Given that the accuracy of a classification model is the fraction of well classified tasks, the objective function in this scenario is to maximize the *expected number of correctly classified images* across the system.

In the following, we use a *standard order* of the tasks by non-decreasing deadlines. The tasks are often denoted by their index in that order. This way the set of tasks is denoted $J = \{1, 2, \dots, n\}$ and $d_i \leq d_j$ if and only $i \leq j$.

4.3.2 Accuracy function

The accuracy of a compressible neural network—such as Once-for-All (OFA) (Cai et al., 2020) can be modeled as a function $a(f)$, where f denotes the total number of floating-point operations executed during an inference task. Recall that $f = s_r \cdot t$, where t is the processing time on a machine r operating at speed s_r . The accuracy function presents the following properties:

- Concave and non decreasing. It is based on the fact that the marginal gain of running an inference model is decreasing with the task processing time, and tending to 0 when the processing time is large. We argue that such dependency can be generally found in many tasks utilities (accuracy functions), not only in "slimmable" neural networks. For example, in Approximation Theory polynomials of degree s can be approximated with Chebyshev's polynomials of degree d up to an error which scales as $e^{-s^2/d}$ (Sachdeva, 2013, Theorem 3.3), while the classical analysis of the famous AdaBoost algorithm shows that, for a weakly PAC learnable class of functions $\mathcal{C}$, T weak learners can be combined to obtain a classifier with empirical error of order $e^{-\Theta(T)}$ (Schapire & Freund, 2012).
- Support $[0, t^{\max}]$. Performing $t = 0$ means accuracy $a^{\min}$. Note that $a^{\min}$ for a classification class with C classes corresponds to a random choice between the classes leading to an accuracy $a^{\min} = 1/C$. On other hand, presenting a processing time $t = t^{\max}$ leads to an accuracy $a^{\max} \leq 1$. Note that our model also allows setting $t^{\max} = \infty$. In this case, $a(s_r \cdot t)$ converges to the constant $a^{\max}$, when t goes to infinity.

We add a technical property:

- Differentiability (and thus continuity) over $\mathbb{R}^+$. Differentiability allows to define the *marginal gain* $a'_j(t)$ of a task $j \in J$ when executed for a time t .

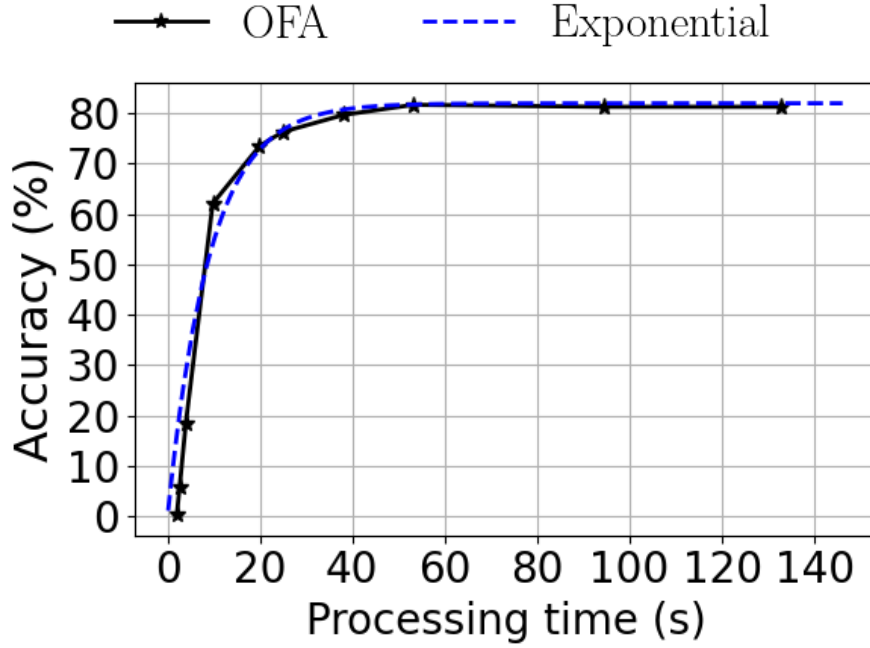


Figure 4.3 – Comparison between OFA experiments and the exponential function for accuracy-latency tradeoff

In the following, we study this *general class* of accuracy functions. We use the *exponential accuracy function* as an illustration, as it models well the behavior of OFA, studied in the previous section (See also the discussion in Section 4.7). It is defined as follows:

$$a(s_r \cdot t) = (a^{\max} - a^{\min}) (1 - e^{-\theta \cdot s_r \cdot t}) + a^{\min} \quad (4.1)$$

Here, θ denotes the slope at the origin of the accuracy function; $a^{\max}$ and $a^{\min}$ denotes the maximum and minimum accuracy, respectively.

In order to evaluate the feasibility of the exponential accuracy function, we present in Fig. 4.3 the accuracy over inference latency (processing time) for the OFA experiments described in Section 4.2 and the exponential accuracy function best fitted. We observe that the exponential function described in Eq. 4.1 closely approximates to the OFA experiments and it represents well the OFA behavior.

4.4 Problem on a single machine

4.4.1 Formulation as a Convex Program

The DSCT on a single machine can be formulated as a convex program (because of the concavity of the accuracy function) with only fractional variables. Indeed, the continuity over $\mathbb{R}^+$ of the accuracy makes that no binary variables are needed to indicate if a task is executed. Not executing a task is equivalent to setting its processing time to 0.

Objective functions. The general objective function is $\max \sum_{j=1}^n a_j(s \cdot t)$. This is equivalent to minimize the error defined as $1 - a(t)$: $\min \sum_{j=1}^n 1 - a_j(s \cdot t)$. As the accuracy $a_j(s \cdot t)$ is concave, the error $1 - a_j(s \cdot t)$ is convex. The sum of the error thus is convex as a sum of convex functions.

Specific cases. For the *exponential accuracy*:

$$\max \sum_{j=0}^n (a_j^{\max} - a_j^{\min})(1 - e^{-\theta_j s t_j}) + a_j^{\min}$$

Taking the above objective function and expanding the terms, we obtain the following objective function:

$$\max \sum_{j=0}^n (a_j^{\max} - a_j^{\min}) + (a_j^{\max} - a_j^{\min})(-e^{-\theta_j s t_j}) + a_j^{\min}$$

We observe that the constants independent of the variable t_j , such as $a_j^{\min}$ and $a_j^{\max}$, do not influence the optimal solution. Furthermore, since $\max(f(x))$ is equivalent to $\min(-f(x))$, we can express the same objective function as a minimization problem, as follows:

$$\min \sum_{j=0}^n (a_j^{\max} - a_j^{\min})e^{-\theta_j s t_j}$$

Convex optimization program with linear constraints. We modeled the DSCT in a single machine as a convex optimization problem, which is modeled in Eq. 4.2. Here, we aim to minimize the problem accuracy error. The decision variables used here are the tasks processing times, denoted by t_j for all $j \in J$. Note that the variables used here are real.

$$\min \quad \sum_{j=1}^n a_j^{\max} - a_j(s \cdot t_j) \quad (4.2)$$

$$\text{s.t.} \quad \sum_{i=1}^j t_i \leq d_j, \quad j \in J = \{1, 2, 3 \dots n\} \quad (4.2a)$$

$$t_j s \leq F_j^{\max}, \quad j \in J \quad (4.2b)$$

$$t_j \in \mathbb{R}^+, \quad j \in J \quad (4.2c)$$

Equation 4.2 defines the minimization problem for the error. In Equation (4.2a), we state that, for any task j , the processing time for each task should end before the task deadline. For this, we consider the cumulative processing time of all previous tasks, which indicates when task j starts, in addition to the processing time of task j . Equations (4.2b) and (4.2c) state that each task must perform at most $F_j^{\max}$ floating-point operations, and $t_j \geq 0$, respectively.

Algorithmic complexity and resolution methods. The DSCT thus is polynomial for a concave accuracy function. For a *differentiable* accuracy function, it can be solved using *Kelley's Cutting-Plane Method* with a linear solver or *gradient descent*. We study faster methods, with closed form formulas for specific accuracy functions, in the following.

4.4.2 Optimal algorithm

We first discuss the KKT (Karush-Kuhn-Tucker) conditions for the optimization problem and, then, we present an algorithm to solve it with polynomial time complexity.

4.4.2.1 Optimality conditions

In our convex problem, the KKT conditions are necessary and sufficient conditions for optimality (Boyd & Vandenberghe, 2004) (Chap. 5 p. 226 and 244), since the Slater's condition is satisfied. In our case, the objective function is convex, but all the constraints are affine. Thus, the Slater's condition simplifies to the existence of a feasible solution. This is clearly the case for our problem, as $t_j = 0$, for all $j \in J$, is a feasible solution. To analyze them, we first write the Lagrangian of the problem:

$$L(t, \mu) = \sum_{j=1}^n (1 - a_j(s \cdot t_j)) + \mu^T g(t) \quad (4.3)$$

where $\mu = [\mu_1, \dots, \mu_{3n}]$ and $g(t) = [t_1 - d_1, \dots, \sum_{i=1}^n t_i - d_n, t_1 s - F_1^{\max}, \dots, t_n s - F_n^{\max}, -t_1, \dots, -t_n]$. For the exponential accuracy function, we get:

$$L(t, \mu) = \sum_{j=1}^n \Delta a_j e^{-\theta_j s t_j} + \mu^T g(t) \quad (4.4)$$

We use the notation $\Delta a_j \stackrel{\text{def}}{=} a_j^{\max} - a_j^{\min}$ here and in what follows.

The *complementary slackness conditions* give that, when a task $j \in J$ is neither constrained by its deadline, nor by its maximum processing time, and has a strictly positive processing time (we say that such tasks are *3-unconstrained*), we have $\mu_j = \mu_{n+j} = \mu_{2n+j} = 0$. Furthermore, if $j+1$ is neither constrained by its maximum time nor by its minimum times (i.e., $\mu_{n+j+1} = \mu_{2n+j+1} = 0$), the optimum processing times of both tasks are derived from the *zero-gradient condition* (i.e., the gradient at the optimum (t^*, μ^*) should be 0) and we have:

$$\frac{\partial}{\partial t_j} a(t_j) = \frac{\partial}{\partial t_{j+1}} a(t_{j+1}). \quad (4.5)$$

The intuition is that, when tasks are *3-unconstrained*, the *optimum schedule* is to choose their processing time so that they have the *same marginal gain*, i.e., the same value for the partial derivative of their accuracy functions. This is due to the concavity of the accuracy function or equivalently to its decreasing marginal gain. In the case of the exponential accuracy function, this gives

$$\Delta a_j \theta_j s e^{-\theta_j s t_j} = \Delta a_{j+1} s \theta_{j+1} e^{-\theta_{j+1} s t_{j+1}}, \quad (4.6)$$

where $\Delta a_j \stackrel{\text{def}}{=} a_j^{\max} - a_j^{\min}$, and the optimum processing time of task j is given by:

$$t_j = \frac{\theta_{j+1}}{\theta_j} t_{j+1} - \frac{1}{\theta_j s} \ln \left(\frac{\Delta a_{j+1} \theta_{j+1}}{\Delta a_j \theta_j} \right), \quad (4.7)$$

The Equation (4.6) holds only if $t_j > 0$, otherwise $t_j = 0$ (in the latter case, task j is constrained by the non negativity bound and $\mu_{2n+j} > 0$).

Note that if both tasks have the same efficiency $\theta_j = \theta_{j+1}$ and same efficiency range $\Delta a_j = \Delta a_{j+1}$, then the condition simplifies to:

$$t_j = t_{j+1}.$$

Thus, the *optimal processing time* for two consecutive tasks j and $j+1$, when j is *not constrained by its deadline*, is taking *equal processing time* $t_j = t_{j+1}$.

On the other hand, the non negativity constraints of the KKT multipliers ($\mu_j \geq 0$) give that, from the zero-gradient condition, $\forall j \in \{1, \dots, n-1\}$,

$$\frac{\partial}{\partial t_j} a(t_j) \geq \frac{\partial}{\partial t_{j+1}} a(t_{j+1}) + \mu_{n+j+1} - \mu_{n+j} - \mu_{2n+j+1} + \mu_{2n+j}. \quad (4.8)$$

If both j and $j+1$ are not constrained by their maximum and have strictly positive processing time, we have

$$\frac{\partial}{\partial t_j} a(t_j) \geq \frac{\partial}{\partial t_{j+1}} a(t_{j+1}). \quad (4.9)$$

The intuition is that, when considering two tasks i and j with $i < j$, j is less constrained by its deadline than i . Thus, its marginal gain may always be lower or equal than the one of i (except if constrained by its maximum processing time). Otherwise, its processing time could have been increased, leading to a better solution.

For the exponential accuracy function, it gives:

$$-\Delta a_i \theta_i s e^{-\theta_i s t_i} \geq -\Delta a_j \theta_j s e^{-\theta_j s t_j}. \quad (4.10)$$

and

$$t_i \leq \frac{\theta_j}{\theta_i} t_j - \frac{1}{\theta_i s} \ln \left(\frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right). \quad (4.11)$$

Note that for two tasks i and j with same efficiency $\theta_i = \theta_j$ and same accuracy range $\Delta a_i = \Delta a_j$, the equation simplifies to:

$$t_i \leq t_j. \quad (4.12)$$

4.4.2.2 Algorithm proposal

Based on the KKT analysis, we propose Algorithm 1, which aims to allocate the processing time t_j for each task $j \in J$. The algorithm takes as input the list of task deadlines and maximum processing times, and returns the processing time for each task. It first sorts the tasks by order of non-decreasing deadlines. It incrementally sets the processing time of task j to $\min(d_j - T_{\text{current}}, t_j^{\max})$, where T_{current} corresponds to the total time already scheduled, i.e., $T_{\text{current}} = \sum_{i=1}^{j-1} t_i$. If $t_j \leq t_j^{\max}$, it then tests if the marginal gain of task j is lower than the ones of tasks i for $i \leq j$, which is an optimality condition (Equation 4.9). If not, it updates their processing times to achieve equal marginal gains.

To compute the optimum processing times, we use the fact that two *3-unconstrained* tasks i and j have equal marginal gains (Equation 4.5). In the case of the exponential function, we have Equation (4.6):

$$t_i = \frac{\theta_j}{\theta_i} t_j - \frac{1}{\theta_i s} \ln \left(\frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right).$$

Algorithm 1 Exact algorithm for scheduling in one machine**Input:** List of task deadlines $[d_1, \dots, d_n]$ and $[t_1^{\max}, \dots, t_n^{\max}]$.**Output:** List of task processing times $[t_1, \dots, t_n]$

```

1: Sort the tasks by non decreasing deadlines.
2:  $t_1 \leftarrow \min(d_1, t_1^{\max})$ 
3:  $T_{\text{current}} = t_1$   $\triangleright T_{\text{current}} \stackrel{\text{def}}{=} \sum_{i=1}^j t_i$  for current task  $j$ 
4: for  $j : 2 \leq j \leq n$  do
5:    $t_j \leftarrow \min(d_j - T_{\text{current}}, t_j^{\max})$ ;  $T_{\text{current}} \leftarrow T_{\text{current}} + t_j$ 
6:   if  $t_j = t_j^{\max}$  then
7:     Continue to next iteration of the for loop
8:    $a' = a'_j(t_j)$   $\triangleright$  current gain
9:    $K \leftarrow \{j\}$   $\triangleright$  Set of 3-unconstrained tasks with  $a'(i) \leq a'$ .  $K$  sorted by non increasing
    $a'(t_k(t_k))$   $a'(t_k)$ 
10:   $Z \leftarrow \emptyset$   $\triangleright$  Set of tasks with  $t_i = 0$ 
11:   $T \leftarrow t_j$ ;  $i \leftarrow j - 1$ 
12:  while  $(a'_i(t_i) < a' \text{ or } t_i = t_i^{\max})$  and  $i \geq 0$  do
13:    if  $(t_i = 0)$  or  $(t_i = t_i^{\max} \text{ and } a'_i(t_i) \geq a')$  then
14:       $i \leftarrow i - 1$ ;
15:      Continue to next while iteration
16:     $K \leftarrow K \cup \{i\}$ ;  $T \leftarrow T + t_i$ 
17:     $t_i \leftarrow \max(\tau(T, K, i), 0)$ 
18:    if  $t_i = 0$  then
19:       $K \leftarrow K \setminus \{i\}$ ;  $Z \leftarrow Z \cup \{i\}$   $\triangleright$  Add in  $Z$  sorted by non increasing  $a'(t_i)$ 
20:    move  $\leftarrow$  True
21:    while move do
22:      move  $\leftarrow$  False
23:      for  $k \in Z$  do  $\triangleright Z$  sorted
24:        if  $\tau(T, K \cup \{k\}, k) \geq 0$  then
25:           $Z \leftarrow Z \setminus \{k\}$ ;  $K \leftarrow K \cup \{k\}$ 
26:          move  $\leftarrow$  True
27:        else Break
28:      for  $k \in K$  do  $\triangleright K$  sorted
29:         $t_k \leftarrow \min(\tau(T, K, k), t_k^{\max})$ 
30:        if  $t_k = t_k^{\max}$  then
31:           $K \leftarrow K \setminus \{k\}$ ;  $T \leftarrow T - t_k^{\max}$ 
32:          move  $\leftarrow$  True
33:        else Break
34:      for  $k \in K$  do
35:         $t_k \leftarrow \tau(T, K, k)$ 
36:         $a' \leftarrow a'_k(t_k)$ 
37:       $i \leftarrow i - 1$ 
38: return  $[t_1, \dots, t_n]$ 

```

Consider now we want to compute the optimum processing times of a set of K 3-unconstrained tasks executed in an interval of time T (i.e., $\sum_{i \in K} t_i = T$). We aim to compute the processing time of task $j \in K$, t_j . For this, we first compute the value of T as described below:

$$T = \sum_{i \in K} t_i = t_j + \sum_{i \in K \setminus \{j\}} t_i = t_j + \sum_{i \in K \setminus \{j\}} \left[\frac{\theta_j}{\theta_i} t_j - \frac{1}{\theta_i s} \ln \left(\frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right) \right]$$

Then, factoring out the variable t_j , we obtain the following equation:

$$T = t_j \left(1 + \sum_{i \in K \setminus \{j\}} \frac{\theta_j}{\theta_i} \right) - \sum_{i \in K \setminus \{j\}} \frac{1}{\theta_i s} \ln \left(\frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right)$$

Then, the processing time of task $j \in K$ is given by

$$\tau(T, K, j \in K) = \frac{1}{1 + \sum_{i \in K \setminus \{j\}} \frac{\theta_j}{\theta_i}} \left(T + \frac{1}{s} \sum_{i \in K \setminus \{j\}} \frac{1}{\theta_i} \ln \frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right). \quad (4.13)$$

The subtlety in the algorithm is that we apply the formula in a context where constrained tasks (by their minimum, maximum processing time, or deadline) may become unconstrained in the process and vice versa. In this case, τ could be negative for a task $i \in K$ or be greater than $t_i^{\max}$. Then, we set $t_i = 0$ or $t_i = t_i^{\max}$, i is deleted from K , and the processing times are recomputed. Note that, for the exponential case, it takes a time linear in the number of tasks in K to compute a value of τ using Equation 4.13 for the first computation. However, computing the value of another element of K can then be done in constant time by updating both sums of the formula. Similarly, when K is updated by addition or removal of an element, updating the value of τ for an element in K can also be done in constant time. Hence, the computations of τ in the algorithm are done in constant amortized times. For a general accuracy function, we note $c(n)$ the amortized complexity of computing a value of τ .

Theorem 4.4.1. *Algorithm 1 returns an optimal schedule in time $O(n^3(\log n + c(n)))$, where $c(n)$ is the amortized complexity of computing a value of $\tau(T, K, j \in K)$ for a set of 3-unconstrained tasks K . In the case of the exponential accuracy function, $c(n) = O(1)$, and the complexity of the algorithm is $O(n^3 \log n)$.*

Proof.

The proof of the current theorem analyzes the optimality and the complexity of Algorithm 1.

Optimality. The KKT conditions state that, when scheduling n tasks for an optimal solution, for all $j \in J$, either (i) the task is constrained by its maximum processing time, i.e., $t_j = t_j^{\max}$ (we denote the set of tasks with maximum processing time as L) (ii) or by its minimum processing time, i.e., $t_j = 0$ and its marginal gain should be lower or equal than the ones of all tasks $i \leq j$, that is $a'_j(t_j) \leq a'_i(t_i)$, or (iii) or $0 < t_j < t_j^{\max}$ and its marginal gain is equal to the one of every 3-unconstrained task $i < j$ executed before ($a'_j(t_j) = a'_i(t_i)$) and lower or equal to the one of every any-constrained task $i < j$. Additionally, for the last task, it is either constrained by its maximum processing time or by its deadline.

Let us prove the correctness of the algorithm by showing by induction that at the end of step j of the `for` loop of Line 4, the KKT conditions are satisfied for $i \in \{1, \dots, j\}$. First, after Line 2,

the KKT conditions are satisfied for the first task, as $t_1 = d_1$ or $t_1 = t_1^{\max}$. Now, suppose that the induction hypothesis is true for $i \in \{1, \dots, j-1\}$ at the end of the step $j-1$ of the `for` loop. At the end of step j of the `for` loop of Line 4, the KKT conditions are satisfied for $i \in \{1, \dots, j\}$. Indeed, first, t_j is initialized in Line 5 with either $t_j = t_j^{\max}$ or $\sum_{i \leq j} t_i = d_j$. In the first case, the KKT conditions are directly satisfied for j and for $i < j$ by the induction hypothesis. In the second case, we have to check if there exist tasks with lower marginal gains processed before j . This is done by the test of the `while` loops of Line 12. Note that, as soon as a task i , not constrained by its maximum processing time, has a larger marginal gain than j , all tasks $k < i$ also have larger marginal gain by the induction hypothesis. If no such task exists, the body of the `while` loop is not entered and step j is finished with the KKT conditions satisfied for j due to Line 5 and for $i < j$ by the induction hypothesis.

If tasks with lower marginal gains than j exist, the processing time t_j will be increased and the one of the other tasks decreased (or kept constant) in order to satisfy the KKT conditions. That is having equal marginal gains with every 3-unconstrained tasks and higher marginal gains for the other ones, or t_j reaches $t_j^{\max}$. To this end, we use the function $\tau(T, K, i \in K)$ which, given a set of 3-unconstrained tasks K and an interval of time $T = \sum_{i \in K} t_i$, computes the optimal processing time t_i of a task $i \in K$ (in the `for` loop of Line 34). We thus track the set, K , of 3-unconstrained tasks, which had a marginal gain larger than the one of j at the beginning of step j . The subtlety is that constrained tasks (by their minimum or maximum processing time or deadline) may become unconstrained in the process (tested e.g. on Line 24) and vice versa (with their processing times becoming null (tested on Line 18) or equal to $t^{\max}$ (tested on Line 30)). We thus also store the set of tasks constrained by their minimum time, Z . We do not have to keep L or the set of tasks constrained by their deadlines explicitly. Indeed, the algorithm works as follow. We track the current marginal gain a' of the tasks in K (of value $a'(t_j)$ at the beginning, see Line 8). Then, during the step i of the `while` loop of Line 12, the processing time of i decreases or stays constant (and its marginal gain increases or stays constant), while the ones of $i < k \leq j$ increase (but never over their original processing time, as a' decreases, except for j) or stay constant (and their marginal gain decreases or stays constant). This means that a task i for which $\sum_{l < i} t_l = d_i$ is not really constrained by its deadline, as its processing time at the end of Step i is lower or equal to the one at the beginning. We thus also put such tasks in K . This also implies that a task i may only enter leave L (and enter K) or enter Z at step i of the Line 12 `while` loop, because the tasks are considered in non-increasing order of gain. It may then leave Z or go back into M at a later step. Note that, if it happens, it will not go back in Z or exit L for the whole step j of the Line 4 `for` loop. Finally, also note that task j may become constrained by its maximum processing time (tested on Line 30). In this case, the end of the `while` loop ensures that the KKT conditions are satisfied for all $i < j$.

Algorithm time complexity. First, sorting the tasks by non decreasing deadlines (Line 1) is done in $O(n \log n)$. Then, the algorithm consists of four nested loops. The `for` loops (of Lines 4, Lines 23 or 28) and the `while` loop (Line 12) are of maximum length n . Let us study the complexity of the `while` loop of Line 21. As discussed above, a task i may only enter Z or leave L at step i of the Line 12 `while` loop. It may then leave Z or return to L at a later step. If it happens, it will not go back in Z or exit L for the whole step j of the `for` loop of Line 4. Thus, the number of moves of the `while` loop of Line 21 is less than 2 number of steps of Line 12 `while` loop. Therefore, the amortized complexity of the Line 21 `while` loop is constant.

In the body of the `for` loops, two kinds of operations are done. First, the function τ is computed. Let us first discuss the case of the exponential accuracy function. It takes a time linear in the number of tasks in K to compute a value of τ using Equation 4.13 for the first computation. However, computing the value of another element of K can then be done in constant time by updating both sums of the formula. Similarly, when K is updated by the addition or removal of an element, updating the value of τ for an element in K can also be done in constant time. Hence, the computations of τ in the algorithm are done in constant amortized time. For a general accuracy function, we note $c(n)$ the amortized complexity of computing a value of τ . Second, the sorted sets Z and K are updated. If using an AVL tree to implement them, inserting and removing an element are done in time $O(\log n)$. Iterate on all elements is done using an in-order traversal with constant amortized time for each new element leading to a $O(n)$ complexity. It leads to a complexity of $O(n^3(\log n + c(n)))$ for a general accuracy function and of $O(n^3 \log n)$ for the exponential accuracy function.

□

4.5 Problem on several machines

We now consider the problem in which the tasks can be processed on m different machines. We consider a scenario in which machines have *different speeds* (e.g., servers, laptops, cellphones, IoT devices, etc.). The accuracy of a task j on a machine r , with speed s_r , is given by $a_j(s_r \cdot t_j)$. Then, for the exponential accuracy function, we have:

$$a_j(s_r \cdot t_j) = (a^{\max} - a^{\min}) (1 - e^{-\theta_j \cdot s_r \cdot t_j}).$$

4.5.1 Formulation as a Mixed Integer Program

The DSCT problem on several machines can be formulated as the mixed integer program given below.

Variables. We introduce the integer variables x for representing the task allocation $x_{jr} = 1$ if task j is executed on machine r , for all $j \in J$ and $r \in M$. Otherwise, $x_{jr} = 0$. The variable indicating the tasks processing time is now chosen for each machine by the variables $t_{jr} \in \mathbb{R}^+$.

Objective function. The processing time, t_j , of task j is given by $\sum_{r=1}^m t_{jr}$. The accuracy of j thus is $a_j(\sum_{r=1}^m s_r \cdot t_{jr})$. The objective function is:

$$\min \sum_{j=1}^n 1 - a_j \left(\sum_{r=1}^m s_r \cdot t_{jr} \right).$$

As an example, for the exponential accuracy function, the objective is:

$$\min \sum_{j=1}^n (a_j^{\max} - a_j^{\min}) e^{-\theta_j \sum_{r=1}^m s_r \cdot t_{jr}}$$

We thus have:

$$\min \quad \sum_{j=1}^n \left[1 - a_j \left(\sum_{r=1}^m s_r \cdot t_{jr} \right) \right] \quad (4.14)$$

$$\text{s.t.} \quad \sum_{i=1}^j t_{ir} \leq d_j, \quad j \in J, r \in M \quad (4.14a)$$

$$\sum_{r \in M} t_{jr} s_r \leq F_j^{\max}, \quad j \in J \quad (4.14b)$$

$$t_{jr} \leq x_{jr} \cdot d_j, \quad j \in J, r \in M \quad (4.14c)$$

$$\sum_{r \in M} x_{jr} \leq 1, \quad j \in J \quad (4.14d)$$

$$t_{jr} \in \mathbb{R}^+, \quad j \in J, r \in M \quad (4.14e)$$

$$x_{jr} \in \{0, 1\}, \quad j \in J, r \in M \quad (4.14f)$$

In the above Mixed Integer Program (MIP), we set in Eq. (4.14) the objective function as the minimization of accuracy error. Eq. (4.14a) ensures that each task j is completed before its corresponding deadline. This constraint takes into account the accumulated processing times of all tasks scheduled before j on machine r , which determine its start time, together with its own execution duration. In Eq. (4.14b), we state the the number of floating operations performed by each task should be at most $F_j^{\max}$, the maximal number of floating operations. Note that the number of floating operations is given by the processing time of a task and the machine speed where it is processed. Eq. (4.14c) allows the task to be processed if it is allocated in a certain machine and Eq. (4.14d) states that each task can be allocated in at most one machine. Eqs (4.14e) and (4.14f) define the variables domain.

4.5.2 NP-Hardness complexity

On the contrary to the problem in a single-machine scenario which is polynomial, DSCT on several machines is formulated with binary variables. In fact, we have the following result.

Definition 4.5.1. The decision problem of DSCT with m machines is to decide if there exists a schedule $\{t_{jr} | j \in J, r \in M\}$ such that the global accuracy is greater than $x \in \mathbb{R}$, that is $\sum_{j \in J} a_j (\sum_{r \in M} t_{jr}) \geq x$.

Theorem 4.5.1. *The decision problem of DSCT (for any strictly increasing accuracy function) is NP-complete when the number of machines is $m \geq 2$.*

Proof.

The proof is done via a reduction from the PARTITION SET PROBLEM.

Recall the definition of this classical problem. Given a set of integers $B = \{b_1, b_2, \dots, b_n\}$, with $S = \sum_{b_i \in B} b_i$ the elements of B must be partitioned into 2 sets B_1 and B_2 such that the sums of the elements in each sets are equal, i.e., $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i = \frac{S}{2}$ and $B_1 \cap B_2 = \emptyset$.

The reduction is done by transforming an instance of the PARTITION SET PROBLEM with n integers into an instance of DSCT with 2 machines with n tasks. The tasks are defined with the uniform efficiency $\theta_i = \theta$ and with $t_i^{\max} = b_i$, for all $i \in \{1, \dots, n\}$. They all have the same

deadline $d_i = \frac{\sum_{i=1}^n b_i}{2} = \frac{S}{2}$. Moreover, we consider the two machines present the same speed $s_0 = s_1 = s$.

Deciding if there exists a partition of A into two subsets of size $\frac{S}{2}$ is equivalent to deciding if there exists a schedule of J with value $\sum_{j \in J} a_j(s \cdot t_j^{\max})$. Indeed, if there exist a partition of B , then the solution $t_{j1} = b_j = t_j^{\max}$ and $t_{j2} = 0$ for $a_j \in B_1$ and $t_{j2} = b_j = t_j^{\max}$ and $t_{j1} = 0$ for $b_j \in B_2$ is feasible. A task j with efficiency θ_j executed for a time $t_j^{\max}$ has an accuracy of $a_j(s \cdot t_j^{\max})$. The global accuracy thus is $\sum_{j \in J} a_j(s \cdot t_j^{\max})$.

Consider now we have a schedule with global accuracy $\sum_{j \in J} a_j(s \cdot t_j^{\max})$. As the accuracy functions are strictly increasing, we must have $t_j = t_j^{\max}$, i.e. $t_j = b_j$, for all $j \in J$. We note J_1 and J_2 the subsets of tasks executed in machine 1 and machine 2, respectively. Because of the deadline constraints, we have $\sum_{j \in J_1} t_{j1} = \sum_{j \in J_1} b_j \leq \frac{S}{2}$ and $\sum_{j \in J_1} t_{j1} = \sum_{j \in J_2} b_j \leq \frac{S}{2}$. Yet, $\sum_{b_i \in B} b_i = S$, therefore we have equality, and, thus, $B_1 = \{b_j | j \in J_1\}$ and $B_2 = \{b_j | j \in J_2\}$ is a partition.

Thus, the decision version of DSCT is NP-complete as PARTITION SET PROBLEM is NP-Hard. Additionally, it is linear to test a solution of the problem, thus DSCT is NP-complete.

□

4.5.3 Approximation Algorithm

We propose here an approximation algorithm, DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - APPROXIMATION ALGORITHM (or DSCT-APPROX in short), presented in Algorithm 3, to solve DSCT with a proved guarantee, see Theorem 4.5.4. DSCT-APPROX consists in three main steps:

- Step 1 (Fractional relaxation). We first solve the problem with the fractional relaxation. In this problem, a single task is allowed to be scheduled in more than one machine. It gives a schedule with maximum accuracy value $OPT_f \geq OPT$ in polynomial time, where OPT is the maximum accuracy of DSCT. In fact, an optimal fractional solution can be found by using the algorithm for one machine (Algorithm 1) with deadlines $d_j^1 = S d_j$ and by taking $t_{jr} = \frac{t_j^1}{s_r}$, (see Lemmas 4.5.2 and 4.5.3), where d_j^1 and t_j^1 denote, respectively, the deadline and the processing time in a single machine equivalent problem.
- Step 2 (Rounding). We consider the tasks by non decreasing deadlines and assign them incrementally to the machine with the least amount of work (allocated time).

At the end of this step, each deadline may only be violated by a single task on every machine.

- Step 3 (Incremental shift.). For each machine $r \in M$, we consider the tasks in the order of their execution on r . If a task j violates the deadline with a violation $v_j = t_{jr}^{(i)} + t_{jr} - d_j$, where $t_{jr}^{(i)}$ denotes the starting time of task j in machine r , we cut the execution of the task done after the deadline, i.e., $t_{jr} \leftarrow t_{jr} - v_j$, and we shift the processing times of the all the tasks executed after j on r , i.e., $t_{ir}^{(i)} \leftarrow t_{ir}^{(i)} - v_j$ for all $i \geq j$.

We first show in Lemma 4.5.3, that Algorithm 2 solves the fractional relaxation of the problem in polynomial time. To this end, we use the preliminary result of Lemma 4.5.2. We then prove that Algorithm 3 is an approximation algorithm and give its approximation guarantee in Theorem 4.5.4.

Algorithm 2 Exact algorithm for solving the fractional relaxation

Input: List of task deadlines $[d_1, \dots, d_n]$, of maximum processing times $[t_1^{\max}, \dots, t_n^{\max}]$, and of machine speeds $[s_1, \dots, s_m]$

Output: List of task processing times $[t_{jr} : \forall j \in J, r \in M]$

```

1:  $S \leftarrow 0$  ▷ The sum of all speeds
2: for  $r \in M$  do
3:    $S \leftarrow S + s_r$ 
4: for  $j \in J$  do
5:    $d_j^1 \leftarrow S d_j$ 
6:  $[t_1^1, \dots, t_n^1] \leftarrow \text{Algorithm 1}([d_1^1, \dots, d_n^1], [t_1^{\max}, \dots, t_n^{\max}])$ 
7: for  $j \in J$  do
8:   for  $r \in M$  do
9:      $t_{jr} \leftarrow \frac{t_j^1}{S}$ 
10: return  $[t_{jr} : \forall j \in J, r \in M]$ 

```

Algorithm 3 DSCT-APPROX: Approximation algorithm for scheduling in several machines

Input: List of task deadlines $[d_1, \dots, d_n]$, $[t_1^{\max}, \dots, t_n^{\max}]$, and of machine speeds $[s_1, \dots, s_m]$

Output: List of task processing times $[t_{jr} : \forall j \in J, r \in M]$

```

1: Sort the tasks by non decreasing deadlines.
2:  $\mathbf{t}^f = \text{Algorithm 2}([d_1, \dots, d_n], [t_1^{\max}, \dots, t_n^{\max}], [s_1, \dots, s_m])$  ▷ Compute the optimal fractional solution  $\mathbf{t}^f$ .
3: for  $r \in M$  do
4:    $S_r \leftarrow []$  ▷ List of tasks scheduled on machine  $r$ 
5:    $work_r \leftarrow 0$  ▷  $work_r$  amount of time on  $r$ 
6: for  $j \in J$  do ▷ Schedule each task on the machine with the least amount of work,  $r_{\min}$ 
7:    $r_{\min} \leftarrow \arg \min_{r \in M} work_r$ 
8:    $t_{jr_{\min}} \leftarrow \frac{\sum_{r=1}^m t_{jr}^f \cdot s_r}{s_{r_{\min}}}$ 
9:    $work_{r_{\min}} \leftarrow work_{r_{\min}} + t_{jr_{\min}}$ 
10:   $S_{r_{\min}} \leftarrow S_{r_{\min}} + [j]$ 
11: for  $r \in M$  do
12:   for  $j \in S_r$  do
13:     $t_{jr}^{(i)} = \sum_{i < j} t_{jr}$  ▷ Starting time of task  $j$  on machine  $r$ 
14: for  $r \in M$  do ▷ Cut tasks violating their deadlines and shift the following ones.
15:   for  $j \in S_r$  do
16:    if  $t_{jr}^{(i)} + t_{jr} \geq d_j$  then
17:       $v_j \leftarrow t_{jr}^{(i)} + t_{jr} - d_j$ 
18:       $t_{jr} \leftarrow t_{jr} - v_j$  ▷ Cut task
19:      for  $i > j \in S_r$  do ▷ Shift the following tasks
20:         $t_{ir}^{(i)} \leftarrow t_{ir}^{(i)} - v_j$ 
21: return  $[t_{jr} : \forall j \in J, r \in M]$ 

```

Lemma 4.5.2. *The fractional relaxation of the DSCT problem with m machines has an optimal fractional solution which allocates the same processing time on each machine, i.e., $t_{jr_1}^{f*} = t_{jr_2}^{f*}$ for all $r_1, r_2 \in M$.*

Proof.

Consider an optimal fractional solution $[t_{jr}^{f*}, \forall j \in J, r \in M]$ with global accuracy OPT_f . Let us define the symmetric solution

$$t_{jr}^s = \frac{\sum_{r \in M} s_r t_{jr}^{f*}}{S} = t_j^s, \quad \forall r \in M, \forall j \in J.$$

We first prove that it is feasible. We have

$$\sum_{i \leq j} t_i^s = \sum_{i \leq j} \frac{\sum_{r \in M} s_r t_{ir}^{f*}}{S} = \frac{1}{S} \sum_{r \in M} \sum_{i \leq j} s_r t_{ir}^{f*} = \frac{1}{S} \sum_{r \in M} s_r \sum_{i \leq j} t_{ir}^{f*}.$$

As the optimal solution is feasible, we have that $\sum_{i \leq j} t_{ir}^{f*} \leq d_j$ for all $r \in M$. We thus get

$$\sum_{i \leq j} t_i^s \leq \frac{1}{S} \sum_{r \in M} s_r d_j = d_j, \forall j \in J.$$

The symmetric solution is thus a feasible solution. Now, task j has an accuracy of

$$\begin{aligned} a_j\left(\sum_{r \in M} s_r t_j^s\right) &= a_j(S t_j^s) = a_j\left(S \frac{\sum_{r \in M} s_r t_{jr}^{f*}}{S}\right) \\ &= a_j\left(\sum_{r \in M} s_r t_{jr}^{f*}\right), \end{aligned}$$

i.e., the same accuracy as the optimal solution. Thus, its global accuracy, $\sum_{j \in J} a_j(\sum_{r \in M} s_r t_j^s)$, is OPT_f , finishing the proof.

□

Lemma 4.5.3. *Given an instance of DSCT with n tasks and m machines, Algorithm 2 provides an optimal solution of the fractional relaxation for the instance in time $O(n^3(\log n + c(n)) + nm)$. Recall that $c(n)$ is the amortized complexity of computing a value of τ , see Theorem 4.4.1.*

Proof.

Consider an instance of DSCT with m machines of speeds $s_1, \dots, s_m$ and n tasks with deadlines $(d_1, \dots, d_n)$. We set $S \stackrel{\text{def}}{=} \sum_{r \in M} s_r$. We define the problem on 1 machine of speed 1, as the problem with n tasks with deadlines $(S \cdot d_1, \dots, S \cdot d_n)$. We now prove that the optimal value for both problems are equal, that is $OPT_1 = OPT_m$.

First, we prove $OPT_m \geq OPT_1$. We consider an optimal solution of the problem with 1 machine $(t_1^1, \dots, t_n^1)$. We can build a solution for the problem with m machines by setting $t_{jr}^m = \frac{t_j^1}{S}$ for all $j \in J$ and all $r \in M$. The solution is feasible as $\sum_{i < j} t_i^1 \leq S \cdot d_j$ directly implies that

$\sum_{i < j} \frac{t_i^1}{S} = \sum_{i < j} t_{ir}^m \leq d_j$. Then, the deadline constraints are not violated when building the solution for m machines. Moreover, the accuracy value of the obtained solution is

$$\sum_{j \in J} a_j \left(\sum_{r \in M} s_r t_{jr}^m \right) = \sum_{j \in J} a_j \left(\sum_{r \in M} s_r \frac{t_j^1}{S} \right) = \sum_{j \in J} a_j(t_j^1) = OPT_1.$$

Thus, $OPT_m \geq OPT_1$.

Then, we prove that $OPT_1 \geq OPT_m$. Conversely, consider the optimal solutions of the problem with m machines. According to Lemma 4.5.2, there exists one satisfying $t_{j1}^m = \dots = t_{jm}^m, \forall j \in J$. We may now build a solution for the problem with 1 machine by setting $t_j^1 = S \cdot t_{j1}^m, \forall j \in J$. This solution is feasible as $\sum_{i \leq j} t_{ir}^m \leq d_j$, from deadlines constraint, directly implies that $\sum_{i \leq j} t_i^1 = \sum_{i \leq j} S \cdot t_{ir}^m \leq S \cdot d_j$. The accuracy value of the solution for 1 machine $\sum_{j \in J} a_j(t_j^1) = \sum_{j \in J} a_j(\sum_{r \in M} s_r t_{jr}^m) = OPT_m$ as seen above. Then, $OPT_1 \geq OPT_m$. Thus, $OPT_1 = OPT_m$.

□

Theorem 4.5.4. DSCT-APPROX (Algorithm 3) is an approximation algorithm with an absolute performance guarantee G :

$$OPT - G \leq SOL \leq OPT,$$

where OPT is the global accuracy of an optimal solution, and SOL is the solution returned by DSCT-APPROX and with

$$G = m \int_0^\infty \max_{j,r} \frac{da_j(s_r t)}{dt} dt.$$

For the exponential accuracy function:

$$G = mA \left(1 + \frac{1}{e} \ln \left(\frac{\theta_{\max}}{\theta_{\min}} \right) \right),$$

where $A \stackrel{\text{def}}{=} \max_{j \in J} (a_j^{\max} - a_j^{\min})$.

SOL is computed in time $O(n^3(\log n + c(n)) + n^2 m)$.

Proof.

The current theorem analyzes the approximation guarantees and the time complexity for DSCT-APPROX (Algorithm 3).

Performance guarantees DSCT-APPROX starts from an optimal solution of the fractional relaxation of the problem of value OPT_f . We know that $OPT \leq OPT_f$.

The algorithm then assigns tasks to the machine with the least amount of work. Note that on any machine $r \in M$, a deadline d may be only violated by a single task. Indeed, consider deadline d is violated on machine r by task i . When Algo 3 places a task $j > i$ with the same deadline d at step j in the for loop on line 6, there exists a machine r' such that $work_{r'} < d$. Indeed, $\sum_{r \in M} work_r + t_j^f = \sum_{i=0}^{j-1} t_{ir}^f \leq md$, because the schedule is feasible for the fractional relaxation. Thus, DSCT-APPROX does not assign j to machine r .

Finally, the algorithm incrementally cuts tasks violating their deadlines (For a cut task j , we denote by v_j the cut time due to the deadline violation) and shifts by v_j the starting times of the

tasks executed afterwards on the same machine. We compute the difference between OPT_f and the total accuracy of the solution returned by the algorithm SOL .

We define the lost accuracy on a machine $r \in M$ as the sum of lost accuracy for each cut task allocated on machine r , which presents speed s_r . We denote by $t_j^{\text{proc}} = d_j - t_{j,r}^{(i)}$ the processing time after part of the task is cut, i.e., the amount of time a task is processed before its deadline. The lost accuracy may be computed using the following equation:

$$LA_j \stackrel{\text{def}}{=} a_j(s_r(t_j^{\text{proc}} + v_j)) - a_j(s_r t_j^{\text{proc}}).$$

First, note that the function is decreasing in t_j^{proc} due to concavity. Moreover, a cut task j was first shifted by $v_j^c \stackrel{\text{def}}{=} \sum_{i < j} v_i$ on the machine where j is allocated. This part is processed. Thus, the worst case is when $t_j^{\text{proc}} = v_j^c$, and

$$LA_j \leq a_j(s_r(v_j^c + v_j)) - a_j(s_r v_j^c).$$

We now bound LA_j by the accuracy of the worst task that could be cut at time v_c . Note that the worst cut task depends on the time at which it is cut; see the discussion for the exponential accuracy function below for an illustration. Note also that the maximal lost accuracy depends on the task i which maximizes the accuracy difference, and consequently its efficiency θ_i ; and also depends on the machine speed s_r .

$$LA_j \leq \max_{i,r} (a_i(s_r(v_j^c + v_j)) - a_i(s_r v_j^c)).$$

Now, we know that two functions $f(x)$ and $g(x)$ such that $f(x_1) = g(x_1)$ and $f'(x) \leq g'(x)$ for all $x \in [x_1, x_2]$ is such that $f(x) \leq g(x)$ for all $x \in [x_1, x_2]$. It gives

$$\begin{aligned} LA_j &\leq \max_{i,r} (a_i(s_r(v_j^c + v_j)) - a_i(s_r v_j^c)) \\ &= \max_{i,r} \int_{v_j^c}^{v_j^c + v_j} \frac{d}{dt} a_i(s_r t) dt \leq \int_{v_j^c}^{v_j^c + v_j} \max_{i,r} \frac{d}{dt} a_i(s_r t) dt \end{aligned}$$

We define the total lost accuracy, TLA , as the sum the lost accuracy for each cut task on a specific machine. We denote by n_c the number of cut tasks on the machine r . Let us now bound the total lost accuracy TLA . We have

$$\begin{aligned} TLA &\stackrel{\text{def}}{=} \sum_{j=1}^{n_c} LA_j \leq \sum_{j=1}^{n_c} \int_{v_j^c}^{v_j^c + v_j} \max_{i,r} \frac{d}{dt} a_i(s_r t) dt \\ &= \int_0^v \max_{i,r} \frac{d}{dt} a_i(s_r t) dt \leq \int_0^\infty \max_{i,r} \frac{d}{dt} a_i(s_r t) dt \end{aligned}$$

where $v = \sum_{j \in J} v_j$. Considering there are m machines, the total lost accuracy of DSCT-APPROX is bounded by

$$G = m \int_0^\infty \max_{j,r} \frac{da_j(s_r t)}{dt} dt$$

Exponential accuracy function. Let us compute $\max_{i \in J, r} \frac{d}{dt} a_i(s_r t)$. We have

$$\begin{aligned} a'_i(s_r t) &= \frac{d}{dt} a_i(s_r t) = \frac{d}{dt} \left((a_i^{\max} - a_i^{\min})(1 - e^{-s_r \theta_i t}) + a_i^{\min} \right) \\ &= (a_i^{\max} - a_i^{\min}) s_r \theta_i e^{-s_r \theta_i t}. \end{aligned}$$

We now consider the function $a'(t, \theta) \stackrel{\text{def}}{=} A s_r \theta e^{-s_r \theta t}$, with $A = \max_{i \in J} a_i^{\max} - a_i^{\min}$. We have

$$\max_{i \in J} a'_i(t) \leq \max_{\theta} a'(t, \theta).$$

Let us find the value of θ for which $a'(t, \theta)$ is maximum, that is the one for which $\frac{d}{d\theta} a'(t, \theta) = 0$.

$$\frac{d}{d\theta} a'(t, \theta) = A s_r e^{-s_r \theta t} - A s_r^2 t \theta e^{-s_r \theta t} = A s_r (1 - s_r t \theta) e^{-s_r \theta t}.$$

The derivative is 0 for $\theta = \frac{1}{s_r t}$. This means that the worst task efficiency θ^* for a task cut after time t on a machine r is $\theta^* = \frac{1}{s_r t}$. For a set of tasks with maximum and minimum efficiencies, respectively $\theta_{\max}$ and $\theta_{\min}$, we have:

$$\theta^*(t) = \arg \max_{\theta} a'(t, \theta) = \begin{cases} \theta_{\max} & \text{when } t \leq \frac{1}{s_r \theta_{\max}} \\ \frac{1}{s_r t} & \text{when } \frac{1}{s_r \theta_{\max}} \leq t \leq \frac{1}{s_r \theta_{\min}} \\ \theta_{\min} & \text{when } t \geq \frac{1}{s_r \theta_{\min}} \end{cases}$$

Note that it means that for small values of t , the worst tasks to be cut have maximum efficiency. However, for large values of t , it is the reverse: the worst cut tasks have minimum efficiency.

Let us now bound the total lost accuracy. We have

$$\begin{aligned} TLA &\leq \int_0^\infty \max_{i \in J, r \in M} \frac{d}{dt} a_i(s_r t) dt \leq \int_0^\infty a'(t, \theta^*(t)) dt \\ &= \int_0^{\frac{1}{s_r \theta_{\max}}} a'(t, \theta_{\max}) dt + \int_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} a' \left(t, \frac{1}{s_r t} \right) dt \\ &\quad + \int_{\frac{1}{s_r \theta_{\min}}}^v a'(t, s_r \theta_{\min}) dt. \end{aligned}$$

Recall that $a'(t, \theta) = A s_r \theta e^{-s_r \theta t}$. We compute each integral:

$$\begin{aligned} \int_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} a' \left(t, \frac{1}{s_r t} \right) dt &= \int_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} A \frac{s_r}{s_r t} e^{-s_r \frac{1}{s_r t} t} dt \\ &= \int_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} \frac{A}{e} \frac{1}{t} dt = \frac{A}{e} [\ln t]_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} = \frac{A}{e} \ln \left(\frac{\theta_{\max}}{\theta_{\min}} \right). \\ \int_0^{\frac{1}{s_r \theta_{\max}}} a'(t, \theta_{\max}) dt &= \int_0^{\frac{1}{s_r \theta_{\max}}} A s_r \theta_{\max} e^{-s_r \theta_{\max} t} dt \\ &= A [-e^{-s_r \theta_{\max} t}]_0^{\frac{1}{s_r \theta_{\max}}} = A \left(1 - \frac{1}{e} \right). \\ \int_{\frac{1}{s_r \theta_{\min}}}^\infty a'(t, \theta_{\min}) dt &= \int_{\frac{1}{s_r \theta_{\min}}}^\infty A s_r \theta_{\min} e^{-s_r \theta_{\min} t} dt \\ &= A [-e^{-s_r \theta_{\min} t}]_{\frac{1}{s_r \theta_{\min}}}^\infty = \frac{A}{e}. \end{aligned}$$

Adding up, it gives

$$TLA \leq A + \frac{A}{e} \ln \left(\frac{\theta_{\max}}{\theta_{\min}} \right).$$

As there are m machines, the global lost accuracy of the algorithm is bounded by

$$G \stackrel{\text{def}}{=} mA \left(1 + \frac{1}{e} \ln \left(\frac{\theta_{\max}}{\theta_{\min}} \right) \right).$$

Time complexity First, in line 1 the algorithm sorts the tasks, which is done in $O(n \cdot \log(n))$. Then, the Algorithm 2 is executed, which is done in $O(n^3(\log n + c(n)) + nm)$, according to Lemma 4.5.3. Then, in line 3, the machines workloads are initialized, which is done in $O(m)$. In line 6, the rounding step is performed. This is done in $O(n)$. Finally, in line 14, the cut and sift step from the algorithm is executed. In this step, there is 3 nested for loop: one iterating over the machines and two iterating over the tasks.

Then, the algorithm final time complexity is $O(n^3(\log n + c(n)) + n^2m)$, where $c(n)$ is the amortized complexity, where, for exponential accuracy function, $c(n) = O(1)$.

□

4.6 Experimental evaluation

In this section, we evaluate the proposed approximation algorithm DSCT-APPROX described in Algorithm 3. We first show that the absolute guarantees given by Theorem 4.5.4 really correspond to a worst case scenario and that in practice DSCT-APPROX provides near optimal solutions. We then discuss the processing time of our algorithm and show that DSCT-APPROX provides near optimal solutions for medium to large instances in a limited time, even using a non optimized Python code. Last, we compare its performance with the one of state of the art solutions using no compression or a limited number of models. We show that using adjustable models allows to reach very significant gains in performance for a large range of scenarios.

Hardware Settings. During the experiments, we used a computer equipped with a CPU Intel Core i9-12900H and with a GPU NVIDIA RTX A2000.

Experimental scenarios and Problem instances. We generate synthetic sets of tasks based on the results of our experiments with OFA in Section 4.2. We set the maximum accuracy to $a^{\max} = 0.82$, i.e., the highest value found by ofa-resnet on ImageNet-1k, and the minimum task accuracy to $a^{\min} = 1/1000$, the accuracy of a random guess. We fix the value of the minimum task efficiency $\theta_{\min} = 0.1$.

We then build scenarios with different (i) task heterogeneity and (ii) deadline tolerance levels. Indeed, for very loose deadlines, there is enough available computation time on the machines, so that all algorithms succeed to schedule all tasks with their maximum processing times. For (i), we vary the *task heterogeneity ratio* of a set of tasks, denoted as $\mu \stackrel{\text{def}}{=} \frac{\theta_{\max}}{\theta_{\min}}$ which measures the similarity between task efficiencies. When μ is chosen, so $\theta_{\max}$, for each task $j \in J$, we select its efficiency θ_j uniformly at random between $\theta_{\min}$ and $\theta_{\max}$. Then, we compute its maximal processing time $t_j^{\max}$, in order to have $a_j(t_j^{\max}) = 0.82$. For (ii), we define the *deadline tolerance level*, denoted by ρ , as following: $\rho = \frac{m \cdot d_{\max}}{\sum_{j \in J} t_j^{\max}}$. The greater ρ is the more time is allocated for the tasks. When $\rho = 0$, no tasks can be executed and the average accuracy is $a^{\min}$. We make ρ vary from 0 to the value, denoted as $\rho_{\max}$, for which all tasks can be executed with their maximum

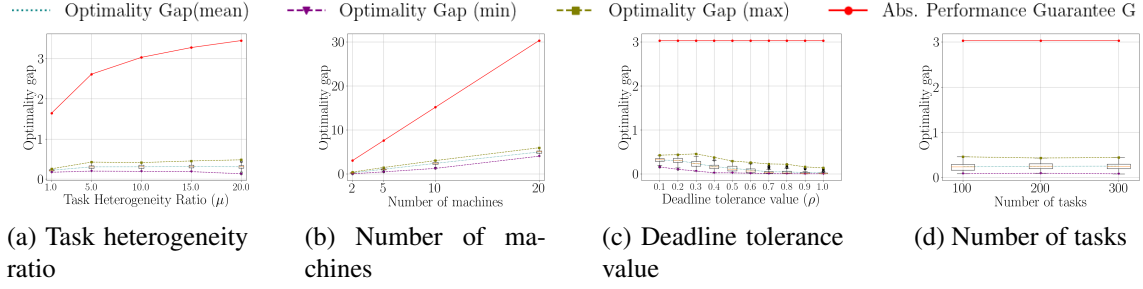


Figure 4.4 – Optimality gap (total accuracy difference between DSCT-UB and DSCT-APPROX) over 100 experiments when varying (a) the task heterogeneity ratio μ , (b) number of machines, (c) deadline tolerance value, and (d) number of tasks. Absolute guarantee G is given as a baseline.

time $t^{\max}$ for all the baseline algorithms. This value is greater than 1, as the fractional relaxation cannot reach such solutions for smaller values of ρ . When $\rho \geq \rho^{\max}$, the average task accuracy is the same for all algorithms, that is $a^{\max}$. When ρ is selected, we set the latest deadline to $d_n = \frac{\rho \cdot \sum_{j \in J} t_j^{\max}}{m}$. Then, we define the first deadline as d_n/n and we draw the difference between consecutive deadlines at random according the uniform distribution $U(0.5 \cdot d_n/n, 1.5 \cdot d_n/n)$. Finally, we make the number of tasks n and the number of machines m vary between 1 and 500 and between 0 and 50, respectively.

Baselines. We benchmark the performance of DSCT-APPROX with 3 different solutions: The second one is proposed in (D. Lee & Song, 2021).

- DSCT-UB, an upper bound solution provided by Algorithm 2 that solves the fractional relaxation of the problem DSCT problem on several machines.
- EDF-NoCOMPRESSION: Here, compression is not allowed. Then, each task must be executed with its maximal processing time ($t_j^{\max}$), which corresponds to the neural network in its maximal configuration. The scheduling strategy used here is the EDF (Earliest Deadline First) for ordering the tasks combined to the `worst-fit` strategy (Y.-W. Zhang, Chen, & Gu, 2023), for deciding the machine on which the task should be executed.
- EDF-3COMPRESSIONLEVELS: The algorithm considers a discrete set of possible configurations for compressing the neural network. In this approach, we use 3 levels of compression, corresponding to 3 accuracy levels compared to the maximal configuration: 27%, 55% and 82%. For this algorithm, we implemented a strategy based on (D. Lee & Song, 2021), which uses the EDF for ordering the tasks and implements a heuristic based on the accuracy for choosing the node and the compression level.

Study of the approximation guarantee. We first compare the performance of our algorithm to the absolute performance guarantee derived in Theorem 4.5.4, i.e. $G = mA(1 + \frac{1}{e} \ln(\mu))$. The performance guarantee G varies with the task heterogeneity ratio, μ , and the number of machines, m . To test the impact of parameters on the performance of our algorithm we conducted experiments for 4 different scenarios, in which we vary (i) the task heterogeneity ratio, (ii) the number of machines, (iii) the number of tasks n , and (iv) the deadline tolerance level ρ . In each scenario, when for the non varying parameters, the values of the other parameters are set to a default values": $\mu = 10$, $m = 2$ machines, $n = 100$ tasks, and $\rho = 0.3$.

In the first scenario, we thus make μ vary from 1 to 20. When $\mu = 1$, all tasks have the same efficiency, i.e. every task present the same slope at origin. When $\mu > 1$, some tasks are more

efficient than others, meaning that they need a smaller processing time to reach the same accuracy as less efficient ones. The guarantee is derived from a worst case analysis. We thus study here the algorithm performance over a large number of experiments (100 for each value of μ). Fig. 4.4 presents the distribution over the experiments of two optimality gaps: (i) the *absolute performance guarantee* G , which is actually a *worst-case optimality gap*; and (ii) the *optimality gap* provided by the difference between the fractional relaxation DSCT-UB and the approximation algorithm DSCT-APPROX solutions. For the latter, we plot the *mean*, *min* and *max* differences over all the tests for a fixed μ . Recall that $A = 0.82$ and $m = 2$ here, thus $G = 1.64$ when $\mu = 1$.

We observe from Fig. 4.4a that the optimality gaps increase with the task heterogeneity ratio as expected, but the gap with respect to the fractional relaxation solution is well below G . In our experiments, in average, the ratio between the *mean of the optimality gap* and G was 10.7%. This is expected as G comes from very specific scenarios, in which tasks with very short processing times and the worst accuracy are cut at each time when doing rounding. These scenarios are rare, as, even with 100 experiments, the maximal optimality gaps were far from G .

In a second scenario, we varied the number of machines from 2 to 20. The total accuracy difference is given in Fig. 4.4b. We observe (i) that the total accuracy difference between DSCT-APPROX and the optimal solution increases linearly with the number of machines. This linear dependency in m is the same as the one of the maximum optimality gap G (Theorem 4.5.4); (ii) that the optimality gap remains well below the performance guarantees, validating the approximation guarantees.

Then, in a third scenario, we evaluate the optimality gap when varying the deadline tolerance value from 0.1 to 1.0. Fig. 4.4c describes the results. We observe that (i) the optimality gap of DSCT-APPROX presents a value well below the performance guarantees, as observed in the previous experiments and (ii) the optimality gap decreases when ρ increases. This occurs due to the fact that, when ρ is high, then the tasks can be scheduled with higher processing times and consequently lower accuracy gains. This leads to lower accuracy differences between DSCT-APPROX and the optimal solution.

Finally, in a fourth scenario, we varied the number of tasks from 100 to 300. Fig. 4.4d describes the optimality gap over the number of tasks. Here, we observe that the total accuracy difference is not really impacted by the number of tasks. This occurs due to the cut-and-shift step in DSCT-APPROX. Indeed, even if several tasks may be cut on a machine, the intuition is that, in spirit, only the accuracy of a single task¹ is lost for each machine, whatever the number of tasks.

Algorithm processing times. We also tested the execution time of DSCT-APPROX against DSCT-Opt, which uses the cvx-MOSEK software (ApS, 2022), a widely used commercial solver which allows solving Mixed-Integer Programs (MIP). Here, we set the time limit of the execution in 60s and for each point, we considered an average over 10 experiments. The results are described in Fig. 4.5.

We considered two different scenarios. In a first scenario, we fixed the number of tasks to 100 and we varied the number of machines from 1 to 10. The results are given in Fig. 4.5b. We verify that, for our solution, the execution time does not present considerable variation. This occurs due to the fact that, in our algorithm, the heaviest complexity corresponds to solving the problem with fractional relaxation using Algorithm 2. On other hand, the processing times using DSCT-Opt

1. In fact, we may lose the accuracy of a virtual "worst" task, defined with the worst possible slope for each cut part. See the proof of Theorem 4.5.4 in Section 4.5 for a precise statement

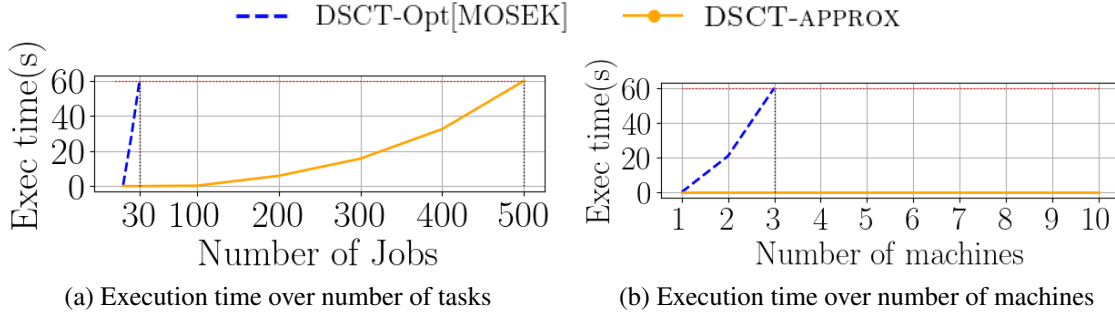


Figure 4.5 – Execution times of DSCT-APPROX vs DSCT-Opt for instances with increasing (a) numbers of tasks, with $m = 5$ and (b) number of machines, with $n = 50$.

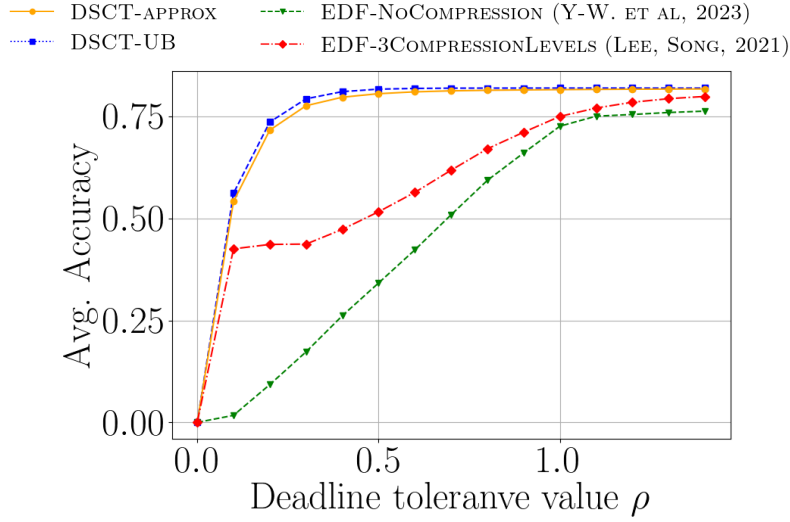


Figure 4.6 – Average task accuracy for DSCT-APPROX and the baselines as a function of the deadline tolerance level ρ , for $m = 10$.

sharply increases with the number of machines, which turns the approach impracticable for an optimization scenario with a high number of machines.

In a second scenario, we fixed the number of machines to 10 and we varied the number of tasks from 10 to 500. We observe in Figure 4.5a that DSCT-APPROX could handle executing until 500 tasks, without overcoming the time limit of 60s. On the other hand, DSCT-Opt executed until 30 tasks before reaching the time limit. Thus, we can observe that DSCT-APPROX has an execution time largely smaller than DSCT-Opt.

Comparison with benchmarks for different deadline tolerance levels. We now compare the global accuracy obtained by DSCT-APPROX with the ones reached by EDF-NoCOMPRESSION (Y.-W. Zhang et al., 2023) and EDF-3COMPRESSIONLEVELS (D. Lee & Song, 2021). To explore the different possible scenarios of usage, we vary the deadline tolerance level in a range between 0 and 1.5, with a step of 0.1. The number of machines is set to $m = 10$ and we consider $n = 100$ tasks with uniform efficiency $\theta_j = 0.1$. The average accuracy value reached by DSCT-APPROX, EDF-NoCOMPRESSION, EDF-3COMPRESSIONLEVELS, and the

fractional relaxation DSCT-UB are given in Fig. 4.6. Each point corresponds to an average over 100 experiments.

We confirm that (i) when ρ is close to zero, all the methods have an average accuracy close to zero, since the space to schedule tasks is very limited; (ii) when $\rho \geq 1$, there is space enough to execute all tasks with their maximum processing time and, thus, maximum accuracy. All algorithms thus reach an accuracy of $a^{\max} = 0.82$. Between these values, we first observe that DSCT-APPROX return quasi-optimal solutions as its values are very close to the ones of the fractional relaxation. Secondly, DSCT-APPROX outperforms EDF-NOCOMPRESSION (Y.-W. Zhang et al., 2023) and EDF-3COMPRESSIONLEVELS (D. Lee & Song, 2021) for each deadline tolerance value tested, as they do not use compression or only a limited number of models. In fact, the fractional relaxation DSCT-UB follows a function which is a composition of 1-exponential functions with limited supports. It thus has a close to exponentially decreasing marginal gain, when the two other algorithms have a (piecewise) linear behavior. The difference between DSCT-APPROX and EDF-NOCOMPRESSION (and EDF-3COMPRESSIONLEVELS) performances is significant and reaches more than .6 point (.3) of accuracy, e.g. representing a gain of 77.7% (43.6%) for $\rho = 0.3$. b

Fairness. A byproduct of allowing for task compression is that the allocation of processing times over tasks is *fairer* than without compression, in the sense that it allows them to reach similar accuracy values. When not using compression, a task is either executed, and reaches its maximum possible accuracy, or not, leading to a minimum one. We study this behavior in Fig. 4.7. For each set of experiments (corresponding to a specific value of deadline tolerance value, ρ), we report the distribution of reached accuracy values for all tasks. If we consider as an example, the case $\rho = 0.3$, we observe that more than 90% of the tasks reach an accuracy between 0.7 and 0.8 for DSCT-APPROX, and that only 2% of the tasks have an accuracy below 0.3, to be compared with 80% for EDF-3COMPRESSIONLEVELS and 70% for EDF-NOCOMPRESSION. Thus, our algorithm allows (i) for having a better average accuracy (as seen before) and (ii) for not sacrificing tasks, which may be particularly important for scenarios in which tasks are requested by different users.

4.7 Conclusion and Discussion

We studied the problem of scheduling with compressible tasks. We analyzed and modeled the tradeoff between accuracy and processing time of a family of "*slimmable*" neural networks. This technique can be used for several applications, such as real-time systems and task offloading. In this chapter, we analyzed the Once-For-All architecture used for image classification.

We proposed an approximation algorithm for multiple machines scenario, which is a NP-hard problem. The algorithm proposed obtained a near-optimal value, outperforming the baseline average accuracy by up to 77%. Also, we observed the algorithm presents smaller execution time compared to commercial MIP solvers.

We studied scheduling problems for specific jobs (classification inference tasks) and specific accuracy functions (concave, differentiable, and with exponentially decreasing marginal gains). However, we believe our work may have implications for a large range of problems.

First, the concavity of the accuracy function is a related to a very general and common phenomena which is the law of diminishing marginal utility (Cournot, 1838).

Second, exponentially decreasing marginal gains can be found for a large variety of optimization problems. While general theoretical results on such phenomenon appear still far from the

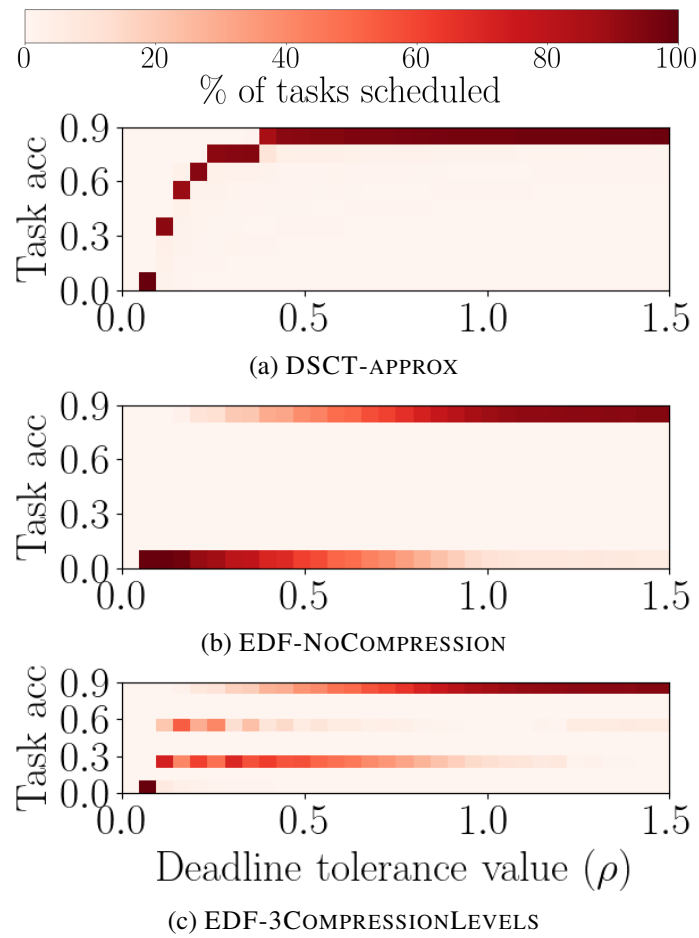


Figure 4.7 – Scheduling Fairness: Heat maps represent the distribution of task accuracy values for sets of experiments with different tolerance ρ . The colors represent the percentage of tasks within an accuracy range.

reach of current techniques for the theory of artificial neural networks, we argue that such dependency is supported by analogous phenomena in similar settings, like approximation error in Chebyshev's polynomials ([Sachdeva, 2013](#), Theorem 3.3), and classifier error of AdaBoost algorithm ([Schapire & Freund, 2012](#)).

Extending our results to a wide range of optimization model, such as energy efficiency and communication latency, is an interesting avenue for future work.

Scheduling Machine Learning Compressible Inference Tasks with Limited Energy Budget

Advancements in cloud computing have boosted Machine Learning as a Service (MLaaS), highlighting the challenge of scheduling tasks under latency and deadline constraints. Neural network compression offers the latency and energy consumption reduction in data centers, aligning with efforts to minimize cloud computing's carbon footprint, despite some accuracy loss.

This chapter investigates the DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - ENERGY AWARE (DSCT-EA) problem, which addresses the scheduling of compressible machine learning tasks on several machines, with different speeds and energy efficiencies, under an energy budget constraint. Solving DSCT-EA involves determining both the machine on which each task will be processed and its processing time, a problem that has been proven to be NP-Hard. We formulate DSCT-EA as a Mixed-Integer Programming (MIP) problem and also provide an approximation algorithm for solving it. The efficacy of our approach is demonstrated through extensive experimentation, revealing its superiority over traditional scheduling techniques. It allows to save up to 70% of the energy budget of image classification tasks, while only losing 2% of accuracy compared to when not using compression.

5.1	Introduction	67
5.2	Problem Formulation	68
5.2.1	Accuracy function	70
5.2.2	Fractional Relaxation	70
5.3	Optimal algorithm for DSCT-EA-FR	73
5.3.1	Algorithm for DSCT-EA-FR, one machine	73
5.3.2	Algorithm for DSCT-EA-FR, multiple machines	74
5.4	Approximation algorithm for DSCT-EA	78
5.5	Results	80
5.6	Conclusion	84

5.1 Introduction

In recent years, there has been a growing usage of Machine Learning (ML) models in cloud computing, contributing to the adoption of Machine Learning as a Service (MLaaS) (Ribeiro et al., 2015), studied in several applications such as image recognition and self-driving cars (Raj et al., 2023).

Cloud and network operators have faced challenges in developing efficient strategies for utilizing computational resources to support machine learning tasks. Among these challenges, scheduling is an important one. A scheduler must determine on which machine each task is executed and its processing order. This becomes especially critical when tasks must adhere to deadline constraints.

In the context of saving computational resources, researchers have investigated neural network compression techniques, including pruning (Z. Liu, Sun, Zhou, Huang, & Darrell, 2018) and quantization (J. Yang et al., 2019). However, these approaches typically involve compressing the model during the training phase, necessitating re-training the model after compression.

Recent approaches compress neural networks at inference time (Cai et al., 2020; J. Yu & Huang, 2019a), reducing network size to varying degrees. Greater compression yields lower latency (i.e., the processing time of a task) but at the expense of accuracy. In Chapter 4 we introduced a scheduling system using compressible neural networks for image classification tasks, in which several heterogeneous machines could be used. We developed an approximation algorithm with proven guarantees for maximizing the average accuracy while respecting deadlines constraints.

Cloud and network operators are compelled to mitigate their cloud carbon footprint, driving researchers and scientists to investigate novel methods for conducting ML inference with greater energy efficiency.

The adoption of MLaaS and the expanding size of neural network models has resulted in increased energy consumption, particularly during the inference stage (McDonald et al., 2022). According to reports from NVIDIA (Freund, 2019), 80-90% of Artificial Intelligence (AI) costs stem from inference. Indeed, processing millions of requests, such as those encountered in social networks, can lead to a significant number of inferences in deep learning models, resulting in elevated energy consumption and a sizable carbon footprint (Desislavov, Martínez-Plumed, & Hernández-Orallo, 2023).

Consequently, new energy-aware scheduling solutions are urgently needed, even though the introduction of energy consumption considerations further complicates the scheduling problem. Neural network compression could be used to reduce the energy consumption of a model by reducing its size during inference. Therefore, we propose constraining cloud inference services to operate within an energy budget, which sets a maximum energy threshold for executing a set of inference tasks. The energy consumption of a task is influenced by both its processing time and the specific machine that executes it, given that power consumption can vary across different machines. In this paper, we extend our previous work to tackle the challenge of limiting cloud energy consumption when scheduling compressible tasks.

Our objective is to maximize accuracy while adhering to an energy budget constraint. The scheduling system determines task-to-machine allocations and processing times for each task, ensuring compliance with deadlines and energy budget limitations. We formulate the problem as a non-linear program and develop an approximation algorithm with proven guarantees. Subsequently, we evaluate our approach against traditional scheduling techniques.

The contributions of this paper can be summarized as follows:

- We investigate the scheduling problem in which tasks can be compressed, and the energy consumption must adhere to an energy budget. The tasks under consideration are inference deep learning tasks, which present a trade-off between latency and accuracy. We believe our approach could be extended to other tasks exhibiting diminishing marginal gain.
- We analyze the problem complexity and propose (i) an exact algorithm to solve the problem using fractional relaxation, which serves as the basis for (ii) an approximation algorithm with proven guarantees for various heterogeneous machines (in terms of speed and energy efficiency) that satisfy the energy budget constraint.
- Finally, we validate and compare our approach with state-of-the-art solutions.

The rest of this chapter is organized as follows. In Section 5.2, we introduce the problem formulation. Sections 5.3 and 5.4 are dedicated to the exact and the approximation scheduling algorithms that we propose, respectively. In Section 5.5, we assess the algorithm’s performance through experiments. Finally, we delve into the main discussions and conclusions in Section 5.6, alongside potential future works.

5.2 Problem Formulation

In this section, we define the DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - ENERGY AWARE (DSCT-EA) problem, and formulate it as a Mixed-Integer Program (MIP). Let us consider a set J of n tasks, which can be scheduled on a set M of m machines.

Each machine $r \in M$ is characterized by its speed s_r , quantified in FLOPS (Floating-point Operations per Second), power consumption P_r , measured in Watts (W), and energy efficiency $E_r = s_r/P_r$, computed as the ratio of speed to power consumption. In this regard, the energy efficiency of servers can exhibit significant heterogeneity, largely depending on the hardware architecture specifications. Figure 5.1, derived from a comprehensive evaluation conducted by Desislavov et al. (Desislavov et al., 2023), illustrates the relationship between energy efficiency and speed for across various NVIDIA GPUs designed for servers. The general observed trend is that devices exhibit linear improvement in energy efficiency with the advancement of hardware speed. We order the machines by non decreasing energy efficiency, i.e., $r < r'$ if $E_r < E_{r'}$.

Each task $j \in J$ represents a compressible inference task that requires f_j^{max} floating-point operations for full execution (i.e., without compression) and must be completed before a deadline d_j (measured in seconds). We order the tasks by non decreasing deadlines, i.e., $i < j$ if $d_i < d_j$. An accuracy function $a_j(f)$ specifies the accuracy reached by task j when dedicating $f < f_j^{max}$ floating operations to it. Let $a_j^{max} = a_j(f_j^{max})$ be the maximum accuracy reached by a task. This setting naturally describes scenarios involving “slimmable” networks, such as OFA and AutoSlim (J. Yu & Huang, 2019a; Cai et al., 2020), where the accuracy of a task depends on its level of compression. Further details regarding accuracy functions are provided in Section 5.2.1.

A solution for the DSCT-EA problem specifies, for each task j , (i) the machine on which it is executed and (ii) its compression level, i.e., the number f_j of floating-point operations allocated to it.

Below, we formally model DSCT-EA as a minimization problem of an objective function, subject to several constraints.

Variables. We have two sets of decision variables. x_{jr} represents a set of binary variables with a value of 1 if task j is processed on machine r . t_{jr} denotes the set of variables indicating the

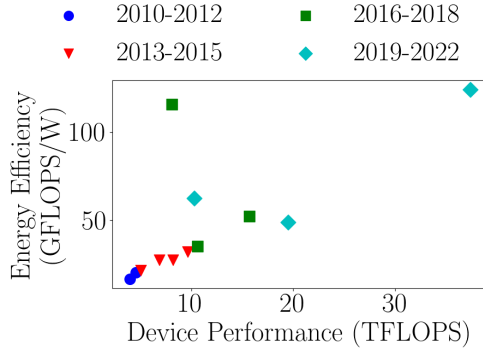


Figure 5.1 – Energy efficiency vs speed for various GPUs (Desislavov et al., 2023).

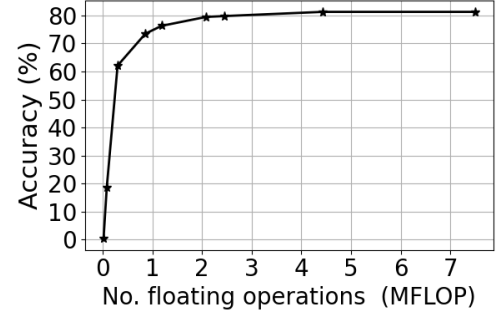


Figure 5.2 – Once-for-All (Cai et al., 2020): Accuracy vs number of floating operations

processing time of task j on machine r . The number of operations dedicated to task j is given by $f_j = \sum_{r \in M} s_r \cdot t_{jr}$.

Model Formulation

$$\min \quad \sum_{j=1}^n \left[1 - a_j \left(\sum_{r \in M} s_r \cdot t_{jr} \right) \right] \quad (5.1a)$$

$$\text{s.t.} \quad \sum_{i=1}^j t_{i,r} \leq d_j, \quad j \in J, r \in M \quad (5.1b)$$

$$t_{jr} \cdot s_r \leq f_j^{\max}, \quad j \in J, r \in M \quad (5.1c)$$

$$t_{jr} \leq x_{jr} \cdot d_j, \quad j \in J, r \in M \quad (5.1d)$$

$$\sum_{r \in M} x_{jr} = 1, \quad j \in J \quad (5.1e)$$

$$\sum_{r \in M} \sum_{j \in J} \frac{s_r}{E_r} \cdot t_{jr} \leq B, \quad (5.1f)$$

$$t_{jr} \in \mathbb{R}^+, x_{jr} \in \{0, 1\} \quad j \in J, r \in M \quad (5.1g)$$

Equation (5.1a) outlines the objective function, which aims to minimize the accuracy error across all tasks (or equivalently, to maximize the accuracy). Constraint (5.1b) ensures that we do not exceed the deadline of a task, accounting for the processing time of previously executed tasks on the same machine. Constraint (5.1c) restricts the total number of floating-point operations dedicated to task j to be no more than $f_j^{\max}$. Constraints (5.1d) and (5.1e) guarantee that each task is allocated processing time on a single machine. Finally, constraint (5.1f) states that the total energy consumed by all machines must not exceed the energy budget B for the system. Here, we consider, in order to evaluate the energy derived from tasks, when some task is processed.

NP Hardness of the problem. DSCT-EA is NP-Complete, as it is an extension of Chapter 4. This can be shown with a reduction from SUM PARTITION PROBLEM.

5.2.1 Accuracy function

We consider accuracy functions that stem from "*slimmable*" neural networks (J. Yu & Huang, 2019a; Cai et al., 2020). Figure 5.2 provides an example of such a function, where the accuracy achieved in an ML inference task depends on the model's size, processing it, and is directly linked to the total number of floating-point operations performed. Such accuracy functions are concave, meaning that the derivative with respect to the number of floating-point operations decreases as the number of operations increases. For a task j where f floating operations have been executed, we define the *marginal gain* as the right-hand derivative of the accuracy function $a'_j^+(f)$, and the *marginal loss* as the left-hand derivative $a'_j^-(f)$. In this work, we will consider piecewise linear functions, where these two derivatives may not always be equal.

Piecewise linear function We consider the special case of linear piecewise functions in the following, as (i) they are a good model for the accuracy function of "*slimmable*" models (see Figure 5.2) and (ii) they can be used to approximate any accuracy functions, for a sufficiently large number of pieces. We use the following notation:

$$a(f) = \begin{cases} \alpha_k \cdot f + b_k, & \text{if } p_k \leq f \leq p_{k+1}, \\ \end{cases} \quad \forall k \in \mathcal{K} \quad (5.2)$$

Here, $\mathcal{K} = \{1, \dots, K\}$, where K is the number of segments, and α_k and b_k , for $k \in \mathcal{K}$, represent, respectively, the slope and y -intercept point for a linear segment k . The values of p_k and p_{k+1} , called breakpoints, represent the start and the end point for each linear segment k . We have $a(p_{K+1}) = a^{\max}$, and $a(0) = b_1 = a^{\min}$.

5.2.2 Fractional Relaxation

Here, we introduce a fractional relaxation of the DSCT-EA problem, named DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - ENERGY AWARE - FRACTIONAL RELAXATION (DSCT-EA-FR), in which a task can be executed on multiple machines. We will utilize a solution to this problem as a starting point for our approximation algorithm for DSCT-EA. We give the formulation for the case in which the accuracy functions are piecewise linear.

$$\min \quad \sum_{j=1}^n -z_j \quad (Dual) \quad (5.3a)$$

$$\text{s.t.} \quad z_j \leq \alpha_{jk} \sum_{r \in M} s_r \cdot t_{jr} + b_{jk}, \quad j \in J, k \in K \quad (\gamma_{jk}) \quad (5.3b)$$

$$\sum_{i=1}^j t_{ir} \leq d_j, \quad j \in J, r \in M \quad (\mu_{jr}) \quad (5.3c)$$

$$\sum_{r \in M} t_{jr} \cdot s_r \leq f_j^{\max}, \quad j \in J \quad (\lambda_j) \quad (5.3d)$$

$$\sum_{r \in M} \sum_{j \in J} \frac{s_r}{E_r} \leq B, \quad (\beta) \quad (5.3e)$$

$$t_{jr} \in \mathbb{R}^+, \quad j \in J, r \in M \quad (\nu_{jr}) \quad (5.3f)$$

A new variable z_j is introduced for each task $j \in J$. Constraints (5.3b) ensures that it is lower than the value of the accuracy function a_j over each segment $k \in K$ for any values of the execution times t_{jr} , $r \in M$. The minimization makes its equal for an optimal solution.

Karush - Kuhn - Tucker Conditions We first discuss the KKT (Karush-Kuhn-Tucker) conditions for the optimization problem and then, we present an algorithm to solve it with polynomial time complexity.

In our convex problem, the KKT conditions are necessary and sufficient conditions for optimality (Boyd & Vandenberghe, 2004) (Chap. 5 p. 226 and 244). First, we write the Lagrangian of the above formulation:

$$L(\mathbf{t}, \boldsymbol{\mu}) = \sum_{j=1}^n -z_j + \boldsymbol{\gamma}^T \mathbf{l}(\mathbf{t}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{t}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{t}) + \boldsymbol{\nu}^T \mathbf{i}(\mathbf{t}) + \beta j(\mathbf{t}),$$

where $\boldsymbol{\gamma} = [\gamma_1 \dots \gamma_{nk}]$, $\mathbf{l}(\mathbf{t}) = [z_j - \alpha_{jk} \sum_{r=1}^m s_r \cdot t_{jr} - \beta_{jk}]_{jk}$, $\boldsymbol{\mu} = [\mu_1 \dots \mu_{nm}]$, $\mathbf{g}(\mathbf{t}) = [\sum_{i \leq j} t_{ir} - d_j]_{j,r}$, $\boldsymbol{\lambda} = [\lambda_1 \dots \lambda_n]$, $\mathbf{h}(\mathbf{t}) = [\sum_{r \in M} t_{jr} \cdot s_r - f_j^{max}]_j$, $\boldsymbol{\nu} = [\nu_1 \dots \nu_{nm}]$, $\mathbf{i}(\mathbf{t}) = [-t_{jr}]_{j,r}$ and $j(\mathbf{t}) = \sum_{j \in J} \sum_{r \in M} t_{jr} \cdot \frac{s_r}{E_r} - B$.

From the *zero-gradient condition* (i.e., the gradient at the optimum $(\mathbf{t}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*, \beta^*)$ should be 0), we have:

$$\frac{\partial L}{\partial t_{jr}} = - \sum_{k \in K} \alpha_{jk} s_r \gamma_{jk} + \sum_{i=j}^n \mu_{ir} + \lambda_j s_r - \nu_{jr} + \beta \frac{s_r}{E_r} = 0, \quad \forall j \in J, r \in M. \quad (5.4)$$

$$\frac{\partial L}{\partial z_j} = -1 + \sum_{k \in K} \gamma_{jk} = 0, \quad \forall j \in J. \quad (5.5)$$

Discussion on Constraints (5.3b). For each task j , we encounter two scenarios for this set of constraints. Either the execution time t_j lies *between 2 breakpoints* (in the line segment l) of the piecewise linear accuracy function or it lies *at a breakpoint* (between the line segments l and $l+1$). In the former case, a single constraint (5.3b) is tight, and its corresponding dual variable is positive ($\gamma_{jl} \geq 0$), while all the dual variables of the other constraints are null. Equation 5.5 implies that $\gamma_{jl} = 1$. In the latter case, two constraints 5.3b are tight, and the two consecutive corresponding dual variables are positive ($\gamma_{jl} \geq 0$ and $\gamma_{j(l+1)} \geq 0$). The dual variables of all the other constraints are null. Equation 5.5 implies that $\gamma_{jl} + \gamma_{j(l+1)} = 1$. Hereafter, we express the equations using a single dual variable γ_{jl} for a task j (which can be set to 1 when j is not at a breakpoint).

Conditions for task execution times on the same machine r . Consider two consecutive tasks $j < n$ and $j+1$ on a machine r . We consider that either the processing time t_j lies *between 2 breakpoints* (in the line segment l) of the piecewise linear accuracy function or it lies *at a breakpoint* (between the line segments l and $l+1$). Similarly, the processing time t_{j+1} either lies in the line segment l' or it lies between the line segments l' and $l'+1$. Applying Equation 5.4 on task j gives:

$$\mu_{jr} + \sum_{i=j+1}^n \mu_{ir} = \alpha_{jl} s_r \gamma_{jl} + \alpha_{j(l+1)} s_r (1 - \gamma_{jl}) - \lambda_j s_r + \nu_{jr} - \beta \frac{s_r}{E_r}$$

For task $j+1$:

$$\sum_{i=j+1}^n \mu_{ir} = \alpha_{(j+1)l'} s_r \gamma_{(j+1)l'} + \alpha_{(j+1)l'(l'+1)} s_r \gamma_{(j+1)(l'+1)} - \lambda_{j+1} s_r + \nu_{(j+1)r} - \beta \frac{s_r}{E_r}$$

Taking the difference, we obtain:

$$\begin{aligned} \mu_{jr} = & (\alpha_{jl} s_r \gamma_{jl} + \alpha_{j(l+1)} s_r (1 - \gamma_{jl})) - (\alpha_{(j+1)l'} s_r \gamma_{(j+1)l'} \\ & + \alpha_{(j+1)l'(l'+1)} s_r \gamma_{(j+1)(l'+1)}) - \lambda_j s_r + \lambda_{j+1} s_r + \nu_{jr} - \nu_{(j+1)r}. \end{aligned} \quad (5.6)$$

If both tasks j and $j + 1$ are not constrained (by f^{max} , by 0, and by their deadlines), the complementary slackness conditions state that $\lambda_j = \lambda_{j+1} = \nu_{jr} = \nu_{(j+1)r} = 0$, giving

$$\alpha_{jl}\gamma_{jl} + \alpha_{j(l+1)}(1 - \gamma_{jl}) = \alpha_{(j+1)l'}\gamma_{(j+1)l'} + \alpha_{(j+1)l'}\gamma_{(j+1)(l'+1)} \quad (5.7)$$

If both tasks are not at a breakpoints (i.e., $\gamma_{j(l+1)} = \gamma_{(j+1)(l'+1)} = 0$), we get

$$\alpha_{jl} = \alpha_{(j+1)l'} \quad (5.8)$$

It means that two unconstrained consecutive tasks should have the same marginal gains (same slopes). Otherwise, it would have been better to execute the one with the larger slope till its deadline or next breakpoint.

If j and/or $j + 1$ tasks are at a breakpoint, using that $\alpha_{jl} \geq \alpha_{j(l+1)}$ and $\alpha_{(j+1)l} \geq \alpha_{(j+1)(l+1)}$ (concave accuracy functions), it gives that

$$\alpha_{(j+1)l'} \geq \alpha_{jl} \geq \alpha_{(j+1)(l'+1)} \quad \text{for } j + 1 \text{ at a breakpoint} \quad (5.9)$$

$$\alpha_{(j+1)l'} \geq \alpha_{j(l+1)} \text{ and } \alpha_{jl} \geq \alpha_{(j+1)(l'+1)} \text{ when} \quad (5.10)$$

both j and $j+1$ at a breakpoint

The first inequality in Eq. 5.10 ensures that it is not better to decrease t_j and increase t_{j+1} , and the second one that is not better to increase t_j and decrease t_{j+1} . We say here that the marginal gains of j and j' are comparable.

If task j is constrained by its deadline, in this case $\mu_{jr} \geq 0$, because of the non-negativity of dual variables. We obtain

$$\alpha_{jl} \geq \alpha_{(j+1)l'} \quad \text{when } j + 1 \text{ is not at a breakpoint and} \quad (5.11)$$

$$\alpha_{jl} \geq \alpha_{(j+1)(l'+1)} \quad \text{otherwise.} \quad (5.12)$$

We thus have decreasing marginal gains (decreasing slopes) of tasks (not constrained by their maximum number of operations) on a given machine in an optimal solution. Indeed, otherwise, it would have been better to increase the execution time of task $j + 1$.

Note that, when $j + 1$ is constrained by its maximum number of operations ($f_{j+1} = f_{j+1}^{max}$), it may happen that the marginal gain and loss of $j + 1$ are higher than the marginal gain of j . Indeed, in this case, the dual variable $\lambda_{j+1} \geq 0$. Equation (5.6) becomes $\alpha_{(j+1)l'} = \alpha_{jls_r} + \lambda_{j+1}$, when j and j' are not at breakpoint and j is not constrained by its deadline nor by f_j^{max} .

The Energy Profiles Let us introduce the concept of energy profile, which will be used extensively throughout the rest of the chapter. The *energy profile* p_r of a machine $r \in M$ indicates the maximum amount of work (in seconds) that can be done on that machine to respect some energy constraints. In particular, the following conditions must be satisfied:

$$\sum_{j \in J} t_{jr} \leq p_r, \forall r \in M \quad \text{and} \quad \sum_{r \in M} p_r \cdot P_r \leq B.$$

These two conditions state that (i) the amount of work on a machine cannot exceed its profile, and (ii) when all machines are fully used up to their profile, the total energy consumption is at most the energy budget. The energy profile for our system is the collection $p = \{p_1, \dots, p_m\}$ of the energy profiles of all machines.

We also define the *energy marginal gain and loss* of a task j executed on machine r , whose number of operation f_j is in segment l , as

$$\begin{aligned} \text{Energy Marginal Gain} &= \frac{E_r}{s_r} \frac{\partial^+ a_j}{\partial t_{jr}} = \begin{cases} E_r \alpha_{j(l+1)} & \text{if } j \text{ at breakpoint} \\ E_r \alpha_{j(l)} & \text{otherwise} \end{cases} \\ \text{Energy Marginal Loss} &= \frac{E_r}{s_r} \frac{\partial^- a_j}{\partial t_{jr}} = E_r \alpha_{j(l)}. \end{aligned}$$

Characteristics of the Energy Profiles. Using again Equation (5.4) and similar technics, we can show that:

- For machines r and r' such that $E_{r'} \leq E_r$, the *energy profile of the most energy efficient machine should be the longest one*.
- The last tasks executed on machines with not a full energy profile should have *equal energy marginal gains* when between breakpoints or *comparable* gain when at breakpoints. Let us consider j and j' as the last tasks scheduled on the machines r and r' , respectively, such that $E_{r'} \leq E_r$. Furthermore, the processing time for the task j is in the breakpoint between the piecewise functions l and $l + 1$. Similarly, the processing time for task j' is between functions l' and $l' + 1$. Then, we have the following inequality:

$$E_r \alpha_{j(l+1)} \leq E_{r'} \alpha_{j'l'} \text{ and } E_{r'} \alpha_{j'(l'+1)} \leq E_r \alpha_{jl}$$

The above inequality states that the energy marginal gain of j (second slope) should be lower than the marginal loss of j' (first slope). Otherwise, it would have been better to decrease the execution time of j' and increase the one of j . Similarly, the energy marginal gain of j' should be lower than the energy marginal loss of j (first slope). Otherwise, it would have been better to decrease the execution time of j and increase the one of j' .

- The *energy slope product of the last task should be lower on machines with a non full energy profile than on machines with a full energy profile*.

5.3 Optimal algorithm for DSCT-EA-FR

In this section, we present an optimal algorithmic solution for DSCT-EA-FR, considering piecewise linear accuracy functions. In particular, we first present an algorithm for solving DSCT-EA-FR when a single machine is used, which is then used as a building block to tackle the multiple-machines scenario.

5.3.1 Algorithm for DSCT-EA-FR, one machine

Algorithm 4 can be used to optimally solve DSCT-EA-FR on one machine. Its inputs are the deadlines of tasks and a list *listSegments* that stores information about the linear segments of the accuracy functions of our tasks. For each segment we know its *slope*, the *task* to which it is associated, its *position* within the accuracy function (e.g., segment number 1 or segment number 2), the number of operations *totalFlops* needed to fully process it, and *usedFlops*, the number of operations currently dedicated to it by the scheduler.

Note that, although Algorithm 4 does not explicitly take energy into consideration, incorporating an energy budget is feasible by treating it as an additional deadline (plus some additional

steps). Further insights into this process will be provided Algorithm 5, where this transformation is explicitly outlined.

The algorithm starts by sorting the segments in non-increasing order of slopes (line 1). Then, for each segment, the algorithms first computes its total processing time based on the value of *totalFlops* and the speed of the machine (line 5), and then adjusts it according to the deadlines of the following tasks (lines 6 - 7). Indeed, increasing too much the processing time for a segment could cause a following task to violate its deadline, which is prevented by line 7. The process is repeated for all the segments. Note that Algorithm 4 always prioritizes segments with higher slope and, for a task $j \in J$, a segment k is considered only if segment $k - 1$ (which has a higher slope) was already processed.

Algorithm 4 Exact algorithm for scheduling on one machine using piecewise linear accuracy functions

Require: List of deadlines $[d_1, \dots, d_n]$, speed s of the machine, and a list of segments *listSegments*, storing information about the linear segments of the accuracy functions of tasks.

Ensure: List of task execution times $[t_1, \dots, t_n]$.

```

1: listSegments.order()                                ▷ Non-increasing order of slopes
2: for seg  $\in$  listSegments do
3:    $j \leftarrow \text{seg.task}$ 
4:    $k \leftarrow \text{seg.position}$ 
5:    $\text{contribution} \leftarrow \frac{\text{seg.totalFlops}}{s}$ 
6:   for  $i > j$  do                                       ▷ For all tasks after
7:      $\text{contribution} \leftarrow \min(\text{contribution}, d_i - \sum_{k \leq i} t_k)$ 
8:    $t_j \leftarrow t_j + \text{contribution}$                    ▷ Update processing time of task  $j$ 
9: return  $[t_1, \dots, t_n]$ 

```

Theorem 5.3.1. Algorithm 4 presents a time complexity $O(n^2)$.

Proof.

At line 1, the algorithm sorts the list of segments in $O(n \log(n))$. Assuming a constant number of segments for each task, the algorithm loops over all n tasks at line 2 and, for each of them, it loops over the following ones (at most n) at line 6. This gives a complexity of $O(n^2)$.

□ The optimality of Algorithm 4 is shown in the next section, contextually with the proof of optimality of the algorithm for multiple machines (Theorem 5.3.2).

5.3.2 Algorithm for DSCT-EA-FR, multiple machines

After dealing with a single machine, we now propose an algorithm for solving DSCT-EA-FR in the multiple-machines scenario, where energy budget is also taken into consideration.

Intuitively, more energy efficient machines should be preferred when we must respect some energy budget; indeed, for a unit of energy, an efficient machine can execute more floating operations than a less efficient one.

Following this intuition, let us define the *naive energy profile* p^{naive} of a system as the energy profile in which we use the most efficient machines until the energy budget is met.

Although intuition might suggest otherwise, the naive energy profile does not always lead to an optimal solution.

We propose an algorithm which computes the optimal energy profile and solution for DSCT-EA-FR when using multiple machines. It comprises two main steps:

- *ComputeNaiveSolution*: It computes the naive profile and the optimal solution for this profile. This procedure is described in Algorithm 5.
- *RefineProfile*: Check if changing the energy profile could lead to a better solution. If so, update accordingly the energy profile and the solution. Repeat this step until no better solution can be found. This procedure is described in Algorithm 6.

ComputeNaiveSolution Algorithm 5 computes the optimal solution for the naive energy profile.

Firstly, the naive energy profile is derived from the energy budget E_{budget} . This is accomplished by sorting the set of machines M in non-increasing order of energy efficiency E and computing, for each of them, the energy profile by selecting the minimum between the maximal execution time for a machine (d^{max}) and the time needed to reach the energy budget B .

Then, the algorithm transforms the problem into an "equivalent" one on a single-machine problem. This is done by checking for each task and for each machine whether the deadline falls inside the machine profile, in which case the machine can be fully used for the task. Otherwise, only a fraction of the machine can be used to process the task. Next, the algorithm uses Algorithm 4 to compute an optimal solution for this single-machine scenario.

We then need to go back to the multiple-machines scenario. It iterates through the tasks in non-decreasing order of deadlines and checks if it's possible to evenly distribute the tasks across the machines, scheduling the same amount of time for each machine. If energy profile limitations prevent this, the algorithm schedules the maximum allowed time on the least energy-efficient machine and repeats the process with the remaining machines. This procedure is repeated for all tasks.

RefineProfile Once we have the optimal solution for the naive energy profile, Algorithm 6 checks whether a better solution could be found by tweaking the energy profile.

Let us define the *accuracy-per-Joule* $\psi_{seg,r}$ as the accuracy gained per unit of energy by increasing the processing time of a linear segment seg in machine r . Note that this is equivalent to the energy marginal gain discussed in Section 5.2, as accuracy-per-Joule is also computed as $\psi_{seg,r} = seg.slope \cdot E_r$.

Firstly, Algorithm 6 generates a list of all pairs $(segment, machine)$, sorted in non-increasing order of accuracy-per-Joule. The pairs at the top of the list offer the highest potential for increasing accuracy.

Next, the algorithm iterates over the list and, for each pair (seg, r) , it computes how much the execution time of seg could be increased on machine r , keeping track of the additional energy E_{add} that it would consume. To do this, it considers the minimum between the energy required for the remaining floating operations of seg and the energy needed to fill the gap between the tasks that were already scheduled and the deadline of $seg.task$. For each pair analyzed, the algorithm iterates over the list of pairs in reverse order, searching for tasks to be reduced. If a pair (seg', r') is found that presents a smaller accuracy-per-Joule value than (seg, r) , it means that it would be beneficial to decrease the execution time of seg' in machine r' and increase that of seg on r . The amount of energy E_{sub} that can be saved by reducing the execution time of seg' is computed, and finally the minimum value between E_{add} and E_{sub} is the actual amount of energy that is "allocated" for seg in machine r , and "deallocated" for seg' in machine r' .

Algorithm 5 *ComputeNaiveSolution*: Computes an optimal scheduling for the naive energy profile

Require: List of segments *listSegments*, the naive energy profile $p^{naive} = [p_1, \dots, p_m]$, the speeds $[s_1, \dots, s_m]$ and the energy efficiencies $[E_1, \dots, E_m]$ of the machines. Deadlines $[d_1, \dots, d_n]$ are in non-decreasing order.

```

1:  $M.order()$  ▷ Order M in non-increasing order of energy efficiency  $E$ 
2:  $E_{temp} = 0$ 
3: for  $r \in M$  do ▷ Compute naive energy profile
4:    $p_r \leftarrow \min(\frac{B-E_{temp}}{\frac{s_r}{E_r}}, d^{max})$ 
5:    $E_{temp} \leftarrow E_{temp} + p_r \cdot \frac{s_r}{E_r}$ 
6: for  $j \in J$  do ▷ Compute temporary deadlines
7:   for  $r \in M$  do
8:      $d_j^{temp} \leftarrow d_j^{temp} + \min(s_r, \frac{p_r \cdot s_r}{d_j}) \cdot d_j$ 
9:  $t \leftarrow \text{Algorithm 4}(d = d^{temp}, s = 1.0)$  ▷ Computes solution in single machine
10:  $K \leftarrow M$ 
11: for  $j \in J$  do ▷ Distribute the task along machines
12:   while  $t_j \geq 0$  do
13:      $k_{min} \leftarrow \arg \min_{k \in K} (E_k)$ 
14:     if  $\sum_{j' \leq j} t_{j'} + \frac{t_j}{\sum_{k \in K} s_k} > p_{k_{min}}$  then ▷ Schedule until the machine energy profile
15:        $t_j \leftarrow t_j - s_{k_{min}} (p_{k_{min}} - \sum_{j' \leq j} t_{j'})$ 
16:        $t_{jk_{min}} \leftarrow p_{k_{min}} - \sum_{j' \leq j} t_{j'}$ 
17:        $K \leftarrow K \setminus \{k_{min}\}$ 
18:     else ▷ Same time for all machines
19:       for  $k \in K$  do
20:          $t_{jk} \leftarrow \frac{t_j}{\sum_{k \in K} s_k}$ 
21:        $t_j \leftarrow 0$ 
22: return  $[t_{jr} : \forall j \in J, \forall r \in M]$ 

```

Algorithm 6 *RefineProfile*: Adapts the naive profile in order to find the optimal solution

Require: List of Segments *listSegments*; List of machines speed *s* and energy efficiency *E*;
List of task deadlines *d*

```

1:  $P \leftarrow \{\}$  ▷ Set containing the pairs (segment, machine)
2: for  $seg \in listSegments$  do
3:   for  $r \in M$  do
4:      $P \leftarrow P + \{(seg, r)\}$ 
5:  $P.order()$  ▷ Order P in non-increasing order of accuracy per Joule
6: for  $(seg, r) \in P$  do
7:    $j \leftarrow seg.task$ 
8:    $E_{add} \leftarrow \min(\frac{seg.totalFlops - seg.usedFlops}{E_r}, (d_j - \sum_{k \leq j} t_{kr}) \frac{s_r}{E_r})$ 
9:   for  $(seg', r') \in P.reverse()$  do
10:    if  $r > r'$  and  $\psi_{seg,r} > \psi_{seg',r'}$  then ▷ Check accuracy-per-Joule for refining Profile
11:       $E_{sub} \leftarrow \min(\frac{seg'.flops}{E_{r'}}, t_{j'r'} \frac{s_{r'}}{E_{r'}})$ 
12:       $E_{transf} \leftarrow \min(E_{add}, E_{sub})$  ▷ Energy transferred between machines
13:       $t_{jr} \leftarrow t_{jr} + \frac{E_{transf} \cdot E_r}{s_r}$ 
14:       $seg.usedFlops \leftarrow seg.usedFlops + E_r \cdot E_{transf}$ 
15:       $t_{j'r'} \leftarrow t_{j'r'} - \frac{E_{transf} \cdot E_{r'}}{s_{r'}}$ 
16:       $seg'.usedFlops \leftarrow seg'.usedFlops + E_{r'} \cdot E_{transf}$ 
17:       $E_{add} \leftarrow E_{add} - E_{transf}$ 
18: return  $[t_{jr} : \forall j \in J, \forall r \in M]$ 

```

Theorem 5.3.2. *Algorithm 7 is optimal and has a time complexity $O(n^2m^2)$.*

Proof.

Optimality. At the end of the first phase, *computeNaiveSolution*, the execution times on all machines satisfy the first set of KKT conditions given by Equations (5.11), (5.12) (non increasing marginal gains on a machine) and Equations (5.8) and (5.10) (comparable marginal gain for tasks not constrained by their deadline and maximum execution time). Indeed, the procedure considers the linear pieces of accuracy functions by decreasing slopes. It is thus not possible to have a task with a lower marginal gain and an earlier deadline, which is processed before one with a higher marginal gain and a later deadline.

At the end of the second phase *RefineProfile*, they satisfy the second set of KKT Conditions stated in the part *Characteristics of the Energy Profiles* of Section 5.2.2: higher energy marginal gains on more energy efficient machines and comparable energy marginal gain for machines with non full energy profiles. We now argue that the conditions of non-increasing marginal gains are maintained following the *RefineProfile* phase, specifically after each iteration of the for loop starting at line 5 of *RefineProfile*. Let's first consider machine r' , where we decrease the processing time of a task j' (line 14 of the *RefineProfile* Algorithm). It's important to note that j' will always be the task on machine r' with the lowest marginal loss. Consider that the number of operations if j' , $f_{j'}$, is in the linear segment $k > 1$ of its accuracy function. If we reduce its number of operations, but not up to the breakpoint $p_{j'k}$, it does not change its marginal gain $\alpha_{j'k}$ and loss $\alpha_{j'k}$, so the condition are still satisfied. Now, if we pass the breakpoint $p_{j'k}$, the marginal gain increases (by concavity of the accuracy function) and is equal to $\alpha_{j'(k-1)}$.

However, we are insured that it won't become higher than the marginal loss of a later task, say α_{il} , with $i > j'$. Indeed, if we decreased the number of floating operations of j' , when it was at $p_{j'k}$, it means that it had the lowest marginal loss $\alpha_{j'(k-1)} \leq \alpha_{il}$. So, the KKT conditions (5.11) and (5.12) are still satisfied. A similar argument shows that, when increasing the number of floating operations of task j on machine r , we cannot decrease its marginal gain lower than the one the marginal loss of a later task (not constrained by its maximum number of floating operations).

The KKT conditions are necessary and sufficient conditions for optimality as the problem is convex. Thus, the algorithm provides an optimal solution.

Complexity. In `computeNaiveSolution` (Algorithm 5), from lines 6-8, the temporary deadlines are computed, which is done in $O(nm)$. In line 9, according to Theorem 5.3.1, it is done in $O(n^2)$. In lines 11-21, the algorithm iterates over every task and each machine.

In `refineProfile` (Algorithm 6), it iterates over each task and each machine and orders the pair list, which is done in $O(nm \cdot \log(nm))$. Then, in lines 6-17, we iterate for each pair (*segment*, *machine*) in a nested loop. Subsequently, Algorithm 6 has a time complexity $O(n^2m^2)$. Therefore, DSCT-EA-FR-OPT has a time complexity $O(n^2m^2)$.

□

Algorithm 7 DSCT-EA-FR-OPT: Computes the optimal solution for DSCT-EA-FR using Piecewise Linear function as accuracy function

Require: List of Segments *listSegments*; List of machines speed s and energy efficiency E ;
 List of task deadlines d

- 1: $t \leftarrow \text{computeNaiveSolution}()$ ▷ Algorithm 5
- 2: $t \leftarrow \text{refineProfile}()$ ▷ Algorithm 6
- 3: **return** $[t_{jr} : \forall j \in J, \forall r \in M]$

5.4 Approximation algorithm for DSCT-EA

Here, we propose an approximation algorithm DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - APPROXIMATION ALGORITHM (DSCT-EA-APPROX for short), which addresses the DSCT-EA problem. We adapted the algorithm developed in Chapter 4. For addressing the energy budget constraints added to the problem, we set that, for each machine, the energy profile found in the fractional relaxation solution works as an upper bound of the load scheduled in the machine. If the work in the machine is equal to the energy profile, i.e. $\sum_{j \in J} t_j = p_r$, the machine is no longer considered in the scheduling. The pseudocode is described in Algorithm 8.

According to the Theorem 4.5.4 in Chapter 4, DSCT-EA-APPROX has an absolute performance guarantee G :

$$OPT - G \leq SOL \leq OPT \quad (5.13)$$

Where OPT is the total accuracy for the optimal solution in DSCT-EA-FR and SOL is the total accuracy for the the solution provided by DSCT-EA-APPROX and the performance guarantee

Algorithm 8 DSCT-EA-APPROX: Approximation algorithm for the scheduling problem on several machines

Input: $[d_1, \dots, d_n], [t_1^{\max}, \dots, t_n^{\max}], [s_1, \dots, s_m], [E_1, \dots, E_m]$

Output: List of task processing times $[t_{jr} : \forall j \in J, r \in M]$

```

1: Sort the segments by non increasing order deadlines.
2:  $\mathbf{t}^f, \mathbf{w}_r^{\max} = \text{DSCT-EA-FR-OPT}([d_1, \dots, d_n], [t_1^{\max}, \dots, t_n^{\max}], [s_1, \dots, s_m], [E_1, \dots, E_m]) \triangleright$ 
   Compute the optimal fractional solution  $\mathbf{t}^f$ , e.g. for piecewise, Algorithm 7.
3: for  $r \in M$  do
4:    $S_r \leftarrow []$   $\triangleright$  List of tasks scheduled on machine  $r$ 
5:    $w_r \leftarrow 0$   $\triangleright w_r$  amount of work on  $r$ 
6:    $F \leftarrow \{\}$   $\triangleright$  Set with fully completed machines
7:   for  $j \in J$  do  $\triangleright$  Schedule each task on the machine with the least amount of work,  $r_{\min}$ 
8:      $r_{\text{best}} \leftarrow \arg \min_{r \in M \setminus F} w_r$ 
9:      $t_{j,r_{\text{best}}} \leftarrow \min(\sum_{r=1}^m t_{j,r}^f, w_{r_{\text{best}}}^{\max} - w_{r_{\text{best}}})$ 
10:     $S_{r_{\text{best}}} \leftarrow S_{r_{\text{best}}} + [j]$ 
11:    if  $w_{r_{\text{best}}} = w_{r_{\text{best}}}^{\max}$  then
12:       $F \leftarrow F + [r_{\text{best}}]$ 
13:   for  $r \in M$  do  $\triangleright$  Cut tasks violating their deadlines and shift the following ones.
14:     for  $j \in S_r$  do
15:       if  $t_{j,r}^{(i)} + t_{jr} \geq d_j$  then
16:          $v_j \leftarrow t_{j,r}^{(i)} + t_{jr} - d_j$ 
17:          $t_{j,r} \leftarrow t_{j,r} - v_j$   $\triangleright$  Cut task
18:         for  $i > j \in S_r$  do  $\triangleright$  Shift the following tasks
19:            $t_{i,r}^{(i)} \leftarrow t_{i,r}^{(i)} - v_j$ 
20: return  $[t_{jr} : \forall j \in J, r \in M]$ 

```

is given by $G = m \int_0^\infty \max_{j,r} \frac{\partial a_j(s_r t)}{\partial t} dt$. For piecewise linear accuracy function, we have

$$G = m(a^{\max} - a^{\min}) \left(1 + \ln \left(\frac{\theta_{\max}}{\theta_{\min}} \right) \right). \quad (5.14)$$

Where $\theta_{\min} = \min_{j \in J} \alpha_{j0}$ and $\theta_{\max} = \max_{j \in J} \alpha_{j|K|}$.

5.5 Results

In this section, we evaluate the approximation algorithm DSCT-EA-APPROX (see Algorithm 8). We first show that the lower bound of Eq. (5.13) is only reached in very specific scenarios: in most cases, DSCT-EA-APPROX achieves solutions that are almost optimal. We then discuss the execution time of our algorithm and demonstrate that DSCT-EA-APPROX provides results within a reasonable time frame. Next, we compare the performance of our algorithm with state-of-the-art methods that either do not employ compression or utilize only limited discrete levels of compression. We show that it can achieve large energy gains with only a small impact on the task accuracy. Lastly, we consider a scenario with 2 heterogeneous machines to investigate the energy profile and the computational contribution of each machine, i.e., the number of floating operations performed.

Hardware Settings. For the experiments, we used a laptop equipped with an Intel Core i9-12900H CPU and an NVIDIA RTX A2000 GPU.

Experiments. We generate synthetic sets of tasks using the Once-For-All (Cai et al., 2020) approach applied to ResNet (He et al., 2016), with 1000 classification classes.

We set the minimum task accuracy to $a^{\min} = 1/1000$, the accuracy of a random guess, and the maximum accuracy to $a^{\max} = 0.82$. These values were chosen based on our tests of ofa-resnet on the ImageNet-1k (Deng et al., 2009) dataset. We denote as θ_j the "task efficiency" of a task $j \in J$, representing the slope of its first segment. We fix the minimum value of task efficiency to $\theta_{\min} = 0.1$. In our experiments, we modeled the accuracy function of a task j as piecewise linear function, constructed by performing a linear regression with 5 segments over an exponential accuracy function of parameter θ_j (Figure 5.2 provides an example, although with more than 5 segments).

We considered machine speeds that are uniformly distributed between 1 TFLOPS and 20 TFLOPS, and energy efficiencies uniformly distributed between 5 GFLOPS/W and 60 GFLOPS/W. These values were selected based on research findings presented in (Desislavov et al., 2023).

We define the *deadline tolerance level*, denoted by ρ , as $\rho = \frac{m^2 \cdot d_{\max}}{\sum_{j \in J} (f_j^{\max} \cdot \sum_{r \in M} s_r)}$. The higher the value of ρ , the more time is allocated for the tasks.

We build scenarios with varying (i) task heterogeneity and (ii) energy budget ratio levels. To this end, we define the *task heterogeneity ratio* μ as $\mu \stackrel{\text{def}}{=} \frac{\theta_{\max}}{\theta_{\min}}$, which reflects the similarity between the accuracy functions of the different tasks. Task efficiencies are uniformly distributed between $\theta_{\min}$ and $\theta_{\max}$, and the value $f_j^{\max}$ for each task is computed so to have $a_j(f_j^{\max}) = a^{\max}$. We also define an energy budget ratio β , which describes how strict is the energy budget. Formally, we have $\beta = \frac{B}{\sum_{r \in M} d^{\max, s_r}}$, where $d^{\max} = \max_{j \in J} (d_j)$. The closer β is to zero, the stricter the energy budget constraint becomes.

Baselines. We benchmark the performance of DSCT-EA-APPROX against 3 different solutions:

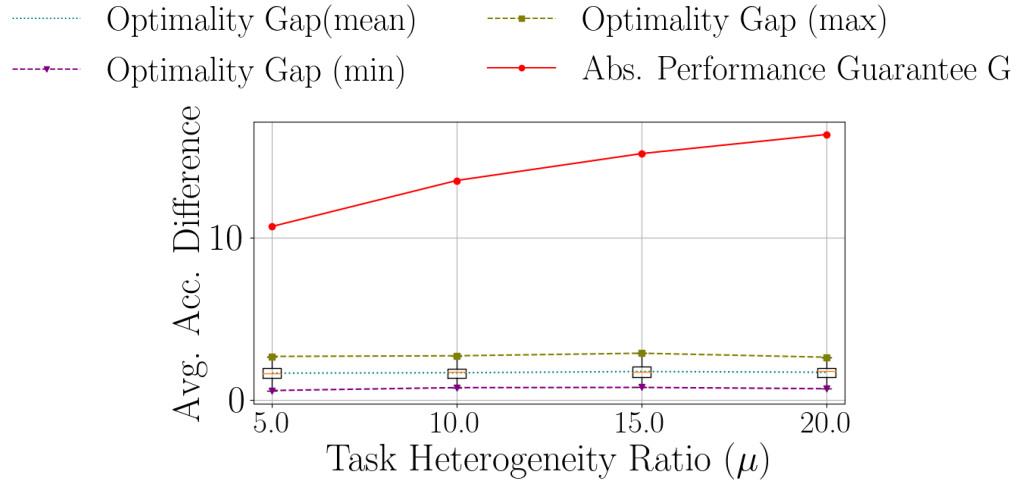


Figure 5.3 – Optimality gap (average accuracy difference between DSCT-EA-UB and DSCT-EA-APPROX) over 100 experiments with varying task heterogeneity.

- DSCT-EA-UB: an upper bound provided by DSCT-EA-FR-OPT, which solves the fractional relaxation DSCT-EA-FR.
- EDF-NOCOMPRESSION: here, no compression is applied, i.e., tasks are always fully processed, performing f^{max} floating operations. We use the EDF (Earliest Deadline First) strategy combined with scheduling on the machine with the least amount of work (Y.-W. Zhang et al., 2023) to determine where tasks should be scheduled. Scheduling is performed until the energy budget is reached, at which point no further tasks are scheduled.
- EDF-3COMPRESSIONLEVELS: this algorithm considers a discrete number of compression levels for neural networks. Three compression levels are used, corresponding to accuracy levels of 27%, 55%, and 82%. A strategy based on prior research (D. Lee & Song, 2021) is implemented. Like the previous approach, tasks are scheduled until the energy budget is reached.

Performance guarantees To assess the performance of our algorithm, we analyze the optimality gap for solutions provided by DSCT-EA-APPROX, which indicates the deviation from optimality. We consider a set of $n = 100$ tasks and $m = 5$ machines, with deadline tolerance $\rho = 0.35$ and energy budget ratio $\beta = 0.5$.

We vary the task heterogeneity ratio μ between 5.0 and 20.0, conducting 100 experiments for each value of μ , and collect the mean, maximum, and minimum accuracy values.

Figure 5.3 illustrates the results of these experiments. It can be observed that, on average, we are quite far from reaching the pessimistic lower bound of Eq. 5.13, which may only be achieved in very specific and rare scenarios.

Execution time analysis We compare the execution time of DSCT-EA-APPROX with that of an optimal solution for DSCT-EA, denoted by DSCT-EA-Opt, which is computed using the cvx-MOSEK software (ApS, 2022), a widely used commercial solver for solving Mixed-Integer Programs (MIP). We considered 2 distinct scenarios: (i) when increasing the number of tasks from $n = 10$ to $n = 500$, keeping fixed $m = 5$ machines, and (ii) when increasing the number of machines from $m = 2$ to $m = 10$, fixing $n = 50$ tasks. We considered an average over 10 instances for each experiment analyzed and a time limit of 60s.

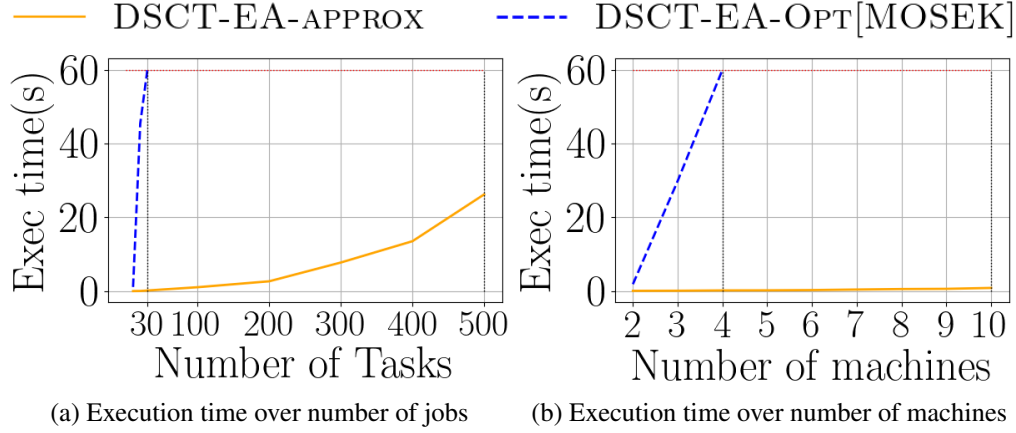


Figure 5.4 – Execution times of DSCT-EA-APPROX vs DSCT-EA-Opt (ApS, 2022) for instances with increasing (a) numbers of tasks, with $m = 5$ and (b) number of machines, with $n = 50$.

The results are shown in Figs. 5.4a and 5.4b respectively. We can observe that the solver (DSCT-EA-UB) could handle for only small instances before reaching the time limit (for experiments 1 and 2, respectively, $n = 30$ and $m = 4$). On other hand, our approach, DSCT-EA-APPROX, was capable of managing large instances with hundreds of tasks and several machines.

We also evaluated the execution time of DSCT-EA-FR-OPT against the Mosek solver applied to DSCT-EA-FR, varying the number the tasks from 100 to 500, fixing $m = 5$. The results are described in Table 5.1. We observe that the developed algorithm DSCT-EA-FR-OPT provides faster results for all instances tested, even with a non-optimized python implementation.

TABLE 5.1 – Comparison of Execution Times for DSCT-EA-FR-OPT and DSCT-EA-FR [Mosek]

Number of tasks	100	200	300	400	500
DSCT-EA-FR-OPT (s)	1.05	2.66	7.75	13.52	26.2
DSCT-EA-FR [Mosek] (s)	1.11	4.26	11.01	21.23	38.07

Comparison with the State-of-the-Art In this section, we compare the performance of DSCT-EA-APPROX against several state-of-the-art approaches under different energy constraints. We tested DSCT-EA-APPROX against EDF-NOCOMPRESSION, EDF-3COMPRESSIONLEVELS, as well as the optimal solution DSCT-EA-UB.

We varied the energy budget ratio from $\beta = 0.1$ to $\beta = 1.0$ to explore a wide range of energy constraints. Additionally, we considered $n = 100$ tasks, $m = 2$ machines, and a deadline tolerance value $\rho = 1.0$. All tasks were assumed to be uniform with $\theta = 0.1$.

Fig. 5.5 illustrates the average accuracy under varying energy budget ratios. We can observe that, for β close to zero, the average accuracy is low due to the stringent energy budget. In general, DSCT-EA-APPROX presents a near optimal average accuracy, which almost matches DSCT-EA-UB and clearly outperforms other solutions. For $\beta = 1.0$, the average accuracy for all methods converges to $a^{\max}$, since the energy budget allows the tasks be fully processed.

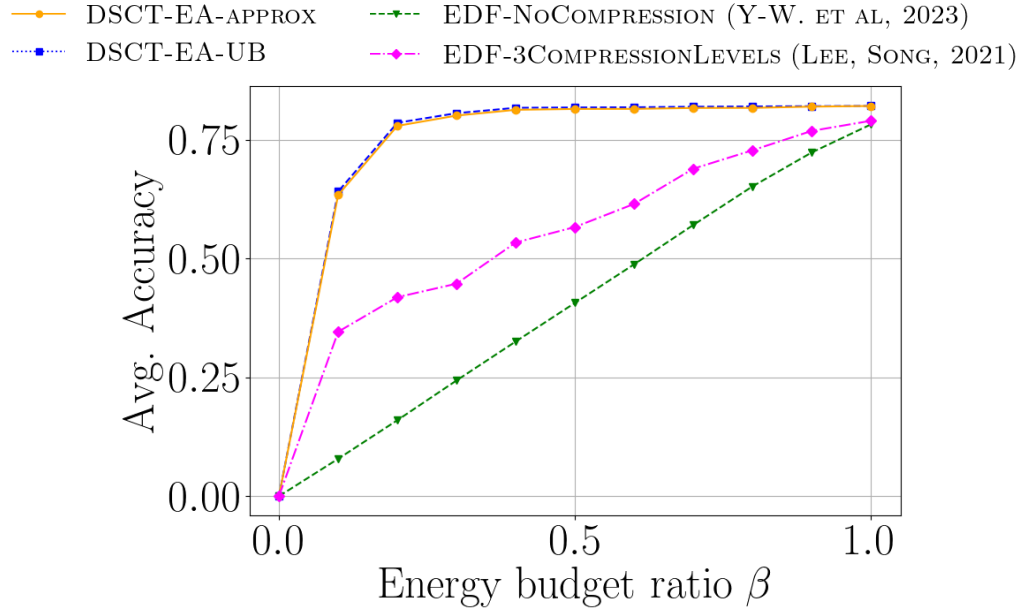


Figure 5.5 – Average task accuracy for DSCT-EA-APPROX and the baselines over energy budget ratio β , for $m = 2$ and $n = 100$.

Energy Gain. Note, that, using our solution, DSCT-EA-APPROX, 70% of the energy can be saved up while only reducing by 2% the average task accuracy, compared to a scenario without compression, achieving a high energy sobriety.

Workload Balancing with Heterogeneous Machines and Tasks. Lastly, we explore how DSCT-EA-APPROX balanced the workload across machines with different energy efficiencies and speeds. We consider a simple scenario with $m = 2$, where machine 1 has speed $s_1 = 2$ TFLOPS and energy efficiency $E_1 = 80$ GFLOPS/W, while for machine 2 we have $s_2 = 5$ TFLOPS and $E_2 = 70$ GFLOPS/W. These values were chosen according to (Desislavov et al., 2023). Essentially, machine 1 is slower but more energy efficient than machine 2.

We used $n = 100$ tasks and looked at two distinct scenarios. In the first one (Uniform Tasks), we used a batch of sets with efficiency value θ uniformly distributed between 0.1 and 4.9. In the second scenario (Earliest High Efficient Tasks), we divide tasks into two sets: the earliest 30% of tasks (according to their deadline) have high efficiency ($4.0 \leq \theta \leq 4.9$), while the remaining ones are not very efficient ($0.1 \leq \theta \leq 1.0$). All the experiments used a deadline tolerance value $\rho = 0.01$, indicating very strict deadlines.

Figs. 5.6a and 5.6b display the final energy profile computed by DSCT-EA-APPROX for both machines, with varying energy budget ratios β , for scenarios Uniform Tasks and Earliest High Efficient Tasks, respectively. Recall that by energy profile we mean how the workload (in seconds) is distributed across the two machines by DSCT-EA-APPROX, which considers energy constraints during scheduling. Fig. 5.6a shows that, unsurprisingly, the computed profile for scenario Uniform Tasks is very close to the naive one. The same cannot be said about the Earliest High Efficient Tasks scenario, as Fig. 5.6b shows a behavior that sensibly deviates from the naive profile.

For $\beta \leq 0.4$, there's a notable difference between the energy profiles of machines 1 and 2 compared to the last deadline d^{max} . This difference arises because there's a significant refinement in the profile caused by tasks with high efficiency values θ being constrained by the deadline on

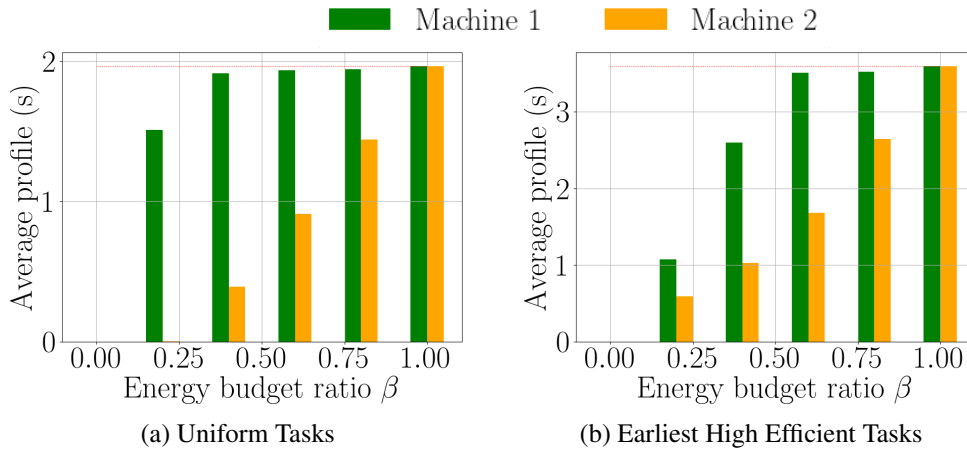


Figure 5.6 – Energy profile of 2 machines when varying the energy budget ratio. Machine 1 is more energy efficient than machine 2.

machine 1. Consequently, the refinement in the profile increases the workload on machine 2 to accommodate these tasks. For example, for $\beta = 0.4$, the naive profile indicates $p_1^{naive} = 1.96s$ and $p_2^{naive} = 0.0$. However, the final profile yields $p_1 = 1.26s$ and $p_2 = 0.25s$. Thus, the profile was refined during the optimal algorithm, leading to an increase of the workload of machine 2.

5.6 Conclusion

In this chapter, we introduced a novel scheduling system for deep learning inference tasks, incorporating neural network compression, deadlines, and energy constraints. We formulated the problem as a Mixed-Integer Program and developed an exact algorithm for its fractional relaxation and an approximation algorithm for the original problem.

Our evaluations show that our algorithm outperforms traditional scheduling methods across various energy budgets and closely approaches optimality. Finally, we offered some insights into the behavior of our algorithm, especially for scenarios involving heterogeneous machines and varying energy constraints.

Looking ahead, we identify the integration of renewable power sources into the scheduling problem as promising avenues for future research. Moreover, we intend to consider in the problem model the energy consumption resulted from communication of devices.

Evaluating AI Environmental Impacts for improving efficiency

Small is Sufficient: Reducing the World AI Energy Consumption Through Model Selection

The energy consumption and carbon footprint of Artificial Intelligence (AI) have become critical concerns due to rising costs and environmental impacts. In response, a new trend in green AI is emerging, shifting from the “bigger is better” paradigm, which prioritizes large models, to “small is sufficient,” emphasizing energy sobriety through smaller, more efficient models.

We explore how the AI community can adopt energy sobriety today by focusing on model selection during inference. Model selection consists of choosing the most appropriate model for a given task, a simple and readily applicable method, unlike approaches requiring new hardware or architectures. Our hypothesis is that, as in many industrial activities, marginal utility gains decrease with increasing model size. Thus, applying model selection can significantly reduce energy consumption while maintaining good utility for AI inference.

We conduct a systematic study of AI tasks, analyzing their popularity, model size, and efficiency. We examine how the maturity of different tasks and model adoption patterns impact the achievable energy savings, ranging from 1% to 98% for different tasks. Our estimates indicate that applying model selection could reduce AI energy consumption by 27.8%, saving 31.9 TWh worldwide in 2025 —equivalent to the annual output of five nuclear power reactors.

6.1	Introduction	89
6.2	Analysis of AI tasks and models	89
6.2.1	Identifying the more popular AI tasks	90
6.2.2	AI benchmarks methodology	93
6.2.2.1	Benchmark sources	93
6.2.2.2	Number of parameters estimation	93
6.2.2.3	Models usage	93
6.2.2.4	Key models selection	95
6.2.3	Utility dependency on model size	95
6.2.4	Model adoption	98
6.3	Estimating the savings of AI model selection	100
6.3.1	AI inference energy consumption measurement methodology	101
6.3.1.1	Carbon Tracker tool	101
6.3.1.2	Experiment 1: Validation of software-based tools for measuring AI energy consumption	102
6.3.1.3	Experiment 2: Energy consumption and number of parameters	103
6.3.1.4	AI models energy estimation methodology	103
6.3.2	Savings by model selection	106
6.4	Conclusion	109

6.1 Introduction

The use of Artificial Intelligence (AI) has skyrocketed in recent years, transforming diverse fields as medicine, education, finance, robotics, games, etc., with amazing successes such as automatic writing, translation, driving, disease diagnosis, etc.

The use of such models, coupled with the growth in data volumes and the infrastructure required to run them, with ever more powerful machines, has led to a rise in energy consumption (C.-J. Wu et al., 2022). Addressing AI energy consumption is, therefore, critical. Then, different methods have been proposed to reduce the energy consumption of AI targeting on the two main phases of a model life cycle: (i) training (e.g., hardware optimization (Tao et al., 2020), model compression (B. Zhang et al., 2018), architecture designing (Mehta et al., 2019)) and (ii) inference (e.g., model selection (Asperti et al., 2021; J.-R. Yu et al., 2022)). We focus on *inference* here. Although the energy expended per inference task is orders of magnitude less than that of training, the sheer volume of inference requests results in a substantial cumulative impact. Indeed, inference has been estimated to account for approximately 60% of ML energy usage (Patterson et al., 2022).

In this chapter, we aim to investigate the *impact of model selection* on inference AI energy consumption. Indeed, the release of new models such as DeepSeek LLM (DeepSeek, s. d.) attracted attention to two potential paradigms for the future of the AI industry. The “*bigger-is-better*”, in which everybody buys the best hardware (e.g., from NVIDIA) to run the best-performing (and usually the largest) models, and the “*small-is-sufficient*”, in which users prioritize much smaller models with similar performance than the state-of-the-art models. In this paper, we estimate the energy consumption of these two potential futures.

Our contributions are as follows. We investigate the energy savings that can be achieved by applying AI model selection globally. To do this, we first identify the most commonly used AI tasks in data centers. For each task, we analyze benchmarks from the *Hugging Face (Papers with Code: The latest in Machine Learning, s. d.)* and *Papers with Code (Hugging Face, s. d.)* platforms to assess model utility and size tradeoffs. By evaluating task maturity and model adoption, we identify key opportunities for energy-efficient AI. Then, we propose a methodology for estimating the energy consumption of AI models during inference, and finally, we estimate the energy savings by applying model selection and redirecting AI inference requests to energy-efficient models.

Our results show the *huge impact of the task stage of development and the patterns of model adoption* on the potential energy reduction, with a range from 1% to 98% across the different AI tasks. Our projection indicates that *applying model selection globally* could lead to a 27.8% reduction in AI energy consumption *without much impact on the model utility*. On the contrary, *always using the state-of-the-art models* to solve all tasks would *increase the global AI energy consumption by 111%*.

In Section 6.2, we discuss what are the main AI tasks, the models to solve them, their utility depending on model size, and their adoption. Then, in Section 6.3, we first present our methodology for measuring and estimating AI model energy consumption. Then, we estimate the energy reduction by applying model selection. Finally, in Section 6.4, we draw the conclusions and point the future works.

6.2 Analysis of AI tasks and models

This section analyzes the potential for energy sobriety in AI inference. We begin by comprehensively investigating the most prevalent AI tasks in data centers, the models employed for these

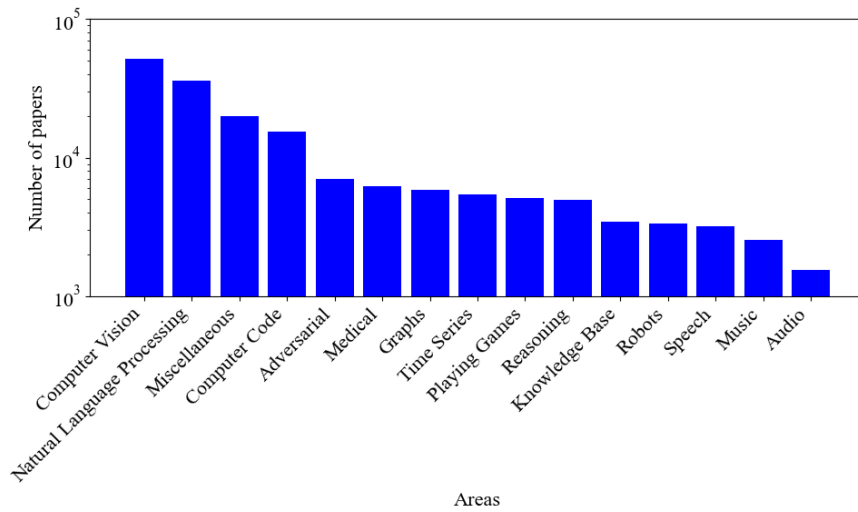


Figure 6.1 – **Research Focus by Area on Papers With Code:** Numbers of papers across different AI research areas on the *Papers With Code* platform. The results highlight significant interest in Computer Vision (image processing), Natural Language Processing, and a diverse range of Multimodal tasks categorized as Miscellaneous.

tasks, their adoption rates, and relevant benchmarks for comparison. Subsequently, for each identified task, we examine the relationship between model size, user adoption, and utility, aiming to pinpoint opportunities for reducing energy consumption through informed model selection.

6.2.1 Identifying the more popular AI tasks

With the rise of cloud computing and the evolution of computational resources, major cloud providers, such as Amazon, Google, and Microsoft now offer platforms for deploying AI models at scale. These platforms facilitate the widespread use of AI across various domains, making it essential to understand which AI tasks are most frequently deployed.

To address this, we conducted an investigation to: (i) identify the most commonly deployed AI tasks in data centers, and (ii) analyze reliable benchmarks used to assess and compare AI models for these tasks.

Several studies have investigated the key AI tasks addressed by industry and academia. For instance, the 2024 AI Index Report (Perrault & Clark, 2024) highlights trending AI tasks and benchmarks associated with notable models, reported in Table 6.1a. Main listed tasks are: *Language*, *Coding*, *Image Computer Vision*, *Video Computer vision*, *Reasoning*, *Audio*, *Agents*, *Robotics*, and *Reinforcement Learning*. Similarly, Bommasani et al. (Bommasani et al., 2021) examined models trained on vast datasets and identified key areas of application, including *language*, *vision*, *robotics*, *reasoning (logic)*, and *interaction*.

In parallel, some platforms, such as *Papers With Code* and *Hugging Face*, propose to serve as collection of multiple models for several AI tasks. The *Papers With Code* platform provides a collection of scientific papers with available implementation across several topics in computer science, including Artificial Intelligence. The number of papers for each AI field is available in Figure 6.1.

The *Hugging Face* platform provides an open source library for hosting and deploying AI models with broad applicability. All AI tasks considered by *Hugging Face* and their respective domains are described in Table 6.1b.

A key metric for assessing the popularity of AI models in *Hugging Face* is the **number of downloads**, which reflects real-world adoption by users and developers. Figure 6.2 describes the total number of downloads for the most popular AI fields and tasks. We observe that tasks belonging to *language*, *vision*, and *audio* fields present a high level of adoption.

This observation is followed by the growing interest of industry and the academia in models for some high-impact tasks, e.g., text generation. Several models, such as *ChatGPT* (OpenAI), *Gemini* (Google), *Deepseek*, and *Llama* (Meta), have attracted a lot of interests by researchers and are widely deployed among several data centers. For instance, data from Meta point that the *Llama* usage in major cloud service providers more than doubled from May through July 2024.

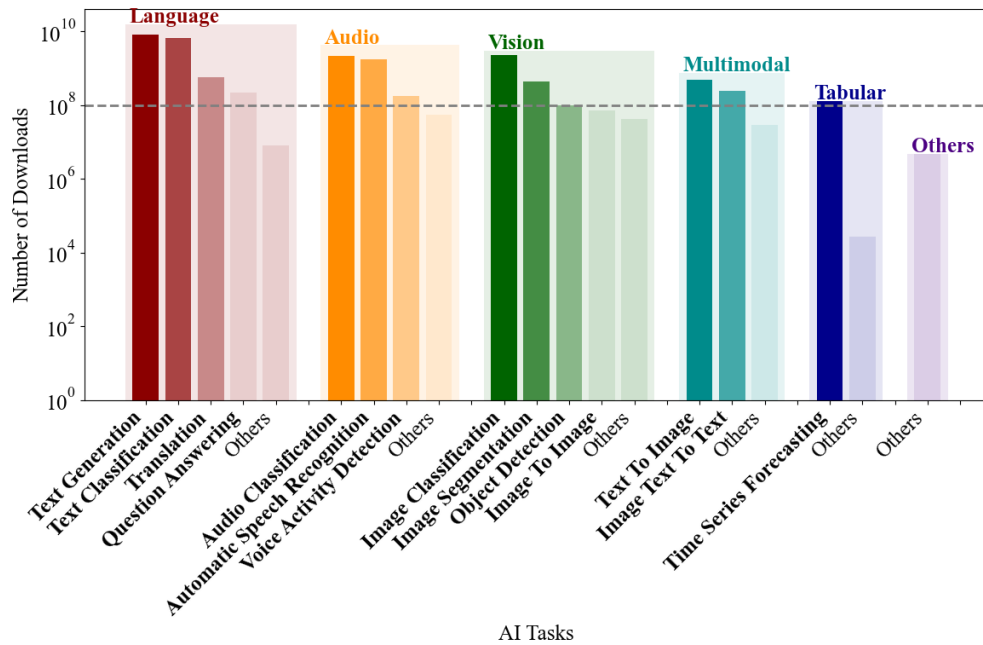


Figure 6.2 – **Most popular AI fields and tasks.** Number of total model downloads on the *Hugging Face* platform, categorized by AI field and AI task. The dashed gray line represents the download threshold used for task evaluation in this study, encompassing 97.3% of all platform downloads.

Based on the popularity analysis, we selected the most frequently used tasks in *Hugging Face* platform to investigate the potential of AI energy sobriety. We focused on tasks with more than 10⁸ downloads in *Hugging Face* platform. The selected tasks grouped by their respective field are the following:

- (i) for the language field: *text generation*, *text classification*, and *translation*;
- (iii) for the audio field: *automatic speech recognition*, and *audio classification*;
- (ii) for the vision field: *image classification*, *object detection*, *image segmentation*;
- (iv) for the multimodal field: *image-text to text* and *text to image*; and
- (v) for the tabular data field: *time series forecasting*.

Field	Sub-field
Language	Understanding
	Generation
	Factuality
Coding	Generation
Image Computer Vision	Generation
	Instruction Following
	Editing
	Segmentation
	3D Reconstruction
Video Computer Vision	Generation
Reasoning	General
	Mathematical
	Visual
	Moral
	Casual
Audio	Generation
Agents	General
	Task Specific
Robotics	-
Reinforcement Learning	-

(a) AI trending fields (with respective sub-fields) reported by The AI Index Report (Perrault & Clark, 2024)

Task	Field
Text Classification	Language
Question Answering	Language
Text Generation	Language
Translation	Language
Token Classification	Language
Text2text Generation	Language
Fill Mask	Language
Sentence Similarity	Language
Summarization	Language
Multiple Choice	Language
Zero Shot Classification	Language
Text Retrieval	Language
Text To Text	Language
Feature Extraction	Language
Table Question Answering	Language
Table To Text	Language
Automatic Speech Recognition	Audio
Audio Classification	Audio
Text To Speech	Audio
Audio To Audio	Audio
Voice Activity Detection	Audio
Text To Audio	Audio
Image Classification	Vision
Object Detection	Vision
Image Segmentation	Vision
Zero Shot Image Classification	Vision
Video Classification	Vision
Image To Text	Vision
Image Feature Extraction	Vision
Image To Image	Vision
Unconditional Image Generation	Vision
Image Text To Text	Vision
Image To Video	Vision
Image To 3D	Vision
Depth Estimation	Vision
Visual Question Answering	Vision
Document Question Answering	Vision
Zero Shot Object Detection	Vision
Reinforcement Learning	Reinf. Learning
Tabular Classification	Tabular
Tabular Regression	Tabular
Time Series Forecasting	Tabular
Graph ML	Graph
Text To Video	Multimodal
Mask Generation	Multimodal
Text To 3D	Multimodal
Video Text To Text	Multimodal
Text To Image	Multimodal
Image To Video	Multimodal
Image Text To Text	Multimodal
Robotics	Robotics
Other	Other

(b) List of tasks and fields on the *Hugging Face* platform.

TABLE 6.1 – **List of Artificial Intelligence (AI) fields and subfields or tasks.** (Left 6.1a) Reported by the AI Index Report and (Right 6.1b) on the *Hugging Face* platform.

Moreover, we evaluate two additional text generation sub-tasks -*mathematical reasoning* and *code generation*- which are widely studied in literature and pointed as trending tasks in Artificial Intelligence.

The tasks are described in Appendix B. The above tasks present more than 22×10^9 downloads and cover approximately 97.3% of all *Hugging Face* downloads.

6.2.2 AI benchmarks methodology

Then, we selected relevant AI benchmarks that analyze and compare models addressing the investigated tasks. For this, we selected benchmarks that assess (i) the model size (in terms of number of parameters) and (ii) the utility value, which evaluates models performance. All the benchmarks are listed in Table 6.2 and are described in Appendix B.

6.2.2.1 Benchmark sources

In our investigations, we selected benchmarks from two different sources: *Hugging Face* and *Papers With Code* platforms.

6.2.2.2 Number of parameters estimation

Although compiling model size is one of our criteria for selecting benchmarks, in some benchmarks, the parameter count of some models was not available.

For *time series forecasting* task, the benchmarks using the *ETTh1* dataset did not include parameter counts. Then, when available, we retrieved this information from the original research papers.

For the *text generation* (more specifically *Lmsys NPHardEval* benchmark), *mathematical reasoning* and *Image-text to text* tasks, some models do not publicly disclose their weights (e.g., GPT-4 or Gemini). In such cases, we relied on size estimates provided by Ecologits ([Impact, 2024](#)).

6.2.2.3 Models usage

In our analysis, we wanted to measure model usage, i.e., the extent to which models are adopted by users in data centers. Since this information is not publicly available, we estimated model usage by looking at the number of downloads:

1. For benchmarks hosted on the *Hugging Face* platform, we considered the number of monthly downloads obtained through the *Hugging Face* API;
2. For benchmarks from *Papers With Code*, for each model, we searched for its implementation on the *Hugging Face* platform and, if available, we retrieved the number of monthly downloads;
3. For models that are not open-source and only accessible through web applications, we estimated popularity based on monthly website visits using data from *SimilarWeb* ([SimilarWeb, 2025](#)).

For the *Papers With Code translation* task benchmark, most of the papers, except for T5-11B, have no implementation in the *Hugging Face* platform. Thus, we decided to extend the benchmark by adding other T5 family models not considered in the original benchmark, as they are the most

Benchmark	AI Task (<i>subtask, if applicable</i>)	Dataset	Field
OpenLLM Leaderboard (Beeching et al., 2023)	Text Generation	HuggingFace	Language
LMSys Chatbot Arena (Zheng et al., 2024)			
NPHardEval (Fan et al., 2023)			
BigCode Leaderboard (Ben Allal, Muennighoff, Kumar Umapathi, Lipkin, & von Werra, 2022)	Text Classification		
mtebLeaderboard (Muennighoff, Tazi, Magne, & Reimers, 2022)			
WMT English-German (Bojar et al., 2014)	Translation	Papers with Code	
Open Object Detection Leaderboard (Rafael Padilla & the Hugging Face Team, 2023)	Object Detection	Hugging Face	Vision
Imagenet (Russakovsky et al., 2015)	Image Classification		
Semantic Segmentation on ADE20K (B. Zhou et al., 2017)	Image Segmentation	Papers With Code	
Open ASR Leaderboard (Srivastav et al., 2023)	Automatic Speech Recognition	Hugging Face	Audio
ARCH (La Quatra et al., 2024)	Audio Classification		
GenAI (Ku et al., 2024)	Text to Image	HuggingFace	Multimodal
MMMU Benchmark (Yue et al., 2024)	Image-Text to Text		
Eth1-336 (H. Zhou et al., 2021)	Time Series Forecasting	Papers with Code	Tabular

TABLE 6.2 – **Evaluated AI Benchmarks.** AI Benchmarks evaluated in this study for analyzing the trade-off between model utility and size. For each benchmark, we also list also the corresponding task, source, and field.

downloaded models for translation on *Hugging Face*. We considered the utility values reported in the original paper (Raffel et al., 2020), introducing the T5 models.

6.2.2.4 Key models selection

For each evaluated task, we select among the available models in the corresponding benchmarks, the two key models (one best-performing and one energy-efficient). For this, we first define the efficiency metric as the ratio between the utility value and the number of parameters.

$$\text{Efficiency} = \frac{\text{Utility}}{\text{Nb. Parameters}}$$

Then, we identify the key models using the following criteria:

1. `best-performing` model: The model with the highest utility value.
2. `energy-efficient` model: The model with the highest efficiency metric, among those whose utility drop does not exceed 5%. For the tasks *Text-to-Image*, *Image-Text-to-Text*, *Speech Recognition*, and *Object Detection*, no model satisfies this constraint due to the nature of the discrete models evaluated. In these cases, we select the model with the highest efficiency overall, even if the utility drop exceeds 5%, reaching up to 18.2% in the worst case.

6.2.3 Utility dependency on model size

The selection of an AI model for inference within data centers is a critical factor influencing both energy consumption and the resulting utility. This section investigates how judicious model selection can promote energy efficiency without substantial compromise to utility. Specifically, we analyze the relationship between model size and utility across a range of benchmarks. Figure 6.3 presents this analysis, displaying the utility value of each evaluated model as a function of its parameter count. Each point on the graph corresponds to a distinct model within a given benchmark.¹ The size of each point is scaled to reflect model popularity, as measured by: (i) download counts for models hosted on *Hugging Face* (blue); (ii) download counts of *Hugging Face* equivalent models for those hosted on *Papers With Code* (orange); and (iii) web visit counts for models accessed through external APIs (gray)."

Within each benchmark, we highlight two key models. The `energy-efficient` model (green) balances high accuracy with a minimal parameter count, offering a favorable trade-off between performance and resource consumption. The `energy-efficient` model is selected as described in Section 6.2.2.4. Conversely, the `best-performing` model (red) is the one with maximum utility and typically has the highest number of parameters, often at the expense of increased energy usage. The key models for each task are reported in Table 6.3.

Across the benchmarks, the utility curves demonstrate a sublinear trend following the law of diminishing returns (Faccarello, 2016). Models with lower parameter counts exhibit a higher marginal gain in utility, reflecting their improved ability to capture data patterns. Conversely, larger models yield diminishing returns in utility with increasing parameter count (see Figure 6.4).

This behavior aligns with scaling laws observed in language models (Kaplan et al., 2020), which posit a power-law relationship between model size, dataset size, and loss.

1. All benchmarks are also available in <https://tsb4.github.io/HF/>

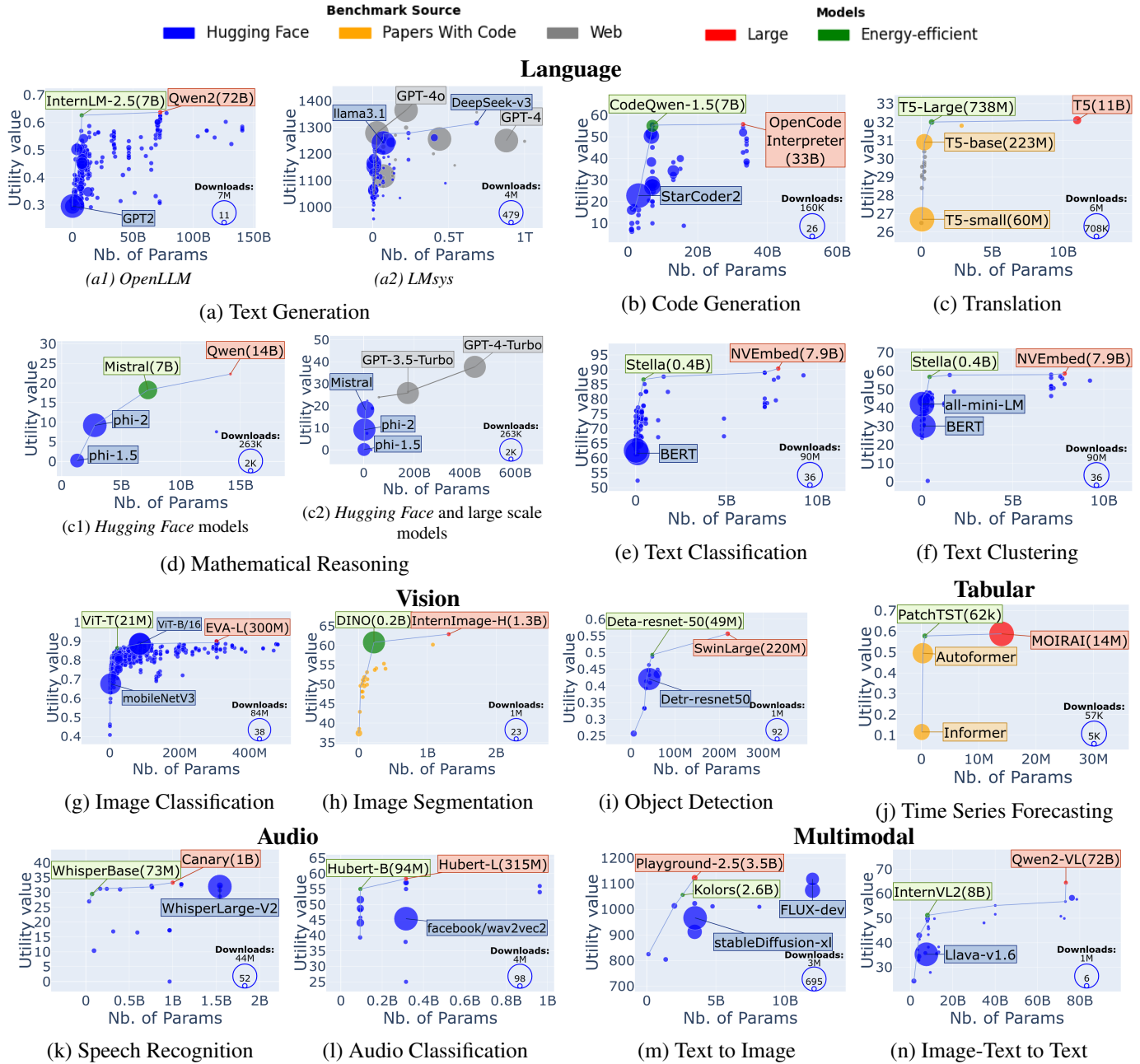


Figure 6.3 – **Tradeoff AI model utility vs. size.** Model utility value versus number of parameters for popular AI tasks. Each point is a single model and its size is proportional to the model popularity. The green and red points indicate the energy-efficient and the best-performing models, respectively. An interactive version of the plots can be found in <https://tsb4.github.io/HF/>.

Task	Model	Params	Utility	Energy(J)	Downloads
Text Generation	<i>internlm/internlm2_5-7b-chat (efficient)</i>	8B	0.6	8035.0	37281
Text Generation	Qwen/Qwen2-72B-Instruct (best)	73B	0.6	35529.4*	92091
Image Classification	<i>timm/tiny_vit_21m_512.dist_in22k_ft_in1k (efficient)</i>	21M	0.9	1907.0	1816
Image Classification	timm/eva02_large_patch14_448.mim_m38m_ft_in22k_in1k (best)	305M	0.9	5501.2	3201
Object Detection	<i>jozhang97/deta-resnet-50-24-epochs (efficient)</i>	49M	0.5	4318.1	210
Object Detection	jozhang97/deta-swin-large (best)	219M	0.6	8653.3	47849
Speech Recognition	<i>openai/whisper-base.en (efficient)</i>	73M	29.5	724.3	605075
Speech Recognition	nvidia/canary-1b (best)	1B	33.3	3726.0*	11597
Image-Text to Text	<i>OpenGVLab/InternVL2-8B (efficient)</i>	8B	51.2	84.4	126306
Image-Text to Text	OpenGVLab/InternVL2-40B (best)	40B	55.2	298.7	5391
Text to Image	<i>Kwai-Kolors/Kolors (efficient)</i>	3B	1056.4	2625.5	1914
Text to Image	playgroundai/playground-v2.5-1024px-aesthetic (best)	3B	1123.0	3214.5	233322
Text Classification	<i>NovaSearch/stella_en_400M_v5 (efficient)</i>	435M	86.7	5824.7	385014
Text Classification	nvidia/NV-Embed-v2 (best)	8B	90.4	12832.5	324552
Translation	<i>google-t5/t5-large (efficient)</i>	738M	32.0	111.0	1028285
Translation	google-t5/t5-11b (best)	11B	32.1	442.0	1648300
Audio Classification	<i>ALM/hubert-base-audioset (efficient)</i>	94M	55.0	388.6	146
Audio Classification	ALM/hubert-large-audioset (best)	315M	58.3	766.2	98
Image Segmentation	<i>IDEA-Research/grounding-dino-base (efficient)</i>	223M	60.8	68.2	1064357
Image Segmentation	OpenGVLab/internimage_h_22kto1k_640 (best)	1k1B	62.9	175.2	23
Time Series Forecasting	<i>ibm-granite/granite-timeseries-patchtst (efficient)</i>	616K	0.6	405.2	7030
Time Series Forecasting	Salesforce/moirai-1.0-R-small (best)	14M	0.6	5606.5	56895
Code Generation	<i>Qwen/CodeQwen1.5-7B-Chat (efficient)</i>	7B	55.1	7518.2	60861
Code Generation	m-a-p/OpenCodeInterpreter-DS-33B (best)	33B	55.8	21035.5*	507
Mathematical Reasoning	<i>mistralai/Mistral-7B-Instruct-v0.1 (efficient)</i>	7B	18.2	7688.2*	200741
Mathematical Reasoning	Qwen/Qwen-14B-Chat (best)	14B	22.3	12004.3*	2457
Text Clustering	<i>NovaSearch/stella_en_400M_v5 (efficient)</i>	435M	56.7	5824.7	385014
Text Clustering	nvidia/NV-Embed-v2 (best)	8B	58.5	12832.5	324552

TABLE 6.3 – **Key models for each AI task.** Energy-efficient models are in *italic* and best-performing models in **bold**.

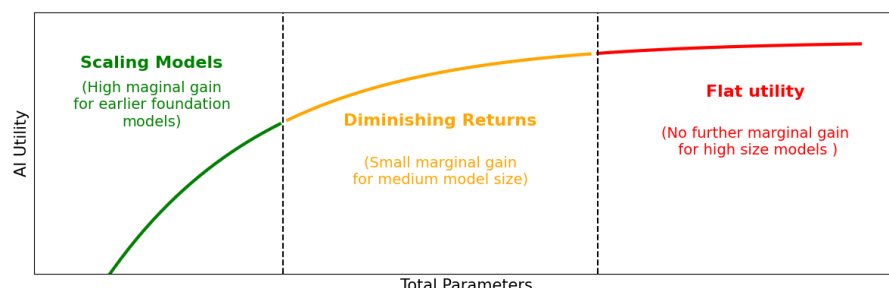


Figure 6.4 – **Law of diminishing return.** Diminishing Marginal Gains in AI Utility with Increasing Model Parameters. The relationship exhibits three phases: (i) Scaling Model: high initial marginal utility gain; (ii) Diminishing Returns: decreasing marginal gain with larger models; and (iii) Flat Utility: minimal marginal gain approaching a plateau.

This observation presents a significant opportunity for enhancing energy efficiency in AI inference. By transitioning from large, high-performing models to smaller, energy-efficient alternatives, substantial reductions in model size and, consequently, energy consumption can be achieved with a minimal impact on utility. For example, in *speech recognition*, the best-performing model is 14 times larger than the energy-efficient model, while providing only a 7.8% improvement in utility.

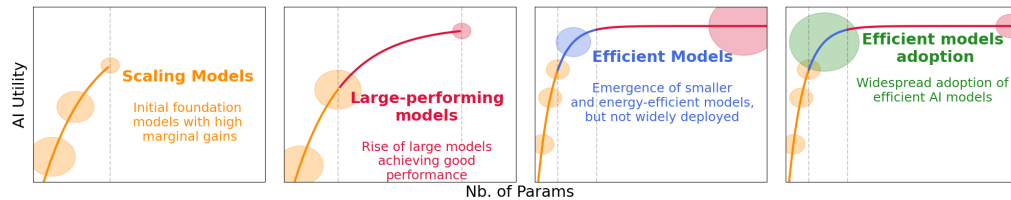
The maturity of an AI task significantly influences the utility-size trade-off. As a task matures with a high-performing model, subsequent development often focuses on creating smaller, yet comparably effective, models (Desislavov et al., 2023). This trend is particularly evident in mature and well-established domains such as *image classification*, *audio classification*, and *speech recognition*. For these tasks, extensive research has optimized model efficiency over time. Consequently, the Pareto frontier relating model size to utility is almost flat for large models, indicating a very small increase in utility with increasing number of parameters.

6.2.4 Model adoption

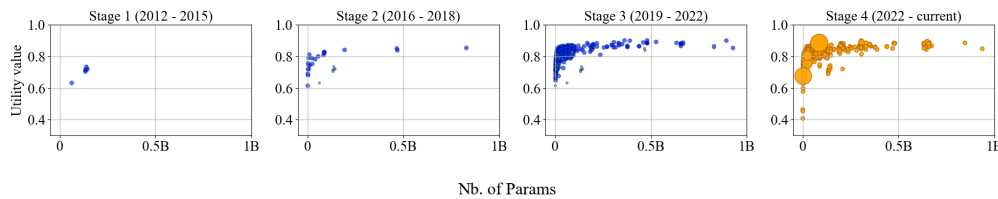
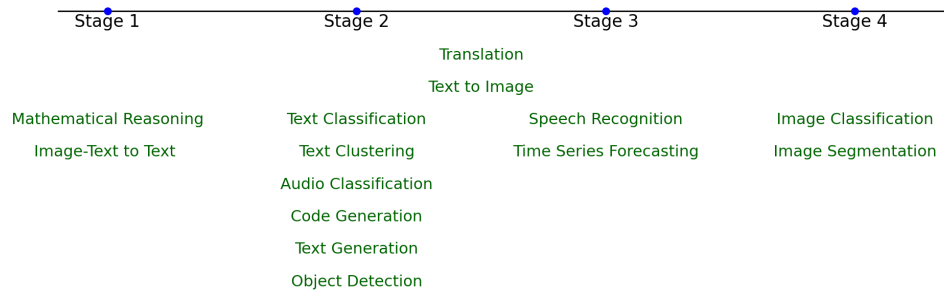
Model usage patterns play an important role in responsible AI. As exemplified in Figure 6.3, all users do not use the same model to solve the same task. Even more, a large number of them do not use the best available models. In fact, model adoption is influenced by several factors: model and brand popularity, habit (Labrecque, Wood, Neal, & Harrington, 2017; Heidenreich & Kraemer, 2016; Ram & Sheth, 1989), model size relative to available hardware, developer vs. general public usage, task maturity, and so on.

First, due to brand popularity and habit, the most widely used models (big blue dot) tend to be models from famous model families: GPT2, GPT4 for LLM, Phi2, Mistral, BERT for *Mathematical Reasoning*, StableDiffusion for *Text to Image*, even if better alternatives exist for some tasks (see e.g., Figure 6.3m).

Second, due to the price and availability of hardware, especially the best GPUs, model size has a large impact on model usage by developers. For *Text Generation*, the *OpenLLM benchmark*, which considers models available on the *Hugging Face* platform, suggests that some small models (e.g., GPT-2) tend to be very popular (with large numbers of downloads) among developers due



(a) The 4 stages of development of an AI task over time

(b) Example: *Image Classification*

(c) Stages of development of the most popular AI tasks

Figure 6.5 – The Four-Stage Development of AI Tasks: Balancing Accuracy and Model Size. Top (6.5a). The four stages of development of an AI task, illustrating the evolving trade-off between accuracy and model size over time. Middle (6.5b) An example of these stages for *Image Classification*. Each point represents a model. The size of the orange circle in Stage 4 represents the model usage (number of downloads). Bottom (6.5c): Current development stages of major AI tasks. In Stage 1, the initial foundation models are developed. In Stage 2, researchers focus on improving utility, leading to larger and performing models. In Stage 3, efficiency is introduced and smaller models with similar performance emerge. Last, in Stage 4, users partially adopt energy-efficient models.

to hardware limitations that limit the adoption of large models. Similarly, BERT is very popular for *Text Classification* and *Text Clustering*.

However, the general public has access to very large models through web applications and APIs. This is especially true for the *Text Generation* task, where users have access to large language models with hundreds of billions of parameters. The *LMsys benchmark* in Figure 6.3a shows the very high usage of such models. Notably, models within the GPT-4 family attract over three billion visits per month, according to SimilarWeb (SimilarWeb, 2025), demonstrating the significant impact of these powerful systems.

The task maturity also influences the size of adopted models. Its evolution over time can be divided into 4 main phases (see Figure 6.5). In the first phase, the task is not mature and high marginal gains are experienced with new larger models. During this phase, a large number of users use models with performance very far from state-of-the-art models, as the field is swiftly evolving. A typical example is reasoning with the two benchmarks *Mathematical Reasoning* (see Figure 6.3d) and *Image-Text to Text* (college-level questions), see Figure 6.3n.

In the second phase, we experience the apparition of large performing models which are not yet massively used by the community. *Text Classification* (see Figure 6.3e) and *Text Clustering* (see Figure 6.3f) have very performing and large models, e.g., the state of the art NvEmbed model. However, users are massively using a BERT model, with a utility 30% lower.

In the third phase, the task is mature. The marginal gain is now small. The best-performing model is adopted (if not too large) and the community introduces small efficient models in parallel. *Speech recognition* (= *Audio-to-text*) is typical of such a phase (see Figure 6.3k). A large performing model, WhisperLarge-V2, is mostly used. Small efficient models have been developed, as the energy-efficient model, WhisperBase.

Other tasks are in transition between phases 2 and 3. One example is *Text to Image* (see Figure 6.3m): while high-performing models like Flux-dev have been proposed and gained significant adoption, lower-performing models such as stable-diffusion-xl remain widely used, likely due to adoption barriers. Efficient smaller models like Playground-2.5 and Kolors have emerged but have not yet achieved widespread use.

The last phase corresponds to very mature tasks for which performing and small models have been developed and adopted. The share of usage between energy-efficient and best-performing models depends on model sizes. If the latter are not too big, both models will be used. The typical example is *Image Classification* (see Figure 6.3g). For this task, efficient models, such as MobileNetV3 and ViT-T, developed by Google, are widely adopted by users.

The maturity level of each AI task, the size of its models, their adoption pattern, the existence or not of small and efficient models will thus have a direct impact on how much energy savings can be achieved through model selection. In the next section, we explore the potential energy savings of each of these tasks using model selection.

6.3 Estimating the savings of AI model selection

This section quantifies the energy savings achievable through model selection. We first estimate the energy consumption of the benchmarked AI models and then analyze the energy reductions resulting from applying model selection techniques.

6.3.1 AI inference energy consumption measurement methodology

Precise measurements of the energy consumption are crucial for selecting energy-efficient models during inference. Several works (A. S. Luccioni, Viguier, & Ligozat, 2023 ; S. Luccioni et al., 2024 ; J. Yang et al., 2024 ; Dodge et al., 2022) have proposed the energy monitoring of AI models in order to identify opportunities for enhancing energy efficiency. These works usually use specialized software-based tools for measuring the models energy consumption, such as CarbonTracker (Anthony, Kanding, & Selvan, 2020), CodeCarbon (Schmidt et al., 2021), Zeus (You, Chung, & Chowdhury, 2023), Scaphandre (Petit, 2023) and powerAPI (Fieni, Acero, Rust, & Rouvoy, 2024). We opted for *CarbonTracker*, which monitors CPU and GPU energy consumption using built-in sensors, since it is well documented and widely used by researchers.

In this section, we present a methodology for assessing the energy consumption of AI models. In Section 6.3.1.1, we describe the *Carbon Tracker* tool used to measure the inference energy consumption of the evaluated models. In Section 6.3.1.2, we validate the tool by comparing its measurements with those of a hardware-based power meter and, show its reliability in our experimental setup.

Given the large number of models per AI task, it was impractical to measure the energy consumption of each one directly. In addition, some models were too large to run. Therefore, for each task, we measured the energy consumption of energy-efficient and best-performing models when feasible. For the remaining models, we estimated their energy consumption using a power-law function, as suggested in the literature (Desislavov et al., 2023). In Section 6.3.1.3, we validate this approach by running models belonging to three different domains: vision, audio, and language, and showing a linear relationship between energy and number of parameters on a log-log scale.

Finally, in Section 6.3.1.4, we describe the methodology used for estimating the AI models energy consumption.

6.3.1.1 Carbon Tracker tool

CarbonTracker (Anthony et al., 2020) is a software-based energy measuring tool designed for predicting the energy consumption and carbon footprint of training deep learning models.

It is implemented in python and it measures the GPU and CPU. For this, it utilizes the energy measuring interfaces `nvidia nvml` and `intel rapl`, respectively. Further information is described below.

Nvidia Management Library (NVML) (NVIDIA, 2022) is an interface which allows managing Nvidia graphical processing units (GPU). The interface allows measuring the power consumption of the GPU with the command `nvml Device Get Power Usage`, which returns the power usage for this GPU in milliwatts, with a 5% of accuracy.

intel rapl (Khan, Hirki, Niemi, Nurminen, & Ou, 2018) is an interface which allows users to estimate, monitor and manage the energy consumption in Intel processors. Intel RAPL provides two main functionalities: (i) energy consumption measurement with a high sampling rate and (ii) limiting the maximal power consumption across the processing domains.

The tool is widely used by researchers (Geißler, Zhou, Liu, Suh, & Lukowicz, 2024; Douwes, Esling, & Briot, 2021; Bannour et al., 2021; Rodriguez, Degioanni, Kameni, Vidal, & Neglia, 2024), is well documented, and provides a user-friendly interface.

6.3.1.2 Experiment 1: Validation of software-based tools for measuring AI energy consumption

Here, we validate the software-based power measurement tool *CarbonTracker* by comparing its measurements with those of *PowerSpy* (ALCIOM, s. d.), a hardware-based power meter that can assess the power consumption of a computer.

Hardware configuration. During our experiments, we used system equipped with an Intel Xeon Gold 6230R CPU and two NVIDIA RTX A5000 GPUs, running Ubuntu 24.04 as the operating system. Additionally, we used the power meter *PowerSpy2* (ALCIOM, s. d.), by Alciom.

Tasks and models. In our experiments, we measured the inference energy consumption by models across six tasks: *object detection*, *text generation*, *speech recognition*, *text classification*, *text to image* and *image-text to text*.

For each task, we selected models evaluated in the benchmarks available on the Hugging Face platform, as these models can be easily retrieved from there. For each task, we evaluated models of different sizes. However, for some tasks, we were unable to run the largest models on our hardware. Below, for each task, we list the benchmark, the number of models we ran, and the size (in terms of number of parameters) of the largest model we were able to run.

1. **Text generation:** OpenLLM Leaderboard. 11 models. Largest model: 1.6B of parameters.
2. **Object detection:** Open Object Detection Leaderboard. 11 models. Largest model: 48M of parameters.
3. **Speech recognition:** Open ASR Leaderboard. 17 models. Largest model: 1.5B of parameters.
4. **Image-text to text:** MMMU. 4 models. Largest model: 4.2B of parameters.
5. **Text to image:** GenAI. 3 models. Largest model: 3.4B of parameters.
6. **Text Classification:** MTEB Leaderboard. 8 models. Largest model: 17M of parameters.

In total, 54 models were evaluated. Each model was executed 10 times in order to obtain consistent results.

Input data. In our experiments, we used the following datasets:

1. **Text generation:** An input prompt containing the following command: “Write a 2000-token story about a futuristic world where AI and humans coexist.”.
2. **Object detection:** A set of 100 images from Coco 2017 dataset.
3. **Speech recognition:** A set containing 100 samples from *LibriSpeech* dataset.
4. **Image-text to text:** Two samples (questions) from MMMU PRO dataset.
5. **Text to image:** An input prompt with the command “Astronaut in a jungle, cold color palette, muted colors, detailed, 8k”.
6. **Text Classification:** A set containing 10000 samples from *imdb* dataset.

Power meter measurement adjustment. Since the power meter measures total system power consumption, including fans, we took idle power consumption into account. We first measured the average power consumption of the system in an idle state (no models running) for 15 seconds. Then, during model execution, we subtracted this idle power from the total power measured by *PowerSpy*, following the methodology of Rodriguez et al. (Rodriguez et al., 2024).

Our energy measurements, comparing *CarbonTracker* and *PowerSpy* (Fig. 6.6), revealed a small average difference of 3.42% for the software-based *CarbonTracker* compared to *PowerSpy* (90% of measurements below 8%, and all below 13%). This close agreement with the external power meter (*PowerSpy*) establishes *CarbonTracker* as a reliable tool for energy measurement in our experiments.

6.3.1.3 Experiment 2: Energy consumption and number of parameters

Here, we investigate the relationship between model size (in number of parameters) and energy consumption across three AI tasks: image classification, speech recognition, and text generation. The energy consumption was measured using *CarbonTracker*

Hardware configuration. During our experiments, we used a system equipped with a Dell R7525 dual-AMD EPYC 7413 and Nvidia A40 GPU card.

Models. During our experiment, we performed inference requests over 241 models, according to the following division:

- Image classification (vision): 213 models. The models were extracted from *PyTorch Images (timm) benchmark*
- Speech recognition (audio): 17 models. Extracted from *Open ASR leaderboard*.
- Text generation (language): 14 models. Extracted from *llm-perf benchmark*.

Each model was performed 10 times.

Input data. In our experiments, we used the following input data for performing the inference tasks.

- Image classification (vision): A set of 100 images from ImageNet dataset.
- Speech recognition (audio): A set containing 50 audio samples from Librispeech dataset.
- Text generation (language): An input prompt containing the following command: *"Write a 50-token story about a futuristic world where AI and humans coexist."*

Figure 6.7 shows the median energy consumption over 10 inference requests per model as a function of model size. Across all tasks, energy consumption scales linearly with model size on a log-log scale. Then, we can approximate by a polynomial relationship on the linear scale.

This result is consistent with Desislavov et al. (Desislavov et al., 2023), who reported a linear relationship between the number of parameters and floating-point operations on a log-log scale for vision models, and a linear scaling of energy consumption with the number of operations.

6.3.1.4 AI models energy estimation methodology

Thus, model size can be used for estimating inference energy consumption when direct measurement is impractical, e.g., when the model is too large.

To estimate the inference energy consumption of a model when direct measurement was not feasible, we employed a regression-based approach based on empirical energy measurements. Our methodology involved evaluating the two key models across several AI tasks.

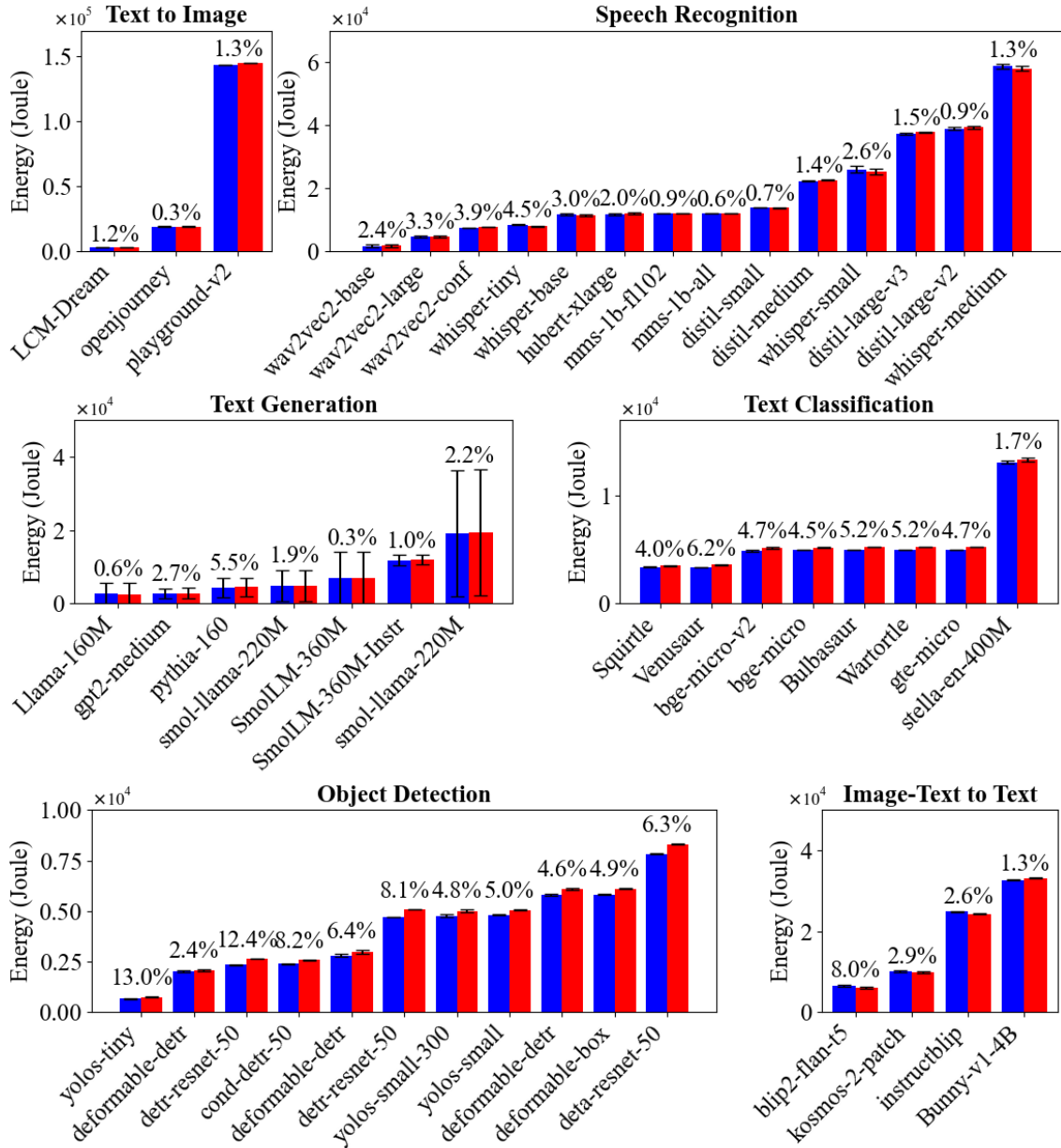


Figure 6.6 – **Validation of the *CabonTracker* tool.** Comparison of energy consumption measurements obtained using the software-based tool *CabonTracker* (CT) and the *PowerSpy* power meter device, across models from several tasks. In blue, we show the *CabonTracker* measurements and in red, the *PowerSpy* measurements.

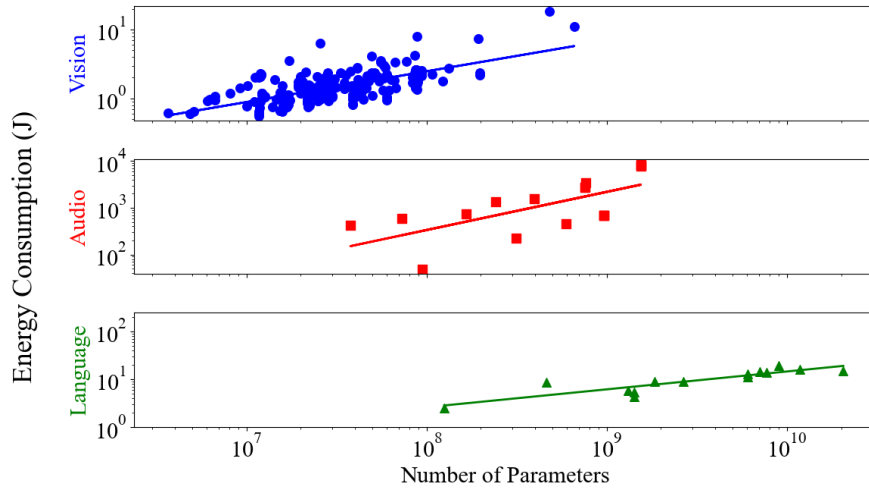


Figure 6.7 – **Modeling AI model energy consumption.** Inference energy consumption versus model size (number of parameters) for *Image Classification* (vision), *Speech Recognition* (audio), and *Text Generation* (language) tasks. Median energy values from 10 repeated inference requests are plotted for each data point.

Hardware Configuration. Experiments were conducted on a Dell R7525 system equipped with a dual-AMD EPYC 7413 processor and an Nvidia A40 GPU.

Tasks and input data. We considered the following tasks in our experiments: *text generation*, *image classification*, *object detection*, *speech recognition*, *text-to-image generation*, *image-text-to-text generation*, *text classification*, *translation*, *image segmentation*, *audio classification*, and *time series forecasting*, as described in Table 6.2.

Some task labels, specifically *text clustering*, *mathematical reasoning*, and *code generation*, are not explicitly available in *Hugging Face*. Therefore, we approximated these as follows: *text clustering* was mapped to *text classification*, while *mathematical reasoning* and *code generation* were treated as *text generation* tasks.

For each task, we used the following data to run the inference requests. The data come from the datasets used in the considered *Hugging Face* and *Paper with codes* benchmarks, except for *Text Generation*, which uses a code example for one of the models.

1. **Image classification:** A set of 500 images of CIFAR-10 dataset.
2. **Object detection:** A set of 100 images from Coco 2017 dataset.
3. **Speech recognition:** A set containing 100 samples from *LibriSpeech* dataset.
4. **Image-text to text:** A single sample (question) from MMMU PRO dataset.
5. **Text to image:** An input prompt with the command "Astronaut in a jungle, cold color palette, muted colors, detailed, 8k".
6. **Translation:** An input prompt command for translating from english to german the following sentence: *This is a beautiful word*.
7. **Image segmentation:** A sample containing one image from *Coco 2017 dataset*.
8. **Audio classification:** A set containing 10 samples from *LibriSpeech* dataset.

9. **Time series forecasting:** A set containing 17420 entries from *ethtl* dataset.
10. **Text Classification:** A set containing 10000 samples from *imdb* dataset.
11. **Text generation:** An input prompt containing the following command: “Write a 50-token story about a futuristic world where AI and humans coexist.”.

Energy Measurement and Regression Analysis. For each task, we used the *Carbon Tracker* tool to measure the inference energy consumption of the two key models: the *energy-efficient* model and the *best-performing* model. Each model was evaluated over five runs.

Using the collected energy consumption data, we performed a linear regression on a log-log scale, following the pattern described in Section 6.3.1.3. The regression model follows:

$$\log_{10}(E) = \alpha \cdot \log_{10}(P) + \beta,$$

where E represents the inference energy consumption and P denotes the number of model parameters. The coefficients α and β were estimated for each task (α ranges from 0.27 to 0.84 with a median value of 0.59, when β ranges from -5.89 to 1.4 with a median value of -2.56).

With these coefficients, the energy consumption of other models within the same task can be estimated using:

$$E = 10^\beta \cdot P^\alpha.$$

The formula enables the estimation of the energy consumed by models other than those directly measured.

6.3.2 Savings by model selection

Equipped with our energy measurements and estimates, we are now able to evaluate how much energy can be saved through model selection for all of the AI tasks under consideration.

We first discuss our choice of the *energy-efficient* (green) model for each task by comparing it to the *best-performing* available (red) model for that task. The methodology used for selecting the key models is described in Section 6.2.2.4 and the models are listed in Table 6.3. Figure 6.8 shows the impact of switching from the *best-performing* model to the *energy-efficient* model for each task. On average, the latter is 65.8% more energy efficient than the former, at the small cost of losing only 3.9% of utility. If we distinguish between tasks, the savings are around 70% or more for mature tasks such as *Time Series Forecasting* (92.8%), *Speech Recognition* (80.6%) or *Image Classification* (69.6%), while they are lower for immature tasks such as *Mathematical Reasoning* (36%) and *Text-to-Image* (image generation) (only 18.3%). The large savings with limited impact on the utility validate the choice of models.

Users may have different tolerances for utility loss depending on their specific needs. To account for this, we evaluated global energy savings and utility variations under different maximum allowed utility drop values when selecting the *energy-efficient* model. The selection criterion for selecting the *energy-efficient* model is based on maximizing the ratio between utility and model size (number of parameters), while ensuring that the global utility drop remains below the specified threshold. Thus, the model selected may change under different maximum utility drop values.

Figure 6.9a illustrates how the choice of the *energy-efficient* model evolves as the maximum allowed utility drop increases. When the allowed utility drop is small, the selected model remains close in performance and energy consumption to the *best-performing* model,

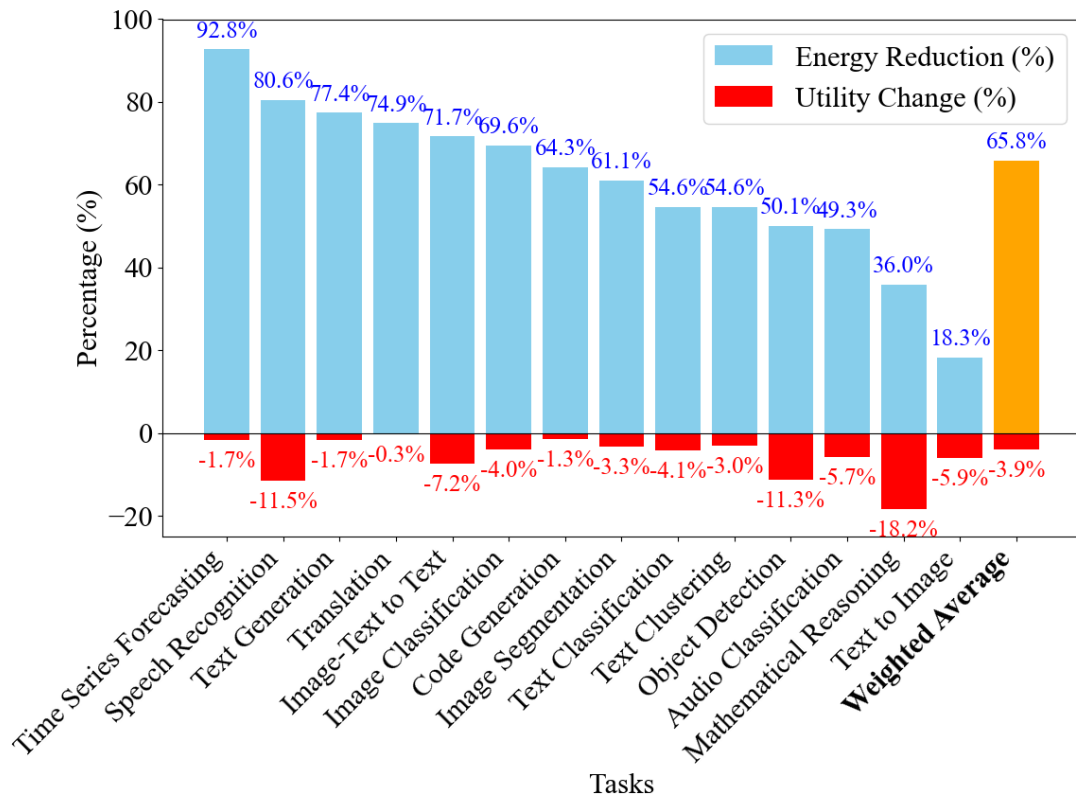


Figure 6.8 – Energy and Utility Impact of Switching to Efficient Inference Models. This figure presents the percentage difference in energy reduction and utility when shifting AI inference from the best-performing model to the energy-efficient model for various tasks. The right-most bars illustrate the weighted average across all considered AI tasks, with weights based on their usage (number of downloads). Notably, these values also represent the energy savings achieved by adopting model selection (as detailed in Figure 6.11) compared to a pessimistic scenario where all inferences are performed using the best-performing models on high-performance hardware.

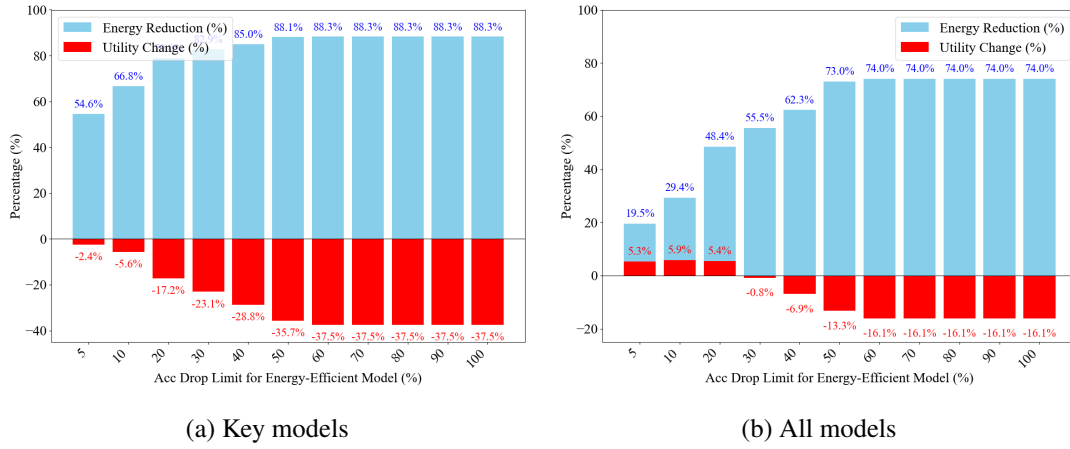


Figure 6.9 – **Trade-off: Energy Savings vs. Maximum Utility Drops.** Global estimated energy savings (%) and utility variation (%) under different maximum utility drop thresholds when choosing the energy-efficient model. Results are shown (a) considering only the two key models (best-performing and energy-efficient), and (b) considering all models. For each threshold, the presented value is a download-weighted average across all tasks.

resulting in limited energy savings. As the maximum allowed utility drop increases, models with lower energy consumption can be selected, resulting in greater energy savings. Beyond a certain point, however, the energy savings reach a plateau, indicating that the most efficient model has already been selected. Our estimations indicate that global energy savings can reach up to 88.3% when a maximum global utility loss of 37.5%.

We now take into account the models usage. We consider a scenario in which all users using a larger model than the energy-efficient (green) model of the task switch to the later model. We examine the energy savings, as well as the impact on utility. Figure 6.10 illustrates the estimated percentage reduction in energy consumption for each task and the average reduction for all the tasks weighted by the total number of downloads per task on *Hugging Face*. **Our findings indicate that model selection applied for AI inference results in a 27.8% reduction in energy consumption.** Among the evaluated tasks, two tasks present a very high potential energy saving, over than 80%: *time series forecasting* and *speech recognition*. For these tasks, the most commonly used models are usually large and power-hungry. Thus, the model selection and the AI inference requests redirecting towards energy-efficient models represent a substantial energy reduction (see Figure 6.3). Although these tasks are mature, the transition to small and efficient models has not yet been made (still in phase 3). We also observe that other tasks with popular large models (e.g., T5-11b for *Translation* and FLUX-dev for *Text-to-Image*) show significant energy savings, over 40%.

Furthermore, Figure 6.10 presents the estimated utility variation in percentage for each task. The largest observed utility decrease was 5.7% for *speech recognition*. **On average, however, utility increased by 4%.** Although this result may seem counter-intuitive, this is expected, since some widely used models are actually bigger and less performing than the energy-efficient model (due to reasons presented in Section 6.2.4 discussing model adoption). Thus, switching to the energy-efficient model can sometimes lead to an improvement in utility. This finding aligns with previous discussions by Abio (Abio, 2023) and Varoquaux et al. (Varoquaux, Luccioni, &

Whittaker, 2024), who challenge the "bigger-is-better" paradigm, arguing that smaller models can perform as well as—or even better than—larger models.

When evaluating energy savings under different utility drop values for choosing `energy-efficient` model, the results (see Figure 6.9b) point that the user can save up to 74.0% of energy consumption by applying model selection with a maximal global utility loss of 16.1%.

Model selection can thus be an efficient method to reduce the impact of AI, while not affecting its global utility. However, several scenarios are possible for the future. We present the estimated energy consumption for AI inference in U.S. data centers for three of them in Figure 6.11. The historical data was retrieved from the U.S. Data Centers report, assuming an inference ratio of 60% (Shehabi et al., 2024). The figure presents three possible future scenarios for energy consumption: (i) a *business-as-usual scenario*, reflecting the natural growth of AI and data center usage; (ii) a *pessimistic scenario*, assuming widespread deployment of the `best-performing` (and often large) models due to hardware evolution; and (iii) a *sobriety scenario*, where model selection redirects inference requests to the `energy-efficient` model. We considered a transition period of a year, 2025-2026, to transition to Scenarios (ii) and (iii). Thus, from 2026, we recompute the energy consumption as if all the used models were `best-performing` or `energy-efficient`. For each scenario, we consider both an upper bound and a lower bound on the estimated energy consumption, as reported in Shehabi et al. (2024). Our projections indicate that **using large and power-hungry models, in a pessimistic scenario, could increase energy consumption by 111.2%**. On the other hand, **adopting model selection** would enhance energy sobriety, saving 27.8%.

For U.S. data centers, we estimate that model selection could save 16.25 TWh in 2025, equivalent to the annual energy production of two nuclear power reactors. **By 2028, these savings could reach 41.8, corresponding to the annual production of seven nuclear power reactors.**

Expanding our analysis to global data centers based on estimations from SemiAnalysis study (Patel, Nishball, & Ontiveros, 2024), **we project energy savings from model selection of 31.9 TWh in 2025 and 106 TWh in 2028 - equivalent to the annual production of 5 and 17 nuclear power reactors, respectively.** These findings highlight the potential of model selection for achieving energy sobriety during AI inference at scale.

6.4 Conclusion

Given the widespread adoption of AI and the significant energy consumption of many popular models, in this work, we investigated how model selection can contribute to energy sobriety. Specifically, model selection aims to identify energy-efficient models that (i) have a small size, thus reducing energy consumption, and (ii) maintain high utility, thus ensuring good performance.

To achieve this, we first analyzed model collection platforms such as *Hugging Face* and *Papers with Code* to identify the most commonly used models in the developer community. We then analyzed the utility-size trade-off across several tasks and observed a diminishing marginal gain that can be exploited by model selection to reduce overall energy consumption.

Based on this pattern, we estimated the potential energy gain from selecting energy-efficient models for inference tasks. Our estimates suggest a reduction in energy consumption of 27.8%. In the United States, this reduction is equivalent to the annual production of two nuclear power reactors.

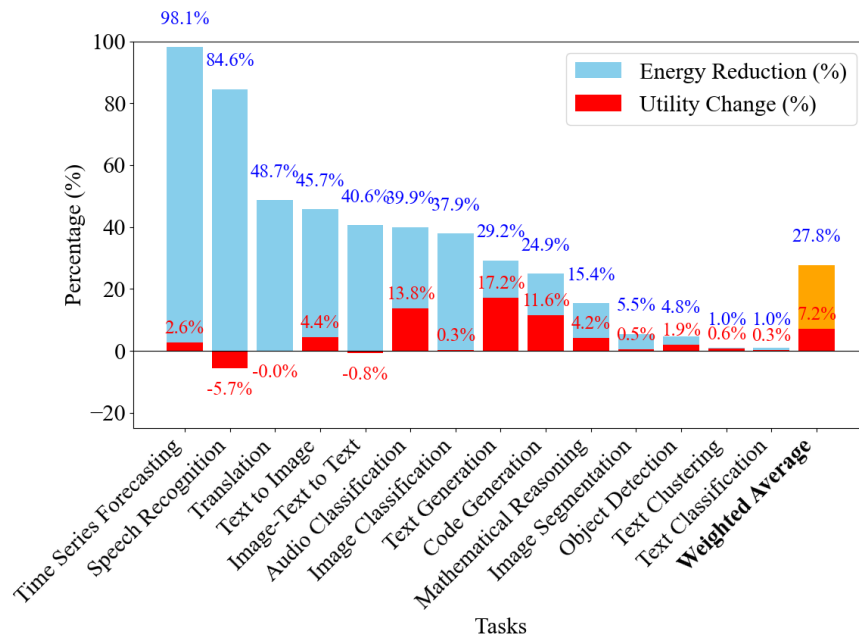


Figure 6.10 – Estimated Impact of Efficient Model Selection on AI Inference. Estimated percentage reduction in energy consumption and the percentage variation in utility for major AI tasks, when applying model selection during inference. For each task, inferences with models larger than the identified energy-efficient model are redirected to it. The rightmost bar (i.e., the bar in orange and in red) represents the weighted average across all considered AI tasks weighted by their usage (number of downloads).

As the urgency of climate change grows, the adoption of sustainable AI practices is critical. We believe that the responsible and efficient use of AI plays a key role in mitigating environmental impact. As a future direction, we aim to expand our analysis to consider the entire life cycle of AI models, from data collection to deployment.

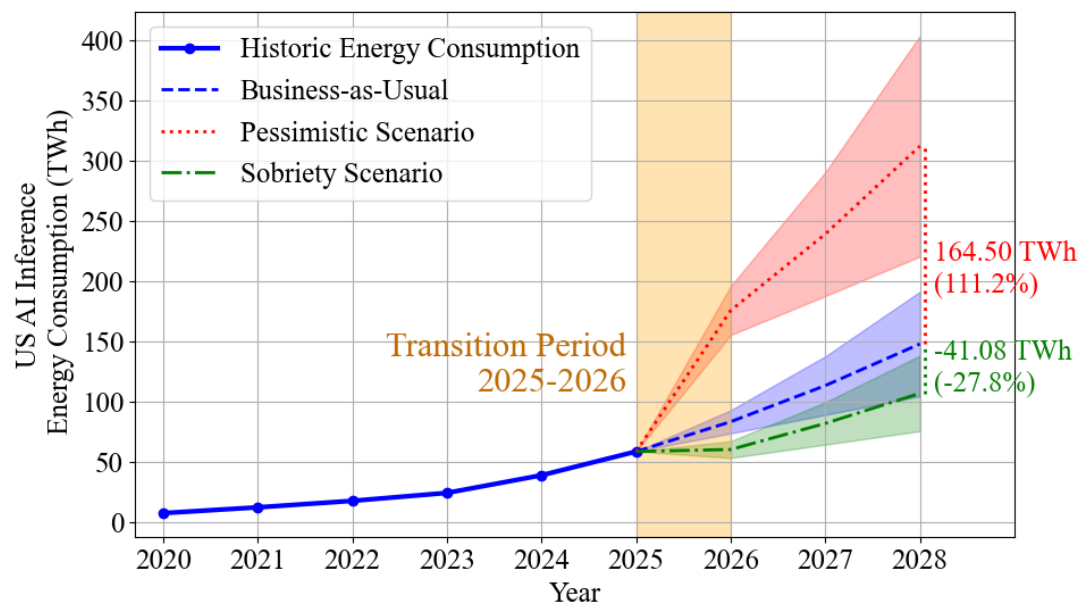


Figure 6.11 – **Projected US Data Center Energy Consumption for AI Inference Under 3 Different Scenarios.** US data centers AI inference energy projection over time for three different scenarios: (i) business as usual, (ii) sobriety using model selection, (iii) use of best-performingmodel with the needed hardware. The values for (i) were extracted from (Shehabi et al., 2024). We considered a one-year transition to reach scenario (ii), applying model selection globally, or (iii), using the best-performingmodel, (during the year 2025).

Evaluating an AI-based application environmental impacts through a Life Cycle Assessment

Given the rapid growth of mobile applications and AI-based models, it is essential to assess their associated environmental impacts. Digital services contribute to various ecological issues, such as metal extraction, energy consumption and greenhouse gas emissions, all of which pose significant environmental challenges. Furthermore, the continuous hardware evolution may amplify these impacts. It is therefore crucial to evaluate not only the environmental effects of service usage but also those associated with equipment manufacturing.

Then, in this chapter, we conduct a Life Cycle Assessment (LCA) on Pl@ntNet, an AI-based vision application used worldwide for plant species identification. We propose to estimate the embodied and operational environmental impacts associated with the application. For this, we utilize a multi-stage and multi-criteria methodology that analyzes five sections of the Pl@ntNet architecture: End Users, Networks, Training, Inference, and Web Hosting.

Our results reveal that Pl@ntnet was responsible for 78.4tCO₂eq emissions in 2024, equivalent to 44 round-trip flights between Paris and New York. The majority of the emissions come from the manufacturing of End Users equipment. These findings highlight the importance of enhancing resource efficiency and promoting responsible technology use to mitigate AI environmental footprint.

7.1	Introduction	115
7.2	PI@ntNet Service	116
7.2.1	Identification Request	117
7.2.2	PI@ntNet Infrastructure	117
7.2.2.1	Training	117
7.2.2.2	Inference	118
7.2.2.3	Webhosting	118
7.3	Related Works	119
7.4	Methodology	121
7.4.1	Study scope and perimeter considered	121
7.4.2	LCA modeling	123
7.4.2.1	End User Terminals	123
7.4.2.2	Network	125
7.4.2.3	Training	125
7.4.2.4	Inference	126
7.4.2.5	Webhosting	127
7.4.3	Parameters and sources	129
7.5	Results	130
7.5.1	Global environmental impacts	131
7.5.2	Detailed assessment of PI@ntNet impacts	132
7.5.2.1	End Users	132
7.5.2.2	Networks	134
7.5.2.3	Training	134
7.5.2.4	Inference	136
7.5.2.5	Webhosting	137
7.6	Conclusion	139

7.1 Introduction

In recent years, the mobile app market has experienced exponential growth, in terms of both users and revenue, and this trend shows no signs of slowing. According to the annual State of Mobile 2025 report ([Sensor Tower, 2025](#)), produced by SensorTower, in 2024, the mobile apps market generated approximately 150 billion dollars in global consumer spending on in-app purchases and subscriptions (with an increasing rate of +13% per year), achieving an annual 136 billion of annual number of downloads. These impressive results underscore the importance of mobile apps in today's digital economy. For businesses and developers, this represents a considerable opportunity, but also significant challenges.

The integration of technologies such as Augmented Reality (AR) and Artificial Intelligence (AI) into mobile apps is a notable trend. These technologies offer enriched and personalized user experiences, such as augmented reality filters in social networks or AI-powered virtual assistants, experiencing a significant growth in the last years. The AI adoption is expected to grow exponentially in the coming years. Consumers have spent nearly 1.1 billion dollars (+200% year-on-year) on applications such as *ChatGPT*, *Google Gemini*, and *Doubao* and the time spent on *CharacterAI* or *ChatGPT* has reached 7.7 billion hours. ([Sensor Tower, 2025](#))

In view of these figures and emerging trends, sustainable development and environmental impact have become key issues for stakeholders for guiding eco-design strategies for digital services. Artificial Intelligence is responsible for a wide range of environmental impacts. Some important environmental impacts of AI usage include *water consumption*, *human toxicity*, and the use of abiotic natural resources, such as *aluminium*, *cobalt*, *copper*, *glass*, *gold*, *tin*, *lithium*, *zinc*, and *plastic*. ([Gupta et al., 2021](#) ; [Bannour et al., 2021](#)).

The multiple impacts associated with the computing industry may occur at different phases of ML-based services development. To evaluate them, researchers have tapped into the Life Cycle Assessment (LCA) methodology ([ADEME, 2025b](#)). The resulting research field is to investigate the societal and environmental impacts of computing systems at different stages of the product life cycle.

In literature, several works ([Gupta et al., 2021](#) ; [A. S. Luccioni et al., 2023](#) ; [Morand, Ligozat, & Névéol, 2024](#) ; [Berthelot, Caron, Jay, & Lefèvre, 2024](#) ; [Berthelot, Caron, De Laage, Lefèvre, & Nicolas, 2024](#)) were proposed to investigate digital services life cycle impacts. Researchers focus their analysis on two types of impacts: (i) *Embodied impacts*, corresponding to the impacts associated to the hardware *production*, *transport* and *end of life*; and (ii) *Operational impacts*, corresponding impacts associated to the computing systems usage (idle and dynamic), mostly related to the energy consumption for powering the computing devices.

In this study, we focus our investigation on PI@ntNet ([Joly et al., 2014](#)), a citizen science platform that utilizes Artificial Intelligence (AI) models to identify plant species. PI@ntNet provides a large-scale infrastructure, achieving millions of users in over 200 countries. At this stage of deployment, with years of experience in developing the platform, PI@ntNet is now committed to understanding the environmental impacts of the identification service and contributing to the search for opportunities to reduce them.

We conduct a Life Cycle Assessment of the PI@ntNet service. Our investigations assess the associated environmental impacts, considering the 2024 perimeter. We propose a multi-criteria and a multi-stage approach. We divide the PI@ntNet service into multiple sections: *End Users*, *Networks*, *Training*, *Inference* and *Webhosting*. For each section, we assess the *Abiotic Deple-*

tion Potential (ADP), the *Global Warming Potential* (GWP) and the *Primary Energy* (PE) for the *embodied* and *operational* impacts.

Our findings point to the PI@ntNet application being responsible for emitting 78.4 tCO₂eq in 2024. The *end users* are responsible for a significant share of the above emissions, equivalent to 62% of the global impacts. Our contributions may be summarized as follows:

- We conduct an LCA on the PI@ntNet application in 2024. We base our methodology on (Berthelot, Caron, Jay, & Lefèvre, 2024), which assesses multiple criteria for environmental impacts.
- We investigate multiple sections concerning PI@ntNet service: *end users*, *networks*, *training*, *inference* and *webhosting*. For each section, we use the associated equipment statistics for estimating the environmental impacts.
- We derive the *embodied* and *operational* impacts, comparing, thus, the multiple phases of the service life cycle.
- We quantify the environmental impacts by identifying the application parts that contribute most to the global impacts and highlighting optimization opportunities.

This chapter is structured as follows. In Section 7.2, we describe the PI@ntNet application and the identification request path, from the moment the user takes the picture until the results are displayed. In Section 7.3, we perform a literature review of Life Cycle Assessment (LCA) applied to AI-based services. In Section 7.4, we present our methodology. We describe the model used to estimate the embodied and operational environmental impacts of each section in PI@ntNet service. Then, in Section 7.5, we present the results of our analysis. Finally, in Section 7.6, we draw the main conclusions and outline the key future perspectives.

7.2 PI@ntNet Service

The aim of the PI@ntNet project is to enable both novices and experienced naturalists to identify and list the plants they encounter in the field, and to share their observations.

Launched in 2009, PI@ntNet is a scientific project bringing together the French National Institute for Research in Computer Science and Control (INRIA), the French Institute for Development Research (IRD), the Centre for International Cooperation in Agronomic Research for Development (CIRAD), the French National Institute for Research in Agriculture, Food, and the Environment (INRAE) and the Tela Botanica network.

The initial aim of the project was to inventory biodiversity for scientific research using image identification. In 2011, the project was transformed into participatory science with the launch of a collaborative web platform, followed by a smartphone application in 2013.

Specialists, amateurs and the curious connect in a collaborative network. The most qualified users contribute to the most expert knowledge and validate observations. In April 2025, the application was able to identify 75,200 species, had 7.8 million registered users and offered 32 million observable images. In addition to plant identification, which remains the main use of the PI@ntNet service, the platform offers a geolocation system to map the species listed, a community network and specific services dedicated to professional structures (private, public and scientific research).

Since 2020, PI@ntNet data has been integrated into the Global Biodiversity Information Facility (GBIF), helping to mobilize and use data on biodiversity trends, as well as promoting cooperation and knowledge sharing.

The platform is now one of the world's largest biodiversity observatories, with over 200 countries represented and several million contributors. Table 7.2 describes the top-10 countries with the most usage and their respective ratio of the total sessions using PI@ntNet. The application is available in 52 languages thanks to the work of the developer and translator communities.

Several thousand scientific publications have utilized PI@ntNet data, and the project is frequently incorporated into other global research initiatives and programs.

7.2.1 Identification Request

The object of our study will be linked to the main service of PI@ntNet : the recognition of a plant or species from a photograph transmitted via the mobile application. When the user requests a plant identification, the request goes through different steps for obtaining the results. The request path is described in Fig. 7.1. First, the user opens the application and takes a picture of the plant, which will serve as input for identifying the species. Then, the picture is compressed and transmitted to the CIRAD data center, where it will be processed. The request is forwarded by the API and stored by the PI@ntnet database system. Then, the identification is transferred to the identification servers, where the ML model performs an image classification task on a GPU card, outputting the plant species. Then, a similarity search is performed in order to get the most likely species and finally, the API transmits the results back to the user. The results consists of a set containing the most likely species of the plant identified, together with some small images of each of the plant possibilities.

Each image in the database is represented by a number of visual attributes that distinguish one species from another. To select the most discriminating attributes, the AI based techniques Visual Transformers (B. Wu et al., 2020) are utilized. They can characterize both fairly general traits, such as flower color or the general shape of the leaf, as well as subtle details, such as the shape of the flower petal tips or the presence of slight hairiness on the leaf margins.

The ML models utilize all the above characteristics to distinguish between different species when identifying a plant species based on a picture. The performance of the ML models used by PI@ntNet is strongly correlated with the model training, which is done in Jan Zay supercomputer using the PI@ntNet dataset.

7.2.2 PI@ntNet Infrastructure

Here, we provide the infrastructure used for hosting the PI@ntnet service. The infrastructure used covers three sections: training, inference and webhosting.

7.2.2.1 Training

During the training, PI@ntNet utilizes the Jean Zay supercomputer, equipped with Nvidia Tesla V100 and Nvidia A100 graphical cards, for performing the training. For the node containing V100 graphical card, we considered the v100-32g node, as it is the most common node in Jean Zay. The node specification provides: 2 processors Intel Cascade Lake 6248, 192 GB RAM, 4 GPU Nvidia Tesla V100 SXM2 32 GB.

For the node containing A100 graphical card, we considered the p5 partition, which contains: 2 processors AMD Milan EPYC 7543, 512 GB RAM, 8 GPU Nvidia A100 SXM4 80 GB.

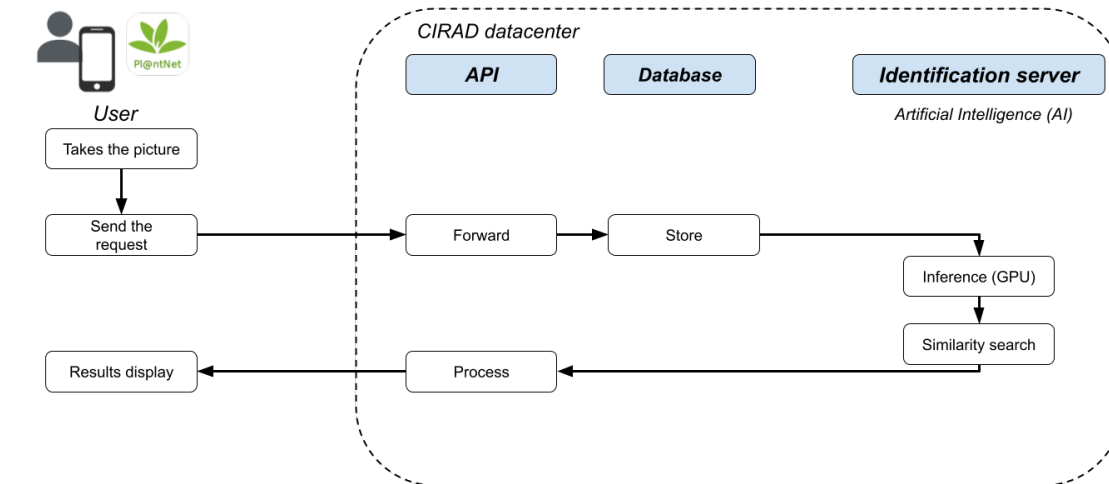


Figure 7.1 – Plant identification path using PI@ntnet service: a user makes an identification request, which is forwarded to PI@ntnet datacenter; the request is saved on the database and the ML model inference is executed on a GPU card; then, the results are processed and returned to the end user.

7.2.2.2 Inference

The inference requests using PI@ntnet service are performed in the CIRAD datacenter. A total of 8 fully dedicated machines are used for inference, named as follows: Capcir, Orxois, Chamois, Devois, Rajol, Huchet, Senart, and Sourdun. The servers present the following specifications:

- The servers CAPCIR and ORXOIS are equipped with 2 Intel Gold 5118 CPUs, an Nvidia Quadro M4000 GPU, 62 GB of RAM, and an HDD with a storage capacity of 2 TB.
- The servers RAJOL and HUCHET present 2 CPU Intel Gold 5118, a GPU Nvidia RTX 8000, 92 GB of RAM, and an SSD with 4TB of storage capacity.
- The servers CHAMOIS, DEVOIS, SENART, and SOURDUN are equipped with 2 Intel Gold 5118 CPUs, an Nvidia A40-48GB GPU, 128 GB of RAM, and an SSD with a 4TB storage capacity.

7.2.2.3 Webhosting

The PI@ntNet *webhosting* section is deployed in the CIRAD data center. The infrastructure may be divided into two parts. The API service is deployed through Virtual Machines (VMs). The physical infrastructure used for this section consists of 12 blades, each containing the following specifications: 2 Intel Gold 5118 CPUs and 512 GB of RAM.

Since the CIRAD cluster does not offer a power meter that provides measures of power consumption, we estimated the power by considering a similar server on the Grid5000 platform (Balouek et al., 2013). We selected the node *roazhon-9*, which contains 4 instances with the following specifications: 2 Intel Xeon Gold 5118 and 512 GB RAM. We extracted the average power consumption in *roazhon-9* node for idle and charge loads, as described in Table 7.1.

For the database service, two machines are used by PI@ntNet in this section: (i) *brivadois*, machine provided with 4 CPU Intel Xeon Gold 5118, 256 GB of RAM memory and 7TB of

Node	Idle Power (W)	Charge Power (W)
roazhon-9-1	50	332
roazhon-9-2	56	256
roazhon-9-3	53	253
roazhon-9-4	53	303
Average	53.75	291.25

TABLE 7.1 – Power measurements in the idle and the charge mode in Roazhon-9 node, in Grid 5000 cluster.

Country	Ratio
France	25.1%
Germany	12.0%
USA	11.1%
Italy	6.5 %
United Kingdom	5.0%
Spain	4.1 %
Netherlands	3.8 %
Switzerland	2.5%
Brazil	2.4%
Canada	2.0%

TABLE 7.2 – Top 10 countries using the Pl@ntNet service worldwide, along with their respective share of total sessions.

SSD Storage; and (ii) *gray*, provided with 4 CPU Intel Xeon Gold 5118, 256 GB of RAM memory and 9TB of SSD Storage. Furthermore, the storage section utilizes the storage equipment *mesoPl@ntnet*, provided with a SSD with 270 TB of capacity.

7.3 Related Works

With the increasing adoption of digital services and the growing influence of Artificial Intelligence (AI) applications, several studies have proposed Life Cycle Assessment (LCA) methodologies to evaluate the environmental and societal impacts of AI-based digital services. This research field aims to evaluate computing systems across several phases of their life cycle, from resource extraction to end-of-life processing.

LCA methodologies. In the literature, several methodologies have been proposed to assess environmental impacts associated with different stages in the life cycle of digital services. For instance, [Gupta et al. \(2021\)](#) identifies four key stages in the life cycle of hardware used in mobile devices and data centers:

- *Production*: Raw metals extraction, manufacturing, assembly, and packing.
- *Transport*: Hardware transportation from the factory to the point of Use, such as consumers and data centers.

- *Use*: Impacts associated with hardware operation, which includes dynamic and static operation and the Power Usage Effectiveness (PUE) overhead.
- *End of Life*: Impacts associated to end-of-life processing and recycling of hardware.

Similarly, in (Bannour et al., 2021), the authors also set four stages of ML life cycle: (i) *production*; (ii) *idle Use* (equipment is ON, but no process is running); (iii) *dynamic use* (equipment is ON and processes are running); and (iv) *end of life*.

However, the authors point out some challenges for assessing the carbon impact of some phases. Particularly during the *Production* and *End of Life* stages, the lack of data poses significant challenges to a precise LCA.

Berthelot, Caron, Jay, et Lefèvre (2024) proposes dividing the impacts into two main stages, as following:

- *Embodied (capex)*: It corresponds to the impacts associated to the hardware *production*, *transport* and *end of life*.
- *Operational (opex)*: It corresponds to the impacts associated with the computing systems usage (idle and dynamic), mostly related to the energy consumption for powering the computing devices.

Quantifying ML environmental impacts. Building upon the LCA frameworks described above, several studies have focused on quantifying the environmental footprint of machine learning (ML) applications in real-world settings.

A. S. Luccioni et al. (2023) investigated the carbon footprint of the BigScience Language Open-science Open-access Multilingual (BLOOM) model. BLOOM is a 176 billion parameter language model, developed for text generation. They estimated that, during *training*, the model emitted 24.7 tCO₂eq due to its *dynamic usage*. The embodied emissions accounted for 11.2 tCO₂eq, equivalent to 22.2% of the total footprint.

Morand et al. (2024) proposed the Machine Learning Life Cycle Assessment (MLCA), a methodology for estimating the embodied and operational environmental impacts of ML models during the *training* phase. Their findings indicate that the embodied impacts are becoming an increasingly significant share of the total impacts. The authors highlight that large models require large specialized hardware, which leverages the environmental impacts.

Then, Berthelot, Caron, Jay, et Lefèvre (2024) proposed a Life Cycle Assessment (LCA) for investigating the associated impacts to *Stable Diffusions* service, which offers generating images from text prompt inputs. The authors investigated embodied and operational impacts of different sectors of the service architecture, such as *end users*, *networks*, *training*, and *inference/webhosting*.

Their findings indicate an elevated cost associated with the embodied impacts. They estimated that the manufacturing represented almost the totality of the Abiotic Depletion Potential (ADP) impact and almost 30% of the carbon footprint.

Tools used for assessing environmental impacts Reliable tools are essential for accurately measuring the environmental impacts of AI services. Some tools have been developed to assess the environmental footprint of AI services, for instance, by tracking the energy consumption and carbon footprint of machine learning (ML) models.

These tools, e.g., Carbon Tracker (Anthony et al., 2020), CodeCarbon (Courty et al., 2024), Zeus (You et al., 2023), and Scaphandre (Petit, 2023) track the energy consumption of CPU and GPU cards using Intel RAPL and Nvidia SMI frameworks.

For instance, in (Bannour et al., 2021), the authors investigated tools to evaluate the energy consumption and carbon footprint of Natural Language Processing (NLP) models. However, they assess only the impact associated with the usage phase, which may lead to an underestimation of the total carbon footprint.

Other tools were developed for estimating the embodied impacts of computing services. The BoaVizta tool (Boavizta, 2025) proposes a multicriteria impacts estimation for server manufacturing. It tracks the contribution of computing servers to Abiotic Depletion Potential (ADP), Global Warming Potential (GWP), and Primary Energy (PE) based on processor size and memory capacity.

7.4 Methodology

In this section, we provide the methodology used for evaluating the environmental impacts related to Pl@ntNet service. We first discuss the scope and the metrics assessed by the current study. Then, we provide the model used to estimate the environmental impacts associated with Pl@ntNet service. All notation used for our analysis is described in Table 7.3.

7.4.1 Study scope and perimeter considered

In this study, we estimate the environmental impacts associated with the Pl@ntNet service for identifying plant species. We define the Functional Unit (FU) of the study as follows: ***Provision of the Pl@ntNet plant identification service over a one-year period (2024), on a global scale.***

The temporal scope for inference activities (i.e., actual plant identification requests) is limited to the year 2024. For the *training*, however, we explore three distinct scenarios to account for different temporal scopes, reflecting the cumulative nature of model development:

- *Final Models* scenario: This scenario accounts for the environmental impacts associated with training the final versions of the models that are deployed to users. Based on information from Pl@ntNet, we assume that a new model is trained approximately every two months, each requiring approximately 40 hours of computation on 64 nodes, with each node equipped with four NVIDIA V100 GPUs.
- *Standard* scenario: Here, we consider all model training activities that occurred during the year 2024. This reflects the training footprint within the same reference period as the inference phase.
- *Legacy* scenario: This scenario takes into account all cumulative training activities from 2020 to 2024. This is based on the understanding that the current model is the product of iterative improvements over multiple years. By evaluating the legacy training efforts, we aim to capture the long-term environmental cost embedded in the service’s current capabilities.

LCA Stages. In our analysis, we propose to investigate the *embodied* and *operational* impacts associated with Pl@ntNet service. Fig. 7.2 illustrates the structure of our Life Cycle Assessment (LCA). The impact evaluation may be divided into five main sections: *end users*, *networks*, *training*, *inference* and *webhosting*.

Notation	Description
C	Set of countries
N	Number of identification requests in 2024
U	Number of users using Pl@ntNet in 2024
$alloc_e$	Allocation factor of equipment e for Pl@ntnet
I_{capex}^e	Impact of manufacturing one unit of equipment e
k_e	Ratio of inference requests performed on equipment e
k_c	Ratio of inference requests originated in country c
t_e	Inference time on equipment e , in seconds
P_e	Power of equipment e , in KW
C_e^c	Energy consumption of equipment e in country c , in KWh
L_e	Lifespan of equipment e , in years
u_e	Usage ratio of equipment e
H_e	Training time of Pl@ntnet service in equipment e , in hours
$nGPU_e$	Number of GPUs in equipment e
ρ	Multiplicative factor used
d	Duration of a request using Pl@ntNet, in min
$timePerDay_e$	Total time of equipment e usage in a day, in min
$energyPerYear_e$	Total energy consumption in equipment e in a year, in KWh
T	Types of networks used for transmitting data
k_t	Ratio of type t used given all types
S	Total data exchanged between the user and the server, in Bytes
I^t	Impact of network type t for GB transmitted
$nCPU$	Number of threads on machine
ram	RAM memory capacity, in GB
sto	Storage capacity, in GB
η	Number of blades managed by the hypervisor in physical machine
$a_{vm}^{cpu,ram,sto,others}$	Share of resources (CPU, RAM, storage or others) allocated to vm .
$P_e^{idle,charge}$	Power of equipment e in idle (or charge) mode, in W
$s_{vm,e}^{idle,charge}$	Share of time that vm (or equipment e) is in idle (or charge) mode
EGM_c	Electricity grid Mix (EGM) of country c

TABLE 7.3 – Notation used in this chapter.

Criteria evaluated. Our investigation proposes a multi-phase and multi-criteria approach. We evaluate three metrics indicating the impacts:

- *Abiotic Depletion Potential (ADP)*, assessing the contribution to the depletion of available mineral and metal resources caused by their extraction. This indicator is estimated in kilograms of extracted antimony equivalent (kgSbeq).
- *Global Warming Potential (GWP)*, assessing the contribution to global warming. This indicator is estimated in kilograms of carbon dioxide equivalent emissions (kgCO₂eq).
- *Primary energy use (PE)*, assessing the cumulative demand for primary energy. This is not strictly speaking an environmental impact category, but it is a relevant indicator of resource demand. It is expressed in MegaJoules (MJ).

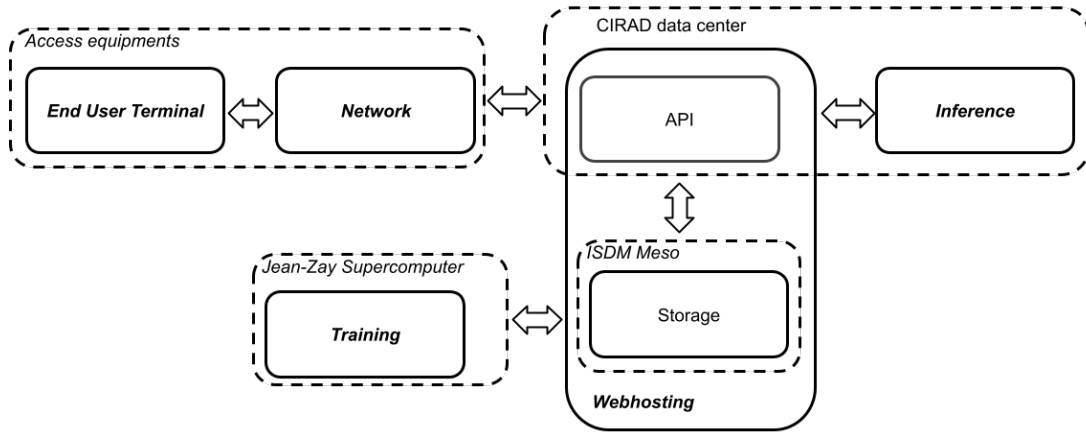


Figure 7.2 – Pl@ntNet service architecture. We divide the service into five sectors: *End Users*, *Network*, *Training*, *Inference*, and *Webhosting* (API and storage).

We have chosen the metrics above due to the alignment with the scientific literature (Berthelot, Caron, Jay, & Lefèvre, 2024) and the existing tools that evaluate environmental impacts (Boavizta, 2025).

Exclusions The following stages and data are excluded from the environmental assessment:

- flows linked to employee transport
- flows linked to administrative and communication services
- construction and maintenance of the building housing the equipment
- spare parts related to equipment maintenance

Most of the data used in the analysis comes from direct measurements and experiments. In the case of work on assumptions and data sensitivity analyses, the study will propose average assumptions.

7.4.2 LCA modeling

In our analysis, we consider the *embodied* and the *operational* impacts associated with five Pl@ntNet architecture sections: *End Users*, *Networks*, *Training*, *Inference*, and *Webhosting*. We describe here the model used to estimate each section of the Pl@ntNet service.

7.4.2.1 End User Terminals

The *End User Terminals* impact refers to the impacts caused by the user's terminal when accessing the Pl@ntnet service. Let N represent the number of identification requests. Each device used to access the service is referred to as equipment e , with a lifespan of L_e years and an average daily usage time of timePerDay_e minutes. A single identification session has an average duration of d minutes.

Embodied Impacts To estimate the end users' embodied impacts, we first calculate, for each equipment e , the allocation factor using the Pl@ntNet service, alloc_e . It corresponds to the time

spent by the PI@ntNet service divided by the total time used by the equipment during a year. For the time spent in PI@ntNet, we consider it to be the product of the session duration d and the total number of requests N . The total time in a year utilizing an equipment e is given by the usage time per day timePerDay_e (in minutes) multiplied by 365 days. The allocation factor computation for each equipment is given by the Equation below.

$$\text{alloc}_e = \frac{d}{365 \times \text{timePerDay}_e \times L_e} \times N$$

Then, we compute the environmental impact for the total number of terminals by computing the sum for all equipment of the product between the impact of manufacturing one equipment I_{capex}^e and the allocation factor alloc_e , as described in the equation below. Note that the notation I_* is used in this section to represent any impact metric, where the superscript and subscript vary depending on the equipment and impact type. These values may correspond to one of the evaluated environmental metrics (ADP in kgSbeq, GWP in kgCO₂eq, or PE in MJ).

$$I_{\text{endUser}}^{\text{capex}} = \sum_e \text{alloc}_e \times I_{\text{capex}}^e$$

Operational Impacts To estimate the operational impacts of terminal usage, we first calculate the equipment power, P_e (in kW), by dividing the equipment's annual energy consumption, denoted as energyPerYear_e , by the total equipment usage time over one year, computed by multiplying timePerDay_e by 365 days. Then, it is divided by 60 to convert to hours. The computation is described in the equation below.

$$P_e(kW) = \frac{\text{energyPerYear}_e}{365 \times \text{timePerDay}_e \times 1/60}$$

Then, we estimate the total energy consumption for equipment e in a country c , denoted as C_e^c (in kWh), during one year due to PI@ntNet usage. The equation below describes the energy computation. The usage time is computed by multiplying the session duration d , the total number of requests N , and the terms k_e and k_c , which refer, respectively, to the proportion of requests allocated to equipment e and originated in country c . Additionally, we divide the time by 60 to convert to hours.

$$C_e^c(kWh) = \left(d \times N \times k_e \times k_c \times \frac{1}{60} \right) \times P_e$$

Finally, we compute the operational environmental impacts by multiplying the energy consumption and Electricity Grid Mix (EGM) for all countries and equipment types. The EGM provides the environmental impact per unit of energy consumed (in kWh), and its value varies by country. It is important to note that the unit of measurement for EGM depends on the specific impact being assessed (e.g., ADP, GWP, or PE). The following equation summarizes the computation of operational impacts.

$$I_{\text{endUser}}^{\text{opex}} = \sum_{c \in C} \sum_{e \in E} C_e^c \times \text{EGM}_c$$

7.4.2.2 Network

For network environmental impacts in Pl@ntNet, the data exchanged between the terminal equipment and the Pl@ntnet server is considered when the user makes an identification request.

When the user requests image identification, it transmits the image to the Pl@ntNet data center. In this process, the image is automatically compressed by the application. Then, after the server processes the request, it retransmits the answer to the user. The data exchanged between the user and the data center has a total volume S (in Bytes). The network link between the end-user terminal and the server may have different types, depending on the equipment used (e.g., mobile networks, Wi-Fi). We denote by T the set of network types used by the end user terminals. We denote the ratio of requests using t as k_t . For each type t , an environmental impact I^t is associated per unit of transmitted data (i.e., byte). The equation below describes the computation of environmental impacts for the *network* section:

$$I_{\text{network}} = \sum_{t \in T} k_t \times S \times I^t$$

7.4.2.3 Training

The training impact is associated with the effects resulting from the model training and development employed by the Pl@ntNet service. We consider that each equipment e used for training contains $nGPU_e$ GPU graphic cards.

Embodied Impacts Here, we describe the estimation of the *embodied* impacts associated with the *training* section. For equipment e , we denote as alloc_e the ratio between the time used for training Pl@ntNet models in equipment e , denoted as H_e (in hours), and the total time used by that equipment. The total usage time of the equipment in a year is estimated by the product of the number of GPUs in the equipment, referred to as $nGPU_e$, the equipment lifespan (in years), L_e , and the utilization ratio, represented by u_e . The following equation describes the computation of the allocation factor.

$$\text{alloc}_e = \frac{H_e}{365 \times 24 \times u_e \times L_e \times nGPU_e}$$

Then, we compute the *embodied* impacts during *training* as the product between the allocation factor and the impact for manufacturing the equipment e , denoted as I_{capex}^e .

$$I_{\text{training}}^{\text{capex}} = \sum_{e \in E} \text{alloc}_e \times I_{\text{capex}}^e$$

Operational Impacts The operational environmental impact refers to the energy consumption that occurs during the training process of machine learning models. A significant portion of this impact comes from the electricity usage of data centers, which includes not only the energy consumed by servers but also that required for supporting infrastructure, such as cooling systems.

To account for the full scope of this energy usage, many studies introduce a multiplicative factor (denoted here as ρ) when estimating operational impact. The most commonly used metric is the *Power Usage Effectiveness (PUE)*, which quantifies the ratio between the total energy consumed by a data center and the energy used specifically by the servers. While widely adopted, PUE often

provides a simplified view by primarily focusing on the energy required during active production periods.

To address the limitations of PUE, recent work by Luccioni et al. (A. S. Luccioni et al., 2023) and Morand et al. (Morand et al., 2024) proposes using the *Dynamic Ratio* as a more precise alternative.

This metric captures a broader range of operational states by considering three distinct modes: (i) Production: nodes are powered on and running jobs; (ii) Idle: nodes are powered on, but no jobs are running; and (iii) Infrastructure: nodes are powered off, while the others of the infrastructure remains operational.

The *Dynamic Ratio* is computed as the ratio of the total energy consumption across all three operating modes to the energy consumption in the production mode alone, in contrast to PUE, which typically includes only the production and infrastructure modes.

Our goal is to test different multiplicative factors when evaluating the operational impacts. For this, we first compute the equipment energy consumption as the product of the equipment TDP power, P_e , the number of training hours, H_e , and the multiplicative factor to account for the server infrastructure consumption, denoted as ρ (PUE or *Dynamic Ratio*). Furthermore, we convert to kWh by multiplying by 0.001, as described in the equation below.

$$C_e(kWh) = P_e \times H_e \times \rho \times 0.001$$

Then, we estimate the operational impact for training by multiplying the energy consumption C_e and the electricity grid mix (EGM), which converts the total energy assessed to one of the evaluated metrics (ADP, GWP, and PE). The following equation describes the computation of operational impacts.

$$I_{\text{training}}^{\text{opex}} = \sum_{e \in E} C_e \times \text{EGM}$$

7.4.2.4 Inference

During inference, the models are asked to predict the image classification labels based on the input containing the raw image data. We consider that for each equipment e , a ratio k_e of the total N inference requests is directed to the equipment. Furthermore, each piece of equipment performs the inference request within time t_e .

Embodied Impacts To compute the embodied environmental impacts of the inference servers, we first calculate the equipment allocation factor, alloc_e . For this, we divide the PI@ntNet usage time, computed by multiplying the total number of requests (N), the ratio of requests handled by the equipment (k_e), and the request processing time (t_e), by the total usage time, computed based on the equipment lifespan, L_e (in years), and the utilization ratio u_e . The total time is converted into seconds by multiplying the number of days in a year (365 days), the number of hours in a day (24 hours), and the number of seconds in an hour (3600 seconds). The allocation factor computation is described in the equation below.

$$\text{alloc}_e = \frac{N \times k_e \times t_e}{365 \times 24 \times 3600 \times u_e \times L_e}$$

Then, the embodied environmental impact of the inference equipment is evaluated using the equation below, where I_{capex}^e refers to the environmental impact of manufacturing the inference equipment e .

$$I_{inference}^{capex} = \sum_{e \in E} alloc_e \times I_{capex}^e$$

Operational Impacts For estimating the operational environmental impacts, we first compute the total energy consumption by multiplying the equipment power P_e , the number of requests directed to the equipment e (obtained by the product of N and k_e), the single request inference time t_e , and the multiplicative factor ρ_e . Furthermore, we multiply by 0.001 to convert to kWh, as described in the following equation.

$$C_e(Wh) = P_e \times N \times k_e \times t_e \times \rho_e \times 0.001$$

Then, we estimate the operational environmental impacts during inference, in a similar way to training, as described in the equation below:

$$I_{inference}^{opex} = \sum_{e \in E} C_e \times EGM$$

7.4.2.5 Webhosting

The webhosting section comprises the servers used for storing and managing the application, processing user requests, and providing the graphical interface. This section also covers the database management and the storage of all Pl@ntnet service data. We divide the analysis into two modules: (i) API, which consists of Virtual Machines (VMs); and (ii) Storage, comprising physical nodes responsible for managing the Pl@ntNet database.

For the API deployment, several Virtual Machines (VMs) are deployed along a physical machine. The VMs may have different requirements for computational resources, such as CPU and RAM. In our analysis, we assess the environmental impacts considering four different resources: CPU, RAM, Storage, and other resources (which comprise other components, such as case, motherboard, assembly, and power).

We denote the set of virtual machines deployed to implement Pl@ntnet services as V . Each virtual machine $vm \in V$ presents an allocation factor a_v , which represents the ratio of resources allocated over the total resources provided by the physical server. The equations below describe the allocation factor for CPU, RAM, and storage.

$$a_{vm}^{cpu} = \frac{nCPU_{vm}}{nCPU_{phy}}; a_{vm}^{ram} = \frac{ram_{vm}}{ram_{phy}}; a_{vm}^{sto} = \frac{sto_{vm}}{sto_{phy}}$$

For the *others* allocation factor, due to the difficulty in precisely quantifying the resource allocation for each virtual machine (VM), we define it as the average of the CPU, RAM, and storage allocation factors. This average serves as a representative approximation for the allocation of other measurable components.

$$a_{vm}^{others} = \frac{a_{vm}^{cpu} + a_{vm}^{ram} + a_{vm}^{sto}}{3}$$

Embodied Impacts The embodied environmental impacts of *webhosting* section are estimated in distinct ways depending on the evaluated services. *For evaluating the API service, which uses Virtual Machines (VMs)*, the embodied impact is given by the product of the physical server manufacturing impact and the allocation factor of each VM for the different resources, as described in the equation below. $I_{\text{capex}}^{\text{cpu}}$, $I_{\text{capex}}^{\text{ram}}$, $I_{\text{capex}}^{\text{sto}}$, and $I_{\text{capex}}^{\text{others}}$ refer to the environmental impact for manufacturing, respectively, the CPU, RAM, storage, and others resources in the webhosting server. Furthermore, the estimation of the impacts also considers the equipment lifetime, L_e and utilization rate, u_e .

$$I_{\text{API}}^{\text{capex}} = \frac{\sum_{\text{vm} \in V} a_{\text{vm}}^{\text{cpu}} \times I_{\text{capex}}^{\text{cpu}} + a_{\text{vm}}^{\text{ram}} \times I_{\text{capex}}^{\text{ram}} + a_{\text{vm}}^{\text{sto}} \times I_{\text{capex}}^{\text{sto}} + a_{\text{vm}}^{\text{others}} \times I_{\text{capex}}^{\text{others}}}{u_e \times L_e}$$

For the storage service, which uses physical machines for deployment, we consider that all the computation resources from the machines are allocated to the PI@ntNet service. We denote as I_{capex}^e the environmental impact of manufacturing the physical machines hosting the database. The equation below describes the estimation of embodied impacts for the database service.

$$I_{\text{storage}}^{\text{capex}} = \sum_e \frac{I_{\text{capex}}^e}{u_e \times L_e}$$

Operational Impacts The operational environmental impacts consider that machines (physical or virtual) operate in two different modes: *idle* (when no process is running) and *charge* (when processes are being executed).

The annual operational impact on physical machines for each operation mode can be computed using the equation below, where η_e represents the number of blades on the physical machine and $P_e^{\text{idle,charge}}$ corresponds to the blade power during idle or charge mode.

In the equation below, we compute the total energy over a year, in kWh, based on a blade power P_e , the total number of blades, η_e , and the usage time (in hours), obtained by multiplying the number of days in a year (365 days) and the number of hours in a day (24 hours). Then, the energy is converted to kWh by multiplying by 0.001.

$$C_e(\text{kWh}) = \eta_e \times P_e^{\text{idle,charge}} \times 365 \times 24 \times 0.001$$

Then, we estimate the physical impacts by multiplying the energy consumption (C_e) and the Electricity Grid Mix (EGM), as shown in the equation below.

$$I_e^{\text{idle,charge}} = C_e \times \text{EGM}$$

For the API service, we consider that the VMs may be in idle or charge mode. When the virtual machine is in charge, we estimate the impact on the VM as the product of the time-share in charge mode, $s_{\text{vm}}^{\text{charge}}$, the impact of the physical machine $I_{\text{charge}}^{\text{phy}}$, and the CPU allocation factor, $a_{\text{vm}}^{\text{cpu}}$, since the CPU corresponds to the highest consuming source.

On the other hand, when the machine is in idle mode, the environmental impact is estimated as the product of the time-share in idle mode, $s_{\text{vm}}^{\text{idle}}$, the operational impact in idle mode, $I_{\text{idle}}^{\text{phy}}$, and the allocation factor of others, $a_{\text{vm}}^{\text{others}}$, since in idle mode, the CPU, RAM and storage are not being effectively used.

The equation below describes the operational environmental impacts of the API service.

$$I_{API}^{opex} = \sum_{vm \in V} s_{vm}^{idle} \times a_{vm}^{others} \times I_e^{idle} + s_{vm}^{charge} \times a_{vm}^{cpu} \times I_e^{charge}$$

In the same way, for estimating the usage impacts for the database service, we utilize the physical machine impact in the idle and charge mode, respectively I_e^{idle} and I_e^{charge} , and share of time that the equipments are in idle and charge mode, respectively s_e^{idle} and s_e^{charge} . The following equation illustrates the computation.

$$I_{storage}^{opex} = \sum_e s_e^{idle} \times I_e^{idle} + s_e^{charge} \times I_e^{charge}$$

7.4.3 Parameters and sources

During our Life Cycle Assessment (LCA), we use the following parameters to evaluate the environmental impacts associated with Pl@ntNet usage. The used parameters for our analysis are described in Table 7.4.

Based on Pl@ntNet statistics, the service accounted for over 220M of identification requests in 2024. We consider that a session for identifying the plant species has a duration d of 0.5 min. We estimate this value based on exchanges with Pl@ntNet engineers and a practical evaluation of the time required, which includes taking the photo, requesting the identification, and obtaining and visualizing the identification results. Furthermore, we consider that the exchanged packets between the Pl@ntNet data center and the application have an average size of 367 kB, based on discussions with Pl@ntNet engineers.

For the smartphone specifications, we consider that the average time of a smartphone during a day is 223 min, according to the Global Overview Report (Kemp, 2022). Furthermore, the ADEME (ADEME, 2025a) reports that the average lifespan is 3 years, and it consumes, on average, 4 KWh per year.

According to Jean Zay, as reported in Morand et al. (Morand et al., 2024), a supercomputer used for training has a lifespan of 6 years, with an 85% utilization ratio. Furthermore, in our analysis, we tested several multiplicative factors: (i) No Factor, when $\rho = 1.0$, (ii) PUE ($\rho = 1.21$) and (iii) dynamic ratio ($\rho = 1.86$), described in Jean Zay specifications and reported in Morand et al. (Morand et al., 2024).

For the *Inference* and *Webhosting* data centers, we consider the average PUE in France to be 1.46, according to the ARCEP report (ARCEP, 2025). Moreover, since the servers are used exclusively for Pl@ntNet, we consider a utilization ratio of 100% and we assume a lifespan of 6 years.

The *Network* impacts were extracted from the European Digital Technology report (Bordage et al., 2021). The report estimates the environmental impacts of mobile data networking in Europe.

Electricity Grid Mix (EGM) Impact The EGM impact for all the countries was retrieved from ADEME database (ADEME, 2025a). The France EGM was used to estimate the impact of the *Training*, *Inference*, and *Webhosting* sections. For End Users, we used the EGM of all evaluated countries.

Parameter	Value	Source
N	220M	Pl@ntNet
d	0.5 min	Pl@ntNet
S	367 KB	Pl@ntNet
timePerDay (smartphone)	223	GWl (Kemp, 2022)
L_e (smartphone)	3 years	ADEME (ADEME, 2025a)
energyPerYear	4 KWh	ADEME (ADEME, 2025a)
u_e (training)	85%	Jean Zay (IDRIS - CNRS, 2019)
u_e (inference)	100%	Pl@ntNet
u_e (webhosting)	100%	Pl@ntNet
L_e (training)	7 years	Jean Zay (IDRIS - CNRS, 2019)
L_e (inference)	6 years	Pl@ntNet
L_e (webhosting)	6 years	Pl@ntNet
ρ (training)	[1.0,1.21,1.86]	Jean Zay (IDRIS - CNRS, 2019)
ρ (inference)	1.46	ADEME (ADEME, 2025a)

TABLE 7.4 – List of parameters used in Pl@ntNet Life Cycle Assessment with their respective values and sources.

Scenario	V100 hours	A100 hours	Source
<i>Final Models (2024)</i>	61,440	0	Pl@ntNet
<i>Standard (2024)</i>	83,704	96,507	Pl@ntNet
<i>Legacy(2020-2024)</i>	292,065	96,507	Pl@ntNet

TABLE 7.5 – Training time (in number of hours) of Pl@ntNet service models for each scenario in Jean Zay supercomputer. We display the time for each type of node used.

Manufacturing impacts of servers For computing the environmental impacts associated with manufacturing the servers used by Pl@ntnet, we utilized the BoaVizta tool ([Boavizta, 2025](#)). The tool estimates the ADP, GWP, and PE impacts based on the specifications of the hardware components. This estimation was performed for the *Training*, *Inference*, and *Webhosting* sections.

Training times Table 7.5 shows the total GPU hours used for training Pl@ntNet models across three scenarios: *Final Models*, *Standard*, and *Legacy*. These values reflect usage on V100 and A100 GPUs in the Jean Zay supercomputer.

7.5 Results

In this section, we present the results of the Life Cycle Assessment (LCA) of the Pl@ntNet service. We performed a multi-criteria and multi-stage evaluation of the environmental impacts based on the methodology described in Section 7.4. All the parameters used are described in Table 7.4. Firstly, we discuss the global impacts associated with the Pl@ntNet service. We derive the Pl@ntNet architecture sections that mostly contributed to the environmental impacts, and we compare the *embodied* and *operational* impacts.

Section	ADP (kgSbeq)	GWP ($\times 10^3$ kgCO ₂ eq)	PE ($\times 10^6$ MJ)
End Users	1.14	49.2	0.69
Networks	0.07	8.0	0.4
Training	0.5	15.1	1.7
Inference	0.15	2.67	0.27
Webhosting	0.19	2.97	0.08
Total	2.03	77.9	3.14

TABLE 7.6 – Pl@ntNet environmental impacts per section.

Then, we describe in detail the environment impacts obtained for each Pl@ntNet section: *End Users*, *Networks*, *Training*, *Inference*, and *Webhosting*. During the analysis, we discuss the main impacts and identify optimization opportunities for the Pl@ntNet service.

7.5.1 Global environmental impacts

In this section, we analyze the environmental impacts of the Pl@ntnet service. Table 7.6 summarizes the environmental impacts caused by Pl@ntNet in 2024. It was responsible for emitting 77.9 tCO₂eq. Furthermore, its impact on primary energy was 3.14×10^6 MJ and its ADP impact was 2.03 kgSbeq.

Fig. 7.3 illustrates the contribution of each service section to the total impact. We observe that, for Abiotic Depletion Potential (ADP), the *End Users* section presents the highest impact. The section was responsible for extracting 1.14 kgSbeq. Similarly, the *End Users* section was responsible for the highest GWP impact. This section emitted 49.2 tCO₂eq, equivalent to 63.2% of the total GWP impact. This is attributed to the resource-intensive manufacturing process for producing the devices (smartphones) used to access the Pl@ntNet service. During this process, a significant amount of raw materials is extracted, and a large quantity of greenhouse gases is emitted.

On the other hand, for Primary Energy (PE), the *Training* presented the highest impact. The section was responsible for consuming 1.7×10^6 MJ, equivalent to 54.1% of the total energy. The Machine Learning training process demands a large amount of computational load, increasing the energy consumption.

Embodied and Operational Impacts Figs. 7.4a and 7.4b describe, respectively, the *embodied* (capex) and *operational* (opex) impacts for each section. The *Networks* category is excluded from this analysis, as it does not decompose into capex and opex impacts. For the *embodied* impacts (Fig. 7.4a), we observe that the *end users* section presents a high impact. The section was responsible for emitting 46 tCO₂eq for manufacturing, which reflects the application widespread usage. In 2024 alone, over 20 million users utilized Pl@ntNet, resulting in more than 220 million identification requests.

For the *operational* impacts (Fig. 7.4b), a significant portion of the environmental impacts is attributed to *Training*. The datacenter usage for *training* accounted for 10.8 tCO₂eq, representing 46.8% of the total GWP impact associated with Pl@ntNet operations. On the other hand, other sectors presented a lower *operational* impact. The *Inference* section, responsible for running Machine Learning inference requests, emitted 1.7 tCO₂eq, while the *Webhosting* section, which manages the API and the Pl@ntnet database, presented a GWP impact of 304 kgCO₂eq. These

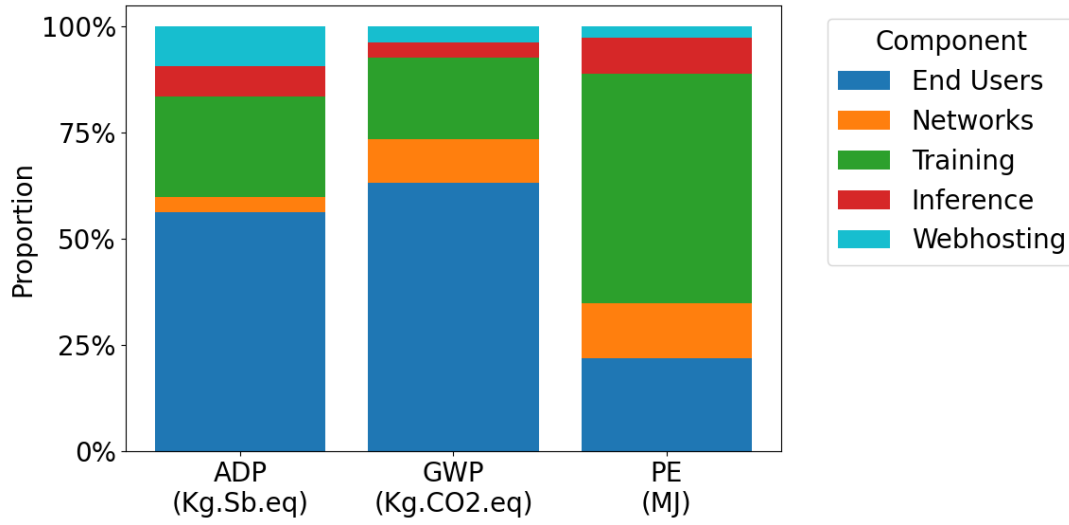


Figure 7.3 – Environmental impacts proportion for each evaluated section of Pl@ntNet service.

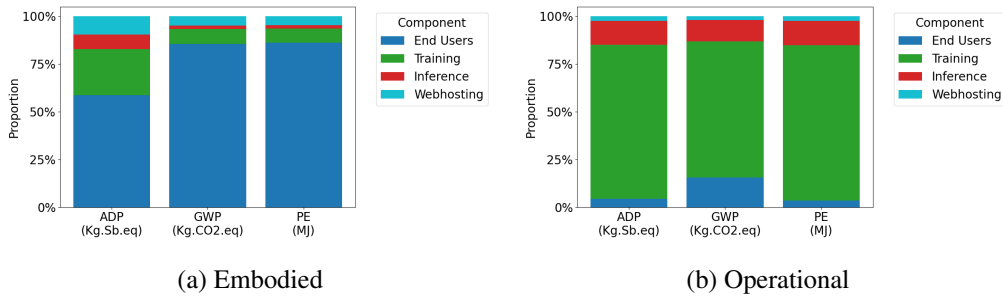


Figure 7.4 – Embodied and operational environmental impacts proportions over different sections in Pl@ntNet.

values are significantly lower than those for the *Training* section. Notably, the GWP impact from *Training* is approximately six times greater than that of *Inference* alone.

7.5.2 Detailed assessment of Pl@ntNet impacts

Following, we provide and discuss LCA results and the environmental impacts associated to each of the evaluated Pl@ntNet sections.

7.5.2.1 End Users

Table 7.7 describes the environmental impacts for the *End Users* section in Pl@ntNet service. Analyzing the total Global Warming Potential (GWP), the *End Users* section emitted 49.4 tCO₂eq. From the total emissions, the embodied stage is responsible for the largest amount of impacts, emitting 46.9 tCO₂eq, equivalent to 95.3% of the total section emissions.

	ADP (kgSbeq)	GWP ($\times 10^3 \text{ kgCO}_2\text{eq}$)	PE ($\times 10^4 \text{ MJ}$)
Embodied	1.14	46.9	62
Operational	0.0003	2.3	7.0
Total	1.14	49.2	69

TABLE 7.7 – *End users* environmental impacts for embodied (capex) and operational (opex) stages.

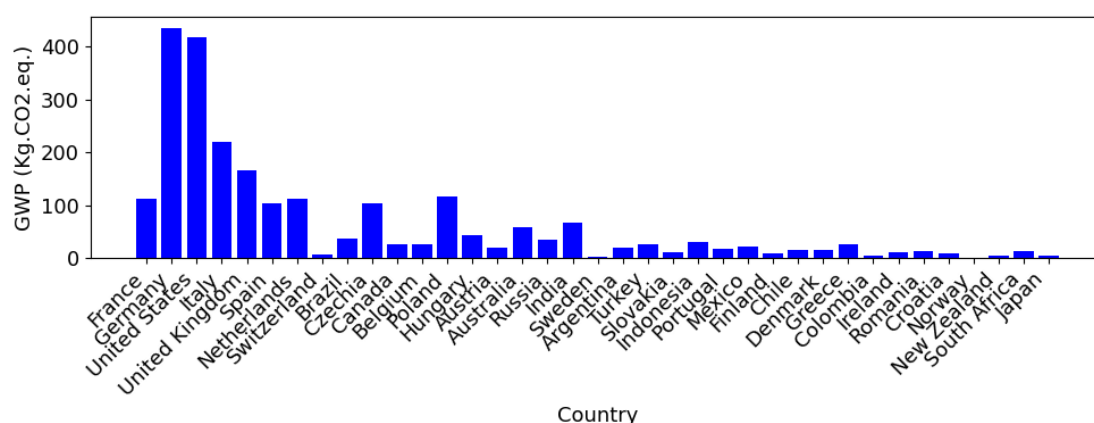


Figure 7.5 – GWP Impact for *End Users* section during usage (OPEX) for different countries. The countries are ordered by number of sessions in non-increasing order.

The observed result aligns with that of Ercan et al. (Ercan, Malmmodin, Bergmark, Kimfalk, & Nilsson, 2016). Performing a Life Cycle Assessment (LCA) of a Sony smartphone, the authors highlight a high Global Warming Potential (GWP) attributed to the production phase. According to them, the integrated circuits, mainly, and display manufacturing are responsible for the main environmental impacts of the evaluated device.

Comparing countries impacts We also analyzed the impact of *End Users* during the *operational* phase, depending on the location of requests. Fig. 7.5 describes the GWP impact during the *operational* phase for each evaluated country. The displayed countries are responsible for a large majority of Pl@ntNet sessions, accounting for over 95%.

The impacts in each may vary depending on Pl@ntNet usage patterns (the number of requests) and the energy grid mix (EGM) of each country. For instance, when comparing the impacts between France and Germany, we observe that France was responsible for 25% of the total sessions, more than twice that of Germany, as observed in Table 7.2. However, the French GWP impact was 160 kgCO₂eq, approximately 4× lower than Germany’s GWP impact of 624 kgCO₂eq. The observed gap occurs since, although France presents more identification requests, the low-carbon sources are larger in the electricity grid mix, leading to attenuated environmental impacts.

On the other hand, some countries, although presenting a small share of Pl@ntNet sessions, provide a high GWP impact. For instance, Czechia and Poland present, respectively, 2% and 1.8% of the sessions, but present 103.7 and 117 kgCO₂eq, respectively.

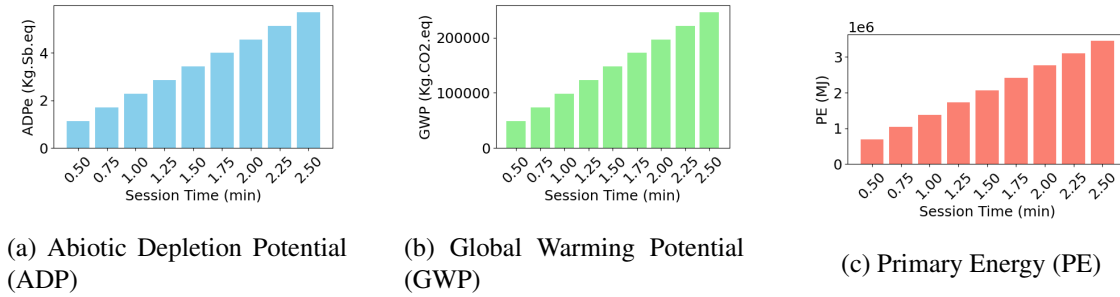


Figure 7.6 – Sensitivity analysis over the identification request duration. We varied the duration in a range from 0.5s to 2.5s and we display the (a) ADP, (b) GWP and (c) PE impacts associated with PI@ntNet *End Users*.

	ADP (kgSbeq)	GWP (kgCO ₂ eq)	PE (MJ)
<i>Networks</i> Impact	7.1×10^{-2}	8.0×10^3	4.6×10^5

TABLE 7.8 – *Networks* environmental for PI@ntNet service in 2024. The data exchanged in the request and answer is considered in this analysis.

Sensitivity analysis Users may present wide behaviors when using PI@ntNet services. Actions such as taking the photo and consulting the results may present a large variation. Thus, we performed a sensitivity analysis by varying the session duration time in the range from 30s to 2min30s with a 15s interval.

In Figs. 7.6a-7.6c, we display the ADP, GWP and PE impacts, respectively, as a function of the request duration time. We observe that the environmental impacts scale linearly with the session duration. For GWP impact, *End Users* section emissions vary from a total of 49 tCO₂eq to 250 tCO₂eq. In the same way, the ADPe and the PE varies from 1.14 to 5.71 kgSbeq and from 691×10^3 to 3.4×10^6 MJ.

7.5.2.2 Networks

We display in Table 7.8 the environmental impacts associated with *Networks* section of PI@ntNet service. The network impacts methodology do not differentiate between *embodied* (capex) and *operational* (opex), considering them as an aggregated value (See Section 7.4).

The *Network* section was responsible for 8 tCO₂eq in 2024, equivalent to 10.2% of total emissions. Although we consider that mobile data links are used, the image compression during transmission between devices and the datacenters leads to resource optimization, attenuating the impacts.

7.5.2.3 Training

We also evaluated the environmental impacts caused by the PI@ntNet models *training*.

Then, in our evaluation, we vary the multiplicative factor, considering three different possibilities: (i) No multiplicative factor ($\rho = 1.0$); (ii) PUE ($\rho = 1.21$); and (iii) dynamic ratio ($\rho = 1.83$).

Furthermore, we considered three distinct scenarios for the training temporal window: (i) *Legacy*, when the total training time from 2020 until 2024 is considered; (ii) *Standard*, when the

training time performed in 2024 is considered; and (iii) *Final Models*, when we consider the training time only for the final models made available for the users.

Table 7.9 describes the environmental impact of the *training* section for each evaluated scenario (*Legacy*, *Standard* and *Final Models*) and each multiplicative factor considered in our analysis.

Training impacts The high-impact behavior observed in the *training* section can be attributed, firstly, to the continuous emergence of new species and observations, which require frequent model retraining to meet users' needs for better and more accurate models. For instance, between 2020 and 2024, the number of species identified by Pl@ntnet increased from 30798 to 75415, representing an increase of 145%.

Furthermore, the development of related products, such as models for disease detection and agricultural diagnostics, has expanded. These new initiatives require additional training efforts, thus leveraging environmental impacts due to the increased computational demand.

Finally, Pl@ntNet researchers have recently focused on optimizing the inference process in order to reduce inference time. While this leads to lower environmental impacts during inference, it introduces a trade-off between inference efficiency and training efficiency. To achieve a low inference time, researchers need to submit their models to longer and more exhaustive training rounds, which demands a longer and more expensive training process, potentially increasing environmental impacts.

Impacts scaling up in 2024 We also observe a considerable impact associated with the *Training* section in 2024. In the *Legacy* scenario, which accounts for the impacts from 2020 to 2024, the GWP impact varies in the range between 13.1 tCO₂eq and 20.6 tCO₂eq. On the other hand, in the *Standard* scenario (2024 year), the GWP impact varies between 6.94 tCO₂eq and 10.9 tCO₂eq, representing, on average, 50% of the total GWP impact observed for *Training* between 2020 and 2024.

The above observation is aligned with the number of training hours used by Pl@ntNet. The training service consumed 208,000 hours of model training from 2020 to 2023, during four years. On the other hand, in 2024, during just one year, Pl@ntNet consumed 180,000 hours, almost the same amount as the previous four years. The new demands of the Pl@ntNet service and the increased availability of data have significantly scaled up the impacts over the last year, posing a substantial impact on the training section due to the Pl@ntNet service.

Final Models vs Standard scenarios We also compared the impacts of the *Final Models* (made available to users) with the *Standard* scenario, which includes developing and testing models. The results described in Table 7.9 illustrate that, in 2024, the *Final Models* accounted for between 1.85 and 2.87 tCO₂eq. On the other hand, the total training in 2024 (*Standard* scenario) ranged from 76.94 to 10.9 tCO₂eq. The *Standard* scenario represents, thus, on average, 4× of the *Final Models* impact.

This illustrates that the impacts may scale up beyond the final model's training. Developing a model to achieve good accuracy may require intense training, leveraging its impacts. Therefore, it is vital that researchers focus on the ML impacts, not only for the *Final Models*, but also for the model development.

Embodied vs Operational impacts Fig. 7.7 describes the share of GWP impact for the *embodied* and *operational* stages in *Legacy* scenario under different multiplicative factors. We observe that *operational* impacts represent the dominant share of the total GWP emissions during *training*. Depending on the multiplicative factor used, between 68% and 79% of the GWP is due to the *operational* phase. This highlights that the emissions generated while training the models, due to electricity usage and the duration of training, constitute the most significant portion of the environmental impact.

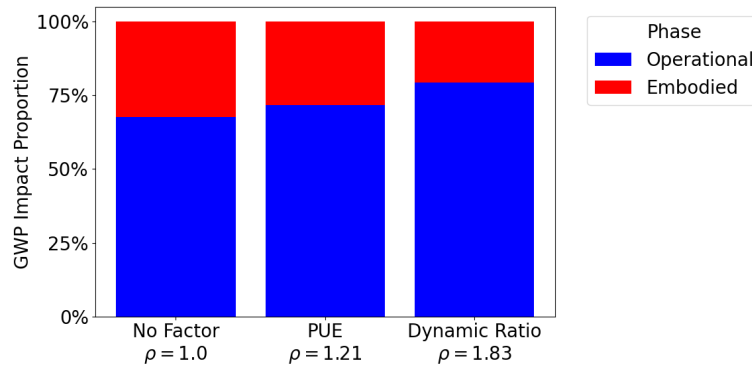


Figure 7.7 – GWP Impact proportion between *embodied* and *operational* stages. We test the stages ratio considering the *Legacy* scenario.

7.5.2.4 Inference

When a plant identification is requested, the model inference is executed on one of the dedicated servers. All the inference requests are performed using GPUs.

Comparing inference machines For each inference server, we collected the average GPU processing time and the proportion of identification requests handled by each machine. All details are described in Table 7.10.

We observed that only a small share of requests were processed in *capcir* and *orxois* machines. This occurs because these machines are older and dedicated to serving requests from legacy devices, which are no longer compatible with updated versions of the PI@ntNet application. The *senart* and *sourdun* machines also accounted for a small share of requests, as they have only been operational since September 2024, and thus have not yet handled a significant volume of traffic.

Fig. 7.8 describes the embodied and operational environmental impacts for each inference machine. We observe a high impact for the *rajol* machine. In 2024, this machine was responsible for generating 1.5 tCO₂eq, accounting for 89% of the total inference impact. This elevated impact is explained by the machine usage for executing multi-species inference, which involves more complex models and demands a significantly longer inference duration. The energy consumption and the environmental impacts become larger.

Scenario	Multiplicative Factor	ADP ($\times 10^{-1}$ kgSbeq)	GWP ($\times 10^3$ kgCO ₂ eq)	PE ($\times 10^6$ MJ)
Final Models (2024)	No Factor $\rho = 1.0$	0.78	1.85	0.19
	PUE $\rho = 1.21$	0.78	2.1	0.23
	<i>Dynamic Ratio</i> $\rho = 1.83$	<i>0.79</i>	<i>2.87</i>	<i>0.35</i>
Standard (2024)	No Factor $\rho = 1.0$	2.12	6.94	0.76
	PUE $\rho = 1.21$	2.13	7.94	0.90
	<i>Dynamic Ratio</i> $\rho = 1.83$	<i>2.15</i>	<i>10.9</i>	<i>1.36</i>
Legacy (2020-2024)	No factor $\rho = 1.0$	4.72	13.2	1.41
	PUE $\rho = 1.21$	4.78	15.1	1.70
	<i>Dynamic Ratio</i> $\rho = 1.83$	<i>4.82</i>	<i>20.6</i>	<i>2.55</i>

TABLE 7.9 – Environmental impacts of training in three different scenarios: *Final models*, *standard* and *legacy*, considering several multiplicative factors.

Server	GPU	Purchase year	Processing time (ms)	Ratio (%)
<i>capcir</i>	Quadro M4000	2016	103.7	2
<i>orxois</i>	Quadro M4000	2016	122.2	3
<i>huchet</i>	RTX 8000	2020	52.2	1
<i>rajol</i>	RTX 8000	2020	3125.6	25
<i>chamois</i>	A40	2022	158.4	37
<i>devois</i>	A40	2022	62.12	24
<i>senart</i>	A40	2024	50.9	3
<i>sourdun</i>	A40	2024	70.4	5

TABLE 7.10 – Average inference processing time (in milliseconds) and percentage of total requests handled by each server.

7.5.2.5 Webhosting

Here, we analyze the main impacts associated with the *webhosting* section. We remind that the analysis is divided into two modules: (i) *API*, consisting of Virtual Machines (VMs); and (ii) *Storage*, composed of physical nodes responsible for managing the Pl@ntNet database.

Table 7.11 summarizes the embodied (capex) and operational (opex) environmental impacts for the *Webhosting* section, divided into *API* and *storage* modules.

We first observe that the ADP impact is, as expected, significantly higher in the *embodied* phase than in the *operational* phase for both modules. This is consistent with the fact that ADP

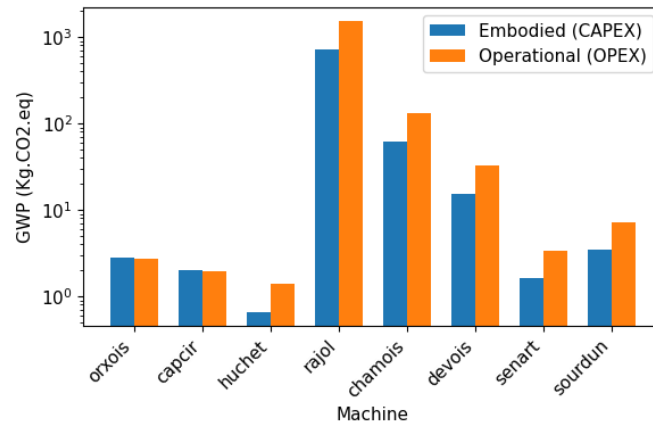


Figure 7.8 – *Inference* environmental impacts per machine. The *capcir* and *orxois* machines use a Nvidia Quadro M4000; *huchet* and *rajol* use a NVIDIA RTX 8000; and *chamois*, *devois*, *senart* and *sourdun* use a NVIDIA A40 GPU.

<i>API (VMs)</i>			
	ADP (kgSbeq)	GWP (tCO ₂ eq)	PE (×10 ⁴ MJ)
Embodied	0.08	0.44	0.57
Operational	0.00012	0.20	3.07
<i>Storage (Physical)</i>			
	ADP (kgSbeq)	GWP (tCO ₂ eq)	PE (×10 ⁴ MJ)
Embodied	0.11	2.23	2.78
Operational	0.00006	0.1	1.56
Total	0.18	2.97	7.98

TABLE 7.11 – *Webhosting* section environmental impacts for the *APIs* (virtual machines) and the *storage* (physical machines) modules.

primarily accounts for the extraction of raw metal resources, which is directly linked to equipment manufacturing processes.

For the *API* module, the GWP is also higher in the embodied stage, totaling 0.44 tCO₂eq, which represents 68.75 % of the total GWP for this module. This is mainly due to two factors: (i) the manufacturing of servers is a carbon-intensive process, significantly increasing the embodied GWP, and (ii) the servers operate in France, where the electricity grid has a low carbon intensity, thus reducing the operational GWP.

In contrast, the PE impact is predominantly concentrated in the operational stage, accounting for 3.07 × 10⁴ MJ, or 84.3 % of the total PE impact. This is expected, given that the servers run continuously, resulting in sustained energy consumption throughout their operational lifetime.

Regarding the *storage* module, we observe that the *embodied* stage dominates all three impact categories. The GWP and PE impacts in this phase are 2.23 tCO₂eq and 2.78 × 10⁴ MJ, accounting for 95.7 % and 64 % of the total module impacts, respectively. This is primarily due to the substantial environmental burden associated with the manufacturing of high-capacity SSDs used for the PI@ntNet application, which involves resource and energy-intensive processes.

7.6 Conclusion

Given the widespread adoption of Artificial Intelligence (AI) models and their integration into several applications, it has become vital to study and quantify the environmental impacts of AI-based services. In this study, we focus on the PI@ntNet service, an image recognition application designed to identify plant species from photographs. We proposed a Life Cycle Assessment (LCA) of the PI@ntNet service to evaluate its environmental impacts.

Our analysis adopts a multi-criteria and multi-stage approach. We investigated the Abiotic Depletion Potential (ADP), the Global Warming Potential (GWP), and the Primary Energy (PE) impacts associated with the evaluated application. These impacts were evaluated across both the embodied phase (production, transport, and end-of-life) and the operational stages of the PI@ntNet service.

PI@ntNet was responsible for a total of 77.9 tCO₂eq emissions in 2024. We structured our analysis into five key sections of the service: *End User*, *Networks*, *Training*, *Inference*, and *Web-hosting*. From the total amount, the *End Users* section accounted for the largest share, contributing 46.9 tCO₂eq, or 60% of the total GWP. We can also see this section's predominant impact on the depletion of abiotic mineral and metal resources.

The *Training* section also exhibited a substantial environmental impact, resulting in GWP of 15 tCO₂eq, along five years, equivalent to 19% of the total emissions; and in PE of 1.7×10^6 MJ, 53% of total primary energy. The *Inference*, on the other hand, in 2024, was responsible for emitting 2.67 tCO₂eq. In large-scale services, the *Inference* stage usually presents a larger impact than the *Training* stage (Patterson et al., 2022). However, in the PI@ntNet service, the *Inference* impact does not significantly contribute when compared to *Training*. This occurs mainly due to the optimization techniques applied to improve *Inference* efficiency, such as code optimization.

For the future, we emphasize the importance of improving optimization techniques to reduce the environmental footprint. These advances, especially for the training, are essential for mitigating the impacts of AI-based services like PI@ntNet.

We also envisage, as future work, assessing indirect and social impacts. Considering these aspects, especially in global and collaborative contexts involving scientists, hobbyists, and AI, can reveal deeper insights into the effective reduction of impact. Although methodologies for such evaluations are still limited, existing work (Solaiman et al., 2023) offers a foundation for further reflection.

CHAPTER 8

Conclusion and Perspectives

In this thesis, we focused on the resource consumption improvement for Machine Learning (ML) inference. For this, we divided the thesis into two parts. In the first part, we investigated optimizing scheduling ML inference jointly with model compression techniques.

Scheduling ML inference tasks jointly with model compression In Chapter 4, we developed an algorithmic approach for scheduling ML inference tasks. We focus our analysis on image classification tasks. We are based on "*slimmable*" neural networks (Cai et al., 2020; J. Yu & Huang, 2019a), which allow for compressing the model at inference time without requiring retraining of weights. These techniques demonstrate an accuracy-latency tradeoff with diminishing marginal gain behavior. When the latency is low, there is a high accuracy marginal gain. When the model size increases (as well as the inference time), the marginal gain diminishes. We investigate, then, a scenario where ML inference tasks present hard deadline constraints, and thus, compression techniques may be used to reduce processing time with minimal accuracy drop.

To achieve this, we first performed experiments to expand the compression techniques of "*slimmable*" models, enabling compression across the full range. Then, we studied the scheduling problem. We modeled the optimization problem and then proposed:

- (i) an optimal algorithm in polynomial time for solving the problem with fractional relaxation. In this scenario, we allow a single task be deployed in multiple machines simultaneously.
- (ii) an approximation algorithm for the original problem (when a task can be deployed in a single machine). In this case, we use the previous algorithm as an intermediary step for building our solutions with proven guarantees.

Our solution provides near-optimal results and outperforms some scheduling techniques in the literature, which do not allow compression or allow only a few discrete levels.

Scheduling with energy budget In Chapter 5, we extended the previous problem. Given the climate change challenges and the limited hardware, for instance, in the cloud-to-edge continuum, many devices may present energy constraints. Then, we considered the previous problem (in which "*slimmable*" neural networks serve ML inference tasks with hard deadlines), and we added an energy budget constraint for executing the inference tasks. In the evaluated scenario, the machines have different speeds and power specifications, which increases the relevance of the place for executing each ML inference task. We propose here:

- (i) An optimal algorithm based on the KKT conditions, which solves the problem with fractional relaxation and a piecewise linear function.
- (ii) an extension of the previous approximation algorithm for adapting to the energy budget constraints

We tested the above solution under several energy budget scenarios and we observed that our proposal achieved near-optimal results. We provided some insights into the behavior of our algorithm, particularly in scenarios involving heterogeneous machines and varying energy constraints.

Estimating AI inference energy savings by model selection In a second part of the thesis, we aim to evaluate the environmental impacts of AI-based applications and opportunities for leveraging consumption efficiency and mitigating impacts. In Chapter 6, we analyze the energy impacts for a wide range of tasks. We first investigated the most commonly addressed AI tasks in data centers based on statistics of platforms, such as *Hugging Face* ([Hugging Face, s. d.](#)) and *Papers With Code* ([Papers with Code: The latest in Machine Learning, s. d.](#)). Then, we selected representative benchmarks for each task and we analyzed the tradeoff between utility and model size. We observed that the tradeoff is governed by the diminishing marginal gain law, which creates opportunities for optimization. We also evaluated model adoption patterns based on statistics in the Hugging Face platforms, which reflect distinct efficiency results for distinct tasks.

We then estimated the energy savings obtained by applying model selection and choosing energy-efficient models for performing AI inference requests. Energy-efficient models provide a high utility and present a comparably small size (leading to small energy consumption). The idea is to estimate the gains that users would achieve by adopting efficient models, in the direction of achieving sufficiency. We estimate that we may achieve a 27.8% energy reduction, which, in a projection for global data centers, translates to a savings of 31.9 TWh, equivalent to the output of 5 nuclear power reactors.

Life Cycle Assessment (LCA) of an AI-based application In Chapter 7.1, we extend our investigation of environmental impacts due to AI activity. Indeed, several factors, such as raw metal extraction and greenhouse gas emissions, are associated with AI usage. We then propose a Life Cycle Assessment (LCA) of a vision-based application. We conducted our investigation on the Pl@ntnet application ([Joly et al., 2014](#)), a widely used platform for identifying plant species. We base our evaluation on a multi-criteria and multi-stage methodology. The aim is to assess the embodied (manufacturing, transport, and end-of-life) and operational (usage) impacts (abiotic depletion potential, global warming potential, and primary energy). We divided the application into 5 sections: end users, networks, training, inference and webhosting.

We estimate that the Pl@ntNet application was responsible for 78.4 tCO₂eq in 2024, equivalent to 44 round-trip flights between Paris and New York. From the application sections, the *End Users* are the ones with the highest contribution to the environmental impacts, as many devices and users access the application. We also identified that the training has a significant impact. This is mainly due to the emergence of new data and species, which require retraining models. On the other hand, *inference* and *webhosting* have a low impact, as many optimization techniques (e.g., coding efficiency and idle state) are applied.

8.1 Future perspectives

Looking ahead, we seek opportunities to expand our work. For the first part, regarding joint scheduling with model compression, we observe that the concept of "*slimmable*" neural networks (compression during inference) may be extended to broader AI tasks, such as *reasoning* and *text generation*. In these tasks, large-scale models, such as LLMs (e.g., *GPT*, *DeepSeek*, *LLama*), are widely adopted by users. These models are characterized by high computing complexity and require a substantial amount of resources. [Kaplan et al. \(2020\)](#), for instance, have demonstrated a decreasing power-law between the test dataset loss and the number of parameters for language models, indicating a convex problem. Furthermore, building "*slimmable*" transformer-based models with the help of MoE (mixture-of-experts) technique has been proposed in the literature as a post-training compression approach ([T. Chen, Zhang, Jaiswal, Liu, & Wang, 2023](#)). Thus, following these trends, we aim to deeply investigate the relation between loss (and accuracy) and model size during inference, in order to extract efficient scheduling algorithms for other tasks than computer vision, such as *reasoning* and *text generation*.

Moreover, searching for efficient configurations across a wide range of possible sub-networks is a future perspective. The Once-For-All approach ([Cai et al., 2020](#)), for example, utilizes an evolutionary algorithm for finding the best sub-network. Although this delay is not currently considered in our model, we recognize that it may be a relevant factor in practical applications. For instance, recently, [Lin, Li, Zhang, et Leon-Garcia \(2024\)](#) proposed a Reinforcement Learning (RL) method for choosing the sub-network. Thus, we envisage developing efficient search algorithms, reducing inference latency and optimizing resource allocation.

Finally, we envisage extending the optimization method for different scenarios. ML inference requests may experience communication delays when arriving at multiple machines, which can depend on several factors, such as network link, distance, and input size. As pointed out by [Ahvar et al. \(2019\)](#), network links may be a significant source of energy consumption in the cloud-to-edge continuum environment. Then, we envisage adding network cost to the optimization problem. The idea, thus, is to investigate the tradeoff posed by the allocation decision: (i) devices closer to the edge may reduce the communication latency and energy, but they may present limited computing resources; (ii) devices closer to the cloud are more powerful, but lead to high communication costs.

For the second part, concerning AI environmental impacts, we envision refining our models to assess the energy consumption of machine learning models in the future. Recently, [Getzner, Charpentier, et Günnemann \(2023\)](#) proposed modelling Convolutional Neural Networks (CNN) inference energy consumption as a per-layer approach using linear and polynomial regression methods to predict energy consumption. Thus, we believe that developing precise models to estimate the energy impact, depending on the model architecture (such as layer width and memory accesses), is essential for accurately assessing the environmental impacts of AI.

Furthermore, we envision developing new algorithms and heuristics for applying *model selection*, as discussed in Chapter 6. Choosing the most efficient model (in terms of energy and utility) may depend on several factors intrinsic to the task, such as the minimum Service-Level Agreement requirements and hardware resource constraints. Furthermore, model performance may vary depending on a sub-task or the data used. These elements pose important challenges when selecting an AI model for performing an inference request. Therefore, designing smart approaches for selecting energy-efficient models under different variants is essential.

Finally, a crucial research direction involves understanding and addressing the rebound effects of AI efficiency improvements, also known as the *Jevons Paradox* ([Jevons, 1866](#)), which states

that increasing efficiency can paradoxically lead to greater overall consumption. In this context, [A. S. Luccioni, Strubell, et Crawford \(2025\)](#) studied the *Jevons paradox* effects for AI-based services. Indeed, the recent advances in energy efficiency of AI models may result in indirect material, economic, and societal effects in the opposite direction. For instance, many optimization techniques have been proposed to improve hardware efficiency. However, the number of shipped GPUs increased by more than one million from 2023 to 2024, contributing to increased greenhouse gas emissions, water consumption, and raw material extraction associated with device manufacturing and transport. Thus, we envisage investigating how the recent proposals for improving energy efficiency can lead to indirect effects on users behaviors, increasing the environmental footprint.

References

- Abio, B. (2023). In ai, is bigger better? *Nature*, 615(7951), 202–205.
- ADEME. (2025a). *Base Impacts: Environmental Impact Database*. Consulté sur <https://base-impacts.ademe.fr> (Accessed May 2025)
- ADEME. (2025b). *Référentiel par catégorie de produit (rcp) des services numériques*. <https://librairie.ademe.fr/industrie-et-production-durable/6022-referentiel-par-categorie-de-produit-rcp-des-services-numeriques.html>.
- Agence Nationale de la Recherche. (2019). *ARTIC: ARTificial Intelligence-based Cloud Network Control*. <https://anr.fr/Project-ANR-19-CE25-0001>. (Project ANR-19-CE25-0001)
- Ahvar, E., Orgerie, A.-C., & Lebre, A. (2019). Estimating energy consumption of cloud, fog, and edge computing infrastructures. *IEEE Transactions on Sustainable Computing*, 7(2), 277–288.
- ALCIOM. (s. d.). *Powerspy2: An advanced power analyzer*. Consulté sur <https://www.alciom.com/en/our-trades/products/powerspy2/> (Accessed: 2025-03-10)
- Alidaee, B., & Ahmadian, A. (1993). Two parallel machine sequencing problems involving controllable job processing times. *European Journal of Operational Research*, 70(3), 335–341.
- Alliche, R. A., Barros, T. D. S., Aparicio-Pardo, R., & Sassatelli, L. (2022, September). Impact evaluation of control signalling onto distributed learning-based packet routing. In *2022 34th international teletraffic congress (itc-34)*. Shenzhen, China.
- Alliche, R. A., Da Silva Barros, T., Aparicio-Pardo, R., & Sassatelli, L. (2022, juin). PRISMA: a packet routing simulator for Multi-Agent reinforcement learning. In *2022 ifip networking wkshps network intelligence*. Catania, Italy.
- Alzahrani, E. J., Tari, Z., Zeephongsekul, P., Lee, Y. C., Alsadie, D., & Zomaya, A. Y. (2016). Sla-aware resource scaling for energy efficiency. In *2016 ieee 18th international conference on high performance computing and communications; ieee 14th international conference on smart city; ieee 2nd international conference on data science and systems (hpcc/smartcity/dss)* (pp. 852–859).
- Angelelli, L., Da Silva, A. A., Georgiou, Y., Mercier, M., Mounié, G., & Trystram, D. (2023). Towards a multi-objective scheduling policy for serverless-based edge-cloud continuum. In *2023 ieee/acm 23rd international symposium on cluster, cloud and internet computing (ccgrid)* (pp. 485–497).
- Anthony, L. F. W., Kanding, B., & Selvan, R. (2020, July). *Carbontracker: Tracking and predicting the carbon footprint of training deep learning models*. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems. (arXiv:2007.03051)
- ApS, M. (2022). The mosek imization toolbox for python manual. version 10.0. [Manuel de logiciel]. Consulté sur <https://docs.mosek.com/latest/pythonapi/index.html>

- ARCEP. (2025). *Enquête annuelle "pour un numérique soutenable" – édition 2025 (données 2023)*. Consulté sur <https://www.arcep.fr/cartes-et-donnees/nos-publications-chiffrees/impact-environnemental/enquete-annuelle-pour-un-numerique-soutenable-edition-2025.html> (Accessed May 2025)
- Aslanpour, M. S., Toosi, A. N., Cheema, M. A., & Gaire, R. (2022). Energy-aware resource scheduling for serverless edge computing. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (pp. 190–199).
- Asperti, A., Evangelista, D., & Loli Piccolomini, E. (2021). A survey on variational autoencoders from a green ai perspective. *SN Computer Science*, 2(4), 301.
- Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., & Sarzyniec, L. (2013). Adding virtualization capabilities to the Grid'5000 testbed. In I. I. Ivanov, M. van Sinderen, F. Leymann, & T. Shan (Eds.), *Cloud computing and services science* (Vol. 367, p. 3-20). Springer International Publishing. doi: 10.1007/978-3-319-04519-1_1
- Bannour, N., Ghannay, S., Névél, A., & Ligozat, A.-L. (2021). 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* (pp. 11–21).
- Beeching, E., Fourrier, C., Habib, N., Han, S., Lambert, N., Rajani, N., Sanseviero, O., Tunstall, L., & Wolf, T. (2023). *Open llm leaderboard (2023-2024)*. https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard. Hugging Face.
- Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, 16(1), 87–90.
- Ben Allal, L., Muennighoff, N., Kumar Umapathi, L., Lipkin, B., & von Werra, L. (2022). *A framework for the evaluation of code generation models*. <https://github.com/bigcode-project/bigcode-evaluation-harness>. GitHub.
- Bengio, Y. (2009). *Learning deep architectures for ai*. Now Publishers.
- Berthelot, A., Caron, E., De Laage, R., Lefèvre, L., & Nicolas, A. (2024). Fine-grained methodology to assess environmental impact of a set of digital services.
- Berthelot, A., Caron, E., Jay, M., & Lefèvre, L. (2024). Estimating the environmental impact of generative-ai services using an lca-based methodology. *Procedia CIRP*, 122, 707–712.
- Bhardwaj, K., Lin, C.-Y., Sartor, A., & Marculescu, R. (2019). Memory-and communication-aware model compression for distributed deep learning inference on iot. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s), 1–22.
- Boavizta. (2025). <https://github.com/Boavizta/boaviztapi>. (Accessed May 2025)
- Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., & Tamchyna, A. (2014, juin). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation* (pp. 12–58). Baltimore, Maryland, USA : Association for Computational Linguistics. Consulté sur <https://aclanthology.org/W14-3302> doi: 10.3115/v1/W14-3302

- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Bordage, F., de Montenay, L., Benqassem, S., Delmas-Orgelet, J., Domon, F., Prunel, D., Vateau, C., & Lees Perasso, E. (2021). Digital technologies in europe: an environmental life cycle approach. *The Greens/EFA, Brussels*.
- Boyan, J. A., & Littman, M. L. (1993). Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proc. neural information processing systems (neurips)* (p. 671-678).
- Boyd, S. P., & Vandenberghe, L. (2004). *Convex imization*. Cambridge university press.
- Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International conference on learning representations*. Consulté sur <https://openreview.net/forum?id=HylxE1HKwS>
- Carastan-Santos, D., Da Costa, G., Poquet, M., Stolf, P., & Trystram, D. (2024). Light-weight prediction for improving energy consumption in hpc platforms. In *European conference on parallel processing* (pp. 152–165).
- Chan, S., Stone, T., Szeto, K. P., & Chan, K. H. (2013). Predictionio: a distributed machine learning server for practical software development. In *Proceedings of the 22nd acm international conference on information & knowledge management* (pp. 2493–2496).
- Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial imization: Volume1–3*, 1493–1641.
- Chen, G., Choi, W., Yu, X., Han, T., & Chandraker, M. (2017). Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems*, 30.
- Chen, L., Hu, B., Guan, Z.-H., Zhao, L., & Shen, X. (2021). Multiagent meta-reinforcement learning for adaptive multipath routing optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 1–13. doi: 10.1109/TNNLS.2021.3070584
- Chen, T., Zhang, Z., Jaiswal, A., Liu, S., & Wang, Z. (2023). Sparse moe as the new dropout: Scaling dense and self-slimmable transformers. *arXiv preprint arXiv:2303.01610*.
- Chen, X., Li, B., Proietti, R., Liu, C.-Y., Zhu, Z., & Yoo, S. B. (2019). Demonstration of distributed collaborative learning with end-to-end qot estimation in multi-domain elastic optical networks. *Optics express*, 27(24), 35700–35709.
- Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1), 126–136.
- Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhang, H., Zhu, B., Jordan, M., Gonzalez, J. E., et al. (2024). Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.
- Cournot, A. A. (1838). *Researches into the mathematical principles of the theory of wealth*. chez L. Hachette. Translated from French by New York: Macmillan Company, 1927 [c1897].

- Courty, B., Schmidt, V., Luccioni, S., Goyal-Kamal, MarionCoutarel, Feld, B., Lecourt, J., Liam-Connell, Saboni, A., Inimaz, supatomic, Léval, M., Blanche, L., Cruveiller, A., ouminasara, Zhao, F., Joshi, A., Bogroff, A., de Lavoreille, H., Laskaris, N., Abati, E., Blank, D., Wang, Z., Catovic, A., Alencon, M., Stkechly, M., Bauer, C., de Araújo, L. O. N., JPW, & Minerva-Books. (2024, mai). *mlco2/codecarbon: v2.4.1*. Zenodo. Consulté sur <https://doi.org/10.5281/zenodo.11171501> doi: 10.5281/zenodo.11171501
- Cplex, I. I. (s. d.). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53), 157.
- Dantzig, G. B. (2016). Linear programming and extensions.
- Day, G. S. (1981). The product life cycle: analysis and applications issues. *Journal of marketing*, 45(4), 60–67.
- Deepseek. (s. d.). <https://www.deepseek.com/>.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848
- Desislavov, R., Martínez-Plumed, F., & Hernández-Orallo, J. (2023). Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38, 100857.
- Docker main page website. (s. d.). Consulté sur <https://www.docker.com/>
- Dodge, J., Prewitt, T., Tachet des Combes, R., Odmark, E., Schwartz, R., Strubell, E., Luccioni, A. S., Smith, N. A., DeCario, N., & Buchanan, W. (2022). Measuring the carbon intensity of ai in cloud instances. In *Proceedings of the 2022 acm conference on fairness, accountability, and transparency* (pp. 1877–1894).
- Douwes, C., Esling, P., & Briot, J.-P. (2021). Energy consumption of deep generative audio models. *arXiv preprint arXiv:2107.02621*.
- Drainakis, G., Pantazopoulos, P., Katsaros, K. V., Sourlas, V., & Amditis, A. (2021). On the distribution of ml workloads to the network edge and beyond. In *Ieee conference on computer communications workshops (infocom wkshps)* (pp. 1–6).
- Epoch AI. (2024). *Data on notable ai models*. Consulté sur <https://epoch.ai/data/notable-ai-models> (Accessed: 2025-01-16)
- Ercan, M., Malmudin, J., Bergmark, P., Kimfalk, E., & Nilsson, E. (2016). Life cycle assessment of a smartphone. In *Ict for sustainability 2016* (pp. 124–133).
- Faccarello, G. (2016). Anne-robert-jacques turgot (1727–1781). *Handbook on the history of economic analysis*, 1, 73–82.
- Fan, L., Hua, W., Li, L., Ling, H., Zhang, Y., & Hemphill, L. (2023). *Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes*.
- Fang, B., Zeng, X., & Zhang, M. (2018). Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th annual international conference on mobile computing and networking* (pp. 115–127).
- Fieni, G., Acero, D. R., Rust, P., & Rouvoy, R. (2024). Powerapi: A python framework for building software-defined power meters. *Journal of Open Source Software*, 9(98), 6670.

- Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., & Whiteson, S. (2017, 06–11 Aug). Stabilising experience replay for deep multi-agent reinforcement learning. In D. Precup & Y. W. Teh (Eds.), *Proc. intl. conf. on machine learning (icml)* (Vol. 70, pp. 1146–1155). PMLR.
- Frantar, E., & Alistarh, D. (2023). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning* (pp. 10323–10337).
- Freund, K. (2019). Google cloud doubles down on nvidia gpus for inference, 2019. URL <https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference>.
- Gandhi, S., von Platen, P., & Rush, A. M. (2022). *Esb: A benchmark for multi-domain end-to-end speech recognition*. Consulté sur <https://arxiv.org/abs/2210.13352>
- Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., & Zou, A. (2021, septembre). *A framework for few-shot language model evaluation*. Zenodo. Consulté sur <https://doi.org/10.5281/zenodo.5371628> doi: 10.5281/zenodo.5371628
- Gao, W., Hu, Q., Ye, Z., Sun, P., Wang, X., Luo, Y., Zhang, T., & Wen, Y. (2022). Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision. *arXiv preprint arXiv:2205.11913*.
- Gawłowicz, P., & Zubow, A. (2019, November). ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research. In *Proc. ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*.
- Ge, D., Huangfu, Q., Wang, Z., Wu, J., & Ye, Y. (s. d.). *Cardinal Optimizer (COPT) user guide*. <https://guide.coap.online/copt/en-doc>. Consulté sur <https://guide.coap.online/copt/en-doc/intro.html> (Version 7.2 (or specify the version you used))
- Geißler, D., Zhou, B., Liu, M., Suh, S., & Lukowicz, P. (2024). The power of training: How different neural network setups influence the energy demand. In *International conference on architecture of computing systems* (pp. 33–47).
- Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., & Ritter, M. (2017). Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE international conference on acoustics, speech and signal processing (icassp)* (pp. 776–780).
- Getzner, J., Charpentier, B., & Günnemann, S. (2023). Accuracy is not the only metric that matters: Estimating the energy consumption of deep learning models. *arXiv preprint arXiv:2304.00897*.
- Giroire, F., Huin, N., Tomassilli, A., & Pérennes, S. (2019). When network matters: Data center scheduling with network tasks. In *Ieee conference on computer communications (infocom)* (pp. 2278–2286).
- Gnibga, W. E., Blavette, A., & Orgerie, A.-C. (2023). Renewable energy in data centers: the dilemma of electrical grid dependency and autonomy costs. *IEEE transactions on sustainable computing*, 9(3), 315–328.
- Gonzalez, T. F. (1995). A simple lp-free approximation algorithm for the minimum weight vertex cover problem. *Information Processing Letters*, 54(3), 129–131.

- Gonzalez, T. F. (2007). *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC.
- Gupta, U., Kim, Y. G., Lee, S., Tse, J., Lee, H.-H. S., Wei, G.-Y., Brooks, D., & Wu, C.-J. (2021). Chasing carbon: The elusive environmental footprint of computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (pp. 854–867).
- Gurobi Optimization, LLC. (s. d.). *Gurobi Optimizer Reference Manual*. Consulté sur <https://www.gurobi.com>
- Gym: Gym toolkit for creating reinforcement learning environments. (s. d.). Consulté sur <https://gym.openai.com> (11/05/22)
- Haddad, M., Da Costa, G., Nicod, J.-M., Péra, M.-C., Pierson, J.-M., Rehn-Sonigo, V., Stolf, P., & Varnier, C. (2021). Combined it and power supply infrastructure sizing for standalone green data centers. *Sustainable Computing: Informatics and Systems*, 30, 100505.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).
- Heidenreich, S., & Kraemer, T. (2016). Innovations—doomed to fail? investigating strategies to overcome passive innovation resistance. *Journal of Product Innovation Management*, 33(3), 277–297.
- Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.
- Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1314–1324).
- Hua, H., Li, Y., Wang, T., Dong, N., Li, W., & Cao, J. (2023). Edge computing with artificial intelligence: A machine learning perspective. *ACM Computing Surveys*, 55(9), 1–35.
- Huang, Y., Wang, F., Wang, F., & Liu, J. (2019). Deepar: A hybrid device-edge-cloud execution framework for mobile deep learning applications. In *IEEE Infocom 2019-IEEE Conference on Computer Communications Workshops (Infocom Wkshps)* (pp. 892–897).
- Hugging face. (s. d.). <https://huggingface.co/>.
- Huin, N., Rifai, M., Giroire, F., Pacheco, D. L., Urvoy-Keller, G., & Moulhierac, J. (2018). Bringing energy aware routing closer to reality with sdn hybrid networks. *IEEE Transactions on Green Communications and Networking*, 2(4), 1128–1139.
- IDRIS - CNRS. (2019). *Jean zay supercomputer*. <https://www.idris.fr/eng/jean-zay/>. (Accessed: 2025-06-22)
- Impact, G. (2024). *Ecologits: A tool for measuring energy consumption of machine learning models*. <https://github.com/genai-impact/ecologits>. (Accessed: 2024-11-27)
- International Energy Agency. (2025). *Energy and ai*. <https://www.iea.org/reports/energy-and-ai>. (IEA, Paris. Licence: CC BY 4.0)
- Jacquet, P., Ledoux, T., & Rouvoy, R. (2024). Scroogevm: Boosting cloud resource utilization with dynamic oversubscription. *IEEE Transactions on Sustainable Computing*.
- Javaid, S., Javaid, N., Saba, T., Wadud, Z., Rehman, A., & Haseeb, A. (2019). Intelligent resource allocation in residential buildings using consumer to fog to cloud based framework. *Energies*, 12(5), 815.

- Jay, M. (2024). *A versatile methodology for assessing the electricity consumption and environmental footprint of machine learning training: from supercomputers to edge devices* (Thèse de doctorat non publiée). Université Grenoble Alpes.
- Jevons, W. S. (1866). *The coal question; an inquiry concerning the progress of the nation and the probable exhaustion of our coal-mines*. Macmillan.
- Jiang, S., Ma, Z., Zeng, X., Xu, C., Zhang, M., Zhang, C., & Liu, Y. (2020). Scylla: Qoe-aware continuous mobile vision with fpga-based dynamic deep neural network reconfiguration. In *Ieee infocom 2020-ieee conference on computer communications*.
- Jin, Y., Jiao, L., Qian, Z., Zhang, S., Chen, N., Lu, S., & Wang, X. (2020). Provisioning edge inference as a service via online learning. In *2020 17th annual ieee international conference on sensing, communication, and networking (secon)* (pp. 1–9).
- Joly, A., Goëau, H., Glotin, H., Spampinato, C., Bonnet, P., Vignoli, V., Planqué, R., Palazzo, S., & Müller, H. (2014). Interactive plant identification based on social image data. *Ecological Informatics*, 23, 22–34. doi: 10.1016/j.ecoinf.2013.07.006
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- Karger, D. R., Stein, C., & Wein, J. (1999). Scheduling algorithms. *Algorithms and theory of computation handbook*, 1, 20–20.
- Kemp, S. (2022). *Digital 2022: Global overview report*. <https://datareportal.com/reports/digital-2022-global-overview-report>. (Accessed: 2025-05-12)
- Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K., & Ou, Z. (2018). Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(2), 1–26.
- Klee, V., & Minty, G. J. (1972). How good is the simplex algorithm. *Inequalities*, 3(3), 159–175.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.
- Ku, M., Li, T., Zhang, K., Lu, Y., Fu, X., Zhuang, W., & Chen, W. (2024). Imagenhub: Standardizing the evaluation of conditional image generation models. In *The twelfth international conference on learning representations*. Consulté sur <https://openreview.net/forum?id=OuV9ZrkQlc>
- Labrecque, J. S., Wood, W., Neal, D. T., & Harrington, N. (2017). Habit slips: When consumers unintentionally resist new products. *Journal of the Academy of Marketing Science*, 45, 119–133.
- Land, A. H., & Doig, A. G. (2009). An automatic method for solving discrete programming problems. In *50 years of integer programming 1958-2008: From the early years to the state-of-the-art* (pp. 105–132). Springer.
- La Quatra, M., Koudounas, A., Vaiani, L., Baralis, E., Cagliero, L., Garza, P., & Siniscalchi, S. M. (2024). Benchmarking representations for speech, music, and acoustic events. In *2024 ieee international conference on acoustics, speech, and signal processing workshops (icasspw)* (p. 505-509). doi: 10.1109/ICASSPW62465.2024.10625960
- LeCun, Y., Denker, J., & Solla, S. (1989). Optimal brain damage. *Advances in neural information processing systems*, 2.

- Lee, D., & Song, M. (2021). Quality-oriented task allocation and scheduling in transcoding servers with heterogeneous processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(3), 1667–1680.
- Lee, H., Calvin, K., Dasgupta, D., Krinner, G., Mukherji, A., Thorne, P., Trisos, C., Romero, J., Aldunce, P., Barret, K., et al. (2023). Ipcc, 2023: Climate change 2023: Synthesis report, summary for policymakers. contribution of working groups i, ii and iii to the sixth assessment report of the intergovernmental panel on climate change [core writing team, h. lee and j. romero (eds.)]. ipcc, geneva, switzerland.
- Li, B., Samsi, S., Gadepally, V., & Tiwari, D. (2023). Clover: Toward sustainable ai with carbon-aware machine learning inference service. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–15).
- Lin, J., Chen, W.-M., Cohn, J., Gan, C., & Han, S. (2020). Mcunet: Tiny deep learning on iot devices. In *Annual conference on neural information processing systems (neurips)*.
- Lin, J., Li, M., Zhang, S. Q., & Leon-Garcia, A. (2024). Murmuration: On-the-fly dnn adaptation for slo-aware distributed inference in dynamic edge environments. In *Proceedings of the 53rd international conference on parallel processing* (pp. 792–801).
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proc. intl. conf. on machine learning (icml)* (p. 157-163).
- Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1), 46–61.
- Liu, P., Jiang, J., Zhu, G., Cheng, L., Jiang, W., Luo, W., Du, Y., & Wang, Z. (2022). Training time minimization for federated edge learning with imized gradient quantization and bandwidth allocation. *Frontiers of Information Technology & Electronic Engineering*, 23(8), 1247–1263.
- Liu, Q., & Fang, Z. (2023). Learning to schedule tasks with deadline and throughput constraints. In *Ieee infocom* (pp. 1–10).
- Liu, S., Yu, G., Yin, R., Yuan, J., Shen, L., & Liu, C. (2021). Joint model pruning and device selection for communication-efficient federated edge learning. *IEEE Transactions on Communications*, 70(1), 231–244.
- Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.
- Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proc. neural information processing systems (neurips)*. Consulté sur <http://papers.nips.cc/paper/770-packet-routing-in-dynamically-changing-networks-a-reinforcement-learning-approach>
- Luccioni, A. S., Strubell, E., & Crawford, K. (2025). From efficiency gains to rebound effects: The problem of jevons’ paradox in ai’s polarized environmental debate. In *Proceedings of the 2025 acm conference on fairness, accountability, and transparency* (pp. 76–88).
- Luccioni, A. S., Viguiet, S., & Ligozat, A.-L. (2023). Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253), 1–15.
- Luccioni, S., Jernite, Y., & Strubell, E. (2024). Power hungry processing: Watts driving the cost of ai deployment? In *The 2024 acm conference on fairness, accountability, and transparency* (pp. 85–99).

- Lumezanu, C., Baden, R., Spring, N., & Bhattacharjee, B. (2009). Triangle inequality and routing policy violations in the internet. In *Passive and active network measurement: 10th international conference, pam 2009, seoul, korea, april 1-3, 2009. proceedings 10* (pp. 45–54).
- Ma, H., Li, R., Zhang, X., Zhou, Z., & Chen, X. (2023). Reliability-aware online scheduling for dnn inference tasks in mobile edge computing. *IEEE Internet of Things Journal*.
- Makhorin, A. (s. d.). *GNU Linear Programming Kit*. <http://www.gnu.org/software/glpk/glpk.html>. Consulté sur <http://www.gnu.org/software/glpk/> (Version 5.0 (or specify the version you used))
- Manfredi, V., Wolfe, A. P., Wang, B., & Zhang, X. (2021-06). Relational deep reinforcement learning for routing in wireless networks. In *2021 IEEE 22nd international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)* (pp. 159–168). doi: 10.1109/WoWMoM51794.2021.00029
- McDonald, J., Li, B., Frey, N., Tiwari, D., Gadepally, V., & Samsi, S. (2022). Great power, great responsibility: Recommendations for reducing energy for training language models. *arXiv preprint arXiv:2205.09646*.
- Mehta, S., Rastegari, M., Shapiro, L., & Hajishirzi, H. (2019). Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9190–9200).
- Mestres, A., Rodriguez-Natal, A., Carner, J., Barlet-Ros, P., Alarcón, E., Solé, M., Muntés-Mulero, V., Meyer, D., Barkai, S., Hibbett, M. J., Estrada, G., Ma'ruf, K., Coras, F., Erman, V., Latapie, H., Cassar, C., Evans, J., Maino, F., Walrand, J., & Cabellos, A. (2017, sep). Knowledge-defined networking. *SIGCOMM Comput. Commun. Rev.*, 47(3), 2–10. doi: 10.1145/3138808.3138810
- Meta AI. (2024, August). *Llama Usage Doubled May Through July 2024*. Consulté sur <https://ai.meta.com/blog/llama-usage-doubled-may-through-july-2024/> (Accessed: 2025-05-27)
- Mitchell, T. (1997). Machine learning. *Publisher: McGraw Hill*, 31.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015, février). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. doi: 10.1038/nature14236
- Mo, X., & Xu, J. (2021). Energy-efficient federated edge learning with joint communication and computation design. *Journal of Communications and Information Networks*, 6(2), 110–124.
- Morand, C., Ligozat, A.-L., & Névéol, A. (2024). Mlca: a tool for machine learning life cycle assessment. In *2024 10th international conference on ict for sustainability (ict4s)* (pp. 227–238).
- Muennighoff, N., Tazi, N., Magne, L., & Reimers, N. (2022). Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*. Consulté sur <https://arxiv.org/abs/2210.07316> doi: 10.48550/ARXIV.2210.07316
- Mukhutdinov, D., Filchenkov, A., Shalyto, A., & Vyatkin, V. (2019). Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system. *Future Generation Computer Systems*, 94, 587–600. doi: <https://doi.org/10.1016/j.future.2018.12.037>

- Nigade, V., Bauszat, P., Bal, H., & Wang, L. (2022). Jellyfish: Timely inference serving for dynamic edge networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)* (pp. 277–290).
- nsnam. (s. d.). *Ns-3 documentation website*. Consulté le 2022-03-14, sur <https://www.nsnam.org/documentation/>
- NVIDIA, S. (2022). *Nvidia management library (nvm)*.
- Ogden, S. S., & Guo, T. (2018). {MODI}: Mobile deep inference made efficient by edge computing. In *Usenix workshop on hot topics in edge computing (hotedge 18)*.
- Ooms, J. (2014). The opencpu system: Towards a universal interface for scientific computing through separation of concerns. *arXiv preprint arXiv:1406.4806*.
- Openai baselines: high-quality implementations of reinforcement learning algorithms. (s. d.). Consulté sur <https://github.com/openai/baselines> (11/05/22)
- Papers with code: The latest in machine learning. (s. d.). <https://paperswithcode.com/>. (Accessed: 2024-08-06)
- Paris, A. (2015). Paris agreement to the united nations framework convention on climate change. *Adopted Dec, 12*.
- Patel, D., Nishball, D., & Ontiveros, J. E. (2024). Ai datacenter energy dilemma-race for ai datacenter space. *SemiAnalysis, March, 13*.
- Patterson, D., Gonzalez, J., Hölzle, U., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D. R., Texier, M., & Dean, J. (2022). The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7), 18–28.
- Perrault, R., & Clark, J. (2024). *Artificial intelligence index report 2024*. < bound method Organization. get_name_with_acronym of< Organization ...
- Petit, B. (2023). *scaphandre*. Consulté sur <https://github.com/hubblo-org/scaphandre>
- Pineau, J., Vincent-Lamarre, P., Sinha, K., Lariviere, V., Beygelzimer, A., d’Alche Buc, F., Fox, E., & Larochelle, H. (2021). Improving reproducibility in machine learning research(a report from the NeurIPS 2019 reproducibility program). *Journal of Machine Learning Research*, 22(164), 1–20.
- Prisma tool: An open marl framework for packet routing. (s. d.). Consulté sur <https://github.com/rapariciopardo/PRISMA> (11/05/22)
- Rafael Padilla, A. R., & the Hugging Face Team. (2023). *Open object detection leaderboard*. https://huggingface.co/spaces/rafaelpadilla/object_detection_leaderboard. Hugging Face.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140), 1–67.
- Raj, S., Gupta, H., & Simmhan, Y. (2023). Scheduling dnn inferencing on edge and cloud for personalized uav fleets. In *Ieee/acm 23rd international symposium on cluster, cloud and internet computing (ccgrid)* (pp. 615–626).
- Ram, S., & Sheth, J. N. (1989). Consumer resistance to innovations: the marketing problem and its solutions. *Journal of consumer marketing*, 6(2), 5–14.

- Ribeiro, M., Grolinger, K., & Capretz, M. A. (2015). Mlaas: Machine learning as a service. In *2015 IEEE 14th international conference on machine learning and applications (icmla)* (pp. 896–902).
- Rodriguez, C., Degioanni, L., Kameni, L., Vidal, R., & Neglia, G. (2024). Evaluating the energy consumption of machine learning: Systematic literature review and experiments. *arXiv preprint arXiv:2408.15128*.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115, 211–252.
- Sachdeva, S. (2013). Faster Algorithms via Approximation Theory. *Foundations and Trends® in Theoretical Computer Science*, 9(2), 125–210. doi: 10.1561/04000000065
- Salem, T. S., Castellano, G., Neglia, G., Pianese, F., & Araldo, A. (2021). Towards inference delivery networks: Distributing machine learning with imality guarantees. In *2021 19th mediterranean communication and computer networking conference (medcomnet)* (pp. 1–8).
- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
- Schapire, R. E., & Freund, Y. (2012). *Boosting: Foundations and algorithms*. Cambridge, MA : MIT Press.
- Schmidt, V., Goyal, K., Joshi, A., Feld, B., Conell, L., Laskaris, N., Blank, D., Wilson, J., Friedler, S., & Luccioni, S. (2021). Codecarbon: estimate and track carbon emissions from machine learning computing. *Cited on*, 20.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green ai. *Communications of the ACM*, 63(12), 54–63.
- Sensor Tower. (2025). *State of mobile 2025*. <https://sensortower.com/state-of-mobile-2025>. (Accessed: 2025-06-22)
- Shabtay, D., & Kaspi, M. (2006). Parallel machine scheduling with a convex resource consumption function. *European Journal of Operational Research*, 173(1), 92–107.
- Shakarami, A., Ghobaei-Arani, M., & Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, 182, 107496.
- Shao, J., & Zhang, J. (2020). Communication-computation trade-off in resource-constrained edge inference. *IEEE Communications Magazine*, 58(12), 20–26.
- Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., Li, Z., Zhang, K., Gao, P., Qiao, Y., & Luo, P. (2023). Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.
- Shehabi, A., Smith, S. J., Hubbard, A., Newkirk, A., Lei, N., Siddik, M. A., Holecek, B., Koomey, J. G., Masanet, E. R., & Sartor, D. A. (2024, 19/12/2024). *2024 united states data center energy usage report*. Lawrence Berkeley National Laboratory. Consulté

sur <https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report.pdf>

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017, octobre). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359. doi: 10.1038/nature24270

SimilarWeb. (2025). *Website traffic analysis*. Consulté sur <https://www.similarweb.com> (Accessed: 2025-03-20)

Sndlib: Library of test instances for survivable fixed telecommunication network design. (s. d.). Consulté sur <http://sndlib.zib.de>

Solaiman, I., Talat, Z., Agnew, W., Ahmad, L., Baker, D., Blodgett, S. L., Chen, C., Daumé III, H., Dodge, J., Duan, I., et al. (2023). Evaluating the social impact of generative ai systems in systems and society. *arXiv preprint arXiv:2306.05949*.

Srivastav, V., Majumdar, S., Koluguri, N., Moumen, A., Gandhi, S., et al. (2023). *Open automatic speech recognition leaderboard*. https://huggingface.co/spaces/hf-audio/open_asr_leaderboard. Hugging Face.

Strubell, E., Ganesh, A., & McCallum, A. (2020). Energy and policy considerations for modern deep learning research. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 13693–13696).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Svorobej, S., Bendechache, M., Griesinger, F., & Domaschka, J. (2020). Orchestration from the cloud to the edge. *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, 61–77.

Tao, Y., Ma, R., Shyu, M.-L., & Chen, S.-C. (2020). Challenges in energy-efficient deep neural network training with fpga. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition workshops* (pp. 400–401).

Tassioulas, L., & Ephremides, A. (1992). Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE transactions on automatic control*, 37(12), 1936–1948.

Tensorboard: Tensorflow's visualization toolkit. (s. d.). Consulté sur <https://www.tensorflow.org/tensorboard> (11/05/22)

Tensorflow: An end-to-end open source machine learning platform. (s. d.). Consulté sur <https://www.tensorflow.org/> (11/05/22)

Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 10.

Tsao, Y.-C., Thanh, V.-V., & Hwang, F.-J. (2020). Energy-efficient single-machine scheduling problem with controllable job processing times under differential electricity pricing. *Resources, Conservation and Recycling*, 161, 104902.

Varoquaux, G., Luccioni, A. S., & Whittaker, M. (2024). Hype, sustainability, and the price of the bigger-is-better paradigm in ai. *arXiv preprint arXiv:2409.14160*.

Vickson, R. (1980). Two single machine sequencing problems involving controllable job processing times. *AIIE transactions*, 12(3), 258–262.

- Wang, Y., Karimi, M., Xiang, Y., & Kim, H. (2021). Balancing energy efficiency and real-time performance in gpu scheduling. In *2021 IEEE Real-time Systems Symposium (RTSS)* (pp. 110–122).
- Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. (Thèse de doctorat non publiée). University of Cambridge.
- Wightman, R. (2019). *Pytorch image models*. <https://github.com/rwightman/pytorch-image-models>. GitHub. doi: 10.5281/zenodo.4414861
- Williamson, D. P., & Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press.
- Wu, B., Xu, C., Dai, X., Wan, A., Zhang, P., Yan, Z., Tomizuka, M., Gonzalez, J., Keutzer, K., & Vajda, P. (2020). *Visual transformers: Token-based image representation and processing for computer vision*.
- Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., et al. (2022). Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4, 795–813.
- Wu, R., Bao, W., & Ge, L. (2023). Online task assignment with controllable processing time. *arXiv preprint arXiv:2305.04453*.
- Wu, X., & Che, A. (2019). A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, 82, 155–165.
- Xu, Z., Zhao, L., Liang, W., Rana, O. F., Zhou, P., Xia, Q., Xu, W., & Wu, G. (2020). Energy-aware inference offloading for dnn-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(4).
- Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Zhong, S., Yin, B., & Hu, X. (2024). Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6), 1–32.
- Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., & Hua, X.-s. (2019). Quantization networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7308–7316).
- Yang, Z., Cui, Y., Li, B., Liu, Y., & Xu, Y. (2019). Software-defined wide area network (sd-wan): Architecture, advances and opportunities. In *2019 28th international conference on computer communication and networks (icccn)*.
- Yin, H., Liu, P., Liu, K., Cao, L., Zhang, L., Gao, Y., & Hei, X. (2020). Ns3-ai: Fostering artificial intelligence algorithms for networking research. In *Proc. acm 2020 workshop on ns-3 (wns3)* (p. 57–64). New York, NY, USA. doi: 10.1145/3389400.3389404
- Yin, L., Li, X., Lu, C., & Gao, L. (2016). Energy-efficient scheduling problem using an effective hybrid multi-objective evolutionary algorithm. *Sustainability*, 8(12), 1268.
- You, J., Chung, J.-W., & Chowdhury, M. (2023). Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *Usenix nsdi*.
- You, X., Li, X., Xu, Y., Feng, H., Zhao, J., & Yan, H. (2020). Toward packet routing with fully distributed multiagent deep reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 1-14. doi: 10.1109/TSMC.2020.3012832
- You, X., Li, X., Xu, Y., Feng, H., Zhao, J., & Yan, H. (2022, février). Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning. *IEEE Transactions on Systems,*

Man, and Cybernetics: Systems, 52(2), 855–868. (Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems) doi: 10.1109/TSMC.2020.3012832

Yu, J., & Huang, T. (2019a). *Autoslim: Towards one-shot architecture search for channel numbers*. arXiv. Consulté sur <https://arxiv.org/abs/1903.11728> doi: 10.48550/ARXIV.1903.11728

Yu, J., & Huang, T. S. (2019b). Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1803–1811).

Yu, J.-R., Chen, C.-H., Huang, T.-W., Lu, J.-J., Chung, C.-R., Lin, T.-W., Wu, M.-H., Tseng, Y.-J., & Wang, H.-Y. (2022). Energy efficiency of inference algorithms for clinical laboratory data sets: Green artificial intelligence study. *Journal of Medical Internet Research*, 24(1), e28036.

Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., Wei, C., Yu, B., Yuan, R., Sun, R., Yin, M., Zheng, B., Yang, Z., Liu, Y., Huang, W., Sun, H., Su, Y., & Chen, W. (2024). Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (cvpr)*.

Zamora, I., Lopez, N. G., Vilches, V. M., & Cordero, A. H. (2016). Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*.

Zeng, D., Ansari, N., Montpetit, M.-J., Schooler, E. M., & Tarchi, D. (2021). Guest editorial: In-network computing: Emerging trends for the edge-cloud continuum. *IEEE Network*, 35(5), 12–13.

Zero mq: An open-source universal messaging library. (s. d.). Consulté sur <https://zeromq.org/> (11/05/22)

Zhang, B., Davoodi, A., & Hu, Y. H. (2018). Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4), 836–848.

Zhang, H., Ananthanarayanan, G., Bodik, P., Philipose, M., Bahl, P., & Freedman, M. J. (2017). Live video analytics at scale with approximation and {Delay-Tolerance}. In *14th unix symposium on networked systems design and implementation (nsdi 17)* (pp. 377–392).

Zhang, Y.-W., Chen, R.-K., & Gu, Z. (2023). Energy-aware partitioned scheduling of imprecise mixed-criticality systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. (2024). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., & Torralba, A. (2017, July). Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition (cvpr)*.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 35, pp. 11106–11115).

Zhu, M., & Gupta, S. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

My publications

Alliche, R. A., Barros, T. D. S., Aparicio-Pardo, R., & Sassatelli, L. (2022, September). Impact evaluation of control signalling onto distributed learning-based packet routing. In *2022 34th international teletraffic congress (itc-34)*. Shenzhen, China.

Alliche, R. A., Barros, T. D. S., Aparicio-Pardo, R., & Sassatelli, L. (2023). prisma-v2: Extension to cloud overlay networks. In *2023 23rd international conference on transparent optical networks (icton)* (pp. 1–4).

Alliche, R. A., Da Silva Barros, T., Aparicio-Pardo, R., & Sassatelli, L. (2022, juin). PRISMA: a packet routing simulator for Multi-Agent reinforcement learning. In *2022 ifip networking wkshps network intelligence*. Catania, Italy.

Barros, T. D. S., Giroire, F., Aparicio-Pardo, R., Perennes, S., & Natale, E. (2024). Scheduling with fully compressible tasks: Application to deep learning inference with neural network compression. In *2024 ieee 24th international symposium on cluster, cloud and internet computing (ccgrid)* (pp. 327–336).

da Silva Barros, T., Ferre, D., Giroire, F., Aparicio-Pardo, R., & Perennes, S. (2024). Scheduling machine learning compressible inference tasks with limited energy budget. In *Proceedings of the 53rd international conference on parallel processing* (pp. 961–970).

da Silva Barros, T., Giroire, F., Aparicio-Pardo, R., & Moulrierac, J. (2025). Small is sufficient: Reducing the world ai energy consumption through model selection. *Under analysis*.

List of figures

1.1	ML models phases during its life cycle	1
1.2	Evolution of AI (a) number of publications and (b) number of patents over time indicating a significant increase of AI popularity.	2
1.3	ML models are deployed along the cloud-to-edge continuum	3
1.4	Number of parameters of foundation models over time	5
1.5	Scheduling scheme system example.	7
1.6	Evolution of <i>Image Classification</i> models over time. Accuracy (%) of image classification models on the <i>ImageNet</i> dataset plotted against their publication date. Each point corresponds to one model. Each data point represents a model, with the marker size proportional to the number of parameters. The plot shows an accuracy "plateau" post-2020, with the continued release of models across various sizes. Data extracted from the <i>Papers With Code</i> platform.	8
2.1	LP problem example. The problem, described in Eq. (2.4) contains two variables and two constraints. The red dashed line represents the objective function. The feasible space is shaded and the optimal point is highlighted.	20
2.2	MILP problem example. The variables are now integers. The points in blue compose the feasible solution set. The lines in blue and orange are the problem constraints. We display two values for the objective function: (i) the solution for the fractional problem, in red and dotted line, with value 45.41; and (ii) the solution for the integral problem (MILP), in red and dashed line, with value 37.20.	21
2.3	Example of three convex functions: (a) exponential, (b) linear, and (c) quadratic	22
2.4	Convex and non-convex sets.	22
3.1	Pruning method description diagram. At each round of pruning (shrinking the model), the weights need to be retrained.	29
3.2	Machine Learning compression during inference time (Cai et al., 2020; J. Yu & Huang, 2019a). The search for the best sub-network is done at inference time, without requiring retraining the model.	31
4.1	Accuracy vs. Latency using the OFA framework for classifying a set of 1000 images from <i>imagenet-1k</i> . Experiments on a CPU (left) and GPU (right).	39
4.2	Scheduling illustrative example with $n = 4$ tasks and $m = 3$ machines.	41
4.3	Comparison between OFA experiments and the exponential function for accuracy-latency tradeoff	42
4.4	Optimality gap (total accuracy difference between DSCT-UB and DSCT-APPROX) over 100 experiments when varying (a) the task heterogeneity ratio μ , (b) number of machines, (c) deadline tolerance value, and (d) number of tasks. Absolute guarantee G is given as a baseline.	58
4.5	Execution times of DSCT-APPROX vs DSCT-Opt for instances with increasing (a) numbers of tasks, with $m = 5$ and (b) number of machines, with $n = 50$	60

4.6	Average task accuracy for DSCT-APPROX and the baselines as a function of the deadline tolerance level ρ , for $m = 10$	60
4.7	Scheduling Fairness: Heat maps represent the distribution of task accuracy values for sets of experiments with different tolerance ρ . The colors represent the percentage of tasks within an accuracy range.	62
5.1	Energy efficiency vs speed for various GPUs (Desislavov et al., 2023).	69
5.2	Once-for-All (Cai et al., 2020): Accuracy vs number of floating operations	69
5.3	Optimality gap (average accuracy difference between DSCT-EA-UB and DSCT-EA-APPROX) over 100 experiments with varying task heterogeneity.	81
5.4	Execution times of DSCT-EA-APPROX vs DSCT-EA-Opt (ApS, 2022) for instances with increasing (a) numbers of tasks, with $m = 5$ and (b) number of machines, with $n = 50$	82
5.5	Average task accuracy for DSCT-EA-APPROX and the baselines over energy budget ratio β , for $m = 2$ and $n = 100$	83
5.6	Energy profile of 2 machines when varying the energy budget ratio. Machine 1 is more energy efficient than machine 2.	84
6.1	Research Focus by Area on Papers With Code: Numbers of papers across different AI research areas on the <i>Papers With Code</i> platform. The results highlight significant interest in Computer Vision (image processing), Natural Language Processing, and a diverse range of Multimodal tasks categorized as Miscellaneous. .	90
6.2	Most popular AI fields and tasks. Number of total model downloads on the <i>Hugging Face</i> platform, categorized by AI field and AI task. The dashed gray line represents the download threshold used for task evaluation in this study, encompassing 97.3% of all platform downloads.	91
6.3	Tradeoff AI model utility vs size. Model utility value versus number of parameters for popular AI tasks. Each point is a single model and its size is proportional to the model popularity. The green and red points indicate the energy-efficient and the best-performing models, respectively. An interactive version of the plots can be found in https://tsb4.github.io/HF/	96
6.4	Law of diminishing return. Diminishing Marginal Gains in AI Utility with Increasing Model Parameters. The relationship exhibits three phases: (i) Scaling Model: high initial marginal utility gain; (ii) Diminishing Returns: decreasing marginal gain with larger models; and (iii) Flat Utility: minimal marginal gain approaching a plateau.	98

6.5	The Four-Stage Development of AI Tasks: Balancing Accuracy and Model Size. Top (6.5a). The four stages of development of an AI task, illustrating the evolving trade-off between accuracy and model size over time. Middle (6.5b) An example of these stages for <i>Image Classification</i> . Each point represents a model. The size of the orange circle in Stage 4 represents the model usage (number of downloads). Bottom (6.5c): Current development stages of major AI tasks. In Stage 1, the initial foundation models are developed. In Stage 2, researchers focus on improving utility, leading to larger and performing models. In Stage 3, efficiency is introduced and smaller models with similar performance emerge. Last, in Stage 4, users partially adopt energy-efficient models.	99
6.6	Validation of the <i>CabonTracker</i> tool. Comparison of energy consumption measurements obtained using the software-based tool <i>CabonTracker</i> (CT) and the <i>PowerSpy</i> power meter device, across models from several tasks. In blue, we show the <i>CabonTracker</i> measurements and in red, the <i>PowerSpy</i> measurements.	104
6.7	Modeling AI model energy consumption. Inference energy consumption versus model size (number of parameters) for <i>Image Classification</i> (vision), <i>Speech Recognition</i> (audio), and <i>Text Generation</i> (language) tasks. Median energy values from 10 repeated inference requests are plotted for each data point.	105
6.8	Energy and Utility Impact of Switching to Efficient Inference Models. This figure presents the percentage difference in energy reduction and utility when shifting AI inference from the <i>best-performingmodel</i> to the <i>energy-efficientmodel</i> for various tasks. The rightmost bars illustrate the weighted average across all considered AI tasks, with weights based on their usage (number of downloads). Notably, these values also represent the energy savings achieved by adopting model selection (as detailed in Figure 6.11) compared to a pessimistic scenario where all inferences are performed using the <i>best-performingmodels</i> on high-performance hardware.	107
6.9	Trade-off: Energy Savings vs. Maximum Utility Drops. Global estimated energy savings (%) and utility variation (%) under different maximum utility drop thresholds when choosing the <i>energy-efficient</i> model. Results are shown (a) considering only the two key models (<i>best-performing</i> and <i>energy-efficient</i>), and (b) considering all models. For each threshold, the presented value is a download-weighted average across all tasks.	108
6.10	Estimated Impact of Efficient Model Selection on AI Inference. Estimated percentage reduction in energy consumption and the percentage variation in utility for major AI tasks, when applying model selection during inference. For each task, inferences with models larger than the identified <i>energy-efficientmodel</i> are redirected to it. The rightmost bar (i.e., the bar in orange and in red) represents the weighted average across all considered AI tasks weighted by their usage (number of downloads).	110

6.11	Projected US Data Center Energy Consumption for AI Inference Under 3 Different Scenarios. US data centers AI inference energy projection over time for three different scenarios: (i) business as usual, (ii) sobriety using model selection, (iii) use of <i>best-performingmodel</i> with the needed hardware. The values for (i) were extracted from (Shehabi et al., 2024). We considered a one-year transition to reach scenario (ii), applying model selection globally, or (iii), using the <i>best-performingmodel</i> , (during the year 2025).	111
7.1	Plant identification path using PI@ntnet service: a user makes an identification request, which is forwarded to PI@ntnet datacenter; the request is saved on the database and the ML model inference is executed on a GPU card; then, the results are processed and returned to the end user.	118
7.2	PI@ntNet service architecture. We divide the service into five sectors: <i>End Users</i> , <i>Network</i> , <i>Training</i> , <i>Inference</i> , and <i>Webhosting</i> (API and storage).	123
7.3	Environmental impacts proportion for each evaluated section of PI@ntNet service.	132
7.4	Embodied and operational environmental impacts proportions over different sections in PI@ntNet.	132
7.5	GWP Impact for <i>End Users</i> section during usage (OPEX) for different countries. The countries are ordered by number of sessions in non-increasing order.	133
7.6	Sensitivity analysis over the identification request duration. We varied the duration in a range from 0.5s to 2.5s and we display the (a) ADP, (b) GWP and (c) PE impacts associated with PI@ntNet <i>End Users</i>	134
7.7	GWP Impact proportion between <i>embodied</i> and <i>operational</i> stages. We test the stages ratio considering the <i>Legacy</i> scenario.	136
7.8	<i>Inference</i> environmental impacts per machine. The <i>capcir</i> and <i>orxois</i> machines use a Nvidia Quadro M4000; <i>huchet</i> and <i>rajol</i> use a NVIDIA RTX 8000; and <i>chamois</i> , <i>devois</i> , <i>senart</i> and <i>sourdun</i> use a NVIDIA A40 GPU.	138
A.1	PRISMA code structure, separated in two parts : Python code and C++ code, highlighting our contribution in green. The figure shows the multi-threaded approach, where each node agent is composed of two threads: one forwarding the packets and another training the model.	177
A.2	Screenshot of <i>tensorboard</i> interface. The x axis represents the time in ns , and the y axis represents the variables tracked during the simulation. For the training metrics (bottom subfigures), each curve corresponds to an agent.	180
A.3	The average cost per packet and TD errors at each node during training for Abilene (rightmost subfigures) and Geant (leftmost subfigures) topology.	181
A.4	The average end-to-end delay per arrived packet and packet loss ratio for different traffic rate intensities for Abilene (right row) and Geant (left row) topology	181
A.5	prisma-v2 code structure	183
A.6	NN layers architecture	191
A.7	Avg. Cost Per Packet vs Replay Memory Size	196
A.8	Avg. Cost Per Packet vs Target Update Period	196
A.9	Control Overhead vs Target Update Period	196
A.10	Avg. Cost Per Packet vs Control Overhead	196
A.11	Average Cost Per Packet.	198

A.12 Packet Loss Ratio	198
A.13 Avg. End-to-End Delay.	198

List of Algorithms

1	Exact algorithm for scheduling in one machine	46
2	Exact algorithm for solving the fractional relaxation	52
3	DSCT-APPROX: Approximation algorithm for scheduling in several machines .	52
4	Exact algorithm for scheduling on one machine using piecewise linear accuracy functions	74
5	ComputeNaiveSolution: Computes an optimal scheduling for the naive energy profile	76
6	RefineProfile: Adapts the naive profile in order to find the optimal solution	77
7	DSCT-EA-FR-OPT: Computes the optimal solution for DSCT-EA-FR using Piecewise Linear function as accuracy function	78
8	DSCT-EA-APPROX: Approximation algorithm for the scheduling problem on several machines	79
9	DQN Routing training routine at a node n with value sharing (or with model sharing)	192

Appendix

APPENDIX A

PRISMA

In this appendix, we describe the additional contribution made during my Master’s thesis and at the beginning of my PhD thesis, when I collaborated with Redha A. Alliche, within the framework of the Artificial Intelligence-based Cloud Network Control (ARTIC) project ([Agence Nationale de la Recherche, 2019](#)). We addressed, in this project, the DPR problem in network environments and its integration with ML-based approaches. In literature, Deep Reinforcement Learning (DRL) methods for DPR are often evaluated using ad-hoc and discrete-time simulators, which present several simplifications; for example, agents are generally not executed in separate threads, and only one packet transmitted is allowed per router per time step. As a result, a major challenge in this field is the lack of realistic and reproducible RL-based simulation environments specifically designed for DPR. For overcoming these challenges, we proposed the PRISMA, a network simulation framework for DPR problem capable of integrating with Multi-Agent Deep Reinforcement Learning (MA-DRL). Then, we evaluated the impact of control signalling mechanisms on the network system.

A.1 Simulation Framework

A.1.1 Introduction

In the last years, Reinforcement Learning (RL) (Sutton & Barto, 2018) using Deep Neural Networks (DNNs) (Bengio, 2009), also called Deep Reinforcement Learning (DRL), has obtained ground-breaking results in solving highly complex tasks, such as human-like performance results at Atari video games (Mnih et al., 2015) or beating AlphaGo (Silver et al., 2017) world champion. In communication networks, DRL has also been widely used in many networking technologies and problems. One of them is the Distributed Packet Routing (Mukhutdinov, Filchenkov, Shalyto, & Vyatkin, 2019; You et al., 2020; L. Chen, Hu, Guan, Zhao, & Shen, 2021; Manfredi, Wolfe, Wang, & Zhang, 2021-06). In this problem, no complete and centralized view of network topology and traffic demands is available, which poses a challenge. This is the case in multi-hop wireless networks (Tassiulas & Ephremides, 1992) or in multi-domain optical networks (X. Chen et al., 2019). More precisely, in the DPR problem, each packet can be potentially routed differently regardless of the flow they belong to. Besides, the per-packet decisions are made locally by distributed agents placed at the routing nodes. These agents exploit local information, such as packet headers and neighboring nodes and link states.

However, in all the above-mentioned works, the proposed DRL approaches were evaluated on ad hoc discrete-time packet-level simulation environments (typically implemented in `python`) tailored to the assumptions and simplifications made by each study. Namely, these works do not generally implement the RL agents as separated threads or processes (although they claim to be multi-agent studies); and, they usually assume that at most one packet per router can be transmitted at each time step, which introduces an artificial synchronization into the routers' dynamics. This has two main consequences: (i) a reduced realism of the simulated communication process; and, (ii) an obstacle to fairly comparing these DRL proposals to state-of-the-art approaches or even against each other. This lack of standardized Machine Learning (ML) tools in the networking community (Mestres et al., 2017; Gawłowicz & Zubow, 2019) represents a major issue to guarantee reproducibility. We point out that ML reproducibility issues are becoming a serious concern (Pineau et al., 2021).

As a consequence, in the last years, some tools (Gawłowicz & Zubow, 2019; H. Yin et al., 2020) devoted to the application of RL to networks have been developed. These tools allow `python` RL agents to interact with the popular `ns-3` network simulator (nsnam, s. d.). Nevertheless, these proposals are not natively compatible with the *collaborative and distributed multi-agent* setting of the DPR problem, where agents collaborate to take decisions in a distributed manner.

In this appendix, we propose an open-source RL simulation environment designed to overcome the aforementioned drawbacks in the application of DRL to the DPR problem. We bring the following main contributions:

1. a RL framework designed specifically for considering the distinctive characteristics of the DPR problem, serving as a playground where the community can easily validate their RL approaches and compare them.
2. more realistic modelling of the communication process based on: (i) the `ns-3` (nsnam, s. d.) network simulator; and, (ii) a multi-threaded implementation for each agent.
3. a modular code design, which allows researchers to test their own RL algorithm, without needing to work on the implementation of the environment.

4. visual tools based on `tensorboard` (*TensorBoard: TensorFlow’s visualization toolkit, s. d.*) allowing to track training and test phases.

The rest of this Appendix A.1 is organized as follows. Section A.1.2 discusses the related works. In Section A.1.3, we present the appendix’s background. Section A.1.4 presents PRISMA. The simulator usage is detailed in Section A.1.5 and illustrated in Section A.1.6. Section A.1.7 introduces an extension of the PRISMA framework for addressing cloud overlay networks. Finally, we conclude the appendix in Section A.1.8.

A.1.2 Related works

Related works can be grouped into the next two categories.

DRL for packet routing. In recent years, the first works applying DRL to the DPR problem have been published. DPR has been studied in (Boyan & Littman, 1993; Mukhutdinov et al., 2019; You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06). The seminal paper (Boyan & Littman, 1993) was the first to apply a RL method, the Q-learning (Watkins, 1989), to DPR, giving rise to the *Q-routing* paradigm. More details about this paradigm will be given in the next section. The study in (Mukhutdinov et al., 2019) applies the Deep Q-Network (DQN) framework (i.e., a neural network fits the Q-value function) to the *Q-routing* algorithm, yielding to the *DQN routing*. In all these works, the node state only codes the destination of the packet to forward. More recently, works (You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06) extended the *DQN routing* by adding to the destination additional inputs, particularly node *buffer occupancy*. *DQRC (DQR with Communication)* (You et al., 2020) also considers *action history* and *future destinations*. Authors in (Manfredi et al., 2021-06) focus on the *relational features* (independent of the network graph), such as the *distance to the destination*. Finally, in (L. Chen et al., 2021), the meta-learning framework is applied to the *DQN routing* to learn different tasks, where a task corresponds to a given traffic matrix.

RL environments for networking. Only two tools have been developed to allow a Python agent to interact with a network simulation: first, `ns3-gym` (Gawłowicz & Zubow, 2019); and, more recently, `ns3-ai` (H. Yin et al., 2020). Both of them are based on the well-known `ns-3` network simulator, but they follow different approaches to implement a RL environment. First, `ns3-ai` is not only restricted to Gym as Python environment (as `ns3-gym` does). Second, it utilizes a shared memory pool as a mechanism to connect the `ns-3` simulator with the Python framework (agent and environment), unlike the `ns3-gym` sockets-based approach. The `ns-3`, Gym, and `ns3-gym` tools are detailed in the next section.

A.1.3 Background

In this section, we present the elements that PRISMA is built over: the RL framework used to solve the packet routing problem, the RL toolkits, and the `ns-3` network simulator.

A.1.3.1 Q-routing: Reinforcement Learning for Packet Routing

The DPR problem can be formalized as Multi-Agent Partially Observable Markov Decision Process (POMDP) (Littman, 1994): a Markov Decision Process where each agent can only locally observe its environment. The first RL method proposed to tackle the DPR was the *Q-routing* (Boyan & Littman, 1993), based on the classical *Q-learning* (Watkins, 1989), which we

will use to describe the POMDP and the RL approach used to solve it. Note that other RL methods could be also adapted to the DPR problem.

Let $\mathcal{N}$ be the set of routing nodes (*agents*), where each *agent* n has its own local observation space $\mathcal{O}_n$ and its own action space $\mathcal{A}_n$. When a new packet arrives at time t at the node n , the node n selects as action a_n the *next hop node* n' to forward the packet. This decision is taken depending on the local observation of the router o_n (e.g., *the current packet destination, the node buffers' occupancy, ...*). As a consequence, the *agent* n receives a reward r_n : the *next-hop packet delay* (i.e. the packet delay to travel from n to n'), and the whole network makes a transition to a new state. When this packet (or another one) arrives to a node, this procedure is repeated. The action a_n is selected to minimize an estimate of the expected *end-to-end packet delay* from node n to its final destination. Under the DRL scheme, this estimate, denoted as $Q_n(o_n, a_n; \theta_n)$ (the *Q-value*), is the output of a DNN with θ_n weights. This DNN is trained to fit the target value Y_n^Q below:

$$Y_n^Q = r_n + \gamma \cdot \tau \cdot (1 - f) \quad (\text{A.1})$$

where f indicates if the next hop is the packet destination, $\gamma \in [0, 1]$ is a discount factor, r is the *next-hop packet delay* and τ is *the remaining end-to-end delay* from the next hop n' to the destination computed as follows:

$$\tau = \min_{a_{n'} \in \mathcal{A}_{n'}} Q_{n'}(o_{n'}, a_{n'}; \theta_{n'}) \quad (\text{A.2})$$

where $Q_{n'}(\cdot; \theta_{n'})$ is the output of next hop agent. The θ_n weights are updated by stochastic gradient descent when minimizing the so-called *Temporal Difference (TD) error*: $(Q(s, a; \theta_n) - Y_n^Q)^2$.

A.1.3.2 The ns-3 Network Simulator

The ns-3 simulator ([nsnam](#), [s. d.](#)) is a stable discrete-event network simulator, largely adopted by the research and educational community and accepted as a standard. It is free and open-source under *GPLv2* and developed in C++ using object-oriented programming. The success of this tool is based on two main features: (i) a realistic modelling of the network systems: the protocol stack is properly abstracted by the simulator classes; (ii) a wide range of networking protocols (e.g., IP, TCP, UDP), communication technologies (e.g., Ethernet, Wi-Fi, LTE) and statistical models for channels, mobility, and traffic generation are supported by built-in classes. As a consequence, ns-3 becomes a natural choice to use as a network simulator for PRISMA.

A.1.3.3 Reinforcement Learning Tools

RL agents are usually implemented using Python packages (e.g., TensorFlow([TensorFlow: An end-to-end open source machine learning platform](#), [s. d.](#))) and interfaced with an environment where agents take decisions. The OpenAI™Gym ([Gym: Gym toolkit for creating reinforcement learning environments](#), [s. d.](#)) is one of the most popular toolkits to define RL environments in Python. Gym is used to test and compare different RL algorithms for various problems, such as Atari games and robotics ([Zamora, Lopez, Vilches, & Cordero, 2016](#)). Gym also provides a simple interface to pass the interactions between the agent and the environment, without making assumptions about the agent structure. Moreover, the Gym toolkit allows the creation of customized environments, which is very convenient since no network environment is natively available in Gym.

The first solution to this lack of networking RL environments was `ns3-gym` (Gawłowicz & Zubow, 2019), which interfaces OpenAI TMGym with the `ns-3` network simulator. In other words, `ns3-gym` "transforms" a `ns-3` network simulation into an RL environment by serving as gateway connecting `ns-3` with Gym. Then, it provides an easy-to-use platform for developing and testing RL algorithms for networking problems. The Python process (the agent and the Gym environment) communicates with the C++ process (the `ns-3` network simulation) via ZMQ sockets (*Zero MQ: An open-source universal messaging library, s. d.*). `ns3-gym` is also open source under a GPL licence and can be easily extended. In PRISMA, we opt for this tool (instead `ns3-ai`) since `ns3-gym` is better documented.

A.1.4 PRISMA description

In this section, we present the technical details of PRISMA. First, a description of the design of the framework is given. Then, each feature is explained, ranging from the network definition to simulation visualization using `tensorboard` (*TensorBoard: TensorFlow's visualization toolkit, s. d.*). The PRISMA source code is publicly available at (*PRISMA tool: An open MARL framework for packet routing., s. d.*).

A.1.4.1 Framework design principles

We consider the following principles:

1. *Training-Forwarding separation*: the agent training does not disturb the packet forwarding.
2. *Modularity*: code easy to reuse and modify.
3. *Realistic simulation*: implement the agent as close as possible to the real world.
4. *Fast prototyping*: easy and rapid modifications of the decision model.
5. *Online simulation tracking*: visualize in real-time the simulation evolution.

For point 1), we choose a multi-threading approach: each node will be run on two separate threads. One is dedicated to the training process, and; the other one to the decision process. Hence, the node agents can be trained at a timestamp without disturbing the action computation at each packet arrival.

For point 2), we built node agents in an object-oriented manner, with separated functions for each task, so that a user can easily modify the behaviour of the agent without affecting the rest of the code. We document each part of the code and provide scripts for running the simulation.

Point 3) is a very important aspect of the PRISMA design, since we need reliable results as close as possible to the deployment of the agents in the real world. For that, we keep the link propagation delay and use `ns-3` to simulate the network. We have also integrated the action handling to the environment and added a Poisson random traffic generator.

For point 4), PRISMA is separated into independent modules, so that a user may easily modify the code. For example, changing the neural network or the inputs of the model will not affect the other parts of the framework.

Finally, for point 5), we give to the user the ability to monitor the simulation progress in real-time. For example, the user can track some network or ML metric (e.g., average delay, buffers occupancy, error progression) to identify training issues in real-time. To do so, `tensorboard` has been integrated into the framework offering network monitoring. In addition, it adds the possibility to customize plots.

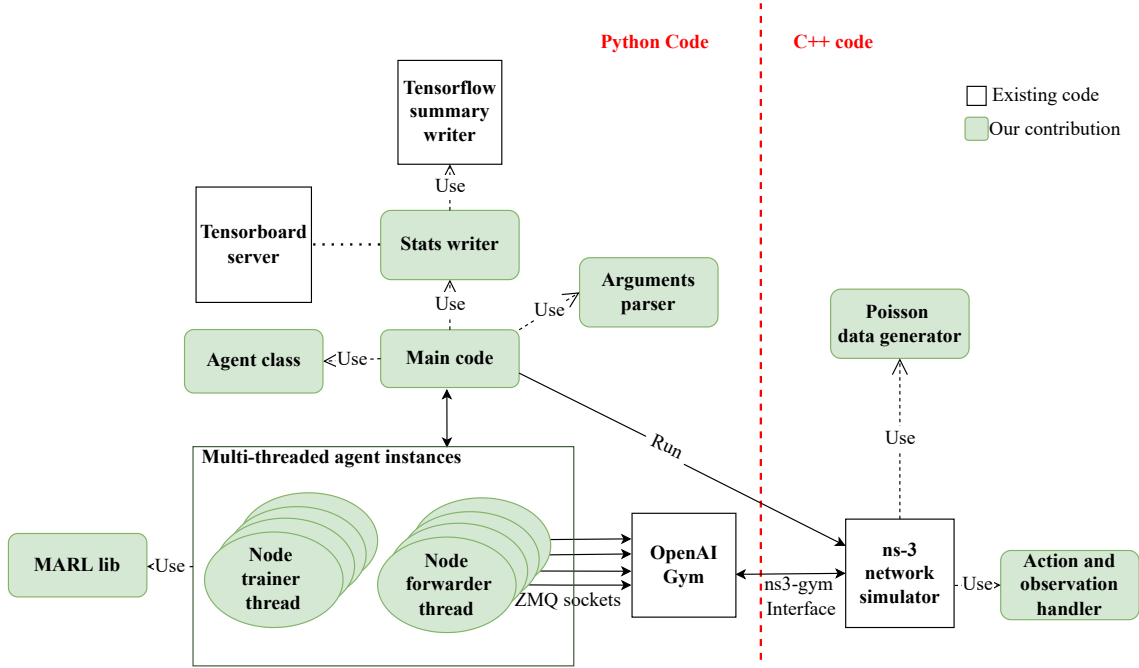


Figure A.1 – PRISMA code structure, separated in two parts : Python code and C++ code, highlighting our contribution in green. The figure shows the multi-threaded approach, where each node agent is composed of two threads: one forwarding the packets and another training the model.

A.1.4.2 Feature description

Figure A.1 depicts the PRISMA code structure, showing in *green* our contribution over the existing `ns-3` and `ns3-gym` modules. We describe next these *green* modules.

Main code. The core of the tool enables to:

- Gather the user arguments from the `Arguments parser` and manage the simulation parameters.
- Instantiate the agents from the `Agent class` and create the threads for each node.
- Run `ns-3` simulator and `tensorboard server` in separated processes.
- Log the simulation statistics using the `Stats writer`.

Agent class. This class represents a node agent, containing two main methods: `run_forwarder` and `run_trainer`. The first one interacts with the `ns3-gym` environment as follows: (i) retrieve the observations, (ii) take action, (iii) compute the reward, (iv) store the states' transitions in the experience replay buffers, and, (v) update the information about the environment. The second method trains the model performing a gradient descent step. The `run_forwarder` also tracks the packets using their IDs to build the target value Y_n^Q at the node n , since the reward r , the next hop agent observation o'_n and the next hop agent Q -value $Q_{n'}(\cdot; \theta_{n'})$ are associated to the packet ID.

Multi-threaded agent instances. These are `Agent class` instances. Each one corresponds to a network node. They share different information among them and with the `Main code` using static variables. As aforementioned, we decide to allocate two threads for each instance: one calling `run_forwarder` method and the other one calling `run_trainer` method

to guarantee that training and forwarding can take place separately.

Stats writer. This module is called by the `Main` code and `Agent` class instances to log information about the environment or the training progress. The following variables are tracked to be visualized using `tensorboard`: average packet delay, loss ratio, Temporal Difference error at each agent, replay buffers' occupancy at each node, exploration ratio for each node, the number of new arriving packets, the number of delivered packets, and, the number of buffered packets. We also add the ability to define custom plots and to compare between execution's hyperparameters.

Argument parser. This module is used to retrieve and parse the arguments given by the user in the script call.

Poisson traffic generator. This is a `ns-3` application class that generates random packets following a *Poisson* process at each node. The average rate is retrieved from a given traffic matrix and the packet length is fixed to a given value. The application is not installed in the node itself, but in a virtual one directly connected to the real node. **Action and observation handler.** As indicated in Section A.1.3.1, the action a_n and the observation o_n are defined as the output network interface index and the packet destination, respectively. For the local observation o_n , other data such as the buffer occupancy of the outgoing interfaces could be considered. This module, implemented as a `ns-3` handler, retrieves o_n from the network simulation when a new packet arrives at the node n , and forwards the packet to the output interface a_n . If the buffer interface is full, then the packet is discarded. We also provide the packet ID in the `info` field, so we can track each packet.

MARL lib. With this module, we extend the OpenAI™ Baselines library (*OpenAI Baselines: high-quality implementations of reinforcement learning algorithms, s. d.*) to the Multi-Agent Deep Reinforcement Learning (MA-DRL) approach. The `MARL lib` provides the tools for defining, training, and testing an agent. It contains the following modules:

- `models`: a module containing the DNN models.
- `Replay buffer`: a class handling the experience replay buffer. It contains methods like `add`, `sample` or `save`.
- `agent`: a class defining an DQN agent in the context of DPR. It contains methods like `step`, `train` and `sync neighbor target neural network`
- `utils`: a set of utility functions like `save_model` and `load_model`.

A.1.5 Usage

In this section, we describe the parameters of PRISMA, and we present the guidelines about how to use the framework.

A.1.5.1 Framework parameters

PRISMA parameters are separated into the following:

- *Global simulation arguments*: it contains the simulation time, the base port, the seed, and whether to run training or not.
- *Network parameters*: it contains network attributes like the path to the adjacency matrix, the maximum output buffer size, or the load factor.
- *DRL agent argument*: it concerns the training of the agent and contains parameters like the batch size, the learning rate, or the training time step.

- *Session logging arguments*: it contains parameters about the session, like the name of the session and the path to store the result of the simulation.
- *Other parameters*: some misc arguments like whether to run a tensorboard server and its port number.

A.1.5.2 Usage guide

First, a user must install the `ns3-gym`, operational system and Python dependencies. To do that, we provide installation files.

The main `prisma` folder is composed of four folders: `source`, `ns-3`, `examples`, and `scripts`. The `source` folder contains the MARL lib, the agent class and utils modules. The `ns3` folder contains the `ns-3` files: (i) the Poisson data generator, and; (ii) the `ns-3` simulation itself (`sim.cc`). The latter file creates and runs the simulation scenario defined by the `examples` folder files, that define the network topologies and traffic matrices. Scripts for launching the simulations of Section A.1.6 are provided in the `scripts` folder.

The DNN model can be changed in `models.py`. The reward, observation, and action definitions can be modified in the methods `_get_reward` (in `agent_class`), `Get_Observation` (in `packet-routing-gym.cc`) and `Get_action` (in `packet-routing-gym.cc`), respectively. Finally, a user may test the framework by calling `main.py` with the corresponding arguments. Calling the latter file with `"-help"` argument will show all the possible parameters.

A.1.6 Illustrative example

This section aims to show how PRISMA can be used to assist the training and testing of a DRL algorithm to solve the Distributed Packet Routing problem. Namely, we make a Deep Q-Network routing model (Mukhutdinov et al., 2019) learn the *Shortest Path (SP) routing* in backbone network scenarios. The tool can be applied to more challenging ad-hoc wireless networks by simply modifying the `sim.cc` file and properly adapting the DQN routing model. Here, we use a more simple case for illustrative purposes.

A.1.6.1 Simulation settings

We ran PRISMA in two different machines: (i) a Dell Precision 7920 workstation equipped with an Intel Xeon Gold 6230R Dual CPU (26 Cores, 2.1-4.0GHz Turbo, 128 GB RAM) with 2 NVIDIA RTX A5000 GPUs; and, (ii) an Intel Inspiron 14 laptop equipped with an Intel Core i7-8565U CPU (Quad Core, 1.80 GHz, 8 GB RAM) with a 1 GPU NVIDIA GeForce MX130 GPU.

The DRL agent is implemented in TensorFlow (*TensorFlow: An end-to-end open source machine learning platform*, s. d.) as a three-layer neural network in `models.py` with a first dense layer of 32 nodes, then two layers of 128 nodes each having a Leaky ReLU activation function. This model considers only the packet destination as input and outputs the estimated end-to-end packet delay based on the selected network interface. The action is retrieved by applying an `argmin` function to the output layer. In the `examples` folder, we define the two topologies considered in this experiment: *Abilene* (11 nodes) and *Geant* (23 nodes), as well as the traffic matrices (randomly generated using a uniform distribution). We fix the packet size to 512 KB, the link propagation delay to 2 ms, and the maximum output buffer length to 30 packets. We

chose the reward r_n (the *next hop packet delay*) to be 1, which means that the agent policy will aim to minimize the total number of hops from source to destination. These parameters along with the model hyperparameters are set as arguments when calling `main.py`. We expect that *DQN-routing* will have a similar performance to *Shortest Path routing*, since both algorithms minimize the number of hops taken by a packet to reach its destination.

A.1.6.2 Model training

In this section, we show how the training process is performed successfully. The model is trained during 1 *minute* at each 7 *ms* (network simulation time) with a learning rate of 0.001, batch size of 512, load factor of 50% and γ of 1. The duration of the simulation time for training was 30 seconds. Moreover, we use an ϵ -greedy approach (ϵ decays from 1 to 0.01) to move from exploration to exploitation.

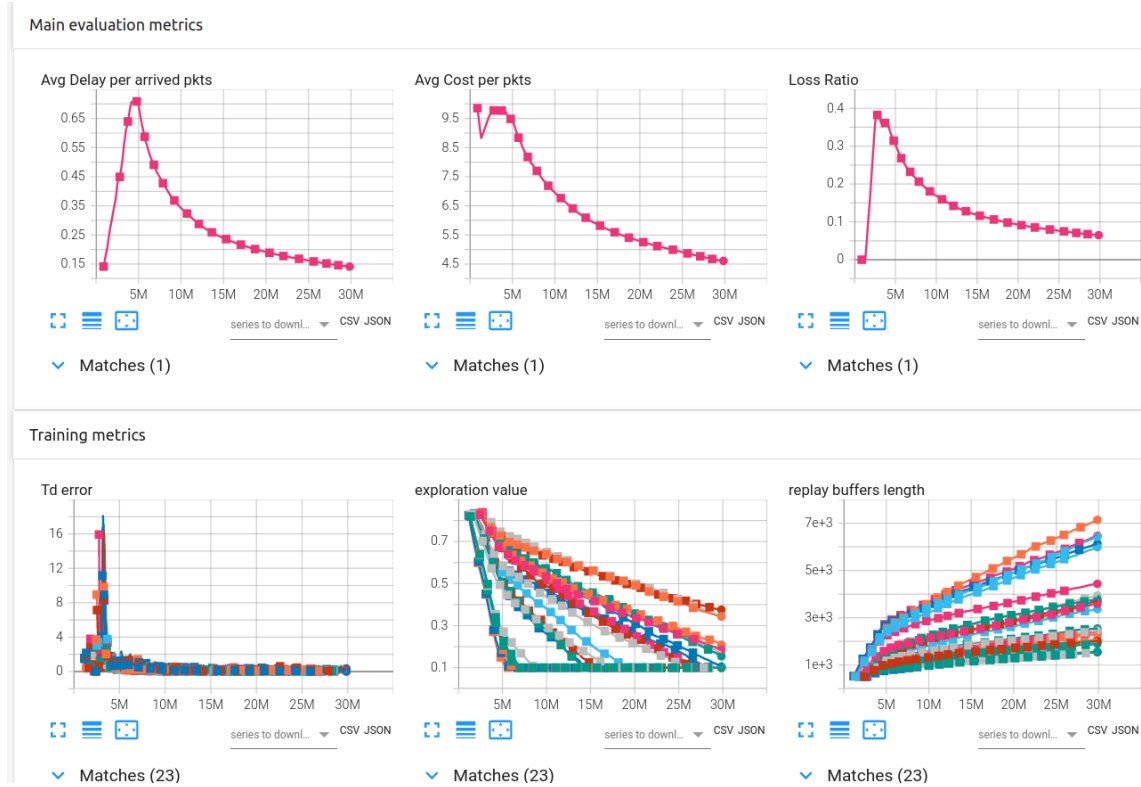


Figure A.2 – Screenshot of `tensorboard` interface. The x axis represents the time in *ns*, and the y axis represents the variables tracked during the simulation. For the training metrics (bottom subfigures), each curve corresponds to an agent.

Thanks to `tensorboard` visualization tools (see Figure A.2), we can watch the training progress. Figures A.3 show in detail the most relevant metrics to watch the learning progress for *Abilene* and *Geant*. Figure A.3a and A.3b show the average cost over the simulation time. The cost is computed by dividing the sum of the rewards over time by the total number of packets. Since the reward is 1 at each hop, the cost represents the average hop count per packet. We can observe an early cost increase due to the initial exploration: the network is flooded with packets needing many

hops to arrive at their destinations, since the forwarding decisions are mainly random. Afterward, we move progressively to the exploitation phase, where the decision comes from the Deep Neural Network (DNN) model. As we see, the packets reduce the hop count to reach their destination, since model decisions improve. Figs A.3c and A.3d show the Temporal Difference (TD) error for each node over the simulation time. We can observe that this error is decreasing, which means that all the agents are converging (learning) to a feasible routing policy (packets are not forwarded indefinitely).

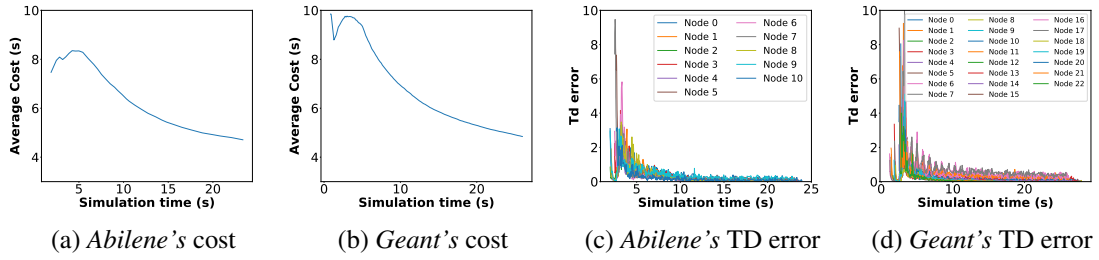


Figure A.3 – The average cost per packet and TD errors at each node during training for Abilene (rightmost subfigures) and Geant (leftmost subfigures) topology.

A.1.6.3 Model testing

In this part, we evaluate the performances of the trained *DQN-routing* agents at different traffic intensities (low, medium, high, and very high traffic, in which the load factor was 50%, 100%, 150% and 200%, respectively). In Figure A.4, we compare the results with SP routing in terms of the *average end-to-end delay* and the *packet loss ratio*. For low and medium traffic rates, *DQN-routing* presents the same performance as *SP* since both methods are using the shortest path decision policy, which is sufficient to handle all the packets at this rate. For high traffic rate, *DQN-routing* outperforms *SP routing* in terms of the *end-to-end delay* for both topologies; and, in terms of *loss ratio*, for Abilene. On the contrary, *SP routing* is slightly better in *loss ratio* in Geant. Anyway, these differences are not significant since we intend to learn a routing close to the SP, which is the case.

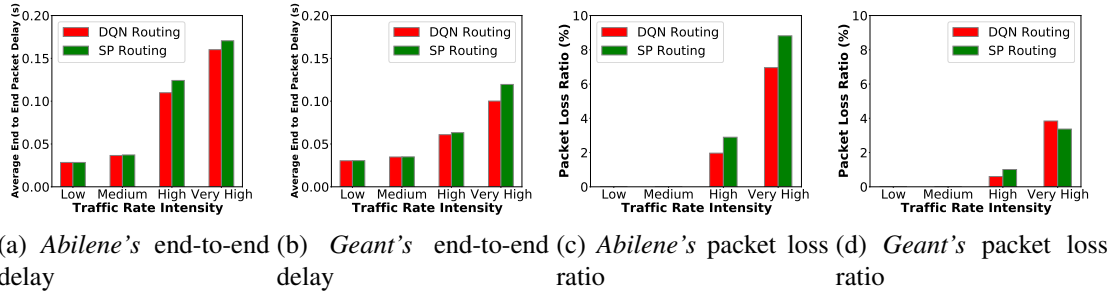


Figure A.4 – The average end-to-end delay per arrived packet and packet loss ratio for different traffic rate intensities for Abilene (right row) and Geant (left row) topology

A.1.7 prisma-v2: Extension to Overlay Cloud Networks

Overlay networks are virtual or logical networks built on top of a physical network (called underlay networks). Overlay networks provide flexible and dynamic traffic routing between nodes that are not directly connected by physical links, but rather by virtual or logical links that correspond to paths in the underlying network. Those virtual links can be established using different technologies, like Generic Routing Encapsulation (GRE), Virtual Private Network (VPN) or network virtualization. The underlay topology is managed by a third party, typically one or more network operators. One particular example of overlay networks is Software-Defined Wide Area Network (SD-WAN)(Z. Yang, Cui, Li, Liu, & Xu, 2019), which fully utilizes the bandwidth of all available transport networks serving one location, like Multiple Protocol Label Switching (MPLS) fabric, Internet and 5G, considering each one of them as an overlay link. In the context of overlay networks, the problem of routing the traffic between the overlay links, especially in multi-hop scenarios, becomes challenging, since the underlay routing policies are unknown and can involve different protocols, like Open Shortest Path First (OSPF), Border Gateway Protocol (BGP), and others. The absence of information about the underlay network topology and routing policies, yields the existence of Triangle Inequality Violation (TIV) (Lumezanu, Baden, Spring, & Bhattacharjee, 2009): it is highly possible to find another path relayed by cloud servers which has a much lower delay than following the shortest path in the overlay topology. There are classical routing protocols that can be used to route in overlay, like Cisco’s Overlay Management Protocol (OMP), which is a control protocol developed by Cisco and works as BGP. This class of protocols highly depends on pre-defined metrics, and they do not handle multi-hop overlay networks.

Then, we proposed a new release of PRISMA, namely `prisma-v2`, which offers the possibility to experiment MA-DRL approaches in the context of cloud overlay networks. The new features brought by **prisma-v2** are listed below:

- Overlay topology simulation and control management.
- Ability to add dynamic underlay traffic along with the overlay one.
- High reproducibility of results by supplying containerizing capability using docker([Docker main page website](#), s. d.).
- Refactoring the code for better readability.
- Improve Tensorboard([TensorBoard: TensorFlow’s visualization toolkit](#), s. d.) logging by incorporating both training and testing phases.
- Implement control packets to realistically simulate the communication and evaluate the overhead.

The **prisma-v2** code source is publicly available as **v0.2** release of PRISMA at ([PRISMA tool: An open MARL framework for packet routing.](#), s. d.). First, we will present **prisma-v2** features and code structure. Then, we will present how to install the tool and present some use cases.

A.1.7.1 prisma-v2 Structure

The main features of this release are illustrated in the Fig. A.5.

ns-3 part . The `prisma-v2` allows controlling the packet routing policy in overlay networks and, also simulating control signalling packets. We implemented three types of control signalling packets: the *big signalling packets* generated at a fixed time interval, and used to simulate the transfer of neural network weights between neighbors; the *small signalling packets* generated

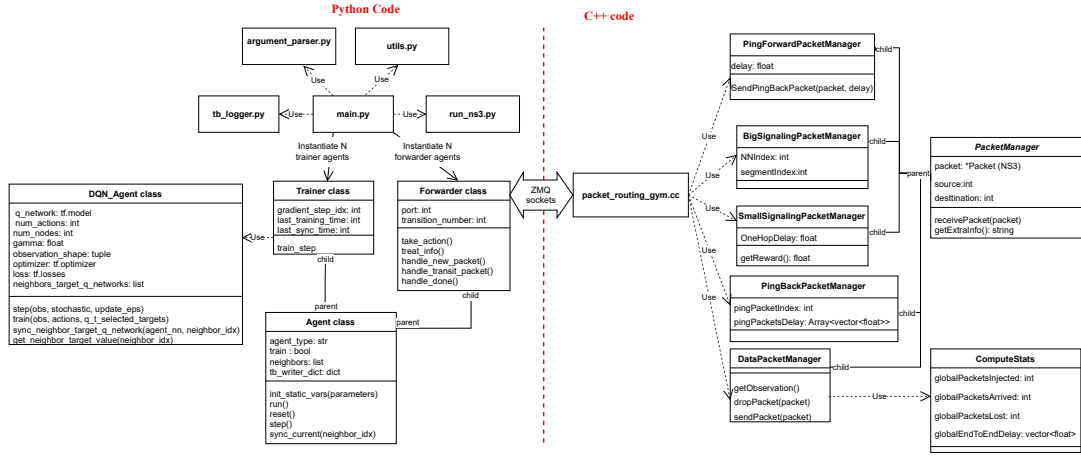


Figure A.5 – prisma-v2 code structure

as a response to data packets, in order to transport the new observation and the reward; and the *ping packets* created according to a parameterized number of data packet transmissions and used to collect the overlay link’s delay. For managing the data and control signalling packets, we developed the class `PacketManager`, which is inherited by five specific packet manager classes. The `SmallSignallingPacketManager` manages the arriving of small signalling packets. The class extracts the overlay link’s delay measured during the sending of the data packet. The `BigSignallingPacketManager` manages the big signalling packet, extracting the information about the neural network weights shared by a node. The `PingForwardPacketManager` receives a ping packet sent, and calculate the overlay link’s delay in the overlay scenario. The `PingBackPacketManager` receives the response of the ping packet containing the tunnel delay and stores it. The `DataPacketManager` manages the data packets. It collects the observation information to transmit to the agent. Furthermore, it sends the packet for the next hop based on the agent action. This class communicates with the `ComputeStats` class, which is responsible for the statistics of the network simulation, such as the number of injected packets by the node, the number of lost packets by the node and the end-to-end delay of the arrived packets.

python part . The left side of figure A.5 presents the python code structure of **prisma-v2**. Like the first version, the program is launched by calling `main.py` with the simulation arguments. Packet Forwarder agent objects are instantiated from the `Forwarder` class stored in `forwarder.py`, while trainer daemon agent objects are instantiated from the `Trainer` class stored in `Trainer.py`. Both classes are children of the `Agent` class stored in `agent.py`. The `Agent` class implements the methods `run`, `reset`, and `step`, which will be overridden by the children’s classes: For the `Trainer` class, the `run` method will launch the daemon main loop that checks if it is the time to train, and calls `step` method to run a training step; For the `Forwarder` class, the `run` method will start the agent episode and connects it to the network node in `ns-3`. It calls the `step` method to run a transition (waiting for a packet to arrive to the node, taking an action using the `take-action` method, and handling the information returned as a response to the action). The `reset` method of the `Forwarder` class will reset the agent environment before starting a new episode. Along with the agent’s classes, we provide `DQN_Agent` class in `agent.py`, which provides the necessary methods to use an adaptation of DQN model

for the DPR problem like in (You et al., 2022) and (Alliche, Barros, et al., 2022). This class relies on the files `replay_buffer.py` and `models.py` for the experience replay buffer and the neural network architecture, respectively.

Tensorboard logger part In **prisma-v2** release, the network metrics(end-to-end delay, packet loss, control overhead) are computed by the `ns-3` part and transmitted to the `python` for being displayed in the tensorboard logger. Moreover, the tensorboard displays in real time all the metrics to the user during the simulation.

We also improved the logger to automatically store in the same instance the train and the test stats. The logs are saved in the session folder, which is given by the argument `session_name`, respectively.

A.1.7.2 Usage

The **prisma-v2** code source is publicly available as **v0.2** release of **prisma** at (*PRISMA tool: An open MARL framework for packet routing*, s. d.). In this section, we will go through different parts of the code, and explain the basic use cases that a user may encounter to launch a simulation.

Installation After cloning the repository, the user is able to run **prisma-v2** by three different ways:

- Run locally by installing the required packages and dependencies by calling `install.sh`.
- Create a local docker image, by running the docker build command on the root folder. This is possible since we provide a docker definition file. This will copy all the folder in the image and so that user may run **prisma-v2** by calling the docker run command and binding the results' folder to be able to retrieve the results in the host machine.
- Pull a docker image provided in Docker Hub. This image only contains the Linux environment with the requirements installed, so the user need to bind the **prisma-v2** folder to the image.

We have provided an illustrative example in the `readme.md` file, guiding the user from the installation to training a Multi-Agent Deep Q-Network (MA-DQN) model to solve the DPR problem in an overlay network.

Modifying or Adding parameters Like the previous version, the parameters are organized by groups and can be visualized by calling “`python3 main.py -help`”. They are accessible in the file `argument_parser.py`, where the user can add a new parameter or modify an existing one. In the file `run_ns3.py`, we parse the arguments to `ns-3`, so the user can modify this function to pass arguments to the NS-3 part.

Changing the DRL algorithm The state can be modified for in `GetObservation` method of the `DataPacketManager`. The user may modify the DRL model input shape to match the new observation shape. Pre-defined neural networks models are already implemented in `models.py`, and a DRL algorithm is implemented in the `DQN_Agent` class. The user may change the latter class to change the learning algorithm.

Sharing the information between the agents The agent forwarder and trainer objects share information with the main process using the Agent class static variables. In those variables, we can find agents which stores the `DQN_Agent` objects, and `pkt_tracking_dict` to keep track of transiting packets in the network.

Changing the topology settings To change the overlay topology configuration, the user may change some parameters: `physical_adjacency_matrix_path`, `overlay_adjacency_matrix_path` and `map_overlay_path` for the paths to the underlay topology’s adjacency matrix file, the overlay topology’s adjacency matrix file, and the map between the indices of overlay and underlay nodes file, respectively; The traffic in underlay topology is handled, by default, using the OSPF protocol, which is used for computing the routing tables. Changing the routing policy is possible by the native `ns-3` method `RecomputeRoutingTables`.

Customize control packets For customizing the control packets, the user may create a new class that inherits the class `PacketManager`. For example, The `receivePacket` method recovers the packet information at its arriving and the method `GetInfo` encapsulate the useful collected information in order to send it to the agent. For generating control signalling packets, the user may be inspired by the methods `sendSmallSignalingPacket` and `sendPingForwardPacket`, providing the packet size and the information which the control signalling packet should encapsulate. For the `bigSignallingPacket`, the parameters `syncStep` and `bigSignalingSize` contain the period time for sharing the model weights and weights total size, respectively. For creating new control signalling mechanisms, the user may be inspired in the current mechanisms developed in **prisma-v2**.

A.1.8 Conclusion

In this appendix A.1, we have presented the PRISMA tool, which is, to the best of our knowledge, the first Deep Reinforcement Learning (DRL) framework specifically devoted to the Distributed Packet Routing (DPR) problem. In this problem, routing agents take packet forwarding decisions based only on local information. When the network state is uncertain or partially hidden (e.g., a global view is not available) or changes very dynamically, DRL appears as a solution to discover an efficient routing policy. We also introduced its extension, `prisma-v2`, which integrate the overlay cloud networks. In this vein, PRISMA aims to provide a playground for researchers interested in the application of this machine learning paradigm to this challenging problem, allowing fast prototyping and bench-marking. We illustrated its main functionalities by applying the tool to learn in a distributed manner a Shortest Path routing policy in two backbone networks.

A.2 Impact Evaluation of Control Signalling onto Distributed Learning-based Packet Routing

A.2.1 Introduction

After the breakthrough results obtained by DRL (Bengio, 2009) in solving highly complex tasks (Mnih et al., 2015), DRL has been also started to be applied to communication networks problems. One of them is the Distributed Packet Routing (DPR) (Mukhutdinov et al., 2019; You et al., 2020; Manfredi et al., 2021-06; L. Chen et al., 2021). This is a challenging problem where no complete and centralized view of network topology and traffic demands is available (e.g., multi-hop wireless networks (Tassiulas & Ephremides, 1992) or multi-domain optical networks (X. Chen et al., 2019)). In the DPR problem, per-packet forwarding decisions are made locally by distributed agents, exploiting local information, such as packet headers and neighboring node states. Nevertheless, these first works (Mukhutdinov et al., 2019; You et al., 2020; Manfredi et al., 2021-06; L. Chen et al., 2021) focused on improving the learning performance by modifying the training mechanisms and the model design, neglecting the impact of control signalling on routing performance. Indeed, the *distributed multi-agent* setting of the DPR problem imposes a non-negligible level of communication between the neighboring agents during the training phase, which is meant to run continuously or frequently to adapt the routing policy to any network change (traffic, topology). This information exchange constitutes the control signalling, and it introduces a certain level of *overhead*. Thus, a trade-off appears between this *overhead* and the *quality* of the learned routing: a minimal amount of signalling (*overhead*) is required to make the agents learn a routing policy at a cost of increased bandwidth requirements. On the other hand, an excessive control *overhead* could slow down data packets and impede the learning process.

This appendix hence focuses on the impact evaluation of this control signalling. To perform the above-mentioned impact evaluation, a realistic modelling of the communication is necessary, in contrast to the simplified simulations performed with ad hoc network simulators in the previous works. For this reason, we make use of our PRISMA tool, detailed in Section A.2.3.6. This is a discrete time packet-level simulator designed to apply MA-DRL to the DPR problem. It is based on the ns-3 (nsnam, s. d.) network simulator and a multi-threaded implementation of each agent, enabling a more realistic network simulation where the control signalling can be implemented in a reproducible manner. In the MA-DRL framework, neighboring agents collaborate by exchanging information, particularly the estimated packet end-to-end delay. In this work, we consider that this information can be shared as the current value of the estimate (*value sharing*), or as the estimating model itself (*model sharing*).

Our contributions are:

- We present, to the best of our knowledge, the first experimental investigation of control signalling mechanisms for distributed learning-based packet routing. We rely on PRISMA, our open simulation ns-3-based module enabling testing MA-DRL-based routing in a reproducible and realistic manner (*PRISMA tool: An open MARL framework for packet routing*, s. d.).
- We formulate two signalling mechanisms for the *DQN routing* (Mukhutdinov et al., 2019) framework, where routing nodes periodically communicate copies (*target*) of their ML models to their neighbors (named *model sharing*), or only communicate delay estimates (named *value sharing*). The control overhead induced by these exchanges can be modulated by tuning the period between two consecutive exchanges.

- We investigate net gains considering in-band signalling and show that (i) *model sharing* between agents is necessary to find routing policies close to the optimal routing of an Oracle with perfect knowledge of topology and traffic, but at the cost of possibly significant control overhead (150 % of extra traffic in the considered network settings with respect to useful traffic); and (ii) *value sharing* incurs in much more reduced overhead (10 %), but barely improves a SP routing policy. We discuss the generality of our results to underline the importance of assessing net gains of MA-DRL-based routing.

Related works are presented in Section A.2.2, the MA-DRL framework and the control signalling mechanisms are described in Section A.2.3. Simulation results are analyzed in Section A.2.4. Section A.2.5 provides some discussion, and Section A.2.6 concludes the section.

A.2.2 Related works

In recent years, works addressing the DPR problem by making use of DRL have been published in (Boyan & Littman, 1993; Mukhutdinov et al., 2019; You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06). The seminal paper (Boyan & Littman, 1993) was the first to apply a RL method, the Q-learning (Watkins, 1989), to DPR, giving rise to the *Q-routing* paradigm. In this work, the routing decisions are taken based on the packet destination. The study in (Mukhutdinov et al., 2019) applies the DQN framework to this *Q-routing* by using a DNN to approximate the Q-value function, yielding its deep learning version: the *DQN routing*.

After *DQN routing*, other proposals of MA-DRL applied to DPR were published in (You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06). These works concentrate on how to improve the quality of the learned routing from a ML perspective (i.e., changing the model, the input features or the RL algorithm), neglecting networking aspects, as the signalling. *DQRC (DQR with Communication)* (You et al., 2020) adds the action history, the future destinations, and the most loaded neighboring node to the packet destination as input. The neural network architecture of *DQRC* also considers a LSTM layer to exploit the new time series features (action history and future destinations). Authors in (Manfredi et al., 2021-06) enlarge the input algorithm with *relational features*, such as the node buffers' occupancy, the distance from the routing device to the packet destination or the packet TTL field. The values of these features can vary for individual packets and devices, but the features' list (and its size) is the same for all the routing nodes, which helps to improve the generalization among the nodes (the same neural network model is used for all the nodes). Finally, the work in (L. Chen et al., 2021) adapts the Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) RL algorithm to the multi-agent scheme of the DPR problem, yielding the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm. In MAPPO, the task to learn is considered as *to find a routing for a given traffic matrix*, using as input the packet destination and the buffers' occupancy. Finally, this section also aims to generalize the routing policies to different traffic matrices (i.e., tasks) by extending MAPPO to the meta-learning framework.

In all the above-mentioned works, the impact of the network control signalling onto the training process is ignored or superficially considered. In the more recent works (L. Chen et al., 2021; Manfredi et al., 2021-06), the training is centralized in a node that gathers all the past experiences (forwarding decisions) of all the forwarded packets at all the routers. The model for each routing node is learned at this central node using this large training set, and is eventually pushed to the routing agents that will forward the packets. The significant overhead associated to these data transfers is not analyzed, questioning the actual gains of the proposed methods. On

the contrary, the first works (Mukhutdinov et al., 2019; You et al., 2020) assumed a distributed training operation where training data are exchanged only locally between neighboring agents at a cost of the *non-stationarity* of multi-agent environments (Littman, 1994), which makes harder the agents’ policies convergence. To tackle this problem, the authors in (Foerster et al., 2017) propose that the neighboring agents share their policies (models) to stabilize the learning when larger datasets of past experiences are used (i.e., replay memories). However, current DQN routing such as (Mukhutdinov et al., 2019) operates by sharing the values of model estimates (*value sharing*), instead of sharing the model at every neighboring node (*model sharing*). Value sharing reduces the signalling overhead, but yields poor results when replay memories are used (known to stabilize the training of DQN (Mnih et al., 2015)). On the contrary, *DQRC* (You et al., 2020) share the neighbors’ routing models (*model sharing*), but at each training step, which introduces a significant communication overhead not properly evaluated. In the present work, we share copies of the neighboring models (i.e., policies) combined with replay memories, which improves policy convergence (Mnih et al., 2015). The overhead entailed by sharing the models between neighbors can be modulated with the sharing update period. In this section, we investigate the performance tradeoff entailed by tuning this update period in MA-DRL.

A.2.3 RL for Distributed Packet Routing: Formulation and Signalling

In this section, we first describe the network model and formulate the DPR problem. We then express the considered Multi-Agent Reinforcement Learning (MA-RL) approach problem based on (Littman, 1994). We detail the required control signalling, proposing two different versions and showing how signalling overhead can be traded with routing performance. We finally introduce our reproducible framework for a realistic evaluation of the impact of signalling in MA-RL.

A.2.3.1 Network Model

Let $\mathcal{G}(\mathcal{N}, \mathcal{E})$ be a directed network graph, where $\mathcal{N}$ is the nodes set and $\mathcal{E}$ is the unidirectional links set. A link $(a, b) \in \mathcal{E}$ initiates at node a , terminates at node b and has a capacity of C traffic units (i.e., *Mbps* or *packets/sec*). The incoming and outgoing neighbor nodes of the node $n \in \mathcal{N}$ are denoted as $\mathcal{N}_n^{in}$ and $\mathcal{N}_n^{out}$, respectively. Each packet is originated from a source node $s \in \mathcal{N}$ and targeted to a destination node $d \in \mathcal{N}$. The traffic matrix $\mathbf{H} = \{h_{sd}, (s, d) \in \mathcal{N} \times \mathcal{N}\}$ counts up the average volumes h_{sd} in traffic units (i.e., *Mbps* or *packets/sec*) of packet flows between the node pairs $(s, d) \in \mathcal{N} \times \mathcal{N}$. Each node $n \in \mathcal{N}$ is a router equipped with $|\mathcal{N}_n^{out}|$ outgoing network interfaces. Each interface $i \in |\mathcal{N}_n^{out}|$ has its own buffer queue of size B . The queue follows a *FIFO* (*First-In First-Out*) policy. Hence, if a packet arrives at a full buffer, it is rejected (i.e., *lost*). Otherwise, the packet is admitted, eventually getting to the buffer head, where it is forwarded to a next-hop node $n' \in \mathcal{N}_n^{out}$. At each hop, this procedure is repeated until the packet finally reaches its final destination d (or is lost elsewhere en route). Table A.1 gathers the notation.

A.2.3.2 An Oracle Routing Policy

In this subsection, we propose an *oracle* policy to be used as a routing optimality benchmark. We devise a centralized version of the DPR problem, where an *Oracle* observer (different from the routing nodes) has a full knowledge of the network topology and the traffic matrix. This centralized version is formulated as the Minimum Cost Multi-Commodity Flow (MCF) (Min Cost MCF)

Name	Description
$x_{ab}^{sd} \in [0, 1]$	Fraction of the packet flow between the node pair (s, d) to be forwarded via the link (a, b) .
$y_{sd} \in [0, 1]$	Fraction of the packet flow between the node pair (s, d) that is rejected.
$h_{sd} \in \mathbb{R}_{\geq 0}$	Average flow volume between the node pair (s, d) in traffic units (i.e., <i>Kbps</i>).
$C \in \mathbb{R}_{\geq 0}$	Link capacity in traffic units (i.e., <i>Kbps</i>).
$B \in \mathbb{R}_{\geq 0}$	Buffer size in data units (i.e., <i>Bytes</i>).
$M \in \mathbb{R}_{\geq 0}$	Large number to penalize a loss more than a large delay.

TABLE A.1 – Notation

problem solved by the Linear Programming (LP) model (A.3). The optimal solution of this model provides a lowest-cost routing policy, that we call *oracle routing* in the reminder of this section.

$$\min_{\{\mathbf{x}, \mathbf{y}\}} \sum_{\substack{s \in \mathcal{N} \\ d \in \mathcal{N}}} h_{sd} \left(\sum_{(a,b) \in \mathcal{E}} x_{ab}^{sd} + M \cdot y_{sd} \right) \quad (\text{A.3a})$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{N}_n^{\text{in}}} x_{an}^{sd} - \sum_{b \in \mathcal{N}_n^{\text{out}}} x_{nb}^{sd} = \begin{cases} -1 + y_{sd}, & \text{if } n = s \\ 1 - y_{sd}, & \text{if } n = d \\ 0, & \text{otherwise} \end{cases} \quad s \in \mathcal{N}, d \in \mathcal{N}, n \in \mathcal{N} \quad (\text{A.3b})$$

$$\sum_{\substack{s \in \mathcal{N} \\ d \in \mathcal{N}}} h_{sd} \cdot x_{ab}^{sd} \leq C, \quad (a, b) \in \mathcal{E} \quad (\text{A.3c})$$

$$x_{ab}^{sd} \in [0, 1], \quad s \in \mathcal{N}, d \in \mathcal{N}, (a, b) \in \mathcal{E} \quad (\text{A.3d})$$

$$y_{sd} \in [0, 1], \quad s \in \mathcal{N}, d \in \mathcal{N} \quad (\text{A.3e})$$

The objective function (A.3a) minimizes a twofold objective: (i) primarily, the total amount of rejected traffic $\sum h_{sd} \cdot y_{sd}$; and, (ii) secondly, the average hop count $\sum h_{sd} \sum x_{ab}^{sd}$. Minimizing (ii) leads to minimize the average end-to-end delay in this model. The flow conservation constraints (A.3b) ensure that all the traffic admitted at source is routed until destination. The constraints (A.3c) limit the link capacity. Finally, the constraints (A.3d) and (A.3e) define the lower and upper bounds on the variables.

The Oracle policy is not necessarily optimal for the DPR problem, since it was computed to solve a centralized and flow-based version of the routing problem. The LP model (A.3) works with averaged packets' flows and is unaware of the traffic dynamics at the packets' scale (e.g., a burst of packet arrivals whose data rate exceeds nominal link capacities C). Hence, a per-packet routing algorithm could find policies exploiting outgoing buffers, which are unknown for the LP model (A.3), able to beat this *Oracle routing* in some circumstances (as we will see in

Section A.2.4.3). Anyway, we use it since it constitutes a high quality benchmark hard to attain without a global view of the exact network topology and the precise traffic matrix.

A.2.3.3 The DPR Problem and our DQN Routing Algorithm

The DPR problem consists of deciding, at each node, the output port (next hop) for each upcoming packet. The objective is to find a global routing policy that minimizes the end-to-end delay for all the packets over the network. This problem can be formalized as Multi-Agent POMDP (Littman, 1994): a Markov Decision Process where each agent can only locally observe its environment. We use an approach based on Q -routing (Boyan & Littman, 1993), but approximating the Q -value as a neural network (as in (Mukhutdinov et al., 2019; You et al., 2020)), and using as input the packet destination and the node buffers' occupancy. We will use this scheme to describe the POMDP and the RL approach used to solve it.

Let $\mathcal{N}$ be the set of routing nodes (*agents*), where each *agent* n has its own local observation space $\mathcal{O}_n$ and its own action space $\mathcal{A}_n$. When a new packet arrives at time t at the node n , the node n selects as action a_n the *next hop node* n' to forward the packet to. This decision is taken depending on the local observation of the router o_n . As a consequence of the decision, the *agent* n receives a reward $r_{n'}$ from the next hop node n' : the *next-hop packet delay* (i.e. the packet delay to travel from n to n'). A transition is hence made to a new state. When the packet arrives to a node, this procedure is repeated. The action a_n is selected to minimize an estimate of the expected *end-to-end packet delay* from the node n to its final destination. Under the DRL scheme, this estimate, denoted as $Q_n(o_n, a_n; \theta_n)$ (the Q -value), is the output of a DNN with weights θ_n . This DNN is trained to fit the target value Y_n^Q below:

$$Y_n^Q = r_{n'} + \gamma \cdot \tau_{n'} \cdot (1 - f) \quad (\text{A.4})$$

where f indicates if the next hop is the packet destination, $\gamma \in [0, 1]$ is a discount factor, r is the *next-hop packet delay* and $\tau_{n'}$ is the *remaining end-to-end delay* from the next hop n' to the destination computed as follows:

$$\tau_{n'} = \min_{a_{n'} \in \mathcal{A}_{n'}} Q_{n'}(o_{n'}, a_{n'}; \theta_{n'}) \quad (\text{A.5})$$

where $Q_{n'}(\cdot; \theta_{n'})$ is the Q -value estimate of the next hop agent.

In order to approach the estimated $Q_n(\cdot; \theta_n)$ to the optimal Q_n^* , at each agent n , the weights θ_n are updated by stochastic gradient descent when minimizing the next square loss called *Temporal Difference (TD) error*:

$$L_{DQN} = \left(Q_n(o_n, a_n; \theta_n) - Y_n^Q \right)^2 \quad (\text{A.6})$$

In the next paragraphs, we detail (i) the *node observation* representation, (ii) the *node action*, and (iii) the *reward* definition.

Node observation. The *node observation* $O_n \in \mathcal{O}_n$ is represented as $o_n = (d, \{b_i, i \in 1 \dots |\mathcal{N}_n^{\text{out}}|\})$. This is the concatenation of *two* components: (i) the *current packet destination* d and, (ii) the *buffers' occupancy* b_i , in *Bytes*, of each node interface i at the node n .

Node action. The *node action* $a_n \in \mathcal{A}_n$ is the choice of the out-neighbor $n' \in \mathcal{N}_n^{\text{out}}$ of the node n (i.e., the outgoing interface to this next hop node n') to forward the packet to the buffer head.

Reward. The reward $r_{n'}$ is the *next-hop packet delay*, defined as the time required by the packet to travel from the buffer tail of n up to the buffer tail of the next hop node n' . Then, it is computed as $r_{n'} = l + q$, where: (i) l denotes the *link transmission delay*, the transmission latency in the link connecting n and n' ; and, (ii) q refers to the *queuing delay*, the time spent by the packet in the outgoing buffer of node n taking to n' . If, in the next hop node n' , the outgoing buffer where the packet should be forwarded is full, the packet is lost, and $r_{n'}$ is considered as infinity, since the packet did not arrive at n' . In practice, we use the *worst-case* end-to-end delay: $\frac{|\mathcal{N}| \times (B+1)}{C}$, that is, the delay of traversing over the $|\mathcal{N}|$ nodes when all the output buffers are full. Thus, the reward can be measured in time units (typically, in *seconds*). This reward definition allows the *end-to-end delay* estimate $Q_n(\cdot; \theta_n)$ to account for both buffer delays and packet losses, but giving more weight to the packet losses to favor its minimization in priority, similarly to the LP model of the *oracle routing* (see Section A.2.3.2).

A.2.3.4 Neural Network Architecture

In this subsection, we detail the architecture of the neural network $Q_n(s_n, a_n; \theta_n)$, which is depicted in Figure A.6. This architecture is inspired from (You et al., 2020).

The input layer is split into two parts according to the description of the node state s_n in Section A.2.3.3: (i) the *packet destination*, as a $|\mathcal{N}|$ -element vector in *one-hot encoding* (i.e. the position corresponding to the destination is set to one, the rest to zero); and, (ii) the *buffer occupancy of outgoing buffers of node n* , as an $|\mathcal{N}_n^{\text{out}}|$ -element vector, whose values are layer normalized.

The first hidden layer is also split into two parts, where each one is a 32-neuron fully connected layer fed by their respective input. Afterward, their outputs are concatenated before feeding two 64-neuron fully connected layers.

Finally, the output layer is fully connected with as many neurons as the node action space (i.e., out-degree $|\mathcal{N}_n^{\text{out}}|$ of n). The value of each output neuron is simply the estimate of the Q -value $Q_n(s_n, a_n; \theta_n)$. All the activations are ELUs (Exponential Linear Units).

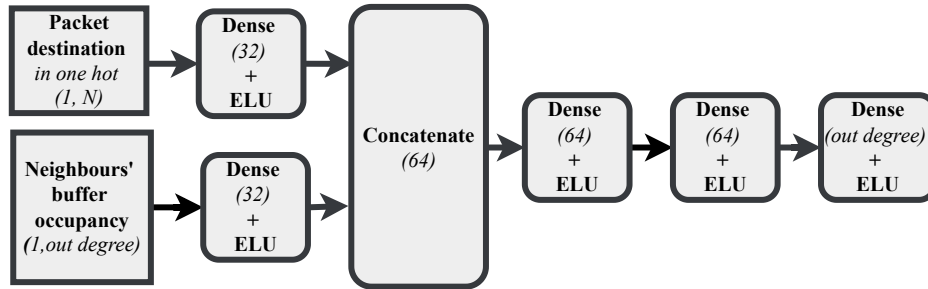


Figure A.6 – NN layers architecture

A.2.3.5 MA-DRL Signalling: Value Sharing or Model Sharing

DPR consists of forwarding decisions taken in a distributed manner without cooperation between the routing nodes. However, with MA-DRL, control signalling exchanges are necessary to enable agent training, since agents need to share their past observations (i.e., experiences), network metric estimates and ML models before the training takes place. We consider two agent information sharing techniques: (i) *value sharing* and (ii) *model sharing*. In the *value sharing*

method, a node n' receiving a packet from n sends back to n a control packet containing the pair $(r_{n'}, \tau_{n'})$. When this packet arrives at n , it is stored with its corresponding observation-action pair (o_n, a_n) in the *replay memory* $\mathcal{M}_n$. We refer to these packets as *replay memory update packets*. We recall that $\tau_{n'}$ is the estimate of the *remaining end-to-end delay* computed by the next hop agent n' at the reception of packet at n' as Equation (A.5). In the *model sharing* technique, the *replay memory update packet* contains the pair $(r_{n'}, o_{n'})$. In this case, the $\tau_{n'}$ estimate is not provided by the neighbor n' but computed at the node n by a copy of the next hop agent DNN $Q_{n'}(\cdot, o_{n'}; \theta_{n'})$. We refer to this copy as *target network*, and we denote it as $\hat{Q}_{n'}(\cdot; \theta_{n'}^-)$. Thus, the model weights $\theta_{n'}$ have to be sent from n' to n periodically to update this *target network*, adding a heavier *control overhead* with respect to the *value sharing* technique. We call the packets carrying the model weights *target update packets*. The time between two consecutive target updates is the *target update period* U , which controls the overhead level of the *model sharing* method. The more often the target networks are updated, the heavier is the control overhead, since control and data packets compete for the same bandwidth, which supposes that both packet types will experience additional queuing delays.

Finally, we conclude this section with Algorithm 9 that depicts the pseudocode of the training routine of our approach. We highlight in red and blue italics the lines corresponding to the *value sharing* and *model sharing* method, respectively.

Algorithm 9 DQN Routing training routine at a node n *with value sharing* (or *with model sharing*)

- 1: **Input:** Training session duration S ; Gradient descent update period T ; Target update period U ; Weights θ_v^{SP} , $v \in \{n \cup \mathcal{N}_n^{out}\}$
 - 2: **Output:** Final routing model weights θ_n
 - 3: Initialize experience replay memory $\mathcal{M}_n$
 - 4: Initialize $Q_n(\cdot)$ with weights $\theta_n \leftarrow \theta_n^{SP}$
 - 5: *Initialize targets $\hat{Q}_v(\cdot)$, $v \in \mathcal{N}_n^{out}$ with weights $\theta_v^- \leftarrow \theta_n^{SP}$*
 - 6: **while** current simulation timestamp $t < S$ **do**
 - 7: **for** each arrival packet **do**
 - 8: Observe the current node State o_n
 - 9: Select action $a_n = n'$, where:
 - 10:
$$n' = \begin{cases} \text{random } v \in \mathcal{N}_n^{out} & \text{with prob. } \epsilon \\ \arg \max_{v_n \in \mathcal{N}_n^{out}} Q_n(o_n, v_n) & \text{otherwise} \end{cases}$$
 - 11: Forward packet p to next hop node n'
 - 12: *Receive back from n' next hop estimate τ and reward r_n*
 - 13: *Receive back from n' next hop state $o_{n'}$ and reward r_n*
 - 14: Set packet destination flag f
 - 15: *Store transition $(o_n, a_n, r_{n'}, \tau_{n'}, f)$ in memory $\mathcal{M}_n$*
 - 16: *Store transition $(o_n, a_n, r_{n'}, o_{n'}, f)$ in memory $\mathcal{M}_n$*
 - 17: **if** $t \bmod T = 0$ **then**
 - 18: Sample random batch $\mathcal{B} \stackrel{i.i.d.}{\sim} \mathcal{M}_n$
 - 19: Set target values Y_n^Q as in Eq. (A.4) for $\mathcal{B}$
 - 20: Update weights θ_n by gradient descent on TD error for $\mathcal{B}$
 - 21: **if** $t \bmod U = 0$ **then**
 - 22: *Get neighbors' weights: θ_v , $v \in \mathcal{N}_n^{out}$*
 - 23: *Update target weights: $\theta_v^- \leftarrow \theta_v$, $v \in \mathcal{N}_n^{out}$*
-

A.2.3.6 A Realistic Evaluation of Signalling with the PRISMA tool

To study the impact of the control signalling, we need a simulation framework reproducible and realistic enough to capture the tradeoffs between the control overhead and the quality of the learned routing. This is enabled by our new PRISMA tool, detailed in Appendix A.1 (Alliche, Da Silva Barros, et al., 2022) and made available to the research community (*PRISMA tool: An open MARL framework for packet routing*, s. d.).

In the previous literature (Mukhutdinov et al., 2019; You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06), the DRL approaches were evaluated on simulation environments (typically implemented in `python`) tailored to the assumptions and simplifications made by each study. Namely, these works do not generally implement the RL agents as separated threads or processes; and, they usually assume a unique buffer queue per node and a fix time slotted operation. As a consequence, only one packet per router is transmitted at each time step, which introduces an artificial synchronization among the state transitions along the network, constraining the diversity of state transitions that the simulator permits to observe. This has two main consequences: (i) a reduced realism of the simulated communication process; and, (ii) an obstacle to fairly comparing these DRL proposals to state-of-the-art approaches or even against each other. This lack of standardized ML tools in the networking community (Gawłowicz & Zubow, 2019) is becoming a major issue to guarantee reproducibility. As a consequence, in the last years, some tools (Gawłowicz & Zubow, 2019; H. Yin et al., 2020) have been developed to apply RL to networks. These tools allow `python` RL agents to interact with the popular `ns-3` network simulator (`nsnam`, s. d.). Nevertheless, these proposals are not natively compatible with the *collaborative and distributed multi-agent* setting of the DPR problem. To bridge this gap, PRISMA has been proposed in the work (Alliche, Da Silva Barros, et al., 2022).

PRISMA is an open-source RL simulation environment designed specifically for considering the distinctive characteristics of the DPR problem, serving as a playground where the community can easily validate their RL approaches and compare them. It introduces a more realistic modelling of the communication process based on: (i) the `ns-3` (`nsnam`, s. d.) network simulator; and, (ii) a multi-threaded implementation for each agent. It makes use of a modular code design, which allows researchers to test their own RL algorithms, without needing to work on the implementation of the environment. PRISMA also provides visualization tools based on `tensorboard` (*TensorBoard: TensorFlow's visualization toolkit*, s. d.) allowing to track training and test phases. All these reasons make PRISMA better suited to evaluate the control signalling exchanges, since their evaluation requires: (i) a higher level of realism to correctly consider the extra delays and traffic introduced by control packets; (ii) a standardized tool where different control signalling techniques between agents can be easily implemented and compared.

A.2.4 Simulation Experiments

In this section, we present the experimental approach to answer quantitatively the following research questions:

1. How does the cost and overhead of *value sharing* and *model sharing* strategies evolve when modulating the target update period U ?
2. How much overhead is required to make the *model sharing* solution close to the Oracle cost solution?

A.2.4.1 Experiment Settings

Hardware and Software settings. We ran the tests in a Dell Precision 7920 workstation equipped with an Intel Xeon Gold 6230R Dual CPU (26 Cores, 2.1-4.0GHz Turbo, 128 GB RAM) with 2 NVIDIA RTX A5000 GPUs, running Linux Ubuntu 20.04. The DRL agent model is implemented in TensorFlow (*TensorFlow: An end-to-end open source machine learning platform*, s. d.) version 2.8 as a described in A.2.3.4. The implementation of the MA-DRL method is based on the OpenAI™ Baselines library (*OpenAI Baselines: high-quality implementations of reinforcement learning algorithms*, s. d.). The PRISMA tool (Alliche, Da Silva Barros, et al., 2022; *PRISMA tool: An open MARL framework for packet routing*, s. d.) is used as RL simulation environment.

Competitors. We compare the routing optimality of the two versions of the *DQN routing* in Algorithm 9 (*value sharing* and *model sharing*) to the *Shortest Path routing* (Bellman, 1958), and the *Oracle routing* from LP model (A.3). The *SP routing* and *Oracle routing* solutions represent performance benchmarks used to assess the quality of the DRL solution. The *DQN routing* with *value sharing* is, in practice, an extension of the original DQN routing (Mukhutdinov et al., 2019), where buffers' occupancy is added to packet destination in the node observation o_n . We cannot compare directly our *DQN routing* to the literature algorithms (You et al., 2020; L. Chen et al., 2021; Manfredi et al., 2021-06) since they are based on different assumptions. In *DQRC* (You et al., 2020), there is a unique buffer at each node, which constraints the outgoing node throughput, since, at each time, only one packet can be transferred from the buffer head to one of the forwarding ports. In the works (L. Chen et al., 2021; Manfredi et al., 2021-06), the training is centralized in a node collecting all the past experiences, in contrast to us, where the training is done on the distributed node agents.

Evaluation metrics. To measure the performance of each routing policy, we use the *average cost per packet*, which is calculated as the accumulated reward along the simulation divided by the number of generated packets. We point out that during a network simulation, we can reward each packet arrival using the same reward definition as in Section A.2.3.3. Thus, the *average cost per packet* represents an average end-to-end delay per packet, where packet losses are also accounted as *worst-case* delays (see Section A.2.3.3).

Topology and traffic. Simulations are performed over the Abilene network ($|\mathcal{N}|=11$) (*SNdlib: Library of test instances for Survivable fixed telecommunication Network Design*, s. d.). We fix link propagation delay and link rate C to 1 ms and 500 Kbps, respectively.

Four traffic matrices $\mathbf{H}$ are generated by sampling each element h_{sd} from a uniform distribution $U(0, 1)$. We scale up these matrices by multiplying by a coefficient α . We increase α up to the largest value α_{max} for which an optimal routing with no packet loss can still be found by the LP model (A.3). The matrix $\alpha_{max} \cdot \mathbf{H}$ is associated to a load factor $\rho = 1$. Data packets are generated as UDP over IP datagrams. Their payload is 512 B long. The UDP and IP headers are 8 B and 20 B long, respectively. Then, the total packet size is 540 B. Packet traces are produced assuming that packet *inter-arrival time* follows an exponential distribution with mean $540B/h_{sd}$. And, finally, the output buffer size B is fixed to 16, 200 B (i.e., 30 data packets of 540 B long).

Control signalling packets. As explained in Section A.2.3.5, DRL agents share information in the form of control signalling packets. For the *replay memory update packets*, we encode each float or integer data type unit in 8 B. Then, a pair $(r_{n'}, \tau_{n'})$ (*value sharing*) is 16 B long and a pair $(r_{n'}, o_{n'})$ (*model sharing*) is $16 + 8 \cdot |\mathcal{N}_n^{out}|$ B long. Control packets are also encapsulated

into UDP over IP datagrams. For a *target update*, the size of a DRL agent model (as described in Section A.2.3.4) is around 36 KB, which we split in *target update packets* of 512 B long.

Training procedure. The training is performed in two phases: a *supervised pre-training phase* followed by the *main reinforcement learning phase*.

Supervised pre-training is done to improve the convergence of the DQN routing, as authors in (Mukhtudinov et al., 2019) demonstrated. They pre-trained the Q-network ($Q_n(\cdot; \theta_n)$) using a dataset of tuples $\{d, L_n^{SP}(d, n')\}$, where d is the packet destination, n' is the next hop node and $L_n^{SP}(d, n')$ is the length of the shortest path between n and d , which contains the next hop node n' . The Q-network is trained till the square loss is minimized. The so-obtained weights are denoted as θ_n^{SP} .

Main reinforcement learning (see Section A.2.3.3). The weights θ_n are initialized with θ_n^{SP} . The model is trained using ADAM optimizer with a learning rate of 0.001, batch size of 512, and γ of 1. The total duration of the training session S is 1 minute (in ns-3 simulation time), which is sufficient to cancel the agents' TD error. The gradient descent is launched every $T = 10$ ms (in ns-3 simulation time). Moreover, we use an ϵ -greedy approach (ϵ decays from 1 to 0.1) to trade between exploration and exploitation. We execute different *training sessions* for each traffic matrix, information sharing technique (*value sharing* and *model sharing*), *replay memory size* and *target update period*. We consider several values for the *replay memory size* (512, 1024, 2500, 5000, 10000, and 15000 experiences), and for the *target update period* U (from 1s to 9s with 1s step). Each session corresponds to a packet trace generated using a traffic matrix scaled to the load factor ρ of 0.4. After each training session, the corresponding model weights are saved.

Testing procedure. For each trained model, corresponding to the tuple $\{\text{traffic matrix, sharing technique, replay memory size, target update period}\}$, a test phase is performed as the following. We ran nine simulations for load factors ρ from 0.6 up to 1.4 with 0.1 step. The rationale behind testing for traffic loads higher than 1.0 is to evaluate the model in highly saturated scenarios, where buffer delays become huge. The test packet traces are generated using the same traffic matrix as training, but scaled to the corresponding load factor ρ . In other words, with respect to the training procedure, we test each DNN model with the same traffic distribution among nodes but with higher loads. The duration (in simulation time) of each *testing* simulation was 20 s, which is sufficient to reach a stationary state in the simulation.

A.2.4.2 Tradeoff between Overhead and Optimality

In this subsection, we evaluate the impact of the control signalling on the learning performance by studying the tradeoff between *average cost per packet* and *control overhead ratio*. The *control overhead ratio* is computed during the training, as the ratio between the total byte count of signalling packets and the total byte count of useful data packets. The results presented are computed as an average over the four traffic matrices, and the nine load factors.

Figures A.7 and A.8 show the *average cost per packet* versus the *replay memory size* and the *target update period* U , respectively. From their observation, first, we see that *DQN routing with value sharing* has a slightly better average cost than SP when choosing the experience replay memory size of 5000 samples. We recall that the original DQN routing algorithm presented in (Mukhtudinov et al., 2019) uses value sharing. However, in (Mukhtudinov et al., 2019), the routing decisions are taken based only on the packet destination, which yields routing policies close to the SP routing since features depending on traffic dynamics, as the buffers' occupancy, are not provided to the neural network. Interestingly, despite the fact that, in the current chapter, we also

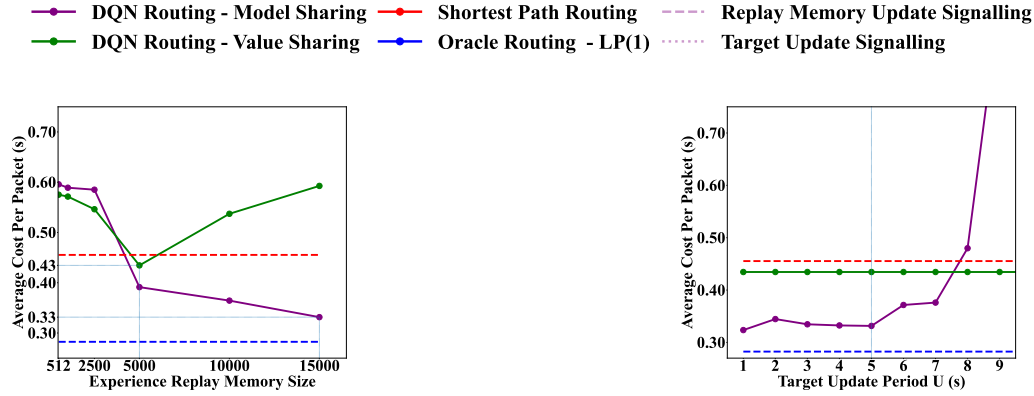


Figure A.7 – Avg. Cost Per Packet vs Replay Memory Size

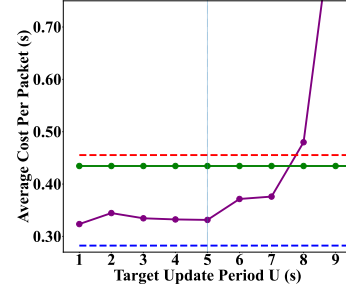


Figure A.8 – Avg. Cost Per Packet vs Target Update Period

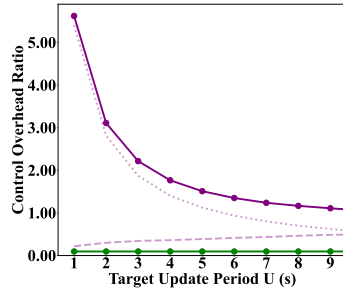


Figure A.9 – Control Overhead vs Target Update Period

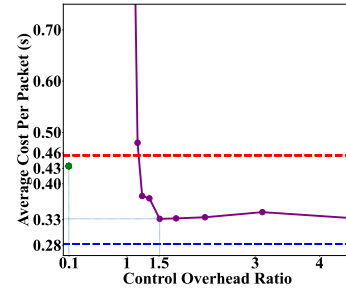


Figure A.10 – Avg. Cost Per Packet vs Control Overhead

add the buffers' occupancy to the neural network input, our enriched version of the original DQN routing in (Mukhutdinov et al., 2019) does not manage to improve significantly over the SP policy, obtaining even worse results for any replay memory size value except 5000, which seems large enough to store the states' diversity, and small enough to not consider too many outdated experiences. In contrast, *model sharing* outperforms the SP routing and gets close to the Oracle routing cost when the replay memory is large enough (more than 5,000 experiences) and the target update period U is small enough (less than 8s). This result suggests that routing agents need to share their DNN models to be able to learn a routing policy close to an Oracle routing. Previous literature (Foerster et al., 2017; Lowe et al., 2017) has shown the importance of conditioning the learning of each agent on an estimate of the other agents' policies to alleviate the *non-stationarity* effect of MA-DRL. Since other agents' policies are continuously updated over time during the training, their Q -value (and, then τ) estimates become obsolete with time. For value sharing, that means that the replay memory can be populated with outdated $\tau_{n'}$ conducting to have different estimates of the *end-to-end delay* from next hop n' to destination for the same state transition $(o_n, a_n, o_{n'})$. Model sharing overcomes this problem by sharing the model $Q_{n'}(\cdot, o_{n'}; \theta_{n'})$. Since $\tau_{n'}$ estimates are computed using Equation (A.5) at the gradient descent update, two identical transitions $(o_n, a_n, o_{n'})$ stored at the replay memory will be associated to the same estimate $\tau_{n'}$ of the *end-to-end delay* from n' .

Figure A.9 depicts the relation between *target update period* U and the *control overhead ratio*. In model sharing strategy, the control overhead strongly decreases with U , since the number of *target update packets* diminishes as well. For value sharing, only the much lighter *replay memory update packets* are sent, giving a much smaller control overhead (0.1). This overhead is shown as constant since it corresponds to only one point (value sharing does not depend on U). Analyzing in detail the overhead for model sharing, we see two different behaviors for the signalling packets. The overhead due to the *target model update packets*, the main overhead source, decreases with the target update period, but the *replay memory update packets* overhead increases linearly. This indicates a growth in packet hops during the training, which is related to a worse training performance for larger U values: the agents do not achieve to find short routing policies (see Figure A.8).

Figure A.10 depicts the tradeoff between *average cost per packet* and *control overhead ratio* for model and value sharing techniques. Replay memory sizes are set to the best values in Figure A.7 for each sharing technique: 5,000 and 15,000 for value and model sharing, respectively. Each point in the curve for DQN routing with model sharing corresponds to a given target update period value U , which is associated to a given cost per packet (see Figure A.8) and a given overhead (see Figure A.9). For value sharing, we have a unique point corresponding to the cost per packet for the best memory size (5,000) and a constant value sharing overhead (0.1). Two observations can be made from this figure: (i) value sharing incurs in a very small overhead (0.1) but cannot improve significantly the SP routing performance, and (ii) model sharing can outperform SP routing and approach the Oracle performance but with much higher overhead (for $U = 5s$, an overhead of 1.5 and a cost 28 % smaller than SP routing and model sharing). These observations provide an important insight: *for the presented settings, the price to pay to become close to the Oracle performance is a control overhead larger than the useful data volume during training*.

A.2.4.3 Packet Cost Performance in Detail

We now analyze the performance of DQN routing for both signalling techniques in terms of: (i) *average cost per packet* (Figure A.11), (ii) *packet loss ratio* (Figure A.12), and (iii) *average end to end packet delay* (Figure A.13). The x -axis represents the load factor, ranging from 0.6 up to 1.4 with 0.1 step, and the y -axis represents the average, the minimum, and the maximum value of the corresponding performance metric over the four traffic matrices. A solid line is used to depict the average value, whereas the area between the minimum and the maximum value is presented with a shaded color. In these figures, the replay memory sizes are fixed to the best values found in the previous subsection: 5,000 and 15,000 for value and model sharing, respectively. Similarly, the target update period for model sharing is set up to the selected value of 5s.

From the observation of Figure A.11, we confirm the remarks stated in the previous subsection: in terms of average cost per packet, the model sharing is always better than value sharing and SP routing, and close to the Oracle routing performance. We recall that average cost per packet is computed as an accumulated reward along a simulation session. Then, it accounts for both buffer delays and packet losses, but giving more importance to the latter to minimize it in the first place (see Section A.2.3.3) as the LP model of the Oracle routing also does (see Section A.2.3.2). Consequently, Figure A.12, that shows the packet loss performance, presents the same trends. Figure A.13 shows the average end-to-end packet delay for the different loads. Interestingly, DQN routing manages to outperform the Oracle routing for medium loads (from 0.7 to 1.1). This can be partly explained by a superior performance in terms of packet loss (if present, the main contributor to the average cost per packet) of the Oracle, namely for the loads 1.0 and 1.1, which leads the

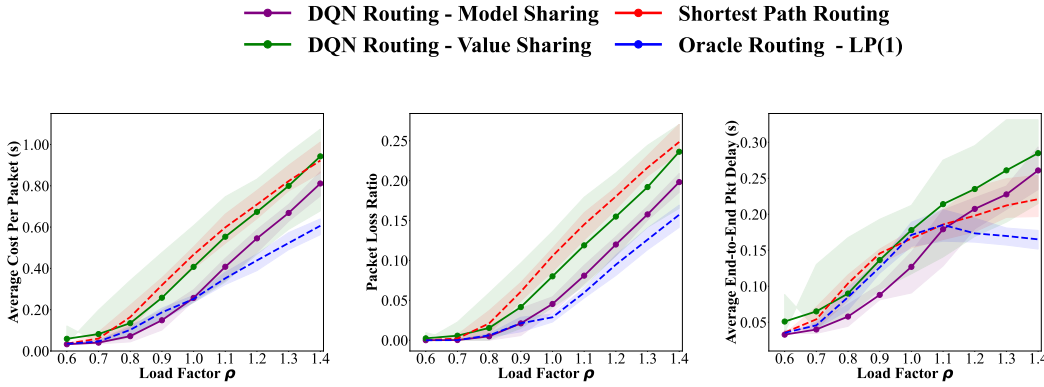


Figure A.11 – Average Cost Figure A.12 – Packet Loss Figure A.13 – Avg. End-to-End Delay.

Oracle to have a poorer behavior in terms of delay. But, for the loads between 0.7 and 0.9, DQN routing outperforms the Oracle in delay and matches its packet loss, yielding a better average cost per packet in Figure A.11. This better behavior of DQN routing can be explained by the flow-based nature of the LP model of the Oracle routing policy. This model has a coarse-grained view of the packets' flows where the notion of packets' buffers is absent: the LP model works with flows defined by an average value in traffic units (i.e., *bps*). On the contrary, the DQN routing has a fine-grained view of the flows, being aware of individual packets. The packets do not arrive at a regular and periodic basis, but randomly following an exponential distribution, giving rise to situations where the instantaneous value of flow in traffic units (i.e., *bps*) exceeds the average flow. In this case, buffers allow absorbing the temporary traffic peaks. Particularly, the packet's buffers can have a bigger impact on the network performance when the average traffic becomes close to the nominal network capacity (medium loads between 0.7 and 0.9), since buffers are mostly empty for lower loads, and saturated for higher loads. Therefore, the DQN routing can benefit from the observation of the buffers' occupancy in these medium loads to overcome the Oracle routing, which is unaware of the buffers.

A.2.5 Discussion

The results presented in this study, especially the ratio between the signalling packets and useful data packets, may change if we change the experimental settings, such as using a larger link rate. Doing so will increase the number of data packets per second and hence the number of *replay memory update packets*. The ratio between *replay memory update signalling* and useful data will remain the same, and so the overhead for *target value sharing*. However, when keeping the same *target update period*, the ratio between *target update signalling* and useful data will decrease, and so the gap in overhead between *model sharing* and *value sharing* will narrow, but the gradient step period T may need to be reduced to maintain the number of experiences per gradient step, and hence possibly have a smaller *target update period*. When changing network topology, the overhead, particularly the one due to *target update signalling*, will scale with the number of nodes, and so if we add more nodes in the network, the overhead will increase. The relationship between the overhead, the performances, and the topology settings will be studied in future works. To conclude, when changing the network configurations, DRL based methods needs

to be re-evaluated in a simulation environment taking into account the overhead of control packets, in order to have a precise estimation of the net gains of such a MA-DRL routing policy.

A.2.6 Conclusion

In this appendix, we have presented an experimental framework to investigate, for the first time to the best of our knowledge, the control signalling mechanisms of distributed learning-based packet routing. In this routing paradigm, node agents must forward packets using only local information, solving the so-called Distributed Packet Routing (DPR) problem. To learn their routing policies, they make use of the DQN routing, a Multi-Agent Deep Reinforcement Learning (MA-DRL) algorithm. This framework imposes the neighboring agents to collaborate during the learning by sharing estimates of the end-to-end delays. We have considered two main control exchange strategies, *value sharing* and *model sharing*, which we have evaluated in terms of routing cost combining packet delay and loss rate, each component, considering the impact of the overhead packets due to control. On the Abilene topology for various loads and traffic matrices, we have shown that *model sharing* yields routing policies able to approach the optimal one provided by an oracle within 18%, considering the additional control packets. These represent a control overhead of 150% of extra traffic in the considered network scenarios. Conversely, *value sharing* adds a very small control overhead (approximately 10 %), but it cannot learn a routing policy close to the Oracle one, only managing to slightly improve a Shortest Path (SP) routing. We finally provide a discussion on the impact of our results, detailing how the ratio of overhead would evolve when changing base link rates and topologies. From the present study, we conclude by underlying the importance of assessing rigorously the signalling entailed by multi-agent DRL packet routing to obtain a precise estimation of net gains considering traffic overhead due to signalling. After this experimental analysis, our future works will investigate how MA-DRL performance scales with changes of network parameters, in particular link rates, topology, and load.

APPENDIX B

Small is Sufficient: AI tasks and benchmarks description

The rapid advancement of Artificial Intelligence (AI) has led to the widespread adoption of AI models across a varied range of tasks and applications, contributing to growing concerns over energy consumption and environmental impact. In this context, in Chapter 5, we propose estimating the potential energy savings achieved by applying model selection during inference. This practice involves choosing energy-efficient models that decrease the energy footprint while maintaining good performance. For this, given the growing accessibility of several pre-trained models on platforms such as Hugging Face and Papers with Code, we first selected a set of widely adopted AI tasks on these platforms. Then, for each evaluated task, we selected representative benchmarks that assess the model's performance and collect its size, which is used for estimating the model's energy consumption. In this Appendix, we describe the set of evaluated AI tasks and the benchmarks used in our analysis.

B.1 Text generation

The text generation task consists of creating a text based on some input which contains the instructions. The text generated should be clear and coherent logically and grammatically.

Benchmark: *Open LLM Leaderboard* (Beeching et al., 2023) The benchmark was developed by *Hugging Face*, using according to Eleuther framework (L. Gao et al., 2021). The benchmarks evaluate the generative models in tasks such as the ability to follow specific formatting instructions, high-school mathematical problems, algorithmic generated problems and multiple-choice knowledge questions.

Benchmark: *LMSys Chatbot arena Leaderboard* (Chiang et al., 2024; Zheng et al., 2024) This benchmark, developed by LMSYS and UC Berkeley, evaluates text generation models using *chatbot arena*, using a score based on humans preferences.

B.2 Mathematical reasoning

In mathematical reasoning, the systems are requested for solving a wide range of mathematical problems.

Benchmark: *NPHard Eval Benchmark* (Fan et al., 2023) (mathematical reasoning) The *NPHard* benchmark evaluates the LLM power for reasoning in mathematical questions. It contains over than 900 questions and the objective is to classify the problems into 3 categories: P, NP-complete and NP-hard.

B.3 Code generation

In code generation tasks, the models, usually based on Natural Language Processing (NLP), are requested for generating source code. The tasks normally involve translating pseudocode to executable code, autocompleting functions and translating code between programming languages.

Benchmark: *BigCode Leaderboard* (Ben Allal et al., 2022) (code generation) The *big-Code* models leaderboard evaluates LLM for generating code scripts. The leaderboard evaluates over two main datasets: (i) *HumanEval*, that analyzes the functional correctness of synthesizing python programs; and (ii) *MultiPL-E*, which translates the previous dataset into other programming languages.

B.4 Translation

In text translation task, the models receive a task in a certain language, and they should generate the task translated into another language.

Benchmark: Machine Translation english-german (Bojar et al., 2014) The *WMT (World Machine Translation)* is a dataset which evaluates pairs of languages. The main metric used is the *BLEU score*, which measures how many *n-grams* (sequences of words) in the machine-generated translation match those to a reference translation.

B.5 Text classification

The text classification task consists of assigning a label or a category based on an input text. This task may be used for several applications such as spam detection, sentiment analysis, content classification and sentence similarity. Usually, the task involves data preprocessing, features extraction and then, the model training and evaluation.

MTEB Leaderboard (Muennighoff et al., 2022) The *MTEB* leaderboard evaluates LLM models for classification tasks. *MTEB* evaluates over 58 datasets and 112 languages. This comprises 8 embedding tasks: Bitext mining, classification, clustering, pair classification, re-ranking, retrieval, STS and summarization.

B.6 Text Clustering

In text clustering, the models are requested for grouping similar textual documents based on the content. The models aim to identify patterns and structures, forming clusters where samples with similar content are clustered in the same group.

The benchmark used for this task was the *MTEB Leaderboard* (Muennighoff et al., 2022), as in *text classification*.

B.6.0.1 Object detection

In the object detection task, the model must identify and locate multiple objects in an image.

Benchmark: Open Object Detection (Rafael Padilla & the Hugging Face Team, 2023) This benchmark was developed by *HuggingFace*. The dataset used is the *Microsoft Common Objects in Context (COCO)*, containing 80 object categories, images with complex scenes, and high-quality and manually annotated labels. The main metric used for evaluating the models is the average precision.

B.7 Image classification

In the image classification task, the models should assign a label to an image based on its visual content. The main metric used is the accuracy, which counts the number of correct classified images over the total of images.

Benchmark: *ImageNet* (Russakovsky et al., 2015) The *ImageNet* dataset is a large-scale visual database for image classification. The subset used in this study, *ImageNet-1k*, comprises over 1.2 M images categorized into 1000 classes.

We used two sources for the *ImageNet* dataset as we tested both models reported in *Hugging Face* and in *Papers With Code* for this task: (i) from the *PyTorch Timm Leaderboard* (Wightman, 2019), which evaluates PyTorch models in the *Hugging face* platform; and (ii) from the *Papers With Code* platform addressing *Image Classification on ImageNet*¹.

B.8 Image segmentation

In the semantic segmentation task, the goal is classifying each pixel in the image, providing a good understanding of the entire scene.

Benchmark: *ADE20K* (B. Zhou et al., 2017) The *ADE20K* consists of a dataset containing over 20k images and more 150 categories, such as cars, trees, and people.

B.9 Text to image

In the text to image task, the models are requested for creating visual content from input text prompts. The goal is to produce high-quality images which can be used for several applications, such as art, design and medical imaging.

Benchmark: *GenAI* (Ku et al., 2024) The *GenAI* benchmark evaluates 17 models using ELO rating system based on public vote between the given results of 2 different models. The benchmark counts more than 8k votes for formulating the model score.

B.10 Automatic speech recognition

The speech recognition task consists of translating an audio extract into a text. It involves processing audio input, identifying the words spoken, and transcribing them accurately.

Benchmark: *Open ASR Leaderboard* (Srivastav et al., 2023) The *Open ASR Leaderboard* was developed by *HuggingFace*. For evaluating the models, the benchmark utilizes a set containing six datasets as described in (Gandhi, von Platen, & Rush, 2022). The metric used is the *Word Error Rate (WER)*, which computes the percentage of words in the system's output that are different from the reference transcript.

1. <https://paperswithcode.com/sota/image-classification-on-imagenet>

B.11 Audio classification

The audio classification task involves assigning a label to an audio signal input.

Benchmark: *ARCH Benchmark* (La Quatra et al., 2024) *ARCH* benchmark aims to benchmark models for audio classification task. The benchmark evaluates the models across 12 datasets with different input data: sound events (4 datasets), music samples (4 datasets) and speech samples (4 datasets).

B.12 Image-text to text

In image-text to text tasks, the models are provided with images and text prompts as inputs and should output text. Also called as vision-language models (VLMs), the models used in this tasks are widely used for several applications, such as multimodal dialogue, visual question answering and image recognition.

Benchmark: *MMMU benchmark* (Yue et al., 2024) The *MMMU* benchmark evaluates multimodal models on massive multidiscipline tasks demanding college-level subject knowledge and reasoning. *MMMU* includes 11.5K multimodal questions from college exams.

B.13 Time series forecasting

The time series forecasting task consists of a regression task, in which the AI models should predict the future elements of a time series based on historical trends in the past.

Benchmark: *etth1-336* (Gemmeke et al., 2017) The *Etth1* benchmark evaluates the models on forecasting a time series. The time series records the operation and environmental characteristics of electricity transformers along time. Each data point consists of the target value “oil temperature” and six power load features.

Optimisation des performances et de la consommation énergétique pour l'inférence dans le domaine de l'apprentissage automatique

Tiago DA SILVA BARROS

Résumé

Les progrès rapides de l'utilisation de l'IA ont conduit à une augmentation significative des besoins en calcul. L'émergence de nouveaux paradigmes de déploiement, tels que le MLaaS (Machine Learning as a Service), et la popularité croissante de Large Language Models (LLM) notables, tels que GPT, Gemini et Deepseek, ont intensifié cette demande. En conséquence, la consommation d'énergie associée aux modèles d'IA a considérablement augmenté, soulevant des inquiétudes quant à leur empreinte environnementale.

Dans cette thèse, nous abordons la demande croissante en ressources et proposons des stratégies d'optimisation pour améliorer l'efficacité énergétique lors de l'inférence IA. Pour répondre à ces questions, nous abordons deux problèmes. Dans le premier, nous considérons un système comportant plusieurs tâches d'inférence avec des délais stricts qui doivent être planifiées sur un ensemble de machines. De plus, les utilitaires de compression de modèles appliqués pendant le temps d'inférence permettent à chaque tâche d'être exécutée avec une certaine compression, ce qui présente un compromis entre son niveau de compression (et son temps de traitement) et son utilité obtenue. L'objectif est de maximiser l'utilité de la tâche. Nous proposons donc des algorithmes d'approximation avec des garanties éprouvées pour résoudre le problème et étendre le modèle afin d'intégrer les contraintes budgétaires énergétiques.

Nous avons ensuite étudié les impacts environnementaux croissants associés à l'utilisation de l'IA. Récemment, de nouveaux paradigmes en matière d'IA verte ont émergé, passant du principe « plus c'est gros, mieux c'est », qui privilégie les grands modèles, à celui « petit suffit », qui met l'accent sur la sobriété énergétique grâce à des modèles plus petits et plus efficaces. Nous étudions comment la communauté de l'IA peut adopter la sobriété énergétique aujourd'hui en se concentrant sur la sélection des modèles pendant l'inférence. Nous avons estimé les économies d'énergie réalisées en sélectionnant le modèle pour exécuter les demandes d'inférence. Enfin, afin d'analyser de manière globale les implications environnementales de l'IA, nous avons réalisé une analyse du cycle de vie (ACV) de Pl@ntNet, une application basée sur l'IA. Celle-ci comprend une évaluation multicritères et multi-étapes afin d'obtenir une image complète de son empreinte écologique.

Mots-clés : Intelligence Artificielle, Efficacité Énergétique, optimisation, Ordonnancement, Compression de Modèles, Sélection de Modèles

Abstract

The rapid advancement of AI usage has led to a significant increase in computing demands. The emergence of new deployment paradigms, such as MLaaS (Machine Learning as a Service), and the growing popularity of notable Large Language Models (LLMs), such as *GPT*, *Gemini* and *Deepseek*, has intensified this demand. Consequently, the energy consumption associated with AI models has increased considerably, raising concerns about their environmental footprint.

In this thesis, we address the AI increasing resource consumption demand and propose optimization strategies for improving energy efficiency during AI inference. To address these issues, we tackle two problems. In the first one, we consider a system with several inference tasks with hard deadlines that must be scheduled onto a set of machines. Furthermore, model compression utilities applied during inference time allow that each task can be executed with a certain compression, presenting a trade-off between its compression level (and, its processing time) and its obtained utility. The objective is to maximize the tasks utilities. We propose, thus, an approximation algorithm with proven guarantees to solve the problem and extend the model to incorporate energy budget constraints.

We then investigated the increasing environmental impacts associated with AI usage. Recently, new paradigms in green AI emerged, shifting from “*bigger is better*”, which prioritizes large models, to “*small is sufficient*” emphasizing energy sobriety through smaller and more efficient models. We investigate how the AI community can adopt energy sobriety today by focusing on model selection during inference. We estimated the energy savings by selecting the model for executing inference requests. Finally, to broadly analyze the environmental implications of AI, we conduct a Life Cycle Assessment (LCA) of Pl@ntNet, an AI-based application. This includes a multi-criteria, multi-stage evaluation to capture a comprehensive picture of its ecological footprint.

Keywords: Artificial Intelligence, Energy Efficiency, Optimization, Scheduling, Model Compression, Model Selection