Energy-aware high-performance artificial intelligence : from measurement and modeling to multi-objective scheduling

Roblex Nana Tchakoute

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Mine Paris - PSL

Energy-Aware High Performance Artificial Intelligence: From Measurement and Modeling to Multi-Objective Scheduling

Intelligence artificielle haute-performance efficace en énergie : de la mesure et la modélisation à l'ordonnancement multi-objectifs

Soutenue par

Roblex NANA TCHAKOUTÉ

Le 05 Décembre 2025

École doctorale n°621

Ingénierie des systèmes, matériaux, mécanique, énergétique

Spécialité

Informatique temps-réel, robotique et automatique

Composition du jury :

Christophe CÉRIN Professeur, Université Paris 13	*Président du jury* *Rapporteur*
Laurent LEFÈVRE Directeur de recherche, INRIA Lyon	*Rapporteur*
Anne BENOIT Full professor, ENS Lyon	*Examinatrice*
Anne-Cécile ORGERIE Directrice de recherche, CNRS	*Examinatrice*
Nahid EMAD Professeure émérite, UVSQ	*Examinatrice*
Petr DOKLADAL Chargé de recherche, Mines Paris - PSL	*Examinateur*
Youssef MESRI Professeur, Mines Paris - PSL	*Examinateur*
Claude TADONKI Chargé de recherche, Mines Paris - PSL	*Directeur de thèse*

MINES PARIS | PSL

Acknowledgments

This doctoral thesis represents the culmination of a long and challenging, yet profoundly rewarding, journey. Its completion would not have been possible without the guidance, support, and encouragement of a great many people to whom I owe my deepest gratitude.

First and foremost, my most profound thanks go to my supervisor, Professor **Claude Tadonki**. His unwavering guidance, scientific rigor, and unwavering trust have been the cornerstones of this work. His mentorship extended far beyond the technical, shaping my approach to research and providing the intellectual freedom necessary for this project to flourish.

I am sincerely grateful to my co-supervisors, Professors **Petr Dokladal** and **Youssef Mesri**. Their invaluable expertise and collaborative spirit were instrumental in navigating the complexities of this research, and their contributions significantly enriched the quality of this manuscript.

I wish to thank my PhD reviewers, Professors **Christophe Cérin** and **Laurent Lefèvre**, for their thorough and constructive feedback. My gratitude also extends to the entire jury (Professors **Nahid Emad**, **Anne Benoit**, and **Anne-Cécile Orgerie**) for their insightful questions and deep engagement with my work.

A special and heartfelt thank you is owed to Professor **Corinne Ancourt**, Head of the Centre de recherche en informatique (CRI). Her immense support was a constant throughout my PhD. Beyond her role as a lab director, she acted as a true mentor, assisting with everything from the perfect organization of my defense to administrative and personal matters. Her daily smile was a source of profound motivation and a vital antidote to the inevitable pressures of doctoral research. I am also deeply grateful for the warm and motherly welcome extended to me by **Claire Medrala** upon my arrival at the laboratory, a gesture of kindness I will not forget.

I would also like to thank the permanent staff members of the CRI. My gratitude goes to Professor **Pierre Jouvelot** for the many critical and rich scientific discussions during seminars, which sharpened my skills in formalism and modeling. A sincere thanks to Professors **Olivier Hermant** and **Fabien Coelho**, who served on my thesis monitoring committee (Comité de Suivi Individuel) and were always available to assist. I am particularly thankful to Olivier for a gesture of kindness that went far beyond his official duties, in driving me to the supermarket upon my arrival in France, and to Fabien for his crucial help with logistics on the day of my defense. My thanks also extend to **Catherine Imbert** for her remarkable efficiency in managing my travel for conferences, and to **Laurent Daveiro** and Professor **Georges-André Silber** for the enriching discussions during seminars and lunch breaks.

This research would not have been possible without institutional support. I am profoundly grateful for the financial support provided by **The Transition Institute 1.5**, and I extend my thanks to the entire **Grid'5000** team for providing the experimental testbeds that made these findings achievable.

My journey into HPC and Computer Architecture began years ago. I owe a significant debt of gratitude to my Master's thesis supervisors, Professor **Jean-François Méhaut** and Professor **Maurice Tchuente**, who first ignited my passion for this field. I must also thank all my professors and collaborators from my master's studies in Yaounde who laid the foundations of my academic career, with a special acknowledgment for Professor **René Ndoundam**, whose recommendation opened the door to this incredible opportunity.

On a peer level, I have been fortunate to share this journey with wonderful colleagues. A special thanks to **Vassili Maillet**, who became a brother to me during these years. I will cherish the many unforgettable moments, both inside and outside the lab, and will greatly miss our after-work activities. My warm thanks go to **Carla Santana** and **Robin le Conte des Floris**, who were indispensable to my integration in France during my first year. To my colleagues on the "Energy Team," **Antoine Solcourt, Benjamin Destal**, and **Antony Bertrand**, I thank you for the weekly technical discussions from which I learned so much. My thanks also go to **Arthur Viens, Luc Saulau, Quentin Petit, Youssef Attia El Hili**, and all the other PhD students of the lab for their kindness and the vibrant scientific environment we shared. I extend a special note of gratitude to **Chahinèze Ztoti**, our outstanding intern, for her kindness and the many great moments we shared. To all the other interns, thank you. My gratitude also extends to my dear friend **Delphine Doutsas** from the CMM (now STIM), and to all the friendly PhD students and researchers from the other Mines Paris research centers.

The support I received during these years also extended beyond the walls of my current laboratory, sustained by friendships forged long ago. I want to express my deepest thanks to **Vanessa Fokou**, for her presence and invaluable assistance during my PhD defense, and for the countless discussions where we navigated the shared challenges of doctoral research. Her readiness to help, irrespective of the subject, was a true testament to her friendship. A special thought goes to **Gabril Matoko**, whose company during our Saturday discoveries and basketball games provided a much-needed and refreshing counterpoint to the intensity of research. I am also deeply grateful to the friends and classmates from my time at the University of Yaounde I, with whom I remained in close contact throughout these three years. Thank you to **Roméo Koati, Brice Tchuenkam, Gaius Libam, Sorelle Kana, Nadia Tsessu, Glwadys Kelodjou, Audrey Dongmo, Cédric Bessala, Antoine Tsagmo**, and **Quentin Kouamo**, for sharing in the painful and happy moments of this long journey. The list of cherished friends is long, and I extend my heartfelt thanks to all those not named here whose friendship has been a source of strength.

On a deeply personal note, none of this would have been possible without my family. Their unconditional love and support since my childhood have been my bedrock. I owe everything to my mother **Mme Florence Nkuika**, and I extend my deepest love and thanks to my **brothers** and **sisters** for their constant encouragement and understanding throughout this long project.

The full list of people who deserve thanks is far too long for a single chapter—it would require a dedicated book. To every friend, colleague, and mentor who played a part : thank you from the bottom of my heart.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACPI Advanced Configuration and Power Interface
ADEPT Analytical Deep-learning Energy and Performance Time-estimator
AI Artificial Intelligence
ALU Arithmetic Logic Unit
AMP Automatic Mixed Precision
API Application Programming Interface
ASIC Application-Specific Integrated Circuit
AVX Advanced Vector Extensions
CI/CD Continuous Integration / Continuous Deployment
CLI Command-Line Interface
CNN Convolutional Neural Network
CPU Central Processing Unit
DCiE Data Center Infrastructure Efficiency
DE Differential Evolution
DL Deep Learning
DRAM Dynamic Random Access Memory
DVFS Dynamic Voltage and Frequency Scaling
EA2P Energy-Aware Application Profiler
EAS-Sim Energy-Aware Scheduling Simulator
FLOPS Floating-point Operations Per Second
FMA Fused Multiply-Add
FPGA Field-Programmable Gate Array
GEMM General Matrix-Matrix Multiplication
GPU Graphics Processing Unit
HBM High Bandwidth Memory
HPC High-Performance Computing
HPCs Hardware Performance Counters
HPL High-Performance LINPACK
I/O Input/Output
IPC Instructions Per Cycle

JCT Job Completion Time
KPI Key Performance Indicator
LLM Large Language Model
MAPE Mean Absolute Percentage Error
MLOps Machine Learning Operations
MPI Message Passing Interface
MSR Model-Specific Register
NAS Neural Architecture Search
NPU Neural Processing Unit
NVM Non-Volatile Memory
NVML NVIDIA Management Library
OOM Out-Of-Memory
OS Operating System
PAPI Performance Application Programming Interface
PMIC Power Management Integrated Circuit
PUE Power Usage Effectiveness
QoS Quality of Service
RAM Random Access Memory
RAPL Running Average Power Limit
RL Reinforcement Learning
SIMD Single Instruction, Multiple Data
SJF Shortest Job First
SLA Service Level Agreement
SoC System-on-Chip
SOTA State-of-the-Art
SPMV Sparse Matrix-Vector Multiplication
SSD Solid-State Drive
TDP Thermal Design Power
TPU Tensor Processing Unit

Glossary

Analytical Model A mathematical model that describes a system's behavior based on an understanding of its underlying physical or structural properties, as opposed to a black-box empirical model derived from statistical correlations.

Backfilling A strategy where a scheduler allocates resources that would otherwise be idle to smaller, lower-priority jobs, without delaying the start time of any higher-priority job.

Baseline A set of quantitative performance and energy measurements of a system in its initial, un-optimized state. It serves as a fixed reference point against which all optimization gains are measured.

Benchmarking The process of running a standardized set of programs (kernels or applications) on a system to measure and assess its performance, energy consumption, or other characteristics under controlled conditions.

Black-Box Model A predictive model, typically based on machine learning, that learns a statistical relationship between inputs and outputs without providing insight into the underlying causal mechanisms.

Characterization The process of systematically measuring and analyzing the performance and energy profile of an application or hardware component under various conditions to understand its fundamental behavior.

Co-Design An engineering methodology where multiple aspects of a system (e.g., scheduling policy, hardware configuration, energy objectives) are designed and optimized concurrently, rather than in isolation.

Compute-bound A state where the execution time of a program is limited by the speed of its computational operations on the CPU or GPU, rather than by the speed of memory access or I/O.

Dennard Scaling An observation from the 1970s which posited that as transistors shrink, their power density remains constant. Its end around 2006 marked a fundamental shift in processor design toward multi-core architectures.

Dynamic Power The component of a processor's power consumption that is due to the switching activity of transistors when performing computations. It is a primary target of optimization techniques like DVFS.

Energy-Delay Product (EDP) A core metric used in this thesis, defined as $E \times T = P \cdot T^2$. It provides a balanced measure of a system's overall efficiency by penalizing configurations that are either excessively slow or excessively power-hungry, and was used as a central heuristic for our energy-aware schedulers.

Fairness A multi-dimensional objective in scheduling that can refer to either ensuring long-term equitable distribution of resources among users (Resource Allocation Fairness) or providing similar job completion times for all users (JCT Fairness).

Heterogeneity In the context of HPC clusters, the state of being composed of different types of computational resources, such as multiple generations or models of CPUs and GPUs, which a scheduler must intelligently manage.

Heuristic A problem-solving approach or rule used in an algorithm, like a scheduler, to find a good-enough solution quickly when finding a perfect, optimal solution is computationally intractable.

Hot Spots In program analysis, the specific sections of code (functions or loops) that are responsible for the vast majority of the execution time or resource consumption.

Inference The phase in the AI model lifecycle where a pre-trained model is used to make predictions on new, unseen data. It is typically less computationally intensive than training but is executed far more frequently.

Malleability The ability of a Deep Learning job to be executed with a variable number of resources (e.g., number of GPUs) and configurations (e.g., batch size per GPU). It is a key lever for advanced scheduling.

Memory-bound A state where the execution time of a program is limited by the speed at which data can be transferred from main memory to the processor, rather than by the speed of computation.

Multi-Objective Optimization The process of optimizing a system based on multiple, often conflicting, objectives simultaneously (e.g., minimizing energy while maximizing performance and ensuring fairness).

Oracle (Performance) A component within a scheduler or simulator that can predict the performance (time) and power consumption of a given job on a given hardware configuration.

Pareto Frontier In multi-objective optimization, a set of solutions where it is impossible to improve one objective without worsening at least one other. In our work, we identified a new energy-performance Pareto frontier defined by our Zeus scheduler.

Pipeline Stall A situation in a processor's pipeline where the execution of instructions is halted, often because the compute units are waiting for data to arrive from a slower memory subsystem. This is a primary source of energy waste in memory-bound workloads.

Preemption A scheduling capability that allows a high-priority job to interrupt a running, lower-priority job, forcing the latter to release its resources. It is critical for ensuring Quality of Service in high-contention environments.

Scheduler A system software component responsible for allocating computational resources (like GPUs) to submitted jobs according to a defined policy, aiming to optimize objectives such as performance, fairness, and energy.

Static Power The component of a processor's power consumption caused by leakage currents, which flows through transistors even when they are not actively switching. It is a significant factor in modern, highly dense microchips.

Throughput A measure of system performance, typically quantified as the number of jobs or tasks completed per unit of time (e.g., jobs per hour).

Training The computationally intensive phase in the AI model lifecycle where the model's parameters (weights) are learned by iteratively processing a large dataset.

White-Box Model An analytical model whose internal workings are understandable and interpretable, allowing for a causal analysis of its predictions.

Workload The collection of computational jobs and tasks that are submitted to a system for execution over a period of time.

Chapter 1

General Introduction

Ce premier chapitre introduit la problématique au cœur de cette thèse, qui émerge de la convergence entre le Calcul Haute Performance (HPC) et l'Intelligence Artificielle (IA). Après avoir défini ces concepts et souligné le rôle central des infrastructures de calcul, le chapitre détaille la crise énergétique latente qui résulte de l'explosion des besoins computationnels. Cette crise est présentée comme un défi multidimensionnel, avec des impacts économiques (coûts d'exploitation), environnementaux (empreinte carbone, consommation de ressources) et des limitations physiques (mur de la puissance, fin de la loi de Dennard) qui freinent l'évolution technologique. Face à cet impératif de soutenabilité, nous argumentons que la réponse ne peut être uniquement matérielle mais doit aussi provenir du système et du logiciel. Le chapitre expose alors les verrous scientifiques majeurs liés à la mesure précise de l'énergie, à la modélisation prédictive de la consommation et à la conception de stratégies d'optimisation intelligentes. Enfin, cette introduction présente les contributions spécifiques de la thèse qui visent à lever ces verrous et en décrit l'organisation dans le manuscrit.

1.1 General Context : High-Performance Computing and Artificial Intelligence

The 21st century is deeply marked by a digital revolution that is transforming how we live, work, and perceive the world. The heart of this transformation relies on exponentially growing of computational capabilities. Two major topics in this evolution are : *High-Performance Computing (HPC)* and *Artificial Intelligence (AI)*. To fully grasp the stakes of this thesis, it is necessary to define these two fundamental concepts and understand their growing synergy, as well as the infrastructure that supports them.

1.1.1 What is High-Performance Computing ?

High-Performance Computing, commonly referred to by the acronym HPC, involves the use of extraordinarily powerful computer systems to solve complex problems that exceed the capabilities of conventional computers. There is a wide spectrum of HPC applications [Tadonki,

2023], including : weather forecasting, cutting-edge simulations, large scale operational research, and large scale Artificial Intelligence, to name a few. These tasks require processing immense amounts of data and performing billions, or even trillions, of mathematical operations per second. The HPC ecosystem [Tadonki, 2013] is characterized by :

— **Computational Power :** The performance of these systems is often measured in *FLOPS* (Floating-point Operations Per Second). The most powerful HPC systems, known as supercomputers, can achieve performance on the order of ExaFLOPS, which is 10^{18} (a billion billion) operations per second. An overview of performance evolution of the top supercomputers form TOP500 List [1] is showed on Figure 1.1.

— **Architecture :** An HPC system is generally not a single giant computer, but rather a cluster—a collection of numerous individual computers (called compute nodes) interconnected by a very high-speed network. Each node contains processors (CPUs), random-access memory (RAM), and often specialized accelerators like Graphics Processing Units (GPUs), which are particularly effective for certain types of parallel computations [Nana, 2024a].

— **Applications :** Historically, HPC has been the domain of numerical simulation in various scientific disciplines (*physics, chemistry, biology, climatology, astrophysics*), engineering (aerospace and automotive design, virtual crash tests), and large-scale data analysis [Tadonki, 2023].

HPC is therefore an essential support for pushing the boundaries of science and technological innovation by enabling the virtual exploration of phenomena that are too complex, too large, too small, too fast, or too dangerous to be studied directly through real experimentation.

1.1.2 What is Artificial Intelligence (AI) ?

Artificial Intelligence (AI) is a computing paradigm that aims to create systems capable of performing tasks that normally require human intelligence. This includes capabilities such as *reasoning, learning, visual or auditory perception, understanding natural language*, and *decision-making* (see Figure 1.2). Rather than being explicitly programmed for every possible situation, many modern AI approaches rely on Machine Learning. Its most popular subfields are :

— **Machine Learning :** These are techniques that allow computers to *learn* from data without being explicitly programmed. The computer analyzes large quantities of examples (images, texts, numerical data) and progressively adjusts a mathematical model to identify patterns or make predictions.

— **Deep Learning :** This is a particularly powerful subfield of machine learning, inspired by the structure and functioning of the human brain (neural networks). Deep neural networks consist of multiple layers of artificial *"neurons"* that process information hierarchically. They have proven extremely effective for complex tasks such as image recognition, automatic translation, or speech comprehension [Goodfellow, 2016].

1. https ://top500.org/

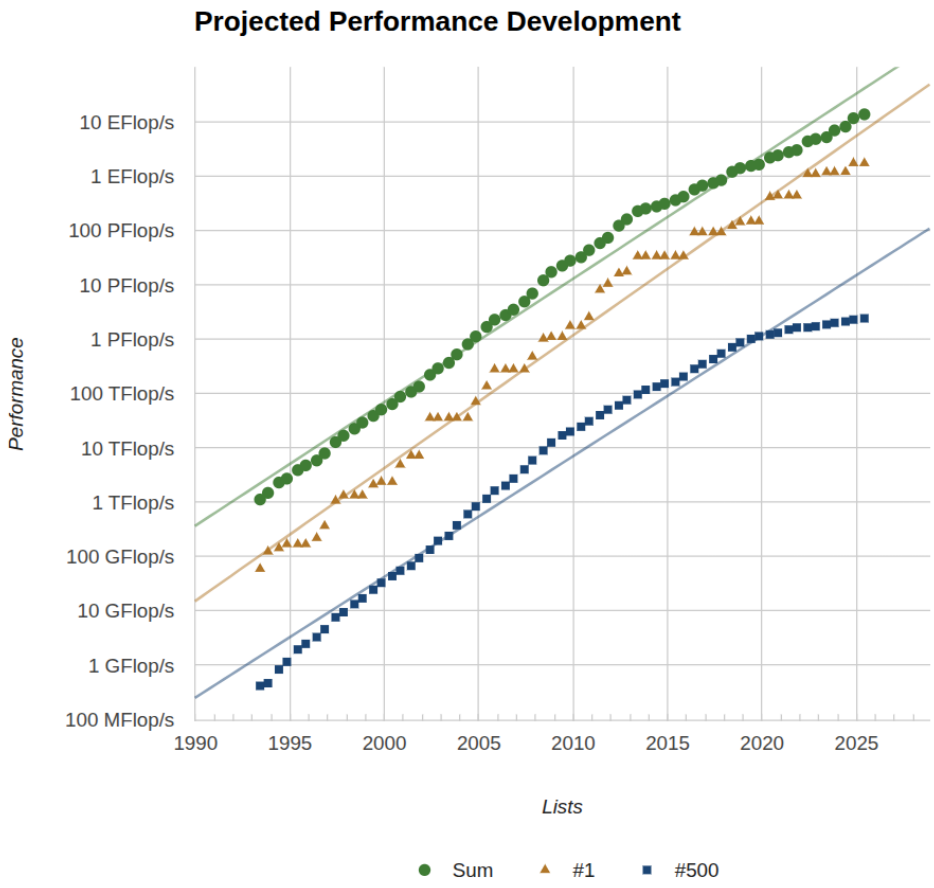


FIGURE 1.1 – Performance prediction of top 500 supercomputers. Source : top500.org

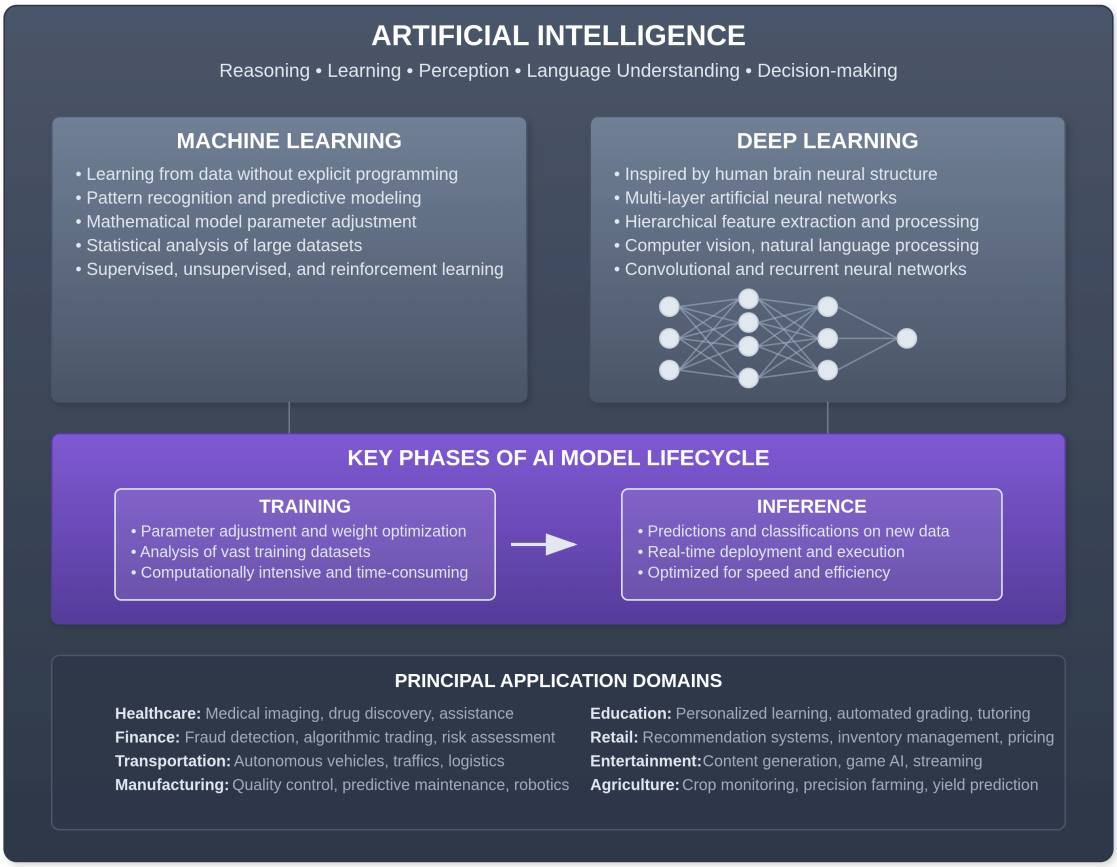


FIGURE 1.2 – General view of AI concepts and mains application domains

The lifecycle of an AI model typically involves two major phases : Training and Inference. **Training** is the initial learning process, where the model adjusts its parameters (called weights) by analyzing a vast dataset of examples. This phase is often very costly in terms of computation and time. **Inference** is the use of the trained model to make predictions on new, unseen data. For example, recognizing a face in a new photo or translating a new sentence. This phase is generally less costly than training but must often be fast and efficient, especially when deployed in real-time.

1.1.3 Data Centers : A Pivotal Infrastructure

Neither HPC nor AI could operate at large scale without a physical infrastructure to manage (house, power, and cool) the thousands, or even millions, of servers required. This is the role of data centers. A **data center** is a building, a dedicated space within a building, or a group of buildings used to house computer systems and associated components, such as telecommunications and storage systems [2]. Its functions include, among others : *Concentration of Resources, Support Infrastructure*, and *Cloud Computing*. Data centers consolidate a high density of servers, data storage systems, and network equipment in a secure location. They also provide all the essential support infrastructure, namely : redundant power supplies (with backup generators and uninterruptible power supplies to prevent outages), sophisticated cooling systems (air conditioning, liquid cooling) to dissipate the heat generated by the equipment, physical security (access control), and high-bandwidth network connectivity [Barroso, 2013]. A large portion of Cloud Computing services is built upon vast data centers operated by companies like Amazon (AWS), Microsoft (Azure), or Google (GCP) [3]. These services allow users (*businesses, researchers, individuals*) to access computing resources (*computation, storage, software, AI platforms*) on demand via the Internet, without having to manage the underlying physical infrastructure themselves. HPC platforms are also increasingly accessible via the Cloud.

Data centers are thus the invisible *"factories"* of the digital age, the material pillars upon which HPC, AI, and a multitude of other online services rest. Their reliable and efficient operation is crucial, but their concentration of power poses major challenges, particularly in terms of *energy consumption*.

1.1.4 HPC-AI Convergence and the Emergence of the Digital Continuum

HPC and AI are increasingly converging, thus creating a virtuous cycle, but also new challenges that include :

— **AI as an HPC Workload :** Training large AI models (LLMs, computer vision models) has become one of the most computationally demanding applications, pushing the capabilities of supercomputers and Cloud infrastructures to their limits. These training runs sometimes require thousands of GPUs operating in parallel for days or weeks.

2. https ://en.wikipedia.org/wiki/Data_center
3. https ://www.cloudzero.com/blog/cloud-service-providers/

— **AI for HPC :** Conversely, AI techniques are being used to enhance HPC. For example, AI models can learn to predict the behavior of complex simulations to accelerate them, analyze the terabytes of data generated by these simulations, or even dynamically optimize resource allocation and the execution parameters of HPC codes [Vallabhajosyula, 2023 ; Brace, 2022].

— **Architectural Evolution :** This convergence is reflected in system architecture. GPUs, originally designed for graphics, have become the workhorses of AI and are now ubiquitous in HPC systems. New types of specialized AI accelerators (TPUs, NPUs, etc.) are also appearing in these systems.

— **The Cloud-to-Edge Continuum :** AI is no longer confined to large data centers. Inference, in particular, is increasingly being deployed *at the edge* (Edge Computing), i.e., on smaller devices closer to the user or data source (smartphones, connected objects, autonomous vehicles, industrial sensors). This creates a digital continuum, where processing can be intelligently distributed between *Edge devices* (for responsiveness and privacy), *regional data centers* (Fog Computing), and *large Cloud centers* (for training and intensive computation).

This HPC-AI synergy, operating on an infrastructure that spans from large data centers to small Edge devices, multiplies the possibilities for innovation but also intensifies the pressure on computing resources and, consequently, on the overall energy consumption of the digital sector (see Figure 1.3).

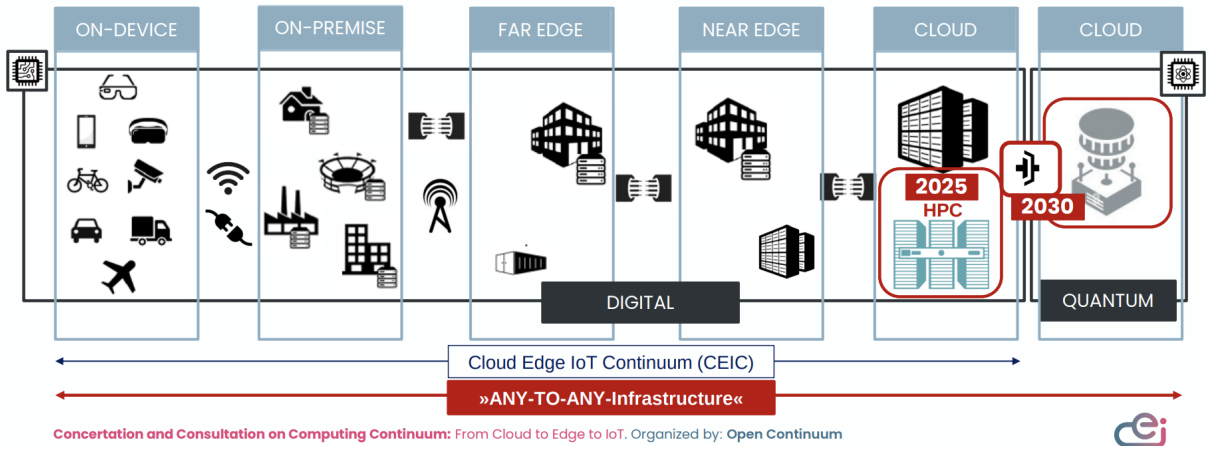


FIGURE 1.3 – The "Big Figure" of new computing continuum. Source : Michael Fritz

1.2 The Digital Energy Crisis : A Multidimensional Challenge

While the advances enabled by HPC and AI are noticeable, they come with an increasingly concerning counterpart : *their energy consumption*. This latent *"energy concern"* in the digital world is a complex phenomenon with multiple facets.

1.2.1 The Exponential Growth of the Energy Needs

The digital sector as a whole (including device manufacturing, networks, and usage) accounts for a growing and already significant share of global electricity consumption. Estimates vary, but values from of 4% to 10% of global electricity production are often indicated for the entire lifetime of digital technology. According to "Bird & Bird" [Lang, 2023], the high-tech sector is responsible for around 7% of global electricity consumption with a prediction of 13% by 2030. Data centers, which concentrate a significant portion of HPC and AI computations, play a major role in this consumption. IEA estimates that by 2026, data centers could use up to 1,050 TWh of electricity, which is equivalent to the total electricity demand of a country like Sweden or Germany nowadays [IEA, 2024]. Some projections suggest they could consume up to 20% or more of the world's electricity by 2030 if current trends continue without drastic optimization [4]. The main drivers of this explosion are following.

The AI Engine : Training AI models is particularly energy-intensive. The complexity of models (measured by the number of parameters, which now reaches hundreds of billions or even trillions) and the size of the required datasets are constantly increasing. Studies have shown that the amount of computation needed to train state-of-the-art AI models doubled approximately every 3 to 4 months between 2012 and 2018, a growth rate much faster than that predicted by Moore's Law [5]. Training a single massive language model can thus consume several GWh (Gigawatt-hours) [Alberto, 2021].

The Cost of Cooling : A non-negligible portion of the energy consumed in a data center is not used for computation itself but to dissipate the heat generated by electronic components. This "overhead" is often measured by the PUE (Power Usage Effectiveness), which is the ratio of the total energy consumed by the data center to the energy consumed by the IT equipment itself. A PUE of 1.5 means that for every Watt consumed by the servers, an additional 0.5 Watts are used for cooling, lighting, power distribution losses, etc. Although the PUEs of modern data centers are improving (approaching 1.1 or 1.2 for the most efficient ones), cooling remains a major consumer of power.

The Data Management and Communication : The movement and storage of massive datasets used by AI and HPC also contribute significantly to energy consumption, both at the level of the storage systems themselves and the network equipment (routers, switches) required to transmit them.

This escalation of energy needs raises fundamental questions about the sustainability of the current trajectory.

1.2.2 Environmental Impacts : Beyond Megawatts

Electricity consumption is only one part of the environmental impact of digital technology. Other well-known impacts include :

4. https ://www.iea.org/news/ai-is-set-to-drive-surging-electricity-demand-from-data-centres-while-offering-the-potential-to-transform-how-the-energy-sector-works

5. https ://openai.com/index/ai-and-compute/

Carbon Footprint : The most discussed consequence is the emission of greenhouse gases (GHG). The carbon footprint depends crucially on the energy mix used to produce the consumed electricity. If the electricity comes predominantly from fossil fuels (coal, gas), the carbon impact will be high. Even though major cloud providers are increasingly committing to using renewable energy, transparency about the real-time origin of consumed electricity is often limited, and coverage is not yet 100% everywhere, all the time. For example, in 2020, the information and communication technologies sector accounted for approximately 1.8% to 2.8% of greenhouse gas (GHG) emission, a figure that surpassed even the emissions from the aviation sector [Freitag, 2021].

Resource and Water Consumption : The manufacturing of servers, accelerators, storage systems, and network equipment requires the extraction of numerous natural resources (*rare metals, rare earths, water, energy*). These extraction and manufacturing processes have their own environmental impacts (*pollution, habitat destruction, GHG emissions*). The often-short lifespan of equipment and the low rate of recycling or reuse of electronic components (the problem of electronic waste or *"e-waste"*) worsen this balance sheet. Evaporative cooling systems (cooling towers), commonly used in many data centers, can consume considerable amounts of water. In regions where water is a scarce resource, this poses a major problem of competition with other uses (agriculture, drinking water).

It is therefore crucial to adopt a broader view of environmental impact, considering the entire life cycle (*manufacturing, use, end-of-life*) and resource depletion, beyond just direct electricity consumption.

1.2.3 Economic Impacts : The Hidden Cost of Performance

The energy bill represents a growing and substantial portion of the Operating Expenses (OPEX) for data center operators, cloud providers, and institutions managing HPC platforms. As example, the HPC market is poised for significant growth, projected to reach USD 49.9 billion by 2027 with a compound annual growth rate (CAGR) of 6.7%, and USD 78.254 Billion by 2032 with 7.1% of CAGR [Aarti, 2023]. These costs can be broken down as follows :

— **Direct Costs :** The purchase of electricity is a major direct expense. In a context of energy price volatility, this represents a significant financial risk.

— **Indirect Costs :** Costs related to the power and cooling infrastructure (initial investment - CAPEX, and maintenance - OPEX) are also directly linked to energy consumption. The more a system consumes, the more the support infrastructure must be oversized and maintained.

— **Total Cost of Ownership (TCO) :** Energy efficiency directly influences the Total Cost of Ownership of an infrastructure over its lifespan. A machine that is cheaper to purchase but highly energy-intensive can turn out to be more expensive in the long run than a machine that is initially more expensive but more energy-efficient.

— **Barrier to Entry :** For research laboratories, universities, or SMEs, particularly in countries with limited resources, the high cost of energy can constitute a significant barrier

to acquiring and operating high-performance HPC systems or AI infrastructures, thereby limiting their capacity for innovation and competition.

Energy optimization is therefore not just an environmental issue, but also a powerful economic lever for controlling costs and improving the accessibility of advanced technologies.

1.2.4 Design Limits : Approaching Fundamental Barriers ?

Beyond environmental and economic aspects, physical and technological limits are beginning to curb the unrestrained growth of computing power and its density.

Power Density and Cooling : Modern processors (CPUs) and especially accelerators (GPUs) individually consume several hundred Watts. Concentrating dozens of these components in a single server, and dozens of servers in a single cabinet (rack), leads to extremely high power densities (several tens of kW per rack). Dissipating such a large amount of heat in a confined space becomes a major technical challenge, pushing the limits of traditional air-cooling systems and necessitating the use of more complex and expensive technologies like liquid cooling (direct-to-chip, immersion) [Pambudi, 2022 ; Schmidt, 2009 ; XENON, 2023 ; Meyer, 2013 ; Nonaka, 2020]. Supplying the necessary electrical power to these high-density racks and ensuring the stability of the electrical grid within the data center also presents a considerable engineering challenge.

End of Dennard Scaling and Slowdown of Moore's Law : For decades, the semiconductor industry benefited from Dennard Scaling, which posited that as transistor size was reduced, their density doubled, their speed increased, and their power consumption per transistor decreased, all at a constant voltage. However, this scaling ended around the mid-2000s due to issues with leakage currents and heat dissipation. Concurrently, the famous Moore's Law (doubling of transistor density every 18-24 months) is also showing signs of slowing down [Schaller, 1997]. **The consequence :** "automatic" energy efficiency gains at the transistor level are becoming harder to achieve. Improvements in energy efficiency must therefore increasingly come from innovations at the level of processor architecture (specialization, multi-core), system architecture (memory, interconnect), and, crucially, from software and algorithms.

These physical and technological limitations reinforce the need to seek optimizations at all levels of the computing stack, from hardware to application software.

1.2.5 Specifics of Energy Constraints at the Edge

Although the energy challenges are noteworthy in data centers, they are critical in the context of Edge Computing. Many Edge devices (*smartphones, connected objects, wireless sensors, drones*) operate on battery power. Available energy is therefore a scarce and precious resource. Energy efficiency is directly linked to the operational lifespan of the device between recharges or battery replacements. Every millijoule counts. These devices are often small, confined, and lack active cooling systems (fans). The amount of heat they can dissipate is limited. Excessive consumption can lead to overheating, degrading performance (throttling), reducing

the lifespan of components, or even causing failures or safety risks. AI inference is a key task for many Edge applications (local voice recognition, image analysis by a smart camera). It is therefore crucial to develop AI models and execution techniques that minimize energy consumption per inference, while respecting the latency constraints (response time) that are often critical in this context.

The shift of AI from the Cloud to the Edge therefore further accentuates the urgency of developing fine-grained energy optimization approaches tailored to these constrained platforms.

1.3 Motivation : Towards Green and Responsible HPC for AI

Faced with the described energy crisis and its multidimensional impacts, the objective of making HPC and AI systems more energy-efficient has become imperative. This endeavor is motivated by several interdependent factors that outline the necessity of a transition towards more responsible computing.

1.3.1 Economic Competitiveness and Democratized Accessibility

Beyond the environmental imperative, energy efficiency is a powerful economic lever. As previously mentioned, controlling the electricity bill can significantly reduce the operating costs of infrastructures, freeing up financial resources for other investments (*research, innovation, personnel*). Companies that succeed in offering more energy-efficient Cloud services or AI/HPC solutions can gain a competitive advantage in terms of price, brand image ("green" marketing [6]), and resilience to energy cost fluctuations. Making HPC and AI technologies less expensive to operate promotes their adoption by a wider range of actors, including SMEs, research labs with constrained budgets, and developing countries. This helps to spread the benefits of these technologies more broadly.

1.3.2 Sustainability Requirement and Regulatory Expectations

The first and arguably most fundamental motivation is ethical and environmental. Technological development cannot be sustained at the expense of the planet. Reducing the energy and environmental footprint of the digital sector is essential to contribute to international goals for combating climate change (such as the Paris Agreement) and to the United Nations' Sustainable Development Goals (SDGs), particularly those related to clean energy (SDG 7), responsible consumption and production (SDG 12), and climate action (SDG 13). The Climate Neutral Data Centre Pact [CNDC, 2021] initiative aimed at making data centers in Europe climate neutral by 2030. Digital stakeholders (*researchers, engineers, companies*) have a growing responsibility to design and deploy technologies that are not only high-performing but also respectful of the environment and resources. The "four Pillar Framework for energy-efficient HPC data centers" [Wilde, 2013] outlines four critical areas for optimization : *applications, system software, system hardware*, and *infrastructures*. It is about avoiding the paradox where tools designed to solve

6. https ://www.investopedia.com/terms/g/green-marketing.asp

complex problems (climate, health) themselves contribute to aggravating others [Lannelongue, 2023]. An **eco-responsible** approach also involves minimizing the use of scarce resources (water, metals) and promoting the circular economy (*repair, reuse, recycle*) for computer hardware.

The pressure for greener computing comes not only from physical or economic limits but also from society and regulatory bodies. The public, consumers, and employees are increasingly **aware** of the environmental impact of the digital sector and expect companies and institutions to adopt more responsible practices. Governments are implementing **regulations** to control the energy consumption of digital products and services. The European Union, for example, has developed **ecodesign** regulations for servers and data storage systems, imposing minimum requirements for energy efficiency and transparency. Carbon taxes or emissions trading schemes could also affect the cost of energy for large consumers. **Standards** (e.g., ISO 50001 on energy management) and **environmental labels** (e.g., EPEAT, Energy Star for equipment) are developing and can influence purchasing decisions or become criteria in public tenders. Moreover, initiatives like the Green Software Foundation [Hussain, 2021] underscore the importance of prioritizing sustainability in software development.

Integrating energy efficiency thus also becomes a strategy for anticipating or complying with these growing expectations and regulations.

1.3.3 A Catalyst for Technological Innovations

Finally, the pursuit of energy efficiency is in itself a powerful driver of innovation. The energy constraint pushes researchers and engineers to fundamentally rethink how computations are performed.

This concerns the development of **low-power processors** (e.g., ARM architectures), specialized accelerators (FPGAs, ASICs for AI) optimized for energy efficiency, new, less energy-intensive memory technologies (HBM, NVM), and advanced cooling systems. We also note **compilation techniques** that optimize code to reduce consumption, improvements in operating systems and hypervisors for better power management, development of energy-efficient parallel computing libraries. The design of inherently more **frugal AI algorithms** (e.g., *"TinyML", quantization, neural network pruning*), development of HPC algorithms that reduce data movement (which is energy-costly) or adapt dynamically to available resources is also a direction. Finally we mention new approaches for job scheduling in clusters, dynamic resource allocation strategies, intelligent power management at the data center scale (taking into account intermittent renewable energies).

The work presented in this thesis is fully aligned with this dynamic of innovation, specifically targeting optimizations at the system software and application execution level. The quest for energy efficiency is therefore not a hindrance to performance but a stimulus for designing systems that are globally smarter and more efficient.

1.4 Scientific Problem and Technological Barriers

Despite the strong motivation to improve the energy efficiency of HPC/Cloud/Edge systems for AI, implementing effective solutions faces major scientific and technical challenges. These "barriers" form the core of the problem addressed in this thesis, which, to reiterate, focuses on optimizations achievable at the system and software levels, without altering the fundamental AI algorithms (such as the structure of a neural network).

1.4.1 Barrier 1 : Measurement and Metrology — Visualizing Energy Consumption

The adage, *"You can not improve what you can not measure"*, applies perfectly to energy optimization. However, obtaining a precise, reliable, low-cost, and sufficiently detailed measurement of the energy consumption of complex computer systems remains a significant challenge.

Hardware mechanisms like Intel RAPL (for CPUs and sometimes the global package including on-package memory) or NVIDIA NVML (for GPUs) are valuable but insufficient. They often do not directly measure the consumption of essential components such as the main *DRAM memory (off-package), storage peripherals (SSD/HDD), or network interfaces*. Yet, these components can account for a significant portion of the total consumption, especially for *"data-intensive"* applications like AI. The **sampling frequency** (often in the order of milliseconds or tens of milliseconds) may be too low to capture very brief or rapid energy events. The **spatial granularity** (*entire CPU package vs. individual cores, entire GPU vs. specific units*) may also be insufficient for fine-grained analysis. The **accuracy** of these sensors is not always guaranteed or documented and can vary from one hardware generation to another. They often rely on internal models rather than a direct physical measurement of current and voltage.

Software tools that collect and aggregate data from these sensors (or use external wattmeters) must overcome other difficulties : *Synchronization, Overhead, Portability and Usability, and Lack of Standardization*. Precisely **correlating the measured** energy consumption with software activity (which function, which line of code was executing at that exact moment ?) is complex, especially in parallel and multitasking systems. The act of measuring itself consumes resources (CPU, memory) and can therefore perturb the behavior of the measured application, introducing a bias in the results. Minimizing this **overhead** is crucial. Developing a tool that works across **different hardware platforms and operating systems**, and is easy to deploy and use by developers or researchers, is a software engineering challenge. The absence of **standardized** and widely accepted methodologies and tools makes it difficult to compare results between different studies or to reproduce experiments.

The scientific challenge here is to design and validate measurement methodologies and tools that offer a complete, accurate, low-overhead, high-resolution (temporal and spatial), and software-synchronized view of the energy consumption for all relevant components of an HPC/Cloud/Edge system. Our contribution, through a tool named **EA2P** aims to provide a

partial answer to this need.

1.4.2 Barrier 2 : Characterization and Modeling — Understand to Predict

Having measurements is not enough ; one must be able to interpret this data to understand the mechanisms of consumption and build predictive models that can be used for optimization.

1. **Complexity of AI Workloads :** AI applications, particularly the training of deep networks, exhibit very complex and dynamic execution profiles. They alternate between compute-intensive phases (e.g., matrix products on GPUs), memory bandwidth-limited phases (e.g., loading weights, activations), and, in distributed settings, phases dominated by network communication. Finely characterizing the energy signature of each of these phases and understanding how it varies with hyperparameters (batch size, model architecture) and the hardware platform is a considerable but necessary task.
2. **Modeling of Uninstrumented Components :** How can one estimate the consumption of a key component like the main DRAM if it is not directly measured by RAPL ? It requires developing indirect models, for example, based on memory activity observed via Hardware Performance Counters (HPCs). These counters (e.g., cache accesses, memory reads/writes) can serve as a proxy for energy activity, but the relationship is not always direct or easy to model.
3. **The Challenge of Analytical vs. Empirical Models :**
   — *Empirical Models (Machine Learning) :* These models (e.g., multiple linear regressions, neural networks) learn a prediction function from numerous experimental measurements. They can achieve good accuracy for configurations seen during training but require a large amount of data, can be time-consuming to train, often lack explainability (*black box*), and may generalize poorly to new architectures or Workloads.
   — *Analytical Models :* These models attempt to describe energy consumption through equations based on a physical or architectural understanding of the system (e.g., $energy = power \times time$, $power = static\_power + dynamic\_power$, dynamic power as a function of frequency, activity, etc.). They are generally faster to evaluate, more generalizable, and more interpretable. However, their development is much more complex, requiring deep knowledge of the hardware and software architecture. Finding the right parameters and analytical relationships to model the energy of an AI training phase or DRAM is a real scientific challenge.

The barrier here is to develop systematic Characterization methods and predictive models (ideally analytical or hybrid) that are simultaneously accurate, fast, explainable, generalizable, and usable for estimating the energy of key components (like DRAM) and complex application phases (AI training/inference). Our contributions in this thesis on modeling aim to address this challenge.

1.4.3 Barrier 3 : Optimization Strategies and Trade-off Management — Acting Effectively

The ultimate goal is to use the information from measurement and modeling to implement effective energy optimization strategies. This raises several sub-barriers related to exploiting system levers and managing the inevitable trade-offs.

1.4.3.1 Fine-Grained Exploitation of Operating System Levers

Modern systems offer various mechanisms to dynamically manage consumption (*DVFS, Power Capping, sleep states, CPU Governors...*). However, their optimal use is far from trivial. These mechanisms interact with each other and with the application in complex ways. For instance, reducing frequency (DVFS) can increase execution time and potentially hinder the expected energy savings, especially if the application is limited memory or I/O bounded. Applying a too-strict Power Cap can severely degrade performance. An application's needs vary greatly during its execution (Compute-bound vs. Memory-bound vs. I/O-bound phases). A static energy policy (e.g., setting a constant low frequency) is rarely optimal. Dynamic strategies capable of adapting in real-time are required. Developing lightweight and effective mechanisms to automatically detect the different phases of a running program (e.g., using hardware performance counters) and apply the most suitable energy policy (the right frequency, the right power cap) to each phase is a difficult open problem. What is the *"right"* setting for a given phase ? How to avoid unstable oscillations ? The barrier is to design intelligent, low-overhead runtime controllers for adaptive energy management.

1.4.3.2 Energy-Aware Scheduling

In shared resources environments (HPC clusters, Cloud), the Scheduler, which decides which job runs where and when, plays a crucial role in overall performance and consumption. Making the scheduler *"energy-aware"* is a major but complex objective.

— *Immense Decision Space :* The scheduler must make complex decisions : to which node (with which hardware characteristics and load level) should a new job be assigned ? In what order should waiting jobs be executed ? What energy parameters (frequency, power cap) should be applied to each job or node ? The space of possibilities is vast.

— *Multiple and Conflicting Objectives :* The scheduler must often reconcile conflicting objectives : minimizing total energy, minimizing job wait or execution time, meeting deadlines or Quality of Service (QoS) levels, ensuring fairness among users.

— *Dynamism and Uncertainty :* Job arrivals are often unpredictable, and their execution time or consumption profile may not be known in advance. The scheduler must make online decisions with incomplete or uncertain information.

The barrier is to develop scheduling algorithms (or other intelligent approaches) that are effective, robust, scalable, and practically deployable for AI workloads with en emphaisis on energy/power consumption. The contribution of this thesis on energy-aware scheduling explores this avenue.

1.4.3.3 The Unavoidable Management of Trade-offs

Any energy optimization approach end-up dealing with *trade-offs*. Reducing energy often implies an increase in execution time (latency) or a decrease in Throughput. In the case of AI, some energy optimization techniques could even affect model accuracy (although this is not the main focus here). The barrier is not to eliminate these trade-offs (which is often impossible), but to understand them, quantify them, and provide users or management systems with the means to consciously navigate this energy-performance(-accuracy) space to choose the operating point best suited to their specific needs.

1.4.3.4 Scalability and Portability — Generalizing Solutions

Finally, a last set of challenges concerns the ability of the developed solutions to operate effectively on large-scale systems and across different generations or types of hardware. The phenomena that dominate consumption and performance on a single node can change radically in a **large-scale system** (thousands of nodes). Inter-node communications (via the interconnect network) and accesses to parallel file systems often become the main bottlenecks and sources of energy consumption. Modern infrastructures are often **heterogeneous**, combining different types of CPUs, GPUs, accelerators, and hardware generations. An optimization solution developed for a specific platform may not be directly applicable or performant on another. Ensuring the portability of measurement tools, models, and optimization strategies is a constant challenge. Hardware and software evolve at a **very fast pace**. Optimization models and strategies must be designed to be adaptable or easy to update to remain relevant in the face of new architectures and programming paradigms.

This thesis does not claim to solve all mentioned barriers exhaustively, but it aims to provide specific and methodological contributions to improve the understanding of power consumption and enhance energy optimization of AI applications at all levels.

1.5 Our Main Contributions

The work carried out during this thesis has led to the following original contributions on the energy efficiency of HPC/Cloud systems for AI, focusing on system and software aspects :

1. **Contribution 1 :** A multi-platform and multi-component energy profiling software tool (EA2P). This tool provides a fine-grained, synchronized, and low-overhead measurement of the power and energy consumed by CPUs, GPUs, and (via estimation) DRAM at the process level.
2. **Contribution 2 :** Reference energy benchmarks. We provide a detailed quantitative analysis of the energy consumption of standard parallel compute kernels (from HPC/AI libraries) and the effectiveness of system power management mechanisms (DVFS, Power Cap) on these kernels, offering a solid foundation for understanding and modeling.

3. **Contribution 3 :** An analytical model for DRAM energy. This model allows estimating the consumption of random-access memory based on hardware performance counters and theoretical information from the manufacturer, which is particularly useful for systems without RAPL sensors for DRAM.
4. **Contribution 4 :** Predictive analytical models for AI energy. We propose models to estimate the energy consumption of training and inference phases, usable for a priori or online optimizations.
5. **Contribution 5 :** An a priori energy-aware scheduling framework. This approach aims to dynamically optimize the allocation and energy configuration of resources in a cluster running AI tasks, evaluated through simulation.
6. **Contribution 6 :** A methodology for energy efficiency. We propose a multi-step methodological framework, integrating the developed tools and strategies, to guide the analysis and energy optimization of AI applications.

1.6 Organization of the Manuscript

This manuscript is organized so as to present the context and problematic of the thesis, the main technical challenges, our contributions, and the concluding remarks/recommendations.

— Chapter 2 establishes a thorough state-of-the-art of existing work on energy measurement, modeling, and optimization in HPC, Cloud, and Edge contexts, highlighting the specifics related to AI applications and clearly positioning our contributions relative to prior research.

— Chapter 3 focuses on energy measurement and Characterization. It presents in detail the design, implementation, and validation of our measurement tool, EA2P. It then discusses the results of Benchmarking campaigns aimed at characterizing the energy of compute kernels and the impact of system power management levers.

— Chapter 4 addresses the problem of analytical energy modeling. It describes the development and validation of our model for estimating the energy consumed by DRAM, as well as the models designed to predict the consumption of the training and inference phases of AI applications.

— Chapter 5 presents our contributions related to energy optimization at the system and application levels. It details the AI job scheduling algorithms.

— Chapter 6 offers a synthesis of our strategies in the form of a structured methodology for energy efficiency. This chapter aims to provide a practical guide, integrating the developed tools (EA2P) and strategies (models, scheduling, runtime adaptation), along with general recommendations.

— Finally, Chapter 7 provides a general conclusion, summarizing the key contributions of the thesis, discussing their limitations, and opening perspectives for future research in this dynamic and crucial field.

— Additional technical and contextual information, particularly on hardware evolution, the physical principles of consumption, and the limits of measurement, are provided in the Appendix.

Chapter 2

Power Consumption Ecosystem : Measurement, Modeling, and Energy Optimization for AI on Distributed Infrastructures

Ce chapitre propose un état de l'art approfondi et critique de la littérature scientifique relative à l'efficacité énergétique dans les écosystèmes HPC et IA. L'analyse est structurée pour suivre le cycle de vie d'une démarche d'optimisation. Nous examinons en premier lieu les outils et méthodologies de mesure, en soulignant les limitations des approches existantes, notamment la difficulté de quantifier la consommation de la mémoire DRAM. Nous explorons ensuite les architectures matérielles modernes, des CPU aux GPU, et leurs mécanismes de gestion d'énergie natifs. Une section substantielle est dédiée à la caractérisation du profil énergétique des workloads d'IA, en différenciant leurs spécificités (densité des calculs, intensité mémoire) de celles des applications HPC traditionnelles. Enfin, le chapitre passe en revue les stratégies de modélisation et d'optimisation, mettant en exergue la prédominance des modèles "boîte noire" peu interprétables et des ordonnanceurs mono-objectifs. Cette revue critique conclut en identifiant des lacunes claires, justifiant ainsi le besoin pour les contributions originales de cette thèse : un outil de mesure flexible, des modèles analytiques interprétables et des stratégies d'ordonnancement nativement multi-objectifs.

2.1 Measurement of Computing Energy Consumption

2.1.1 Introduction

The first indispensable step towards the energy optimization of computer systems, particularly those running HPC and AI workloads, lies in the ability to precisely measure and quantify their consumption and performance. Without reliable metrology and a set of relevant metrics,

any optimization attempt remains blind or misdirected. This state-of-the-art section explores the different metrics used to characterize performance and energy consumption at various scales (from the individual component to the complete data center) as well as the fundamental measurement approaches. We will highlight key concepts, standard and emerging metrics, commonly used tools, and the challenges inherent in obtaining and interpreting actionable energy and performance data, emphasizing the crucial importance of managing trade-offs.

2.1.2 Global Metrics : Performance, Efficiency, and Infrastructure Impact

At the scale of large infrastructures (supercomputers, data centers), where energy consumption represents considerable economic and environmental stakes, several metrics are used to evaluate raw performance, the efficiency of converting energy into computation, and the overall impact.

2.1.2.1 Computational Performance (FLOPS)

The historical and fundamental performance metric in HPC is FLOPS (Floating-point Operations Per Second). It quantifies the raw ability of a system to perform calculations on real numbers (floating-point). We typically distinguish :

— *Theoretical Peak Performance (Peak FLOPS) :* Calculated by summing the maximum theoretical capabilities of all computing cores and accelerators in the system, assuming ideal utilization.

— *Sustained Performance (Sustained FLOPS or Max FLOPS) :* Measured on a real application or benchmark (LINPACK benchmark [Dongarra, 2003 ; Top500, 2025] used for the Top500 ranking). This performance is generally much lower than the theoretical peak performance as it reflects the real limitations of the system (*memory bandwidth, network, code efficiency*, etc.). FLOPS measures computational power but says nothing about the energy required to achieve it.

2.1.2.2 Computational Energy Efficiency (FLOPS/Watt)

To link performance and energy, the most widespread efficiency metric in HPC is FLOPS per Watt. Popularized by the Green500 list [Green500, 2025], it divides the performance (usually the sustained FLOPS from the HPL benchmark) by the electrical power consumed by the IT components during the benchmark's execution. The objective is to maximize this ratio to obtain the most *"useful computation"* per unit of energy consumed. A system optimized for FLOPS/Watt is not necessarily the one that delivers the highest absolute peak FLOPS. It represents a compromise between raw speed and energy sobriety. The energy efficiency improvement over years could be found on Figure 2.1 with a two years doubling tendency.

2.1.2.3 Application Performance (Throughput and Latency)

While FLOPS is a measure of raw computational capacity, the performance perceived by the user often depends on other, more application-oriented metrics :

— *Throughput :* Quantifies the amount of work completed per unit of time. For AI, this could be the number of images processed per second in inference, the number of training epochs completed per hour, or the number of user jobs processed per day in an HPC cluster. High throughput is crucial for batch processing.

— *Latency :* Measures the time required to complete a specific task. For AI, this could be the time to perform a single inference (critical for real-time or interactive applications), or the total execution time of a training run. Low latency is essential when response time is important.

Optimizing energy can impact throughput or latency, and vice versa. Measuring these metrics is indispensable for evaluating trade-offs.

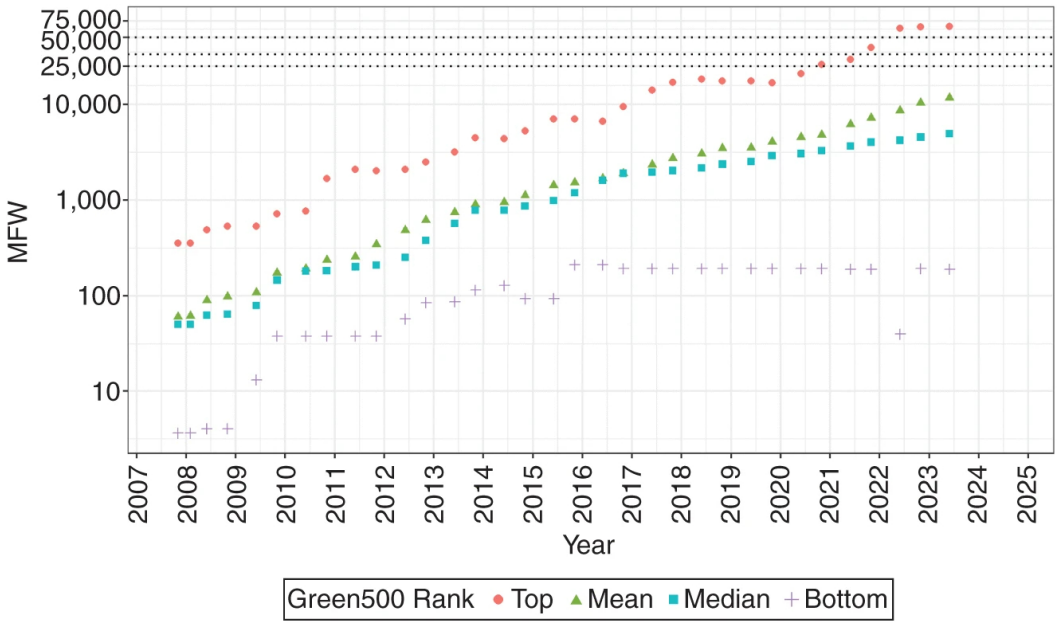


FIGURE 2.1 – Green500 energy efficiency trends over years [Adhinarayanan, 2025]

2.1.2.4 Infrastructure-related Energy Metrics (PUE and DCiE)

To evaluate the efficiency of the entire data center, including non-IT systems (cooling, power distribution, lighting), the PUE (Power Usage Effectiveness) [Belady, 2008] is the de facto standard. Calculated as Total Facility Energy / IT Equipment Energy (Eq. 2.1), an ideal PUE is 1.0 (no overhead outside of IT), while modern data centers aim for values below 1.5 or even close to 1.1 [Daniel, 2021]. Its inverse is the DCiE (Data Center Infrastructure Efficiency).

$$PUE = \frac{Total\_Facility\_Energy}{IT\_Energy} = 1 + \frac{Non\_IT\_Energy}{IT\_Energy} \tag{2.1}$$

Limitations : PUE measures the efficiency of the data center's "shell" but not that of the servers themselves. A data center can have an excellent PUE while hosting very energy-inefficient IT servers. It also does not capture dynamic variations related to load or the efficiency of IT resource utilization. There are calls for complementary metrics [Jacqueline, 2022]. Jin et al. [Jin, 2016] provide a comprehensive overview of metrics and techniques for *"green"* data centers.

2.1.2.5 Other Derived Metrics

Energy Cost : The direct financial impact, calculated by multiplying the energy consumed (kWh) by the unit cost of electricity ($/kWh), remains a major concern for operators [GlobalPetrolPricecom, 2025]. This cost depends on geographical and temporal factors and influences investment and operational decisions as we can see on Table 2.1 for top supercomputers from the June 2025 TOP500 list.

Table 2.1 – Electricity cost per hour for the top ten supercomputers (June 2025).

Machine	Peak Perf.	Power(MW)	$/KWh	Total(K$)
El Capitan	1.742 EFLOPS	29.581	0.148	4.378
Frontier	1.353 EFLOPS	24.607	0.148	3.642
Aurora	1.012 EFLOPS	38.698	0.148	5.727
JUPITER Booster	793.4 PFLOPS	13.088	0.286	3.743
Eagle	561.2 PFLOPS	8.468*	0.148	1.253
HPC6	477.9 PFLOPS	8.461	0.442	3.740
Fugaku	442.0 PFLOPS	29.899	0.207	6.189
Alps	434.9 PFLOPS	7.124	0.284	2.023
LUMI	379.7 PFLOPS	7.107	0.127	0.903
Leonardo	241.2 PFLOPS	7.494	0.442	3.312

Carbon Footprint (CO2e) : Translating energy consumption into environmental impact, the calculation of the *CO2 equivalent (CO2e)* [Patterson, 2022] incorporates the *carbon intensity (gCO2e/kWh)* of the electricity mix used (Eq. 2.2). This intensity varies greatly depending on the geographical location and the time of day [ourworldindataorg, 2025], making the choice of location and execution time for computations crucial for minimizing the footprint (cf. Table 2.2). A distinction is often made between :

— *Operational Carbon :* Emissions due to electricity consumption during use.

— *Embodied Carbon :* Emissions due to the manufacturing and transportation of hardware, and its end-of-life. Considering the entire life cycle is essential for a complete assessment.

$$CO_2e = Wh \times (CO_2e \ per \ Wh) \tag{2.2}$$

Sources and Notes : TOP500 performance data from June 2025 rankings [Top500, 2025]. Business electricity prices from GlobalPetrolPrices.com[GlobalPetrolPricecom, 2025]. Carbon intensity values from OurWorldInData.org (2024)[ourworldindataorg, 2025]. Power consumption values from TOP500 official submissions. *Eagle power consumption estimated based on system configuration. Notable contrast : Switzerland (Alps) and Finland (LUMI) show lower carbon footprints due to clean energy grids.

2.1.3 Hardware Absolute and Relative Metrics

To refine the analysis and guide optimization at the hardware and software level, other metrics focus on individual components, servers, and especially the explicit quantification of the energy-performance trade-off.

TABLE 2.2 – CO_2 per hour for the top ten supercomputers (2024/2025 data).

Machine	**Peak Perf.**	**Power**	**gCO_2/KWh**	**Kg(CO_2)**
EL CAPITAN	1.742 EFLOPS	29.581MW	384	11,359
FRONTIER	1.353 EFLOPS	24.607MW	384	9,449
AURORA	1.012 EFLOPS	38.698MW	384	14,860
JUPITER BOOSTER	793.4 PFLOPS	13.088MW	344	4,502
EAGLE	561.2 PFLOPS	8.468MW	384	3,252
HPC6	477.9 PFLOPS	8.461MW	288	2,437
FUGAKU	442.0 PFLOPS	29.899MW	382	11,422
ALPS	434.9 PFLOPS	7.124MW	37	264
LUMI	379.7 PFLOPS	7.107MW	72	512
LEONARDO	241.2 PFLOPS	7.494MW	288	2,158

2.1.3.1 Thermal Design Power (TDP)

A key thermal specification provided by CPU/GPU manufacturers, TDP indicates the maximum heat (in Watts) that the cooling system must be able to continuously dissipate under a "typical" workload defined by the manufacturer. It does not represent the absolute maximum power consumption (which can be exceeded, cf. "turbo" modes) [Hennessy, 2012], but serves as an essential reference for thermal design and chip binning. **Average CPU Power (ACP)** is a historical, less standardized metric from AMD [Scott, 2011 ; Arne, 2022], tied to a specific workload.

2.1.3.2 SWaP (Space, Wattage and Performance)

This metric [David, 2013] evaluates the energy-efficient performance density of a server by integrating application performance, power consumption (Wattage), and physical footprint (Space, often in 'U' -rack units-) into a single formula (Eq. 2.3). Useful for comparing the volumetric efficiency of different server solutions.

$$SWaP = \frac{Performance}{Space \times Power} \tag{2.3}$$

where *Performance and Power* are measured by any relevant benchmarks, and *Space* is related to the size of the computer infrastructure in meter square.

2.1.3.3 Energy-Performance Trade-off Metrics

To simultaneously evaluate the energy consumed and the performance (typically execution time or latency), several combined metrics have been proposed. They are crucial for optimization because they aim to find a balanced *"operating point"* rather than blindly minimizing energy (which could make the system unusably slow) or maximizing performance (which can be very energy-costly).

— *EDP (Energy-Delay Product) :* Defined as $Energy \times Time$ [Gonzalez, 1996]. Minimizing EDP seeks to avoid extremes : neither a very fast but extremely power-hungry system, nor

a very economical but too slow system. It favors a balance between the two.

— *ED^2P (Energy-Delay Squared Product) :* Defined as $Energy \times Time^2$. By squaring the time, this metric penalizes an increase in execution duration more heavily than a proportional increase in energy consumption. It is relevant when performance (low latency) is a stronger constraint than energy, while still seeking a compromise. Other variants (E^2DP, etc.) exist depending on the priority given to each dimension.

Usage : These metrics are often used as the objective function in dynamic optimization algorithms (e.g., choosing the CPU/GPU frequency that minimizes the EDP for a given computation phase ; objective for scheduling policies).

2.1.3.4 Performance Indicators as Proxies for Power Consumption

Although they do not directly measure energy, certain performance metrics from Hardware Performance Counters (HPCs) are extremely useful indirect indicators (proxies) for understanding consumption variations and for feeding energy models :

— **IPC (Instructions Per Cycle) :** Reflects the efficiency of compute core utilization. A high IPC suggests a compute-bound phase, often correlated with high CPU/GPU activity and consumption. A low IPC may indicate waiting for memory or I/O.

— **Cache Miss Rate :** A high rate indicates numerous accesses to main memory (DRAM), which are often costly in time and energy compared to cache accesses.

— **Memory Bandwidth Utilized :** A direct measure of the intensity of memory usage, strongly correlated with the consumption of DRAM and memory controllers.

— **Network/Disk Activity :** Number of packets sent/received, number of bytes read/written to the disk.

These counters, often accessible via interfaces like PAPI [McCraw, 2014], are essential for characterizing application behavior (compute-bound vs. memory-bound vs. I/O-bound) and for building analytical or learning-based energy models, as we will see later (Chapter 4).

2.1.4 Fundamental Measurement Approaches

To acquire the raw energy data that feed the metrics discussed previously, two main families of technical approaches are primarily used, each with its intrinsic strengths and weaknesses. The Figure 2.2 illustrates the general workflow of both scenarios.

2.1.4.1 Out-of-Band Measurement

This approach consists of using external measurement devices that are physically inserted into the power supply circuit of the target computer equipment.

Principle : Standard wattmeters can measure the overall consumption of a server or a rack at the power outlet. For finer granularity, more sophisticated systems exist, such as current and voltage probes placed on the specific power lines of certain components (e.g., CPU 12V

ATX supply, PCIe lines for GPUs) or dedicated hardware platforms (often based on microcontrollers or FPGAs) that acquire and aggregate data from multiple probes. Examples of such dedicated hardware platforms, often developed in an HPC research context, *include HDEEM (High Definition Energy Efficiency Monitoring)* [Hackenberg, 2014], *WattProf* [Rashti, 2015], and *DiG (Dwarf in a Giant)* [Libri, 2018]. These systems are designed for high-frequency sampling (kHz) and data collection that is minimally intrusive to the measured compute node. For embedded systems, modules like *PSoC 5LP (Programmable System on Chip)* [Janković, 2015] or *Dr. Wattson* [UpbeatLabs, 2023] play a similar role.

Advantages :

1. *Potential Accuracy :* A direct physical measurement of current and voltage can offer high accuracy if the sensors and acquisition system are of good quality.
2. *Non-Intrusiveness :* The execution of software on the target system is generally not (or very little) affected by the measurement itself, thus avoiding performance overhead.
3. *Independence :* Does not depend on the internal sensors, drivers, or operating system of the measured system. Allows for measuring total consumption or that of specific subsystems if probes are placed appropriately.

Disadvantages :

1. *Cost and Complexity :* Requires additional hardware, which can be costly and complex to install, especially for fine-grained per-component measurement (requiring modification of internal wiring).
2. *Software Synchronization :* Precisely correlating external energy measurements with software events (which function, which phase of the program ?) is a major challenge, requiring rigorous timestamping and post-processing mechanisms.
3. *Limited Spatial Granularity :* The fineness of the measurement depends on where the probes can be physically placed. Measuring the individual consumption of each CPU core or the different internal domains of a SoC is often impossible with this approach.
4. *Variable Temporal Resolution :* While dedicated platforms (HDEEM, WattProf) aim for high frequencies, consumer-grade wattmeters may have much lower sampling rates (on the order of a second).

2.1.4.2 In-Band Measurement

This approach relies on power sensors or energy models integrated into the hardware components themselves, whose data are accessible via software through standardized or proprietary interfaces.

Principle : Recent processors (Intel, AMD), GPUs (NVIDIA, AMD), and some other components integrate temperature, voltage, or current sensors, or more often, internal models that estimate energy consumption based on activity and operating parameters (frequency, voltage). This information is exposed to the operating system or applications via Model-Specific Registers (MSRs on x86) or programming interfaces (APIs). The most well-known examples are the RAPL

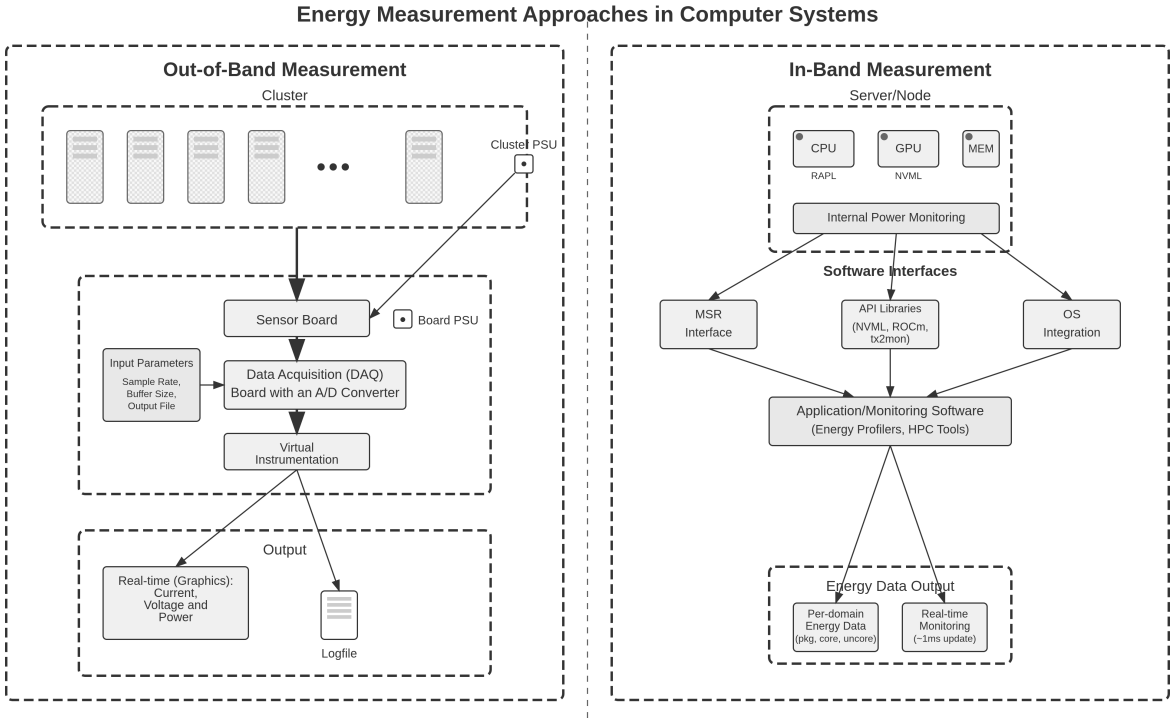


FIGURE 2.2 – Complementary approaches for comprehensive energy monitoring in HPC systems

(Running Average Power Limit) interface for Intel [Intel, 2022c] and AMD [Schöne, 2021] CPUs, and the NVML (NVIDIA Management Library) [NVIDIA, 2016] and ROCm SMI (ROCm System Management Interface) [AMD, 2014] libraries for NVIDIA and AMD GPUs, respectively. Other platforms like Marvell's ThunderX2 [Marvell, 2019a] have their own sensors and access tools (tx2mon [Marvell, 2019b]).

Advantages :

1. *Low Cost and Ease of Access :* Uses existing hardware, no extra cost or complex installation. Data access is purely software-based.
2. *Fine Spatial Granularity :* Can provide consumption information for different processor domains (e.g., *pkg* for the entire package, core for the compute cores, uncore for the *memory/L3 cache controllers, dram* or *psys* on certain Intel or AMD generations/platforms) or per core (AMD RAPL).
3. *High Temporal Resolution :* RAPL, for example, typically updates its energy counters approximately every 1 ms, allowing the capture of relatively fast variations.
4. *Easier Synchronization :* Direct access from software allows for easier correlation between measurement and the executing code.

Disadvantages :

1. *Incomplete Coverage :* This is a major limitation. Interfaces like RAPL often ignore a significant part of the server's total consumption [Orgerie, 2021], notably the main *DRAM memory* (not integrated into the CPU package), *network cards, storage drives, fans,* etc.

Even the RAPL measurement for DRAM, when available, is limited to certain types of platforms (notably Intel servers).

2. *Questionable Accuracy and Calibration :* The accuracy of the sensors and especially the internal models is not always documented or guaranteed by manufacturers. Studies have shown discrepancies or biases [NVIDIA, 2016]. These measurements may be more useful for observing relative trends than for obtaining exact absolute values without calibration. The influence of certain factors (like instruction type or operands) on the measurement can vary [NVIDIA, 2016].
3. *Measurement Overhead :* Frequently reading these registers (polling) consumes CPU cycles and can introduce a slight overhead, although it is generally low.
4. *Dependency :* Requires software support (specific drivers in the Linux kernel, vendor libraries) and sometimes administrative privileges. Portability between different architectures or vendors may be limited.

2.1.4.3 Hybrid Approaches and Activity-Based Estimations

Faced with the limitations of the two pure approaches, hybrid methods or those based on indirect models are often necessary.

In-Band Calibration via Out-of-Band : Using precise out-of-band measurements (wattmeter) to calibrate or validate the readings of in-band sensors (RAPL, NVML) for a given platform.

Proxy-Based Modeling : Using hardware performance counters (HPCs), which measure activity (*instructions executed, cache/memory accesses*, etc.), as input variables for models (analytical or ML) that estimate the energy consumption of measured or unmeasured components (e.g., estimating DRAM energy from the memory bandwidth measured by HPCs).

TDP or Utilization-Based Estimation : Some tools (e.g., Eco2AI [Budennyy, 2023]) provide a very coarse estimation of energy based on the utilization rate (e.g., CPU%) and the component's nominal power (TDP). This approach is generally very inaccurate as it does not account for the non-linearity of consumption or the actual nature of the workload.

The choice of approach heavily depends on the required level of precision, the desired granularity (component vs. system), the available budget, and the acceptable intrusiveness. For a fine-grained analysis aimed at software optimization, a combination of in-band measurements (for granularity and synchronization) and activity-based models (to fill coverage gaps, like DRAM) often appears to be a pragmatic solution.

2.1.5 The Ecosystem of Energy Measurement Tools and APIs

A wide range of software tools, libraries, and specifications has been developed to exploit the measurement approaches described above, facilitate access to energy data, and, in some cases, enable active management of consumption. A typical taxonomy of software tools for energy concerns could be found in figure 2.3

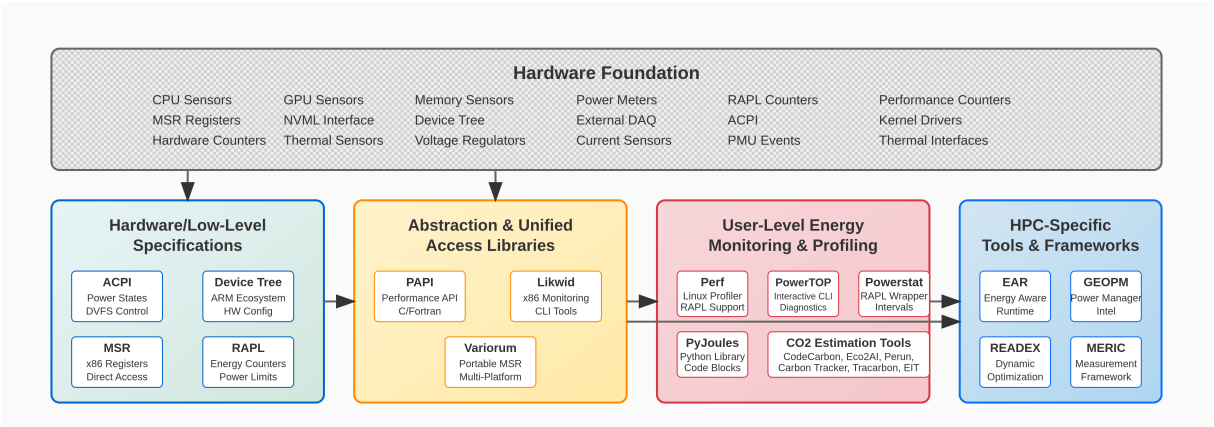


FIGURE 2.3 – Typical ecosystem of energy measurement in HPC systems

2.1.5.1 Hardware/Low-Level Specifications and Interfaces

They define how the hardware exposes information and how the OS can interact with it. Among them, **ACPI (Advanced Configuration and Power Interface)** [UEFI, 2021] is an industry standard (initially for x86) defining how the OS discovers components, manages their power states (C-states for CPU idle, D-states for peripherals), and controls their energy performance (P-states for CPU frequency/voltage levels). It is essential for OS-implemented DVFS. Similarly, **Device Tree (DT)**[Linaro, 2022] is an alternative to ACPI, mainly in the ARM ecosystem, describing the hardware configuration statically for the OS. Also allows passing power management-related information. In terms of access, **MSR (Model-Specific Registers)** are x86 architectures specific registers, used (among other things) to access performance counters and RAPL data. Direct access often requires high privileges. The **RAPL (Running Average Power Limit) concept**[Intel, 2022c ; Schöne, 2021] (an interface, not the tool itself) implemented by Intel and AMD to read estimated energy counters and sometimes to set power limits could be viewed as a specification. The implementation is accessible via MSRs or kernel drivers (`intel_rapl, amd_energy`).

2.1.5.2 Abstraction and Unified Access Libraries

These aim to provide a portable and higher-level interface for accessing performance and energy information.

PAPI (Performance API) [Browne, 1999 ; McCraw, 2014] is a widespread C/Fortran library, standardizing access to hardware performance counters (HPCs) on various architectures. It integrates a `'rapl'` component to read RAPL data portably. **Likwid**[Treibig, 2010a] is a suite of command-line tools and a library (C API) for performance monitoring on x86. It includes `'likwid-powermeter'` which uses RAPL for energy measurements. Another tool in this logic is **Variorum** [LLNS, 2023], a C library aiming to provide a portable abstraction over MSRs and other low-level vendor-specific interfaces to control power and frequency on different HPC platforms (*Intel, IBM Power, ARM, NVIDIA GPU*). Often used as a building block by higher-level tools like GEOPM (Global Extensible Open Power Manager) [Eastep, 2017] or PowerAPI

[Bourdon, 2013].

2.1.5.3 User-Level Energy Monitoring and Profiling Tools

These are ready-to-use tools for measuring the energy consumption of applications or the system operating. **Perf** [Firefox, 2023] is the most popular and the standard profiling tool of the Linux kernel. Can be used to read RAPL counters when running a command (`'perf stat -e power/energy-pkg/ ...'`). More focused on energy/power consumption, **PowerTOP**[Intel, 2020] is an interactive tool (CLI) on Linux for diagnosing consumption issues, estimating power per process/device, and modifying some power management settings. It uses RAPL and analyzes system activity. **Powerstat**[Colin, 2021] is a simple tool (CLI) for Linux, a wrapper around RAPL, similar to PowerTOP. It measures average consumption over intervals. **PyJoules** [INRIA-Lille, 2019] is a Python library specifically designed to measure the energy consumed by Python functions or code blocks. It uses RAPL and NVML/Nvidia-SMI for CPU (Intel/AMD), RAM (some Intel), and GPU (NVIDIA, some integrated Intel). It aims for ease of instrumentation for Python developers.

Other similar tools : perun [Gutiérrez, 2023], **CodeCarbon** [BCG, 2020], **Eco2AI** [Budennyy, 2023], **Carbon Tracker** [Anthony, 2020], **Tracarbon** [Valeye, 2022], **Experiment-impact-tracker(EIT)** [Henderson, 2020], **Green Algorithms** [Lannelongue, 2021] are often Python libraries focused on estimating CO2e from energy measurements (via RAPL/NVML) or estimations (using TDP), with mentioned limitations on coverage and granularity.

2.1.5.4 HPC-Specific Tools and Frameworks

These are often more complex tools, integrating measurement, analysis, and sometimes dynamic optimization for large-scale parallel applications. **EAR (Energy Aware Runtime)** [Lenovo, 2022] is a comprehensive framework for measurement, management, and optimization for HPC clusters (CPU/GPU). It includes a runtime library and system-level management (power capping, DVFS). **GEOPM (Global Extensible Open Power Manager)** [Eastep, 2017] is an open-source framework (Intel) for power management at the node and cluster scale, with a plugin architecture allowing the implementation of different strategies (e.g., power balancing to respect a global budget while maximizing performance). It uses Variorum internally. In same order of dynamic tools, **READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing)** [Oleynik, 2015] is a project and suite of tools for dynamic optimization. It combines an offline analysis phase (to identify code regions and their sensitivity to parameters like frequency) and a runtime library to apply the detected optimal configurations. Figure 2.4 provides an overview of READEX working diagram. Additionally, **MERIC (Measurement-based Energy RankIng and Optimization framework for HPC Clusters)** [Vysocky, 2017; Schuchart, 2017] is a C/C++ library (with a Fortran interface) inspired by READEX. Allows annotating code regions to measure energy and time, analyzes behavior under different configurations (DVFS/UFS), and can apply dynamic tuning. Also supports hardware tools (HDEEM/DiG). **lo2s** [Ilsche, 2017] is a lightweight monitoring tool

that collects performance traces (HPCs, kernel events) and energy traces (e.g., via HDEEM), and stores them in OTF2 format for offline analysis (e.g., with Vampir [Knüpfer, 2008]).

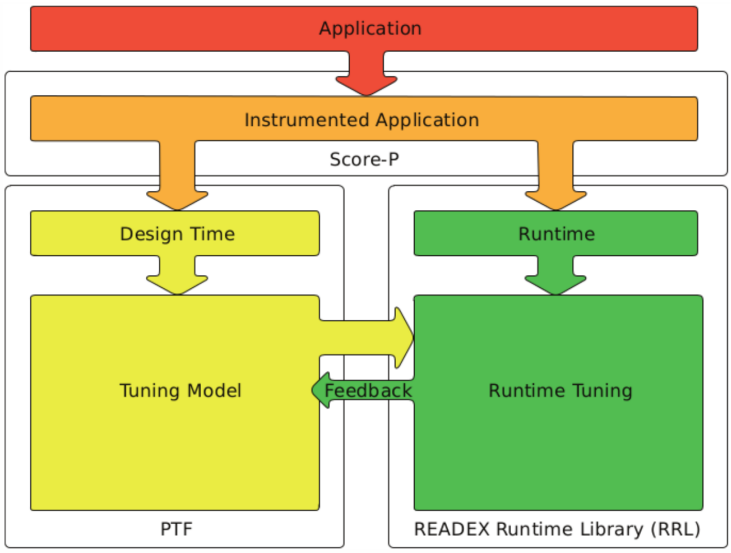


FIGURE 2.4 – READEX working diagram

2.1.6 Operational Limitations

The ecosystem of energy measurement and management tools is rich and diverse, ranging from low-level specifications to complete HPC frameworks. However, despite this abundance, several limitations persist and motivate the contributions of this thesis :

— **The DRAM "Black Hole" :** The direct and reliable measurement of main DRAM memory consumption remains a major challenge on many platforms where RAPL does not cover it, or covers it poorly. Existing tools often lack an integrated and validated solution to accurately estimate this crucial energy component, especially for memory-intensive AI applications.

— **Lack of Flexibility and Portability :** Many tools are specific to a vendor (Intel, NVIDIA), an architecture (x86), or an operating system (Linux). Developing, adapting, or extending them to new hardware platforms (ARM, future accelerators) or to new needs (finer-grained profiling) can be complex. Existing Python tools, while convenient, show limitations in terms of device coverage or granularity of RAPL measurements.

— **Complexity of Integration and Use :** Setting up a complete measurement chain, combining multiple tools (e.g., RAPL + NVML + DRAM model), synchronizing the data, and making it easily exploitable by developers (especially in Python, a very popular language in AI) remains a difficult task.

— **Accuracy and Validation :** The accuracy of in-band measurements remains questionable without external validation, and the overhead of some profiling tools is not always well characterized. The non-additive complexity of consumption makes per-process profiling difficult in multitasking environments.

It is in this context, to specifically address the lack of a flexible and portable tool in Python capable of aggregating multi-component measurements (CPU, GPU) and integrating a robust model for DRAM energy, that the EA2P tool was designed and developed as part of this thesis (cf. Chapter 3). It aims to provide a pragmatic and extensible solution for the fine-grained energy profiling of applications, especially those related to Artificial Intelligence.

2.1.7 Conclusion

A vast landscape of approaches, specifications, libraries, and tools exists for energy measurement and management. Out-of-band methods offer potential accuracy at the cost of complexity, while in-band methods provide granularity and ease of software access at the cost of incomplete coverage and sometimes uncertain accuracy. The tools range from low-level interfaces (RAPL, NVML) to sophisticated HPC frameworks (EAR, GEOPM). Nevertheless, key challenges remain regarding the complete coverage of components (especially DRAM), portability, flexibility, and ease of integration, particularly for modern development environments like Python. These gaps justify the need for new approaches and tools, such as those proposed in this work.

2.2 Target Infrastructures and Native Power Management

AI applications do not currently run in a vacuum. They are deployed on a variety of hardware infrastructures, ranging from powerful supercomputers to small embedded systems at the periphery, including vast Cloud data centers. Each of these infrastructures has its own architectural characteristics, key components, and integrated mechanisms for managing energy consumption. Understanding these target architectures and the native control levers they offer is fundamental before considering higher-level software or system optimization strategies. This section explores the main types of hardware components encountered (see Figure 2.6), their energy specifics, and the standard mechanisms (ACPI, DVFS) that allow for initial management of their consumption.

2.2.1 Key Hardware Architectures for HPC-AI

The performance and energy consumption of AI and HPC applications are intrinsically linked to the hardware components that execute the computations and manage the data. The simplified general view of a typical HPC systems is shows on Figure 2.5. Several families of processors and systems coexist, each with its strengths and weaknesses from an energy perspective.

2.2.1.1 Specialized Accelerators : Highly Parallel, but Power-Hungry Computing Units

Massively parallel workloads, typical of the training and inference of deep neural networks, benefit enormously from dedicated hardware architectures capable of executing a large number of simple operations simultaneously.

GPU (Graphics Processing Unit) : Initially designed for graphics rendering, GPUs have become the reference accelerators for AI and many HPC applications [Ikram, 2018]. Their

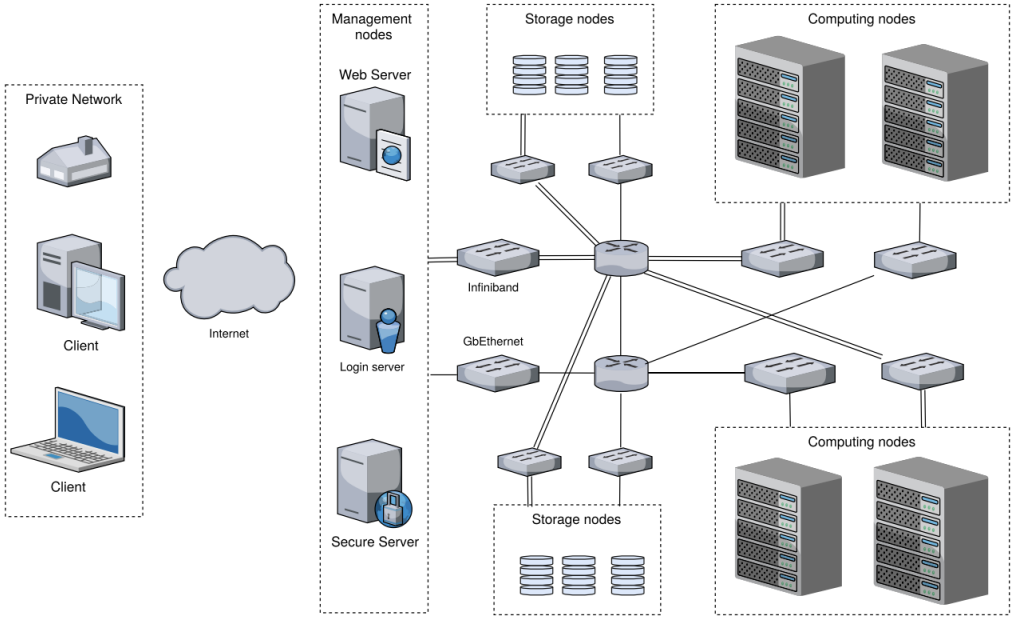


FIGURE 2.5 – Simplified overview of a typical HPC systems [Reghenzani, 2020].

massively parallel architecture (thousands of simple cores) makes them extremely effective for the matrix and tensor operations that dominate Deep Learning computations.

High-end models like the NVIDIA H100 [NVIDIA, 2022] or the AMD Instinct MI250X/MI300X [AMD, 2021] offer phenomenal peak performance (TFLOPS or even PFLOPS in reduced precision), but at the cost of considerable power consumption, with TDPs reaching 500W, 600W, or even 700-750W [NVIDIA, 2022]. Although their efficiency (FLOPS/Watt) is often superior to that of CPUs for parallel tasks, their absolute consumption is a major factor in the overall energy budget of a compute node. Projects like Intel Ponte Vecchio [Intel, 2022a] also target this market with high TDPs (e.g., 600W). Tools like NVIDIA-SMI and ROCm-SMI offer management capabilities (*power capping, specific operation modes, frequency control*) to try to control this consumption.

TPU (Tensor Processing Unit) : Accelerators developed specifically by Google for computations related to neural networks (matrix multiplications, convolutions). They often use a systolic array architecture to optimize data flow and maximize arithmetic efficiency.

TPUs are renowned for their high energy efficiency on target AI workloads. For example, the Edge TPU can achieve 4 TOPS (Tera-Operations Per Second) for only 2 Watts [GoogleLLC, 2020a ; Google, 2023], an efficiency of 2 TOPS/Watt, ideal for embedded applications. The data center versions (e.g., TPU v4 with an average TDP of 192W) are also designed for a good performance/watt ratio. Research such as that by Pandey et al. [Pandey, 2021] explores intelligent power gating techniques (UPTPU) to exploit the potential underutilization of compute units and further improve energy efficiency (factors of 3.5× to 6.5× reported). The efficiency of Edge TPUs has led to their integration into development boards for embedded systems (e.g., Coral Dev Board [GoogleLLC, 2020b]).

FPGA (Field-Programmable Gate Array) : Integrated circuits whose internal logic

can be reconfigured after manufacturing to implement a specific digital circuit. They offer the potential for highly advanced hardware optimization for a given application.

By implementing the algorithm directly in hardware, FPGAs can offer excellent energy efficiency for certain tasks, sometimes surpassing GPUs or CPUs on specific workloads, particularly those involving arbitrary-precision arithmetic or non-standard data flows. Their reconfigurability offers unique flexibility, but their programming (in hardware description languages like VHDL or Verilog, or via HLS - High-Level Synthesis tools) is significantly more complex than classic software programming. Initially confined to niches, FPGAs are increasingly finding their place in servers (e.g., SmartNIC accelerator cards, database or AI acceleration) [Boku, 2022] and embedded systems. Dynamic power management on FPGAs (e.g., via DVFS) is less standardized than on CPUs/GPUs. Works like Hosseinabady and Nunez-Yanez [Hosseinabady, 2018] explore hybrid FPGA-CPU strategies and the application of DVFS on FPGAs to reduce energy but note that efficiency depends on the nature and period of the task.

2.2.1.2 General-Purpose Processors

Despite the rise of accelerators, classical CPUs remain the central component of any computer system, running the operating system, managing resources, and performing a large share of the computations, especially sequential or irregular tasks.

x86 Architectures (Intel, AMD) : *x86* is a family of CISC *(Complex Instruction Set Computer)* instruction set architectures, initially developed by Intel from Intel 8086 microprocessor and its 8088 variant. It was introduced in 1978 as a fully 16-bit extension of Intel's 8-bit 8080 microprocessor, with memory segmentation as a solution for addressing more memory than can be covered by a plain 16-bit address. Embedded systems and general-purpose computers used x86 chips before the IBM Personal Computer in 1981. Nowadays, most desktop, workstation, laptop and server computers are based on the x86 architecture family, while mobile categories such as smartphones or tablets are dominated by ARM. The latest processor generations like the Intel Xeon Scalable (e.g., Sapphire Rapids, and its predecessor Icelake) or the AMD EPYC (e.g., Genoa/Bergamo based on Zen4, and their predecessors Zen3/Zen2) equip the majority of current HPC and Cloud systems [AMD, 2023], including the most powerful supercomputers [Top500, 2025].

These are very complex processors (CISC - Complex Instruction Set Computer), integrating many cores (up to 64, 96, or more), large caches, advanced vector instructions (AVX, AVX-512), and sophisticated memory controllers. Their nominal consumption (TDP) is also high, commonly reaching 200W to 400W per socket for high-end server models, contributing significantly to the node's overall consumption. Competition between Intel and AMD also drives improvements in energy efficiency[AMD, 2023]. They benefit from the most mature power management mechanisms (*ACPI, P-states, C-states, RAPL for monitoring/capping*) which are widely supported by OSes and tools (cf. next section).

ARM Architectures : Initially dominant in the mobile and embedded world for their low power consumption (RISC - Reduced Instruction Set Computer), ARM processors are increasingly penetrating the server and HPC market [Mantovani, 2020].

Their main argument is better energy efficiency (performance/watt) compared to x86 for many workloads, thanks to an often simpler architecture optimized for low power. Supercomputers like Fugaku (with A64FX processors developed by Fujitsu/RIKEN based on ARM) or systems based on Marvell ThunderX2 chips [Wikichiporg, 2020], or more recently NVIDIA's Grace CPUs (Grace Hopper Superchip), demonstrate the viability of ARM for high-performance computing. The software ecosystem (compilers, libraries) for HPC on ARM is less mature than on x86, although it is progressing rapidly. Power management relies on ACPI or Device Tree (DT), but specific monitoring/control tools may be less standardized than RAPL (e.g., requiring tx2mon for ThunderX2).

2.2.1.3 Embedded Systems and Microcontrollers : Constrained Sobriety

At the other end of the spectrum, embedded systems and microcontrollers are designed for specific applications with strong constraints on cost, size, and especially energy. They are optimized to consume very little, often on the order of Watts, milliwatts, or even microwatts in standby, allowing battery operation for weeks or months. This sobriety comes at the expense of computing power, memory capacity, and storage, which are very limited compared to servers or PCs.

Popular Platforms for Embedded Computing :

— *Arduino [Arduino, 2023] :* A very popular open-source platform for beginners, based on simple microcontrollers (AVR, ARM Cortex-M), very low power (< 1W) but limited computing power.

— *Raspberry Pi [RaspberryPi, 2020] :* A complete mini-computer based on a more powerful ARM SoC (e.g., Broadcom), capable of running a Linux OS. Offers a good performance/price/consumption compromise (10W max for RPi 4). Often serves as a prototyping platform or for more complex IoT applications.

— *Nvidia Jetson [Seeedstudio, 2020 ; Nvidia, 2023] :* A range of embedded boards (Nano, AGX Orin) combining ARM CPUs and integrated NVIDIA GPUs, designed specifically for AI at the edge (Edge AI). They offer significant AI computing power (up to 275 TOPS for Orin) for moderate consumption (5-10W for Nano, up to 60W for Orin).

— *Coral Dev Board [GoogleLLC, 2020a] :* A Google platform based on an NXP SoC with an integrated Edge TPU, very energy-efficient for ML inference (4 TOPS for $\sim$ 2W).

— *Intel NCS2 (Neural Compute Stick 2)* [Intel, 2022b ; Paul, 2017] : A VPU (Movidius) accelerator in USB stick form, designed for low-power AI inference (up to 4 TOPS for $\sim$ 1.5W).

Despite the intrinsically low consumption, software optimization remains crucial to maximize battery life or respect the thermal envelope. Benchmarks like EEMBC ULPBench [EEMBC, 2014 ; UpbeatLabs, 2023] aim to measure this efficiency.

1. https ://www.seeedstudio.com/blog/2024/08/12/cpu-vs-gpu-vs-tpu-vs-n
2. https ://anewtech.wordpress.com/2020/09/17/edge-ai-accelerator-ultimate-performance-for-edge-devices/

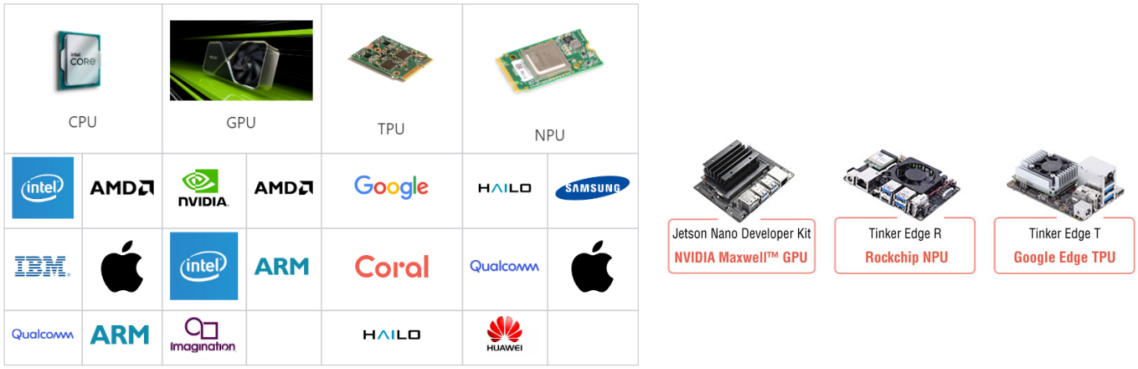


FIGURE 2.6 – Key Hardware and Main Vendors for AI and HPC systems. Sources : *seedstudio*[1] (left) and *anewtech*[2] (right)

2.2.1.4 Other Components

Beyond the main compute units (CPU, GPU, accelerators), other components contribute non-negligibly to consumption and thermal challenges.

Memory (DRAM) : The consumption of DRAM is no longer negligible, especially in HPC and server systems equipped with large memory capacities. With terabytes of RAM, DRAM consumption can equal or even exceed that of idle CPUs [Appuswamy, 2015]. It depends on the technology (DDR4, DDR5, HBM...), capacity, module density (higher density DIMMs are more power-hungry [Skhynix, 2021]), and access activity. It is a power-hungry component often poorly covered by in-band measurement tools (cf. section 2.1).

Storage (HDD, SSD) : Traditional hard disk drives (HDDs) consume energy mainly to spin the platters and move the read/write heads. Solid State Drives (SSDs), with no moving parts, are generally more energy-efficient, especially at rest, but their consumption increases during intensive read/write operations. Parallel storage systems (e.g., Lustre, GPFS) used in HPC can involve hundreds or thousands of disks, representing a significant overall consumption.

Network (NICs, Switches) : Network Interface Cards (NICs) and interconnection equipment (switches, routers) consume energy for communication (data transmission and reception). Very high-performance networks used in HPC (e.g., InfiniBand, high-speed Ethernet) with bandwidths of hundreds of Gb/s can be particularly power-hungry, especially at the level of core switches. Technologies like RDMA aim to optimize transfers, but network consumption remains a factor.

Other System Components : Motherboard, fans, power supply units (PSUs - with their own efficiency losses), etc., add to the total node consumption.

2.2.2 The Crucial Role of Cooling

The electrical energy consumed by IT components is almost entirely converted into heat. This heat must be evacuated to keep the components within their optimal operating temperature range and avoid damage or performance degradation (thermal throttling). The cooling system is therefore an indirect but major energy consumer, as evidenced by the PUE (cf. Section2.1.1).

Key technologies :

— *Air Cooling :* The most common method, using fans to circulate ambient air (often pre-cooled by room or row-based air conditioners) through the servers and dissipate heat via heatsinks. Simple, but reaches its limits with current high power densities (>30-40 kW per rack).

— *Rear Door Heat Exchangers (RDHX) [Schmidt, 2009 ; AKCP, 2021] :* An air/water combination where a rack's rear door integrates a water-based heat exchanger that cools the hot air exiting the servers before it returns to the room. Improves efficiency compared to all-air systems. The general process is shows on Figure 2.7

— *Direct Liquid Cooling (DLC) [XENON, 2023 ; Meyer, 2013] :* A liquid (often water or a water/glycol mix) circulates directly via pipes to "cold plates" placed on the hottest components (CPU, GPU). Much more thermally effective than air, allowing for very high power densities. Requires a complex plumbing infrastructure but can significantly reduce cooling-related energy consumption. The heat recovered in the water can potentially be reused (e.g., heating buildings) [Ljungdahl, 2022 ; Meyer, 2013]. The impact of water temperature on system performance must be considered [Nonaka, 2020].

— *Immersion Cooling [Pambudi, 2022 ; scientific-computingcom, 2019] :* Complete servers are submerged in a dielectric (non-conductive) but thermally conductive liquid. Potentially offers the best cooling efficiency and allows the elimination of internal fans. However, maintenance (accessing components) is more complex and initial costs can be high.

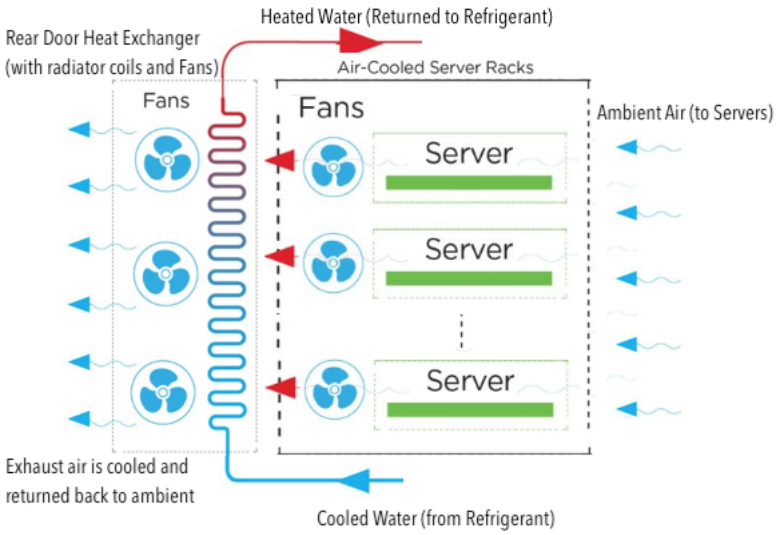


FIGURE 2.7 – RDHX main principle

With CPUs and GPUs reaching TDPs of 300W to 750W, liquid cooling solutions (DLC or immersion) are becoming indispensable for next-generation HPC and AI systems to manage the thermal load and improve overall energy efficiency [scientific-computingcom, 2019]. The choice and design of the cooling system have a direct impact on PUE and operational costs. The cooling infrastructure, unlike servers ($\approx$ 5 years), has a long lifespan (20-30 years), making

initial design choices critical. As example, the Nvidia DGX H100 server node, powered by eight H100 GPUs, projects a maximum power consumption of approximately 10.2 kW [Nvidia, 2024]. Figure 2.8 show's a breakdown of the physical server's node power consumption for modern HPC-AI datacenter. More than 55% of power is used for the 8 × 700 Watt GPUs.

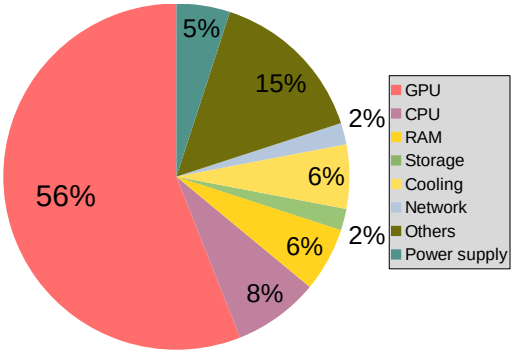


FIGURE 2.8 – Typical power breakdown inside a modern data center server node.

2.2.3 Native System Power Management Mechanisms

To allow for dynamic control of consumption based on load, hardware architectures (notably CPUs and, to a lesser extent, GPUs) and operating systems implement native power management mechanisms, often standardized by ACPI (or Device Tree on ARM). We describe some of them.

2.2.3.1 Processor Idle States (C-states)

ACPI defines several idle levels ("C-states") for processor cores or the entire package, ranging from C0 (active state) to increasingly deep states (C1, C3, C6, etc.). In these deep idle states, parts of the core (clocks, caches, functional units) are disabled and their voltage is reduced or even cut off, allowing for substantial energy savings during periods of inactivity. The deeper the C-state, the greater the energy saving, but the longer the wake-up time (latency to return to the C0 state). C-state management is primarily handled by the operating system based on detected inactivity.

2.2.3.2 Processor Performance States (P-states) / DVFS

ACPI also defines performance states ("P-states") which correspond to different combinations of clock frequency and supply voltage for the processor (P0 is usually the highest, P1, P2... are states with reduced frequency/voltage). This mechanism is known as DVFS (Dynamic Voltage and Frequency Scaling).

Physical Principle : The dynamic power consumed by a CMOS circuit is approximately proportional to Capacitance * Voltage² * Frequency ($P_{dyn} \approx C * V^2 * f$). Reducing the frequency (f) allows for a proportional reduction in power. But if the voltage (V) associated with this lower frequency can also be reduced (which is often the case), the power saving becomes quadratic (proportional to V²), hence the potential effectiveness of DVFS.

Implementation : The operating system, via CPU governors in Linux, decides which P-state to use based on the system load. Common governors include :

— *performance :* Maintains the maximum frequency (ignores lower P-states). Good for raw performance, bad for energy.

— *powersave :* Maintains the minimum frequency. Very energy-efficient, but can severely degrade performance.

— *ondemand :* Ramps up frequency quickly upon detecting load, then gradually decreases it if the load drops. Responsive but can oscillate.

— *conservative :* Similar to ondemand, but increases frequency more gradually. More stable, less responsive.

— *schedutil :* Uses information from the kernel's scheduler to estimate future load and adjust the frequency. It is considered as the most advanced and effective in recent Linux kernels.

— *userspace :* Allows the user or an application to manually set the frequency via system interfaces (sysfs). Useful for benchmarking or fine-grained control by "energy-aware" applications.

Impact on Performance : Reducing frequency via DVFS slows down the execution of compute-bound tasks. However, for memory-bound tasks, the impact on execution time can be much smaller, as the CPU already spends a lot of time waiting for memory. The frequency-memory bandwidth relationship depends on the microarchitecture. Dynamic and adaptive DVFS management is therefore key.

2.2.3.3 Power Capping

Allows setting an upper limit on the average power that the processor (CPU or GPU) or package is allowed to consume over a certain time window.

Mechanism : Implemented via interfaces like Intel RAPL or NVML/ROCm-SMI. When consumption tends to exceed the set limit, the hardware reacts by dynamically reducing the frequency (throttling), or even activating other limiting mechanisms, to stay under the defined threshold.

Usage : Very useful in energy-budget-constrained environments (data centers, Edge systems) to ensure that overall consumption does not exceed a given envelope, potentially at the expense of maximum performance (if the cap is too aggressive). It allows for predictive management of consumption at the rack or data center scale.

2.2.3.4 Thermal Throttling

This is not an intended optimization mechanism, but an ultimate protection. If a component's temperature (CPU, GPU) exceeds a critical threshold (defined by the manufacturer), the hardware automatically and aggressively reduces its frequency and voltage to decrease heat production and prevent physical damage. This leads to a severe and often unpredictable degradation of performance. A good cooling system and active power management (DVFS, power capping) are precisely aimed at avoiding reaching this thermal throttling threshold.

These native mechanisms provide a first layer of control over energy consumption. However, their default configuration by the OS (e.g., ondemand or schedutil governor) is often generic and not necessarily optimal for specific applications like AI, which have very particular execution profiles (rapid alternation of compute-bound and memory-bound phases). This opens the door for finer-grained optimization strategies at the application or runtime level, which would exploit these levers in a more informed way, as we will see in Section 2.5.

2.2.4 Conclusion

The infrastructures for AI and HPC are diverse, ranging from ultra-low-power embedded systems (*Arduino, Raspberry Pi, Nvidia Jetson, Google Coral Dev, Intel NCS2*) to massively parallel servers based on x86/ARM CPUs and powerful accelerators (*GPU, TPU, FPGA*), all housed in data centers where cooling efficiency is critical. The consumption of each component (*CPU, GPU, DRAM, storage, network*) and the native management mechanisms (*C-states, P-states/DVFS, Power Capping*) must be understood to consider energy optimizations. While these native mechanisms provide important foundations, their standard control by the OS is not always sufficient to achieve optimal efficiency, justifying the exploration of more advanced and application-specific control strategies.

2.3 Energy Profile and Environmental Impact of AI Applications

The previous sections have laid the general framework for energy metrics, infrastructures, and native management mechanisms. This section focuses specifically on AI applications and their energy footprint. AI workloads, particularly those related to deep learning, possess unique computational characteristics that translate into specific consumption profiles and optimization challenges, distinguishing them from traditional HPC applications. We have seen that the environmental impact of AI, especially its carbon footprint, has become a major concern. Here, we will explore the key phases of the AI lifecycle (training vs. inference), workload characteristics, the main models, and their documented energy and carbon impact.

2.3.1 The Energy Lifecycle of AI : Training and Inference

The development and deployment of an AI model generally follow a lifecycle 2.9 whose two most important computational phases from an energy perspective are training and inference.

2.3.1.1 Training Phase

This is the learning process where the model (e.g., a neural network) adjusts its internal parameters (weights) by iteratively processing a large volume of training data.

Characteristics : Typically very compute-intensive (especially matrix multiplications and convolutions), memory-hungry (*storage of weights, intermediate activations, gradients*), and often distributed across multiple nodes/accelerators (GPUs/TPUs), inducing significant network

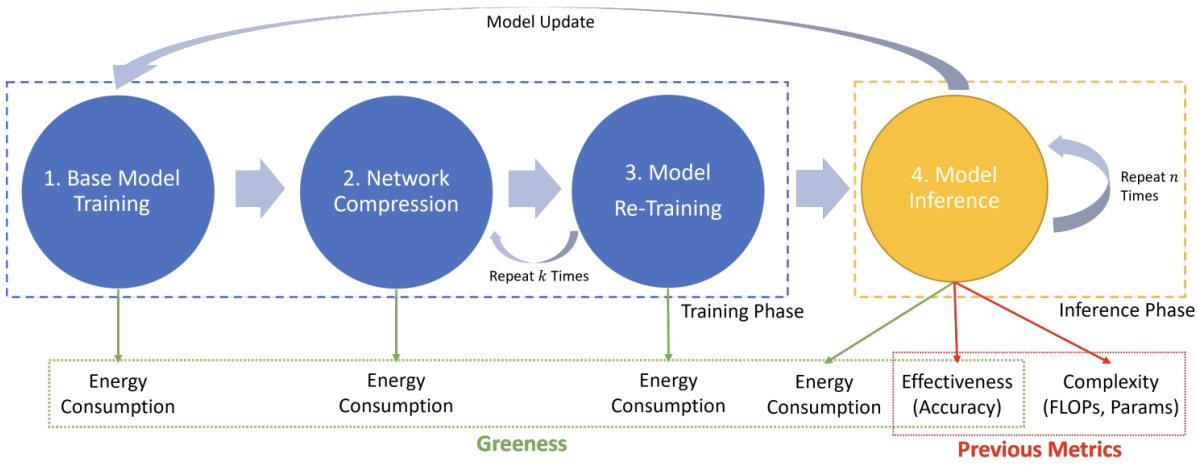


FIGURE 2.9 – Full Life cycle of efficient deep learning model [Li, 2021]

communication (exchange of gradients or parameters). It is characterized by repetitive cycles (*forward pass, backward pass, weight update*).

Consumption : Due to the intensive and often prolonged use (hours, days, or even weeks) of powerful HPC resources (multiple GPUs/TPUs), the training phase is extremely energy-intensive and represents a considerable financial and environmental cost for large models [Wu, 2021 ; Strubell, 2019 ; Alberto, 2021].

2.3.1.2 Inference Phase

This is the use of the model, once trained, to make predictions on new data inputs.

Characteristics : Typically less computationally expensive than a training iteration (often, only the forward pass is needed), but may need to be executed very frequently, at a large scale, and often with low-latency constraints (real-time). It can be deployed on a much wider range of infrastructures, from large Cloud data centers to low-power Edge devices.

Consumption : The energy consumed by a single inference is much lower than that of the complete training. However, the Model Usage Intensity (MUI), i.e., the total number of inferences performed during the model's lifetime before a potential update, is a critical factor. If the MUI is very high (e.g., a model used billions of times a day in a web service), the cumulative consumption of inference can largely exceed that of the initial training. Estimates suggest that inference can account for up to 80-90% of the total energy cost over the lifecycle for large-scale deployed systems [Desislavov, 2023].

2.3.1.3 Importance of a Holistic Approach

Much research focuses on optimizing energy efficiency for inference alone (e.g., *quantization, pruning* for Edge deployment). However, as pointed out by [Li, 2021], a comprehensive evaluation of the energy efficiency of an AI solution must consider the entire lifecycle (data collection, pre-processing, training, deployment/inference, updating), taking into account the expected MUI. A technique that drastically reduces inference energy but requires much longer or more frequent

training might not be globally beneficial if the MUI is low. Conversely, investing more energy in Neural Architecture Search (NAS) or training can sometimes lead to a much more efficient model in inference, thus reducing the overall cost if the MUI is high [Gholami, 2021a].

2.3.2 Specific Characteristics of AI Workloads versus Traditional HPC

Although AI uses HPC infrastructures, the characteristics of its workloads present notable differences compared to more classic scientific HPC applications (simulations, modeling), which has implications for the energy profile and optimization strategies.

Nature of Computations :

— *AI (Deep Learning) :* Dominated by dense operations like matrix-matrix products (GEMM) and convolutions, very well suited to the parallel SIMD/SIMT architectures of GPUs and TPUs. Often uses reduced-precision arithmetic (FP16, BF16, INT8) to accelerate computations and reduce memory consumption and footprint, at the potential cost of a slight loss of accuracy. The computational patterns can be regular within a layer, but the sequence of operations is dictated by the complex network architecture [Kwon, 2022 ; Park, 2022].

— *Scientific HPC :* More varied workloads can include dense computations (linear algebra), but also sparse computations (e.g., mesh-based simulations, sparse matrix-vector multipliers - SpMV), Fourier transforms (FFT), solving systems of differential equations. Double precision (FP64) is often required for numerical stability. Computational patterns are often more regular and predictable, related to the discretization of the physical problem [Tadonki, 2023].

Memory Access and Data Movement :

— *AI :* Manipulates huge datasets (images, texts, sounds) and models with billions of parameters. Training is very memory-intensive, requiring the storage of weights, activations, gradients, and optimizer states. Data movement between main memory (DRAM) and accelerator memory (HBM on GPU/TPU), as well as between nodes in distributed training, is a major performance bottleneck and a significant source of energy consumption (every memory access or network communication costs energy). The nature of input data access can also be different (stream of small files vs. large files).

— *Scientific HPC :* Memory intensity is highly variable depending on the application (some very compute-bound, others memory-bound). Memory accesses can be more structured (constant stride accesses), but inter-process communications (via MPI) remain critical and energy-intensive for large-scale parallel simulations.

Dependence on Accelerators :

— *AI :* Strongly dependent on GPUs, TPUs, or other specialized accelerators to achieve acceptable performance. The consumption of these accelerators often dominates the node's energy budget (e.g., >55% for 8 GPUs in a DGX H100 server [Nvidia, 2024]).

— *Scientific HPC :* Also uses GPUs, but a larger share of computations can still run efficiently on multi-core CPUs. The CPU/GPU balance varies more depending on the codes.

Software Stack :

— *AI :* Relies heavily on high-level frameworks like TensorFlow, PyTorch, JAX, which abstract much of the hardware management and parallelization, but can also introduce their own overhead and complexity for fine-grained energy optimization.

— *Scientific HPC :* Often uses compiled languages (C, C++, Fortran) with explicit parallelization libraries (MPI, OpenMP, OpenACC) and optimized numerical libraries (BLAS, LAPACK, FFTW). Control over execution is potentially finer.

Specific Energy Efficiency Challenges :

— *AI :* Optimization strongly focuses on the efficient use of accelerators (keeping compute units busy), reducing data movement (optimizing data flows, using caches/fast memories), and managing the consumption of inter-node communications in distributed settings.

— *Scientific HPC :* Places more emphasis on code optimization for the CPU (vectorization, data locality), managing the energy of CPU cores (DVFS adapted to phases), and the efficiency of MPI communications, in addition to using GPUs where applicable.

Understanding these differences is crucial for applying the right characterization, modeling, and energy optimization techniques in the specific context of AI applications on HPC infrastructures. The typical system stack of modern HPC-AI cluster is show on figure 2.10

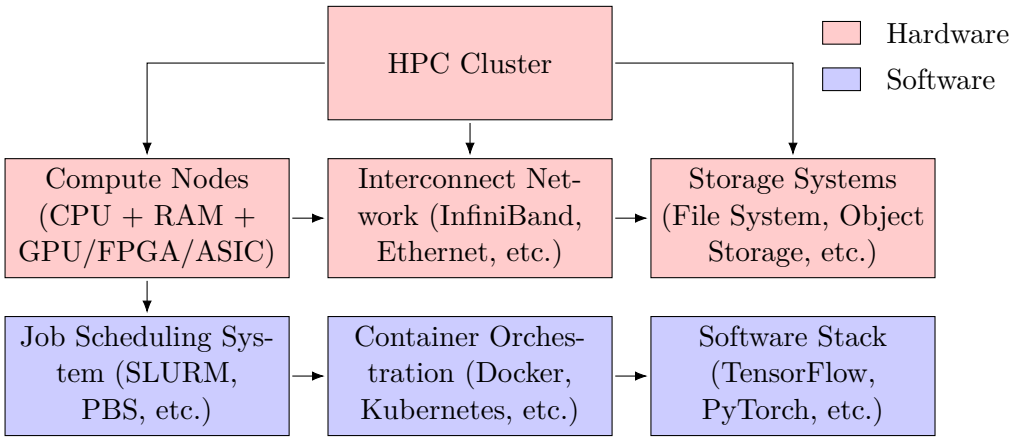


FIGURE 2.10 – Illustration of a typical HPC-AI architecture.

2.3.3 Representative AI Models and their Energy/Carbon Impact

Deep learning models, particularly in domains such as computer vision and natural language processing (NLP), demand substantial computational power (especially during the training phase). These workloads typically involve large-scale clusters equipped with GPUs or TPUs operating continuously over extended periods, which results in significant energy consumption. As models and datasets scale, the associated power requirements grow rapidly, posing serious concerns regarding sustainability. This section examines two prominent AI paradigms (*Convolutional Neural Networks (CNNs)* and *Large Language Models (LLMs)*) to illustrate how energy consumption manifests in real-world HPC-AI scenarios.

2.3.3.1 Convolutional Neural Networks (CNNs)

CNNs are foundational models in computer vision, well-suited for tasks such as image classification, segmentation, and object detection. Their architecture leverages local connectivity and spatial hierarchies to efficiently process visual data. In high-performance computing contexts, CNNs benefit from the parallel computation capabilities of modern accelerators, such as Nvidia GPUs and Google TPUs, significantly reducing training and inference time.

However, this performance gain comes at the cost of elevated energy use. Large-scale CNN architectures (such as ResNet [He, 2015a], VGG [Simonyan, 2015a], EfficientNet [Tan, 2019], and U-Net [Ronneberger, 2015]) demand considerable computational throughput, resulting in increased power consumption and thermal stress. As these models grow deeper and wider, their energy profile becomes a critical design consideration, especially in large-scale deployments.

A detailed study by Castro et al. [Castro, 2018] provides empirical insights into CNN energy performance on Nvidia's Maxwell and Pascal GPU architectures. The study showed that energy efficiency could fall significantly below vendor-stated peaks, with Pascal achieving only 20 GFLOPS/W in some 64-bit workloads. It also revealed a 60% gap in performance-per-watt across different CNNs and batch sizes, highlighting the influence of batch configuration and dataset complexity. Notably, larger batches improved energy efficiency but sometimes introduced accuracy trade-offs, particularly with smaller datasets.

Beyond algorithmic complexity, hardware characteristics and deployment configurations (e.g., single vs. multi-GPU environments) further affect energy profiles. Communication and synchronization overhead in distributed training scenarios can reduce arithmetic intensity and exacerbate inefficiencies.

To mitigate these impacts, researchers are developing energy-conscious strategies, including model compression techniques like pruning [Yang, 2016], quantization [Gholami, 2021a], and knowledge distillation [Gou, 2021]. These methods aim to reduce model size and computational load without compromising accuracy. On the hardware side, advances in low-power processors and AI accelerators, along with efforts to utilize renewable energy sources in data centers, offer promising avenues for improving the energy sustainability of CNNs [Ting, 2021 ; Chen, 2017 ; Xu, 2023 ; Desislavov, 2023].

2.3.3.2 Large Language Models (LLMs)

Large Language Models, such as GPT [Radford, 2018], BERT [Devlin, 2019], and their successors, have become central to modern NLP. These models are built on the Transformer architecture [Vaswani, 2017a], which uses attention mechanisms to capture complex dependencies in sequential data. While this design enables high-quality language understanding and generation, it also requires enormous computational resources, particularly during pretraining on large corpora.

Training LLMs involves billions to trillions of parameters and terabytes of data, making HPC infrastructures with powerful GPUs or TPUs essential. However, this translates to high energy use and a considerable carbon footprint. Recent studies indicate that inference, rather than

training, is emerging as the primary contributor to environmental costs, potentially accounting for up to 90% of a model's total lifecycle energy use [Jegham, 2025]. This shift reflects the transition from one-time training to continuous, large-scale deployment across millions of daily queries.

Modern LLMs have scaled dramatically in both size and computational requirements. GPT-4 is estimated to have 1.76 trillion parameters, making it ten times larger than GPT-3's 175 billion parameters, while LLaMA 3.1 features variants with up to 405 billion parameters and was trained on over 15 trillion tokens using 16,000 Nvidia H100 GPUs [3]. The emergence of reasoning models such as o3, DeepSeek-R1, and Claude-3.7 Sonnet represents a shift toward multi-step logic and chain-of-thought reasoning, further increasing computational demands.

A notable example from earlier generation models is the training of the BLOOM model [Luccioni, 2022], a multilingual LLM comprising 176 billion parameters. BLOOM was trained on 1.6 TB of data over 118 days, consuming 433,196 kWh of electricity and 1,082,990 GPU hours. The total carbon emissions, including those from embodied hardware and idle energy use, were estimated at 50 tonnes of CO_2e. Table 2.3 presents a comparative overview of emissions for major LLMs, including both training and inference metrics.

Table 2.3 – Carbon footprint and energy consumption for training and inference of SOTA LLMs (2024-2025 data). Training data adapted from [Luccioni, 2022; Patterson, 2022; Chowdhery, 2022]. Inference data from [Jegham, 2025].

Model	**Parameters**	**Training**		**Inference**	
		Energy (MWh)	Emissions (tCO_2e)	Energy/ Query (Wh)	Annual (tCO_2e)*
GPT-3	175B	1,287	552	0.43	62
BLOOM	176B	433	30	0.28	40
GLaM	1,162B	456	40	1.2	172
PaLM	540B	3,435	271	0.85	122
GPT-4o	∼1.76T†	∼10,000 †	∼4,200 †	0.43	62
LLaMA-3.1 405B	405B	∼5,000 †	∼1,900 †	0.65	93
Claude-3.7 Sonnet	∼200B†	∼2,500 †	∼950 †	0.35	50
Gemini 1.5 Pro	∼400B†	∼4,500 †	∼1,800 †	0.24	34
DeepSeek-R1	∼671B†	∼6,500 †	∼2,600 †	33+	4,700+
o3	Unknown†	Unknown†	Unknown†	33+	4,700+

Notes : † Estimated values based on industry reports and scaling analyses. ∗Annual CO_2e for inference assumes 100M queries/year at global average carbon intensity of 400 gCO_2/kWh. Training emissions calculated using regional carbon intensities where training occurred. Recent reasoning models (o3, DeepSeek-R1) show dramatically higher per-query consumption due to multi-step inference processes.

The inference footprint has become particularly significant with large-scale deployment. GPT-4o processes an estimated 700 million queries daily as of December 2024, and scaling this to annual consumption results in electricity use comparable to 35,000 U.S. homes, freshwater evaporation matching the annual drinking needs of 1.2 million people, and carbon emissions

3. https ://ai.meta.com/blog/meta-llama-3-1/

requiring a Chicago-sized forest to offset. Advanced reasoning models like o3 and DeepSeek-R1 consume over 33 Wh per long prompt, more than 70 times the consumption of smaller models.

Recent benchmarking studies reveal significant variations in model efficiency. Google's Gemini LLM uses around 0.24 Wh per text query, equivalent to using a microwave for one second, while Claude-3.7 Sonnet ranks highest in eco-efficiency when considering performance relative to environmental cost. These findings highlight the growing paradox : although individual queries are becoming more efficient, their global scale drives disproportionate resource consumption.

To address these concerns, researchers are investigating model optimization techniques such as distillation [Sanh, 2019], pruning, quantization, and architectural innovations that aim to reduce model size and complexity while maintaining performance. Large but sparsely activated DNNs can consume less than 1/10th the energy of large, dense DNNs without sacrificing accuracy despite using as many or even more parameters. Furthermore, hardware-level improvements, including specialized AI chips, and the adoption of cleaner energy sources are being explored to reduce the environmental impact of LLMs. However, gaps in power supply, combined with the rush to build data centers to power AI, often result in increased reliance on fossil fuels, which still account for just under 60% of electricity supply in the US.

In summary, both CNNs and LLMs demonstrate the dual challenge of achieving state-of-the-art performance while managing the energy cost of training and deployment. As the demand for HPC-AI systems continues to grow, with GPU shipments to data centers increasing from 2.67 million in 2022 to 3.85 million in 2023, with even greater increases expected in 2024, it becomes essential to integrate energy-aware practices (*spanning algorithmic, architectural*, and *infrastructural layers*) to ensure the sustainability of future AI developments.

2.3.3.3 Other Models (RNNs, GNNs, Diffusion Models, etc.)

Beyond the commonly discussed Convolutional Neural Networks (CNNs) and Large Language Models (LLMs), other classes of models also have notable computational and energy characteristics. Recurrent Neural Networks (*RNNs, LSTMs, GRUs*) for sequential data or Graph Neural Networks (GNNs) for graph-structured information often receive less attention in energy-efficiency studies but can still represent substantial computational loads in specific domains.

Additionally, modern generative architectures such as diffusion-based models (e.g., *Stable Diffusion*[Esser, 2024], $DALL \cdot E\ 2$ [Ramesh, 2022]) have emerged as highly energy-demanding due to their iterative denoising processes and large-scale training requirements. These models often involve billions of parameters and require many forward passes to generate a single sample, leading to significant energy consumption during both training and inference. As generative AI grows in popularity, understanding and optimizing the energy profiles of these models becomes increasingly important.

2.3.4 Growing Awareness and Paths to More Sustainable AI

Awareness of the energy and environmental impact of AI is growing in the scientific and industrial community [Wu, 2021 ; Strubell, 2019 ; Qiu, 2022 ; Patterson, 2021 ; Zhao, 2022]. Several

avenues are being explored to mitigate this impact :

— **Algorithmic Optimization and Frugal Models :** Designing inherently more efficient models (NAS), compressing them (*pruning, quantization, distillation*), using more efficient training techniques [Patterson, 2022].

— **Hardware and Software Optimization :** Improving the efficiency of accelerators, developing specialized low-power hardware architectures, optimizing compilers and runtimes [Wu, 2021].

— **Infrastructure Optimization :** Improving data center efficiency (PUE), using more performant cooling systems, choosing locations with low-carbon electricity [Patterson, 2022 ; Patterson, 2021].

— **Use of Renewable Energies :** Powering data centers with clean energy sources is the most direct lever to reduce the operational carbon footprint [Patterson, 2022].

— **Best Practices and Transparency :** Systematically including energy and carbon evaluation in scientific publications [Patterson, 2022 ; Patterson, 2021], developing accessible estimation tools [Ludvigsen, 2022 ; BCG, 2020 ; Lacoste, 2019], judiciously choosing model size and retraining frequency, sharing resources [Zhao, 2022].

The work of this thesis, by focusing on energy optimization at the system and software level (via measurement, modeling, scheduling, and runtime control), is part of this global movement towards AI that is not only more powerful but also more energy-responsible.

2.3.5 Section Conclusion

AI applications present an energy lifecycle dominated by the training and inference phases, with workload characteristics (*parallelism, memory access, accelerator dependency*) distinct from traditional HPC. Training large models (CNNs, and especially LLMs like GPT, BLOOM, Gemini...) generates considerable energy consumption and carbon footprint, raising sustainability issues. Inference, though less costly per unit, can dominate the overall energy balance under intensive use. A growing awareness is stimulating research for solutions at all levels (algorithms, hardware, software, infrastructure, best practices) for a *"greener"* AI. Our work is positioned in software and system optimization, an essential link in this chain of efforts.

2.4 Modeling Energy Consumption : Approaches, Tools, and Challenges

While measurement (Section 2.1) provides the raw consumption data, energy modeling aims to establish quantitative relationships to understand, predict, and ultimately, optimize this consumption. Developing models capable of estimating the energy consumed by a system or application based on its state, configuration, or activity is crucial for many tasks : *a priori evaluation of the energy impact of different design choices, informing energy-aware scheduling algorithms, or driving real-time dynamic optimization mechanisms.* This section explores the

different approaches to energy modeling, from simple linear models to more complex methods based on machine learning or theoretical analysis, examining their advantages, limitations, and the specific challenges posed by modeling AI applications and critical components like DRAM (see Figure 2.11).

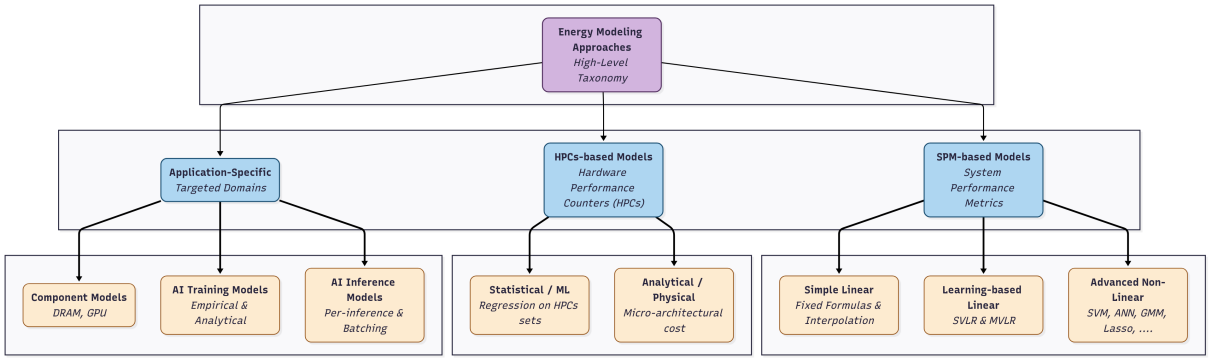


FIGURE 2.11 – Full modeling taxonomy in this thesis

2.4.1 Taxonomy of Software-based Energy Models

The literature on energy modeling of servers and computer systems is rich, with approaches evolving over time in parallel with hardware technological progress and measurement capabilities [Dayarathna, 2016]. A common classification distinguishes models along two main axes :

2.4.1.1 Type of Relationship (Linear vs. Non-linear)

Linear Models : They postulate a directly proportional (or affine) relationship between power consumption and one or more explanatory variables (e.g., CPU utilization rate). Simple to build and interpret, they were the first to be developed.

Non-linear Models : They recognize that the energy-activity relationship is often more complex (*quadratic, exponential, polynomial*, or other). They are potentially more accurate but can be more difficult to develop or require more advanced mathematical techniques.

2.4.1.2 Construction Method : Mathematical Formula versus Machine Learning

Based on Mathematical Formulas (Fixed Parameters) : These models rely on predefined equations whose parameters (slope, y-intercept) are often derived from specific measurement points (e.g., idle and full-load consumption) or simplified theoretical assumptions.

Based on Machine Learning (Variable Parameters) : These models use statistical or learning techniques (*linear/polynomial regression, SVM, neural networks...*) to learn the relationship between input variables and consumption from a large experimental dataset. The model's parameters are adjusted to minimize the prediction error on the training data.

2.4.2 Models Based on Component Utilization : System Performance Metrics

The earliest and simplest models have historically been based on aggregate system resource utilization metrics (SPM - System Performance Metrics), particularly the CPU utilization rate (u_{cpu}), initially considered the main driver of consumption [Bohrer, 2002 ; Fan, 2007b ; Fan, 2007a].

2.4.2.1 Simple Linear Models

The Founding Model : The observation of a quasi-linear relationship between power and u_{cpu} on early 2000s servers [Bohrer, 2002] led to simple models like :

$$P = (P_{MAX} - P_{MIN}) \times u_{cpu} + P_{MIN} \tag{2.4}$$

where P_{MIN} is the power at idle and P_{MAX} is the power at 100% CPU utilization. This model, sometimes reformulated with the ratio $k = P_{MIN}/P_{MAX}$ (Eq. 2.5), has been widely used for quick estimations [Qureshi, 2009 ; Gmach, 2009 ; Dai, 2016 ; Raycroft, 2014 ; Sharma, 2019].

$$P = ([1 - k] \times PMAX \times u_{cpu}) + (k \times P_{MAX}) \tag{2.5}$$

"70% Rule" Simplification : Based on the empirical observation [Fan, 2007b] that an idle server consumes about 70% of its maximum power (k=0.7), the model was simplified to (Eq. 2.6) [Beloglazov, 2012].

$$P = P_{MAX} \times (0.7 + 0.3 \times u_{cpu}) \tag{2.6}$$

Piecewise Linear Interpolation : Tools like CloudSim [Calheiros, 2011] adopted a slightly finer approach by using piecewise linear interpolation between several measurement points (u_{cpu}, P) rather than a single line (Eq. 2.7) to evaluate the energy efficiency of the cloud resource allocation algorithms.

$$P = P1 + (\Delta \times (\frac{u_{cpu} - u1_{cpu}}{10}) \times 100) \tag{2.7}$$

where P1 and P2 are the power values corresponding to the CPU utilization "u1" and "u2" respectively ($u1 \leq u \leq u2$) and Δ is the slope of the line between points (u1, P1) and (u2, P2).

Based on Application Throughput : For I/O-bound or network-oriented applications, a linear model based on application throughput (e.g., requests/sec) was proposed (Eq. 2.8) [Jin, 2013].

$$P = ([P_{MAX} - P_{MIN}] \times \frac{throughput}{throughput_{Max}}) + P_{MIN} \tag{2.8}$$

Limitations : These very simple models ignore the contribution of other components (*Memory, Disk, Network*) and do not capture fine non-linearities or the impact of the workload type

for the same ucpu. Their accuracy is therefore often limited, especially on modern architectures [Fan, 2007b ; Cheung, 2018].

2.4.2.2 Learning-Based Linear Models

To improve accuracy, linear regression techniques have been applied, learning the optimal parameters (slope β, y-intercept α) from real data.

Single-Variable Linear Regression (SVLR) : Fits a line (Eq. 2.9) to minimize the squared error on a set of measurements [Raghavendra, 2008 ; Berral, 2010 ; Berral, 2011 ; Chen, 2008 ; Pedram, 2010 ; Zhang, 2013]. Sometimes the intercept α is fixed to P_{MIN} [Wang, 2010]. It can also use application throughput as a variable [Koller, 2010] instead of u_{cpu}.

$$P = \alpha + \beta \times u_{cpu} \tag{2.9}$$

Multi-Variable Linear Regression (MVLR) : Recognizing the importance of other components, these models incorporate several SPM variables (Eq. 2.10) [Economou, 2006].

$$P = \alpha + \beta 1 \times u_{cpu} + \beta 2 \times u_{mem} + \beta 3 \times u_{disk} + \beta 4 \times u_{net} \tag{2.10}$$

One variant fixes the intercept to P_{MIN} [Alan, 2014], others use a subset of these variables (e.g., *CPU, Mem, Disk* without Network) [Kansal, 2010].

— *Advantages* : More accurate than fixed-parameter models because they are better fitted to the real data of the target machine.

— *Limitations* : Still assume a linear relationship, which can be inaccurate. Selecting the right variables is crucial.

2.4.2.3 Non-Linear Models

Simple Non-Linear Formulas (SVNLF) : Empirical non-linear relationships based on u_{cpu} have been proposed, such as a polynomial form (Eq. 2.11) [Fan, 2007b ; Qureshi, 2009] or an exponential form (Eq. 2.12) [Lien, 2007], where parameters r, α, β are calibrated experimentally.

$$P = ((P_{MAX} - P_{MIN}) \times (2 \times u_{cpu} - u_{cpu}^{r})) + P_{MIN} \tag{2.11}$$

$$P = ((P_{MAX} - P_{MIN}) \times (\alpha u_{cpu}^{\beta})) + P_{MIN} \tag{2.12}$$

Polynomial Regression (SVPR) : Uses u_{cpu}, u_{cpu}^2, u_{cpu}^3 ... as variables in a regression to capture the non-linearity (often related to the effect of DVFS where $P \approx f^2$, and f can vary with u_{cpu}) [Janacek, 2012 ; Zhang, 2013 ; Canuto, 2016] (Eq. 2.13). It can be extended to multiple variables (e.g., CPU², mem², cross-terms...).

$$P = \alpha + \beta 1 u_{cpu} + \beta 2 u_{cpu}^2 + \beta 3 u_{cpu}^3 \tag{2.13}$$

Others techniques often used for power modeling : A technique with *Lasso Regression (LR)* is used with polynomial or multi-variable regression to automatically select the most relevant variables and avoid overfitting, by penalizing non-zero coefficients (L1 regularization) [McCullough, 2011]. It is generally more useful when many variables (SPM or Hardware Performance Counters (HPCs)) are available. *Support Vector Machines (SVM)* is also a learning model capable of capturing complex non-linear relationships by projecting data into a higher-dimensional space via a kernel function [McCullough, 2011 ; Schölkopf, 2001]. It is suitable when variables are interdependent. *Neural Networks (ANN / DNN)* with Deep Learning approaches like *Multi-Layer Perceptrons (MLPs)* [Cupertino, 2015] or *Recursive Auto-Encoders* [Li, 2016] can model very complex and non-linear relationships between a large number of input variables (SPM or HPCs) and power consumption. Can take into account the history of consumption (recursive models). *Gaussian Mixture Models (GMM)* is another probabilistic approach that models the data distribution as a mixture of several Gaussians [Reynolds, 2008 ; Dhiman, 2010]. It can be used to cluster similar energy behaviors based on several variables (e.g., *CPU, IPC, memory access, cache*) and apply a specific model to each cluster. It helps to recognize that the application type strongly influences the u_{cpu}-power relationship.

2.4.2.4 Conclusion on SPM-based models

Models based on aggregate system utilization metrics (CPU%, Mem%, ...) have evolved from simple linear formulas to sophisticated learning techniques (*MVLR, Polynomial, SVM, DNN*). While their accuracy has improved, they still suffer from a fundamental limitation : the same aggregate utilization (e.g., 50% CPU) can correspond to very different underlying activities (intense computation vs. memory waiting) and thus to different consumptions. This has motivated the development of models based on finer-grained indicators : hardware performance counters.

2.4.3 Models Based on Hardware Performance Counters (HPCs)

These models exploit Hardware Performance Counters (HPCs), available through interfaces like PAPI [McCraw, 2014] (see the architecture on Figure 2.12), which provide very detailed information about the internal activity of the processor and memory system (*number of instructions executed, cycles, cache accesses/misses at different levels, memory reads/writes, floating-point instructions, mispredicted branches*, etc.). The assumption is that energy consumption is more directly related to this fine-grained micro-architectural activity than to aggregate utilization rates.

Pioneering Approaches : Works like [Economou, 2006] were among the first to show the correlation between certain HPCs (memory access, disk/network I/O) and consumption, in addition to the CPU. [Husain Bohra, 2010] proposed an MVLR model including cache utilization.

MVLR/Lasso Models Based on HPCs : Numerous works have applied multiple linear regression or Lasso regression to a large number (tens or even hundreds) of HPCs to predict

4. https ://www.vi-hps.org/cms/upload/material/tw42/PAPI.pdf

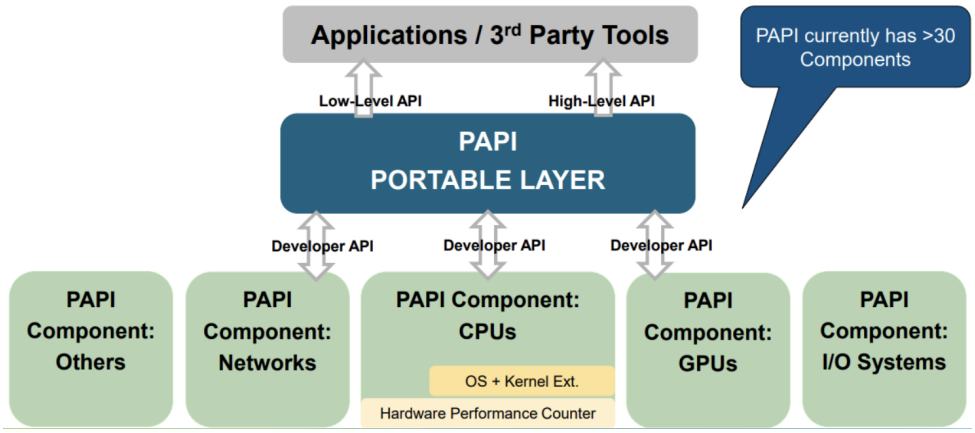


FIGURE 2.12 – PAPI Framework. Image by *Radita Liem*[4]

power [Da Costa, 2010; Jarus, 2014; Witkowski, 2013]. Selecting the relevant counters (those with a physical correlation to consumption) is a key challenge. [Jarus, 2014] proposed combining application clustering with HPC-based models specific to each cluster. [Cupertino, 2015] and others have also used ANN/MLP with a large number of HPCs as inputs.

Analytical/Physical Models : Instead of statistically learning the relationship, these models attempt to derive it from a physical understanding of the consumption of micro-architectural units. For example, one could try to estimate the energy per instruction, per cache access, per DRAM access, etc., and then sum these contributions weighted by the number of events measured by HPCs. Works like [Dolz, 2016] and [Bertran, 2010] are cited as having analyzed consumption via HPCs, but robust and general analytical models remain difficult to build, as the energy per event strongly depends on the system state (frequency, voltage) and complex interactions [IPCC, 2014].

— *Advantages of HPC-based models :* Potentially much more accurate and informative than SPM models because they reflect the actual activity at the micro-architectural level. Can allow for breaking down consumption by activity type.

— *Challenges :*

  — High Number of Counters : Choosing the relevant counters from the thousands available is complex.

  — Multiplexing : The hardware can often only measure a small subset of counters simultaneously, requiring time-division multiplexing which can introduce noise.

  — Overhead : Frequent reading of many counters has a cost.

  — Portability : The meaning and availability of counters vary greatly between architectures (Intel, AMD, ARM).

  — Correlation vs. Causation : A statistical correlation between a counter and power does not always mean a direct and stable causal relationship.

  — Lack of general theoretical models : Despite attempts, a generalizable theoretical framework for estimating energy from HPCs without direct measurement remains a challenge [IPCC, 2014].

2.4.4 Modeling Related to Specific Components and AI Phases

Beyond general models for the CPU or the server, efforts are needed to model specific components or particular application phases.

2.4.4.1 DRAM Modeling

As seen previously, DRAM is a significant consumer (see Figure 2.13) but often not measured by RAPL. Developing models to estimate its energy (static and dynamic) is crucial. These models typically rely on :

— *Memory-related HPCs :* Memory access (reads/writes), utilized bandwidth, cache miss.

— *DRAM states :* Consumption varies depending on whether rows are active, precharging, or refreshing. Finer models may attempt to track these states.

— *Physical characteristics :* DRAM type (DDRx), capacity, density, voltage.

— *Approach proposed in this thesis :* The development of an analytical model for DRAM based on SPM (cf. Chapter 4) is a response to this gap identified in the state of the art.

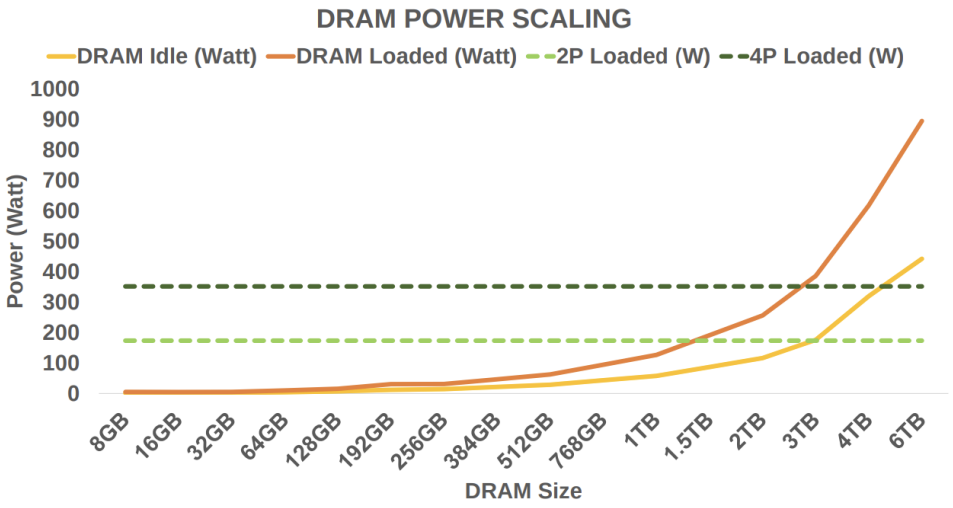


FIGURE 2.13 – DRAM vs CPU (2P and 4P for 2 and 4 CPUs systems packages loaded) power consumption scaling [Appuswamy, 2015].

2.4.4.2 GPU Energy Modeling

While NVML/ROCm-SMI provide an overall measure of GPU power, finer models seeking to link consumption to types of operations (computations vs. HBM memory accesses), shader/memory frequency, or the utilization of different units (Tensor Cores, etc.) can be useful for more advanced optimization. The approach is often empirical, based on profiling a wide range of GPU "kernels". Performance modeling tools like the Roofline model [Williams, 2009] offer useful insights related to the nature of the workload (i.e compute bound or memory bound) but often lack the necessary granularity for detailed analysis of DL operators on modern GPUs

[Ding, 2019]. Some analytical models [Kumar, 2025 ; Hoffmann, 2022] typically relied on FLOP counting and simple scaling, often overlooking detailed nuances of hardware efficiency. These models while insightful are not sufficient to tackle the energy efficiency problems.

2.4.4.3 AI Training Energy Modeling

Modeling the energy of a full training epoch is complex because it involves a sequence of distinct phases (*forward, backward, update, communication*). The approaches can be :

Empirical/ML : Learn a model (e.g., regression, ANN) linking hyperparameters (batch size, model architecture), system characteristics (GPU/CPU frequency), and potentially HPCs, to the measured energy per epoch or per phase. Some works on energy prediction have focused on correlating hardware PMCs with power consumption for general HPC workloads [Molka, 2010 ; Bertran, 2010]. Althouh being insightful, managing the technical complexity and ensuring the interpretability of models with a large number of PMCs can be challenging and not easy to generalize [McCullough, 2011]. Simpler PMC-based regression models have also been explored [Dolz, 2016 ; Cupertino, 2015], but these models may lack a consistent formal basis that can handle the hardware flexibility and architectural diversity of cutting-edge AI models.

Analytical/Composite : Model the energy of each fundamental operation (*convolution, GEMM, activation, communication*), then compose these models to estimate the total energy based on the network architecture and hyperparameters. This approach is more complex but potentially more explainable and generalizable. More advanced analytical approaches have considered incorporating greater architectural detail [Eberius, 2022 ; Wang, 2020]. Frameworks like PALEO [Qi, 2017] provides detailed operator-level accounting. However, a robust modeling of non-linear efficiency saturation with varying operator dimensions together with a systematic calibration across various architectures still need to be investigated. The work in this thesis on analytical models for training (cf. Chapter 4) falls into this category.

Estimation/Proxy for NAS : In the context of Neural Architecture Search (NAS), fast estimators of performance or energy ("proxies") are used to rank candidate architectures without fully training them [Frey, 2022]. These estimators can be based on simple theoretical calculations (e.g., number of FLOPs, memory accesses) or very short training runs. Work is being done to more explicitly integrate energy into the search (e.g. ENOS [Nasrin, 2022] and others [Wang, 2019b ; Wu, 2019]).

2.4.4.4 AI Inference Energy Modeling

Inference is often simpler to model (usually, a single forward pass), but the constraints (low latency, execution on Edge) are different. Models often seek to predict the energy consumed per inference based on input size, model complexity, and hardware frequency/configuration [Poddar, 2025 ; Lazuka, 2024]. Grouping inferences into batches (batching) often improves throughput and energy efficiency (amortizing fixed overheads), but increases latency. Modeling must capture this effect. Systems like BatchSizer [Nabavinejad, 2021] dynamically optimize the batch size. Models must be very lightweight and account for the specifics of Edge accelerators (TPU, VPU, NPU)

and very strict power/thermal constraints [Rodrigues, 2017 ; Kakolyris, 2024 ; Merkel, 2020 ; Archet, 2023 ; Lahmer, 2022].

2.4.4.5 Hybrid and Self-Calibrating Models

Recent approaches [Amaral, 2023 ; Fieni, 2020 ; Qi, 2023] combine hardware measurements (HPCs, RAPL), statistical models, and learning to improve accuracy and adaptability. SmartWatts [Fieni, 2020] is an example of a self-calibrating system that does not need prior manual training. HighRPM [Qi, 2023] uses statistical models to improve the temporal resolution of power measurements.

2.4.5 Conclusion

Energy modeling has progressed from linear models based on CPU utilization to sophisticated approaches using dozens of hardware performance counters and advanced learning techniques (*ANN, SVM, GMM, Lasso*). However, significant challenges remain : the precise modeling of key components like DRAM, the construction of explainable and generalizable analytical models (especially for complex phases like AI training), the selection of the right indicators (HPCs), portability across architectures, and the ease of use of models for optimization. The work presented in this thesis (*analytical DRAM model, analytical training/inference models*) aims to make specific contributions to fill some of these identified gaps.

2.5 Overview of Energy Optimization Techniques

Having explored how to measure and model energy consumption (Sections 2.1 and 2.4), as well as the characteristics of infrastructures (Section 2.2) and AI workloads (Section 2.3), this section turns to the solutions : the techniques and strategies aimed at actively reducing the energy consumption of computer systems, especially when they are running AI applications. The spectrum of these techniques is broad, ranging from static hardware or software optimizations (applied once and for all) to dynamic adaptations during execution, and covering different levels of abstraction (*circuit, architecture, operating system, application, AI algorithm*) [Shuja, 2016]. We will first examine general approaches, before focusing more specifically on optimizations at the system and software level, which form the core of this thesis's contributions, without forgetting the techniques specific to the energy optimization of AI models themselves, as they influence the overall context. A typical taxonomy in this thesis is shown on Figure 2.14.

2.5.1 Main Categorization of Energy Optimization Approaches

Energy optimization strategies can be classified along several axes.

2.5.1.1 Static versus Dynamic

Static Approaches : Optimizations are decided upon and applied before the application runs or during the system design. They are generally simpler to implement but less adaptive

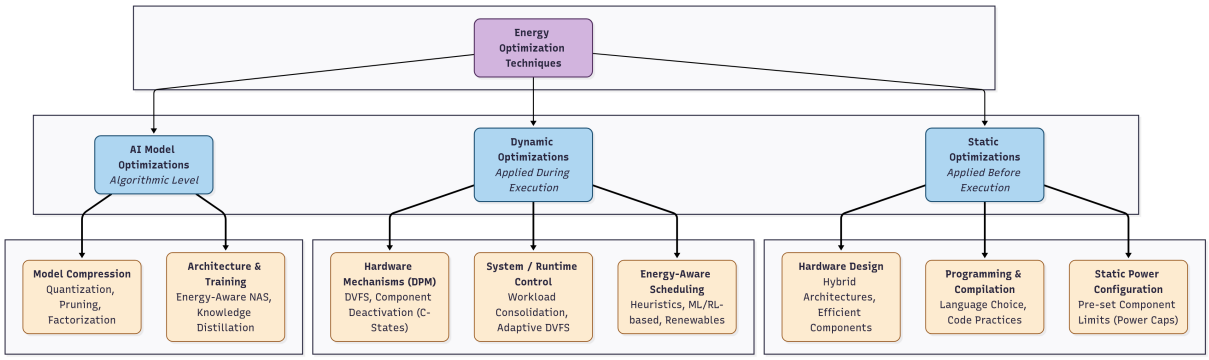


FIGURE 2.14 – Full optimization taxonomy in this thesis

to load variations. Examples : choosing low-power hardware, using an efficient programming language, energy-optimized compilation, a fixed *power cap* setting.

Dynamic Approaches : Optimization decisions are made during the application's execution, in response to current conditions (workload, system state). They are more complex but offer greater potential for savings by adapting to real needs. Examples : dynamic DVFS, adaptive scheduling, VM/container consolidation.

2.5.1.2 Hardware versus Software

Hardware Optimizations : Intervene at the physical design level of components. Examples : low-power circuit design [Beloglazov, 2010], hybrid architectures (big.LITTLE) [Intel, 2019], choice of efficient memory technologies, design of high-performance cooling systems. These optimizations are the responsibility of hardware designers.

Software/System Optimizations : Act at the level of the operating system, middleware, libraries, or the application itself. Examples : OS management of P-states/C-states, energy-aware scheduling algorithms, compilation techniques, application code optimization, dynamic adaptation of execution parameters. It is primarily this level that is addressed in this thesis.

2.5.1.3 Algorithmic Level

For AI applications, a third crucial level is the optimization of the algorithm itself (the model structure, the learning method). Examples : quantization, pruning, distillation, NAS. Although not directly addressed as a contribution in this thesis, these techniques are essential in the ecosystem of energy-efficient AI and are therefore briefly presented here.

2.5.2 Static Optimization Approaches

These techniques aim to establish a baseline of energy efficiency even before execution begins.

2.5.2.1 Energy-Aware Hardware Design

Hybrid Architectures (CPU) like the ARM's big.LITTLE approach, adopted by Intel (Alder Lake/Lakefield [Intel, 2019]), combines *"performance"* cores (powerful but power-hungry) and

"efficiency" cores (less powerful but very economical) on the same chip. The operating system can then assign tasks to the most suitable cores to save energy. Selecting processors (ARM vs. x86), accelerators (GPU/TPU/FPGA with the best perf/watt ratio for the task), memory, or storage that are inherently more economical. Techniques at the chip design level to reduce current leakage (Static Power) or switching energy (Dynamic Power) [Beloglazov, 2010].

2.5.2.2 Energy-Efficient Programming and Compilation

Choice of Language : Energy efficiency varies considerably between programming languages. Studies [Pereira, 2017] have shown that compiled languages like C are significantly more energy-efficient than interpreted languages like Python (a factor of 59× reported in one case). However, productivity and the ecosystem (e.g., AI libraries) often favor high level and less efficient languages like Python. Compilation (e.g., via Cython, Numba for Python) or the use of underlying libraries optimized in C/C++/Fortran are crucial. Adopting **Good Programming Practices** (coding styles that promote efficiency like data locality to optimize cache usage, avoiding frequent memory allocations/deallocations, using appropriate data structures) helps reduce execution time and often the energy consumed [Meyers, 2005]. Modern compilers apply many optimizations (loop unrolling, vectorization, inlining...) that improve performance and can reduce energy. Research aims to explicitly integrate energy objectives into the compilation process [Lin, 2013], for example, by relying on energy cost models per instruction [Leite, 2014a].

2.5.2.3 Static Power Configuration

Setting, via vendor-provided tools (NVIDIA-SMI, RAPL, etc.), a maximum power limit for a component (CPU, GPU) before launching the application. If the "cap" is well-chosen, this can reduce energy without impacting performance too much for certain applications [Cabrera, 2019]. Finding the right setting a priori, however, remains difficult and often sub-optimal compared to dynamic adaptation. A methodology based on analyzing the energy-performance trade-off (Pareto front) can help choose a relevant static cap [Cabrera, 2019], but automating this choice remains a challenge.

2.5.3 Dynamic Approaches

These techniques adapt to real execution conditions to maximize energy savings.

DVFS (Dynamic Voltage and Frequency Scaling) : A fundamental technique that adjusts the CPU (and sometimes GPU) frequency and voltage according to the load [Sueur, 2010]. This is the mechanism behind ACPI P-states and the OS CPU governors (ondemand, conservative, schedutil). Its effectiveness is based on the fact that dynamic power varies quadratically with voltage. It is very effective for reducing energy during low-activity phases or for memory-bound applications [Sueur, 2010]. Its fine-grained control (how and when to change P-state) is an active research topic (cf. system/runtime optimizations below). Specific strategies may be needed in case of system failure to limit consumption during recovery [Morán, 2020] .

DFS alone (Dynamic Frequency Scaling) : Sometimes, only the frequency is adjustable, or the voltage is not independently controllable over all frequency ranges. The energy savings are then smaller (proportional to f) but still possible.

Dynamic Component Deactivation (DCD) : Putting components or CPU cores into a deep sleep state (C-states > C0) or turning them off completely when they are idle. The OS management of C-states is a typical example. On/Off Policies for entire servers in a cluster based on the overall load are an extension of this [Benoit, 2018], but must consider the cost (time, energy) of rebooting.

2.5.3.1 System and Runtime Level Techniques

These aim to control the native hardware DPM mechanisms more intelligently or to manage resources at the system level.

Workload Consolidation : In virtualized (Cloud) or containerized environments, grouping virtual machines (VMs) or containers onto a minimum number of physical servers to allow idle servers to be turned off [Sanjeevi, 2017 ; Leite, 2014b]. This is a DCD technique at the data center scale.

Dynamic and Adaptive DVFS/Power Capping Management : Rather than letting the OS generically manage P-states (e.g., with schedutil), finer-grained approaches seek to : Detect application phases (compute-bound, memory-bound, I/O-bound, communication) during execution (via HPCs, annotations...). Apply the optimal energy configuration (frequency, power cap) for each detected phase, based on predictive models (cf. 2.4) or pre-established profiles. Tools like BDPO, READEX, MERIC, or works like [Vaddina, 2021] explore this path. Recent works use Reinforcement Learning to dynamically control DVFS or Power Capping at the node or GPU level, learning the optimal policy based on the observed state (load, temperature...).

Workload Peak Reduction and Balancing : Techniques aiming to smooth the workload in time or space to avoid consumption peaks and better utilize resources. *"Space-Time Multiplexing" (STM)* [Sai, 2013] is an example that seeks to balance the load via DVFS to reduce peaks.

Energy-Aware Scheduling : The scheduler (HPC job scheduler like Slurm/PBS, or Edge orchestrator like Kubernetes) makes allocation (where to run ?) and sequencing (when to run ?) decisions by integrating energy objectives. This is a very active research area.

— *Consideration of Renewable Energies :* Scheduling non-urgent tasks when renewable energy is abundant (e.g., solar during the day) [Toosi, 2022 ; Nardin, 2022].

— *Heuristic Approaches :* Development of fast heuristics [Litzinger, 2023] or optimization models (sometimes stochastic [Filippini, 2024] or hierarchical [Guan, 2022]) to find good energy/performance trade-offs.

— *Knowledge/ML-Based Approaches :* Using predictive models of performance or consumption [Diel, 2025] or GNNs [Li, 2022] to guide the scheduler's decisions.

— *RL-Based Approaches :* Reinforcement learning is increasingly explored [Singh, 2017] for energy-aware scheduling, allowing the scheduler to learn a complex adaptive policy [Yu,

2020 ; Liu, 2022 ; Zhang, 2019].

— *Hybrid Approaches :* Combination of static and dynamic techniques, or different levels of optimization. For example, statically profiling an application to identify phases, then applying a dynamic DVFS adapted to these phases [Vaddina, 2021 ; Grant, 2017]. Or using static planning for initial placement and dynamic adjustment for power [Jafari-Nodoushan, 2020].

2.5.4 AI Model-Specific Optimizations at Algorithmic Level

Although not the core of the system/software contributions of this thesis, it is essential to mention the optimization techniques acting directly on the AI models themselves, as they have a major impact on the final consumption. They mainly aim to reduce the size and computational complexity of models, especially for inference.

Quantization : [Gholami, 2021b] Reducing the numerical precision of weights and/or activations (e.g., FP32 -> FP16, BF16, INT8, or even less). Decreases memory footprint, required bandwidth, and the complexity of arithmetic operations, at the cost of a potential (often small if done well) loss of accuracy. Can be applied post-training or integrated into training (Quantization-Aware Training - QAT). Very popular for Edge deployment.

Pruning : [Yang, 2016] Removing weights or neural connections deemed "less important" (close to zero or having little impact on the output) in a pre-trained neural network. Leads to "sparser" models, reducing storage and potentially the number of computations (if the hardware supports sparse acceleration). *"Energy-aware pruning"* techniques specifically aim to prune to maximize energy gain.

Filter Compression / Matrix Factorization : Techniques aiming to reduce redundancy in convolutional or dense layers. The "bottleneck architecture" [He, 2015c] or tensor decomposition (e.g., Tucker, TensorTrain) are examples that reduce the number of parameters and operations.

Energy-Aware Neural Architecture Search (NAS) : Automating the search for neural network architectures that offer a good trade-off between accuracy, size, latency, and energy [Frey, 2022 ; Nasrin, 2022 ; Wang, 2019b ; Wu, 2019]. Often uses fast estimators ("proxies") of energy or integrates energy as an objective in the search function (e.g., ENOS(Energy-Aware Network Operator Search in DNNs) approach to address the *energy-accuracy trade-off* of a deep neural network acceleration [Nasrin, 2022]).

Knowledge Distillation : [Gou, 2021] Training a small "student" model to imitate the behavior (the outputs or internal representations) of a large pre-trained "teacher" model. Allows transferring knowledge into a more compact and faster model (e.g., DistilBERT [Sanh, 2019]).

These techniques are often complementary and can be combined (e.g., Pruning + Quantization) to obtain highly optimized models for energy-efficient inference. They modify the very nature of the AI model, unlike the system/software approaches explored in this thesis which seek to optimize the execution of a given model.

2.5.5 Conclusion

The arsenal of energy optimization techniques is vast, ranging from low-power hardware design to frugal AI algorithms. Static approaches provide a baseline efficiency, while dynamic approaches (DPM, intelligently driven DVFS, Power Capping, adaptive scheduling) allow for adjustment to load variations for more substantial savings. Optimization at the system and software level, by finely exploiting hardware levers and making intelligent decisions (phase detection, RL scheduling), represents a promising area and constitutes the main field of investigation for this thesis. Understanding AI-specific techniques (quantization, pruning...) is also important to grasp the overall context and potential interactions with system optimizations.

2.6 Synthesis and Overview of our Contributions

The detailed literature analysis conducted throughout this chapter has provided a comprehensive overview of the knowledge and tools available to address the energy efficiency of AI on distributed infrastructures. From fundamental measurement to complex optimization strategies, notable progress has been made. However, this review has also highlighted persistent limitations and areas where further research is needed. This section aims to synthesize these main gaps identified in existing works and to explain how the specific contributions of this thesis strive to provide targeted and original answers to them.

2.6.1 Gaps and Challenges Identified in the Literature

Our journey through the state of the art (sections 2.1 to 2.5) highlights several recurring challenges and areas for improvement : **On the Measurement Side, Synchronization and Integration, Optimization Side, and Scalability & Complexity of Scheduling**.

Despite the availability of in-band interfaces like RAPL and NVML, the fine-grained and reliable measurement of all major consuming components, particularly the main DRAM memory on many platforms, remains a major gap. Furthermore, the absolute accuracy of in-band estimators is often questionable without external calibration. Precisely correlating measurements (whether in-band or out-of-band) with fine-grained software execution, and easily integrating different data sources into a single, user-friendly tool (especially for languages like Python, prevalent in AI), remains complex with existing solutions.

While ML-based models (ANN, SVM...) sometimes achieve good local accuracy, they often lack explainability and their generalization to other platforms or workloads is difficult. The construction of analytical models, which are more interpretable and potentially more robust, remains a challenge, especially for complex phenomena like DRAM energy or complete AI training/inference phases. The rigorous validation of energy models over a wide range of scenarios and their actual domain of validity are often insufficiently documented.

The OS's default strategies for DVFS (governors) or the static use of Power Capping are often too generic and do not capture the fine dynamics of AI applications. The need for adaptive runtime controllers, based for example on phase detection, is clearly identified, but practical and

robust solutions are lacking. Energy-aware scheduling strategies for HPC/Cloud clusters face growing complexity.

While many building blocks exist (*measurement tools, models, optimization techniques*), there is often a lack of a clear, integrated methodology to guide a developer or system administrator through the complete process of improving the energy efficiency of a specific AI application on a given infrastructure, using a coherent set of tools and techniques.

2.6.2 Positioning of our Contributions

The work presented in this thesis aims to directly address several of the gaps identified above in the literature, by focusing on pragmatic and experimentally evaluated solutions at the system and software level.

2.6.2.1 Addressing the Gaps in Measurement

EA2P (Contribution 1 in section 3.3) is positioned as an attempt to provide a flexible software tool (Python) that aggregates several in-band sources (CPU, GPU) and integrates an initial solution for the indirect measurement of DRAM (via the model below), thus aiming for better coverage and ease of use for AI practitioners. Its architecture is designed to be more easily extensible than some existing tools.

Benchmarking (Contribution 2 in section 3.4 & 3.5) provides specific quantitative data, which may be missing for some recent architectures or for the fine-grained evaluation of the impact of system levers, thus serving as a validated basis for our own models and strategies.

2.6.2.2 Addressing the Challenges in Modeling

Our analytical DRAM model (Contribution 3 in section 4.2) directly targets the DRAM measurement "black hole", by proposing an HPC-based approach that aims for explainability and portability, in contrast to purely empirical or specific approaches for certain Intel server chips.

Our **analytical models for AI training/inference (Contribution 4 in section 4.3)** seek to provide an alternative to "black box" ML models, by proposing an analytical decomposition of phases that could offer better understanding of the sources of consumption and potentially improved generalization for a priori optimization.

2.6.2.3 Addressing the Need for Finer System/Software Optimization

Our **A priori-based scheduling approach (Contribution 5 in Chapter 5)** specifically explores the application of "what-if" analysis to the joint management of resources and energy for heterogeneous AI tasks (training/inference) in a simulated cluster environment.

2.6.2.4 Addressing the Lack of an Integrated Approach

The **proposed methodology (Contribution 6 in Chapter 6)** aims to structure the energy optimization process, by explicitly integrating the tools (EA2P), models, and strategies

developed in the thesis to provide practical guidance that is often lacking in the literature.

In summary, this thesis does not claim to reinvent the entire field, but it provides targeted and interconnected contributions that, starting from an improved measurement (EA2P integrating a DRAM estimation), develop specific analytical models (DRAM, AI), explore advanced optimization strategies (A-priori approach for scheduling), and synthesize everything into a practical methodology. The focus on analytical models, dynamic control based, and integration into a flexible Python tool (EA2P) constitute axes of originality compared to a significant portion of existing works.

Chapter 3

Systematic Energy Measurement Methodology and Application Energy Characterization

*Ce chapitre établit le socle empirique de la thèse en se concentrant sur le premier pilier de notre démarche : la mesure. Il introduit **EA2P**, notre première contribution logicielle, un profileur énergétique conçu en Python. Le chapitre détaille sa conception modulaire et extensible, qui permet d'agréger les données de multiples sources (CPU Intel/AMD, GPU NVIDIA/AMD) et qui intègre une innovation majeure : un modèle analytique pour estimer la consommation de la mémoire vive (DRAM), comblant ainsi une lacune critique des outils existants. La seconde partie du chapitre met en application cet outil à travers deux campagnes de benchmarking fondamentales. La première caractérise finement l'impact sur la performance et l'énergie des différents paradigmes de parallélisme (SIMD, OpenMP, GPU) sur des noyaux de calcul représentatifs. La seconde évalue quantitativement les compromis (trade-offs) induits par les leviers de gestion d'énergie système (DVFS, Power Capping). Ces résultats fournissent une base de données factuelle et indispensable qui informe et valide les travaux de modélisation et d'optimisation présentés ultérieurement.*

3.1 Introduction

We have so far described and analysed the rise of energy consumption in High-Performance Computing (HPC) systems and with Artificial Intelligence (AI) applications, as well as the state of knowledge on existing metrics, infrastructures, profiling, and optimization techniques. This chapter transitions to the presentation of our original contributions, starting with the foundation of any energy optimization process, namely the ability to precisely measure and characterize energy consumption.

An adage says, *"you can not improve what you can not measure."* In the field of energy efficiency, this assertion takes a particular importance. The literature (cf. Chapter 2) shows that

consumption estimates are often based on simplified models or global measurements that mask the complexity of interactions between hardware and software. Facing the growing diversity and Heterogeneity of hardware architectures (*multi-core CPUs, GPUs, various accelerators*), the variety of compute kernels used (*compute-bound, memory-bound, mixed*), and the complexity of software stacks (*AI frameworks, low-level libraries, operating systems*), a fine-grained and detailed understanding of energy consumption at the component and application-phase level becomes indispensable.

This chapter is therefore dedicated to the foundation of rigorous energy analysis. We first address the imperative need for precise and granular energy measurements, detailing the associated challenges and justifying the necessity of developing adequate tools. Subsequently, we present our first major contribution : the design and implementation of an energy profiling tool named EA2P. This tool aims to overcome some of the limitations identified in existing solutions by offering a flexible, multi-platform solution for profiling Python applications, a language that is ubiquitous in the AI ecosystem. Finally, in order to illustrate the utility of such metrology and to provide essential reference data, this chapter present the results of systematic benchmarking campaigns. The aim is on one hand, to characterize the energy profile of fundamental compute kernels on different CPU and GPU architectures, and on the other hand, to evaluate the quantitative impact and trade-offs of native power management mechanisms (such as *DVFS, power capping, and ACPI governors*).

All of this work (development of a measurement tool, characterization of basic building blocks) constitutes the necessary foundation upon which the subsequent contributions of this thesis regarding energy modeling and optimization will be built.

3.2 The Need for Accurate and Detailed Energy Measurements

3.2.1 Motivation and Objectives : Beyond Rough Estimations

The energy optimization of computer systems, particularly in the demanding context of AI and HPC applications, has become a central concern, driven by economic, environmental, and technological imperatives (cf. Chapter 1). However, to undertake relevant and effective optimization actions, it is essential to have a clear understanding and quantitative views of the actual energy consumption. Yet, as highlighted by the literature review (Section 2.1), many assessments still rely on global estimates, low-granularity outlet-level measurements, or manufacturer's nominal specifications (TDP), which very weakly reflect the dynamic consumption of applications. The fundamental motivation for precise and detailed energy measurement stems from several observations :

— **Limitations of Approximations :** Simplistic energy models based solely on CPU utilization or TDP often provide very coarse estimates of actual consumption, failing to capture the fine dynamics of the application or the contribution of other components (GPU, memory, etc.). Optimization decisions based on such approximations can be flawed or suboptimal.

— **Architectural Heterogeneity :** Modern HPC systems are intrinsically heterogeneous, combining multi-core CPUs (Intel, AMD, ARM), powerful GPUs (NVIDIA, AMD, Intel), and potentially other accelerators (FPGAs, TPUs). Each component has its own consumption profile and interacts in complex ways with others. A global measurement masks this complexity and prevents the identification of energy "Hot Spots" within the system.

— **Workload Diversity :** Even on a given architecture, the consumption profile varies considerably depending on the nature of the workload (compute-intensive, memory-intensive, communication-intensive, or mixed), the algorithms used, and even the software frameworks. Fine-grained measurements are necessary to capture these nuances.

— **Complexity of the Software Stack :** Modern applications, especially in AI, rely on complex software stacks (OS, drivers, runtimes, libraries, frameworks). Each layer can influence consumption. Understanding the energy impact of specific software choices requires a metrology capable of correlating software activity with hardware consumption.

— **Identifying Optimization Paths :** A detailed measurement, capable of breaking down consumption by component or by a program's execution phase, is essential for identifying the critical areas where optimization efforts (hardware, software, or algorithmic) would be most fruitful. It allows for targeting actions and quantifying their real impact.

The primary objective of this section is to highlight the necessity of advancing beyond rudimentary energy measurement techniques towards a comprehensive and robust energy metrology framework. Such a framework must fulfill several critical requirements to accurately support research and development in energy-aware HPC and AI systems.

3.2.2 Analysis of Energy Measurement Challenges

Although the need for accurate measurements is clear, obtaining them faces several significant technical and methodological challenges, as outlined in the state of the art (Section 2.1.3) and confirmed by the literature on measurement tools. Mains challenges we analysed are : access to raw data, component coverage, granularity, portability, flexibility, accuracy, and overheads.

Access to Raw Data : Accessing energy information often involves hardware interfaces (onboard sensors) or specific registers (MSRs) whose documentation by manufacturers is sometimes incomplete or difficult to access. Understanding the exact behavior of these counters can require extensive investigation, or even reverse engineering. The APIs provided by manufacturers (*Intel RAPL, AMD RAPL, Nvidia-SMI, ROCm-SMI*) are indispensable but have their own specifics and limitations. They may provide aggregated data for an entire board (GPU) or socket (CPU), making it difficult to estimate the consumption of a specific program if several are running simultaneously (complexity of non-additivity and static power loss).

Component Coverage : The direct measurement of main random-access memory (DRAM) consumption remains a major challenge, as many RAPL interfaces (especially on Intel *"client"* platforms or some AMD chips) do not cover it, while its energy impact is growing, especially in overloaded systems or those with large capacities [Appuswamy, 2015]. The consumption of disks, network interfaces, the motherboard, or cooling systems is generally not captured by in-band

tools, requiring out-of-band approaches for a global view.

Granularity and Synchronization : The refresh rate of internal sensors (e.g., $\approx 1ms$ for RAPL) is much better than that of many external wattmeters but may still be insufficient to capture very brief power spikes. Finely correlating a power/energy measurement with a specific code section or function executed at the same instant remains complex, especially in parallel environments or with high operating system activity.

Portability and Flexibility : Many tools are specific to an architecture (x86 vs. ARM), a vendor (*Intel, NVIDIA, AMD*), or even a hardware generation. Support for heterogeneous platforms with a single unified tool is rare. Accessing sensors is often easier on Linux than on Windows or macOS in direct mode. Many tools (*Perf, Likwid* in C/C++ mode) require command-line instrumentation or a low-level API, which can be a barrier for developers using higher-level languages like Python, which is widespread in AI. The need for a simple API to instrument Python code is pressing.

Accuracy and Overhead : The absolute accuracy of in-band sensors is not always guaranteed and may require calibration against out-of-band measurements. The process of frequently reading sensors, especially if many counters are polled in parallel by threads, can induce non-negligible energy consumption and execution time overhead, potentially biasing the measurements, especially for very fast-running codes.

The previous challenges highlight the inherent complexity of developing energy measurement solutions that are at once robust, accurate, flexible, and simple to use for fine-grained application profiling.

3.2.3 Why designing the New Tool EA2P

Our previous analysis (also the more detailed one in Chapter 2) demonstrates that, several energy measurement tools are available, but most of them suffer from significant shortcomings in terms of : *flexibility, portability, component coverage, programmability, maintainability and extensibility.*

Existing tools for Python, such as Perun [Gutiérrez, 2023], PyJoules [INRIA-Lille, 2019], CodeCarbon [BCG, 2020], Eco2AI [Budennyy, 2023], Carbon Tracker [Anthony, 2020], Tracarbon [Valeye, 2022], Experiment-impact-tracker(EIT) [Henderson, 2020], show limitations in the detailed measurement of RAPL domains, support for multiple GPUs, or the precise estimation of RAM energy (often limited to Intel servers or based on coarse TDPs). Tools like Likwid [Treibig, 2010a] are not designed for Python, and Variorum [LLNS, 2023], although flexible, targets integration into higher-level tools rather than fine-grained application profiling. Table 3.1 provides a compatibility overview of the aforementioned tools including ours (EA2P [Nana, 2024c]), *where CC = CodeCarbon, EIT = Experimental Impact Tracker, CT = Carbon tracker, Eco = Eco2AI, TC=Tracarbon, LW = LIKWID, PG = Intel PowerGadget, PJ = PyJoules, PT = Powertop, SP = Score-P, VR = Variorum, CP = CrayPat, DDT = Linaro Forge DDT & MAP and our **"EA2P"**.*

Considering our observations, it appeared relevant and necessary to develop a new measurement tool, named **EA2P (Energy-Aware Application Profiler)** [Nana, 2024c], with the

TABLE 3.1 – Compatibility overview of energy measurement frameworks.

Support	**Perun**	**CC**	**EIT**	**CT**	**Eco**	**TC**	**PJ**	**Perf**	**LW**	**PAPI**	**PG**	**PT**	**SP**	**VR**	**CP**	**DDT**	**EA2P**
GPU support																	
Nvidia GPU	✓	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓
AMD GPU	✓													✓		✓	✓
Intel GPU														✓			
CPU and RAM supports																	
Intel CPU	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AMD CPU					✓			✓	✓			✓	✓	✓			✓
RAM	✓	✓	✓	✓	✓	✓			✓					✓			✓
OS support																	
Linux	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Windows		✓									✓						✓
Mac OS		✓	✓			✓					✓						
Other important characteristics																	
Documentation	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
Configurable	✓	✓	✓									✓					✓
Code API	✓	✓	✓	✓	✓	✓	✓		✓	✓							✓
Open Source	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓
Perf oriented								✓	✓	✓			✓		✓	✓	
Energy oriented	✓	✓	✓	✓	✓	✓	✓				✓	✓		✓			✓
Multi-Nodes	✓								✓				✓	✓	✓	✓	✓
Device details																	✓

specific objectives of overcoming some of these limitations. EA2P aims to offer a flexible and modular framework in Python for the energy profiling of the main components (*CPU, GPU, RAM* via estimation) during the execution of applications, especially those developed with popular AI frameworks. Its design, implementation, and validation constitute the first original contribution of this thesis. We now present and describe it.

3.3 EA2P : A Multi-Platform Energy Profiler for Python Applications

This section presents our first original contribution : ***EA2P (Energy-Aware Application Profiler)***, a software tool designed for the fine-grained energy profiling of applications, with a particular focus on Python applications commonly used in the fields of Artificial Intelligence and High-Performance Computing. The development of EA2P was motivated by the limitations of existing tools in terms of flexibility, component coverage, and ease of use for Python developers (cf. Section 3.1.3).

The specific objectives for the development of EA2P were therefore to design and implement :

— A multi-device energy profiler for Python, capable of simultaneously measuring the consumption of the main contributors (CPU, GPU) and estimating that of RAM.

— A Python-based solution, leveraging its popularity and ecosystem to facilitate integration into AI/HPC workflows.

— A tool relying on existing hardware APIs (RAPL interfaces, Linux Perf, Nvidia-SMI, ROCm-SMI) for raw data acquisition.

— An analytical model for estimating RAM energy for platforms lacking dedicated sensors (a contribution that will also be detailed in Chapter 4, but its integration into EA2P is an aspect of the tool).

— A tool experimentally validated for its flexibility, relative accuracy, and utility in various profiling scenarios.

3.3.1 General Design and Architecture of EA2P

To achieve the set objectives, EA2P was designed with a modular and flexible architecture, centered around the Python ecosystem.

3.3.1.1 Design Philosophy

The tool is structured into distinct modules for managing each type of device (*Intel CPU, AMD CPU, NVIDIA GPU, AMD GPU, RAM*). This facilitates adding new modules for future hardware. EA2P creates an abstraction layer over the low-level APIs (*RAPL, Perf, SMI*), offering a unified interface to the user. The choice of Python allows leveraging its clear syntax, its numerous libraries (e.g., `'Threading'` for parallel monitoring of multiple devices to reduces the main program execution overheads, `'Pandas'` for data manipulation to process log easily), and its ease of integration with AI frameworks (*TensorFlow, PyTorch, JAX*). When possible and relevant, EA2P relies on low-level tools that directly access registers or hardware interfaces to improve the accuracy of measurements compared to higher-level estimates.

3.3.1.2 Technical Architecture and Dependencies

The overall architecture of EA2P is illustrated by the Figure 3.1. The EA2P core (in Python) constitutes the central logic of the tool, managing configuration, instrumentation (via Python API or CLI), launching monitoring threads, data collection, processing (aggregation, calculating energy from power), and report generation.

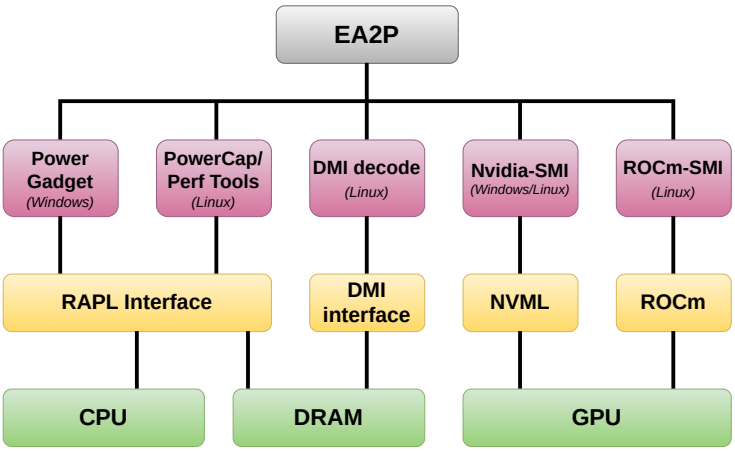


FIGURE 3.1 – Main diagram of our framework

EA2P interfaces with a set of APIs and system tools to access energy data :

— *For Intel CPUs :* Mainly via `/sys/class/powercap/intel-rapl/`, the Linux kernel's PowerCap interface, which exposes RAPL domains (*pkg, core, uncore, dram* if available). On Windows, Intel Power Gadget is an option.

— *For AMD CPUs :* RAPL access is more complex and less standardized. EA2P primarily uses the 'power/energy-pkg' events of the Linux Perf tool, which provides a measurement at the CPU package level.

— *For NVIDIA GPUs :* The Nvidia-SMI, via its CLI, is used to query the instantaneous power of the card.

— *For AMD GPUs (ROCm-compatible) :* The ROCm-SMI, via its CLI, is used similarly for AMD GPUs.

— *For RAM information (Type, Slots, Size) :* The Linux `'dmidecode'` tool is used to retrieve DMI (Desktop Management Interface) information necessary for applying the RAM energy estimation model (see 3.2.3).

EA2P implements a mechanism to automatically detect the system configuration and the presence of necessary tools/drivers, simplifying installation and use. EA2P is designed to be used in two main ways : *API Mode* and *Command-Line Interface (CLI) Mode*. The **API Mode** allows developers to directly instrument their Python code by importing EA2P as a library and using decorators or function calls to delineate the code regions to be profiled (fine-grained profiling). The **CLI Mode** allows profiling the execution of an entire Python script or an executable binary (e.g., C/C++, Fortran) without source code modification, by launching EA2P as a wrapper around the target command (global profiling).

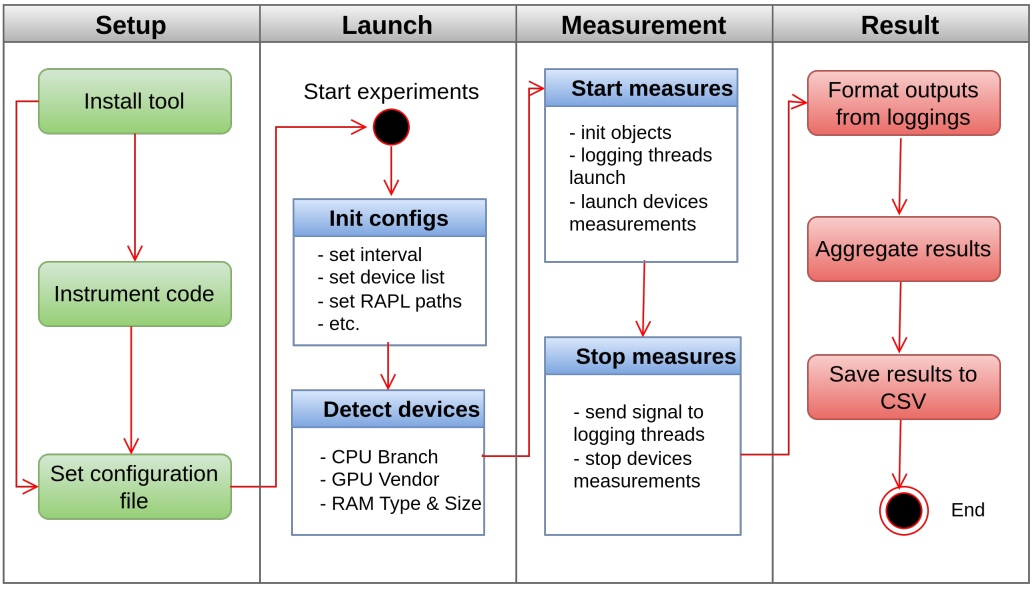


FIGURE 3.2 – EA2P general workflow.

The profiling process with EA2P follows a structured workflow, illustrated by the Figure 3.2. The key steps are :

Step 1 : Environment and Configuration (Setup)

Installation (1.1) : Installation of EA2P and dependencies (e.g., `'dmidecode'`, GPU drivers, `'perf'` for AMD CPU, PowerGadget for Windows/Intel). Requires appropriate access rights to the sensors (often root or via specific permissions). Code Instrumentation (1.2) : (Optional, for API mode) Annotation of the Python code with EA2P decorators to mark the sections to be profiled. Configuration File (2.1) : Setting up the experiment via a configuration file (e.g., sampling frequency, list of devices to monitor, output file path).

Step 2 : Initialization and Launch of Profiling (Launch & Running)

Initialization (2.3) : The tool reads the configuration, automatically detects active devices (CPU, GPU) and available measurement interfaces (2.2), and initializes internal variables. Parallel Energy Measurement (3.1) : EA2P launches dedicated Python threads for each monitored device. These threads run in the background and collect power data (or accumulated energy) at the specified sampling frequency, independently and in parallel to minimize overhead. Execution of Instrumented Code (3.2) : The user code (or target executable) is launched. The measurement threads continue their collection in the background.

Step 3 : End of Execution and Result Generation (Epilogue & Result)

Termination (4.1) : When the profiled code section (or the entire application) finishes, EA2P signals the measurement threads to stop. For MPI applications, data from the different ranks is aggregated at the master node. Data Processing and Output (4.2) : The raw collected data (time series of power/energy) are processed (calculation of total energy per interval and per device), aggregated, and formatted. The results are saved in CSV or text files, including measurements per device, time intervals, and metadata about the execution. This format facilitates subsequent analysis and visualization.

This modular design and structured workflow aim to provide a tool that is both powerful for detailed profiling and simple enough for wide adoption by the community. The next section will detail the implementation of measurement features for each component.

3.3.2 Implementation of Key Measurement Features per Hardware Component

EA2P's modular architecture allows for a specific implementation of measurement mechanisms for each major type of hardware component (*CPU, RAM, GPU*), adapting to the available interfaces on different platforms.

3.3.2.1 CPU Energy Measurement

The approach to measuring CPU energy varies mainly between Intel and AMD processors due to differences in their RAPL interfaces.

Intel CPU : EA2P uses the PowerCap interface (`/sys/class/powercap/intel-rapl/`). During profiling, it periodically polls (at the user-defined sampling frequency) the `'energy_uj'` files corresponding to the various available RAPL domains (e.g., *intel-rapl :0* for the package, *intel-rapl :0 :0* for the cores, etc.). These files provide the accumulated energy in microjoules. EA2P stores these successive readings and calculates the energy consumed during each interval

by difference. Particular attention is paid to handling the wrap-around of RAPL counters : if a reading is lower than the previous one (indicating a reset of the hardware counter, which has reached its maximum value `'max_energy_range_uj'`), the energy for that interval is calculated accordingly to reflect the full cycle of the counter. The tool also takes `'max_energy_range_uj'` into account as an upper bound of the energy recordable per request.

AMD CPU : The PowerCap interface is generally not implemented in the same way by AMD RAPL. EA2P therefore relies on the Linux `'perf stat'` tool for AMD processors. The `'power/energy-pkg'` event (or an equivalent depending on the kernel/perf version) is used to obtain an energy measurement at the CPU package level. Since `'perf stat'` is better suited for profiling an entire application rather than fine-grained sections continuously, EA2P launches an instance of `'perf stat'` in parallel with the execution of the instrumented code region, then terminates the `'perf'` process at the end of the region to collect the total energy measurement for that execution. The data is logged and processed to obtain a final report. The drawback is that this generally provides fewer measurement points than with Intel's continuous PowerCap interface (often one measurement at the beginning and one at the end of the `'perf stat'` execution for a given section).

3.3.2.2 RAM Energy Measurement & Estimation

As previously highlighted (Sections 2.1, 2.4, and 3.1.2), the direct measurement of main RAM consumption is a major limitation of many in-band tools. On Intel server architectures, energy profiling of the main memory (RAM) system aligns with the PowerCap interface, similar to that of the CPU, as RAPL covers the RAM. However, this approach does not apply to *Intel client* and *AMD CPU* systems, for which an alternative method is thus needed to estimate the energy consumed by the RAM.

A typical solution uses analytical models considering the workload and TDP or nominal power values. However, there is a genuine difficulty that comes from the variations among different generations of DDR RAM, particularly in terms of the Power Management Circuit (PMC). For instance, DDR5 technology stands out as a significant advance over DDR4, where a Power Management Integrated Circuit (PMIC) is directly integrated into the dual in-line memory module (DIMM), thereby resulting in a noticeable 30% increase in power efficiency [Samsung, 2022]. Consequently, any analytical model must consider this evolution with DDR technology.

EA2P integrates an analytical model to estimate the energy consumed by RAM, particularly for Intel Client and AMD CPU systems where the "DRAM" RAPL domain is not exposed. The full analytical model design and validation is presented in Section 4.2.

3.3.2.3 GPU Energy Measurement

EA2P uses the native interfaces provided by manufacturers to query the instantaneous power of GPUs. NVIDIA GPU uses the `'nvidia-smi'` command (or the underlying NVML library). EA2P periodically queries the card's power (in Watts) at the defined sampling frequency. AMD GPU (ROCm) uses the `'rocm-smi'` command (or the ROCm SMI library). The approach is similar to that for NVIDIA.

GPU energy is calculated by integrating the measured power over time (Formula 3.1) :

$$Energy_{GPU} = \sum_{i=1}^{N} P_i \times \Delta T_i, \tag{3.1}$$

where P_i is the power measured during the i^{th} measurement interval and ΔT_i is the duration of that interval (equal to the sampling frequency if the measurement is regular). The power data is collected in a log file and then processed to obtain the total energy.

Challenges (Nvidia-SMI) : We noted that `'nvidia-smi'` reports power rather than accumulated energy, which can pose challenges for direct correlation with other energy counters. Furthermore, at very high polling frequencies (milliseconds or less), `'nvidia-smi'` can return **"Unknown"** values, limiting the effective temporal resolution.

3.3.2.4 MPI Extension for Multi-Node (Distributed) Measurement

For profiling parallel applications running on multiple nodes of a cluster, EA2P integrates support based on MPI (Message Passing Interface), via the Python library `'mpi4py'` [Dalcin, 2021].

Principle : Each node (MPI process) runs its own instance of EA2P to locally profile its components (CPU, GPU, RAM). `'mpi4py'` is used to ensure that the measurement phases (start, stop) are globally synchronized among the nodes participating in the computation.

Aggregation : At the end of profiling, the energy data collected by each node is sent to the master node (MPI rank 0), which aggregates the results to provide a view of the total energy consumption of the cluster for the profiled execution, as well as the details per node.

3.3.2.5 Docker Containerization for Portability and Ease of Deployment

To simplify deployment and ensure the reproducibility of measurement environments, a containerized version of EA2P with Docker is proposed.

Advantages : Allows packaging EA2P and its software dependencies into a single Docker image, which can then be run identically on any system supporting Docker. This avoids manual installations and configuration conflicts, especially for users less familiar with system aspects.

Constraints : The Docker container must be launched in privileged mode (`'-privileged'`), with the correct device mounts (`'-device'`) and file system mounts (`'-v'`) (e.g., `/dev/cpu/0/msr, /sys/class/powercap, /proc, /dev/mem`) so that EA2P can access the hardware counters from within the container. Kernel-level dependencies (*PowerCap, SMI drivers, DMI Interface*, etc...) must of course be present on the host.

3.3.3 Experimental Validation of EA2P

After detailing the design, architecture, and implementation of EA2P, this section presents the results of its experimental validation. The main objective of this validation is to evaluate the reliability, accuracy, robustness, and practical utility of our tool in various energy profiling

scenarios. In line with EA2P's design goals, our validation process focused on the following aspects :

— *Tool accuracy assessment :* Evaluate the accuracy of the energy measurements provided by EA2P for the different covered components (CPU, RAM via model, GPU) by comparing them to established reference tools (e.g., *Linux 'perf'*, *CodeCarbon* for the GPU) on different hardware platforms.

— *Energy profiling consistency :* Verify the reproducibility and consistency of the profiling results obtained with EA2P over multiple executions of the same workloads on different platforms (Intel, AMD, NVIDIA).

— *Workload characterization :* Test EA2P's ability to correctly profile different types of workloads, including CPU-intensive, GPU-intensive, or mixed (CPU/GPU) operations.

— *Cross-platform compatibility :* Demonstrate EA2P's portability and its ability to function on heterogeneous systems equipped with Intel/AMD CPUs and NVIDIA/AMD GPUs.

— *Profiling overhead analysis :* Measure the time overhead induced by EA2P during profiling to ensure it remains within acceptable limits and does not significantly bias the measurements.

— *RAM energy model validation :* Specifically evaluate the accuracy of our RAM energy estimation model by comparing it to direct measurements from the RAPL DRAM domain on Intel server platforms that support it.

— *Evaluation of advanced features :* Demonstrate the utility of EA2P for more complex use cases, such as analyzing the impact of sampling frequency, and profiling multi-threaded and multi-node (MPI) parallel applications.

Our experiments were conducted on a set of heterogeneous machines to ensure the broadest possible validation. The main characteristics of these platforms are summarized in the Table 3.2.

Table 3.2 – Platform Characteristics

name	**Intel-1**	**AMD-1**	**AMD-2**	**Intel-2**
CPU name	i9 12950HX	EPYC 7452	EPYC 7513	E5-2698v4
GPU name	RTX 3080Ti	A100 SXM4	A100 SXM4	Tesla V100
CPU TDP	55W	155W (x2)	200W(x1)	135W (x2)
GPU TDP	150W	400W (x2)	400W (x4)	300W (x8)
CPU threads	24	64 (x2)	64 (x1)	40 (x2)
GPU VRAM	16GB	40GB (x2)	40GB (x4)	32GB (x8)
CPU RAM	32 GB	128 GB	512 GB	512 GB
Multi-SoC	No	Yes	No	Yes

The following subsections detail the experimental results obtained during the validation of EA2P on the various aspects previously stated. The experiments were repeated five times for each scenario to assess the consistency of the measurements, and the average values are generally reported, unless otherwise indicated.

We describe the items of Tables :

— **cores :** is Power Plane 0 (PP0), associated with the CPU cores. It tracks the energy consumption of all cores.

— **uncore** : is Power Plane 1 (PP1), it measures the energy consumption of the integrated GPU (not to be confused with an external Nvidia or AMD GPU) on Intel client architectures only. Remember that powerCap Linux interface name this package as *"uncore"* while in Linux perf tools it correspond to *"energy-gpu"* event.

— **DRAM/RAM :** It specifically measures the energy consumption of the RAM.

— **Package (Pkg) :** It tracks the overall energy consumption of the System on Chip (SoC). It includes the consumption of all cores, integrated graphics and also the uncore components (*QuickPath Interconnect (QPI) controllers, L3 cache, snoop agent pipeline, on-die memory controller, on-die PCI Express Root Complex, Thunderbolt controller*).

— **Psys :** It provides an aggregate view of power consumption across various components : package domain, System Agent, Platform Controller Hub(PCH), eDRAM and a few more domains on a single socket SoC. It is useful especially when the main source of the power consumption is neither the CPU nor the GPU, but the whole system instead.

The primary objective of our experiment is to measure and compare the energy consumption during the fine-tuning of the VGG16 [Simonyan, 2015a] deep learning model on two different hardware configurations : CPU-only and GPU-accelerated. We employ two datasets for this purpose : CIFAR-10 [Krizhevsky, 2009b] and Stanford Dogs [Deng, 2009 ; Khosla, 2011]. For all experiments, we use the full training set of each dataset and run training for 10 epochs to ensure a reasonable duration, as longer training would consume significantly more energy and is unnecessary for our tool validation context. All other training parameters are kept at their default settings following standard conventions. TensorFlow 2.12 with CUDA 12.2 was used for implementations.

Additionally, we use the `sleep` function from Python's `time` module to measure idle energy consumption as a baseline.

3.3.3.1 Measurement Accuracy and Consistency Compared with Reference Tools

A fundamental aspect of validation is to assess whether the measurements provided by EA2P are consistent and comparable to those obtained by established and recognized tools in the community. We conducted comparisons with the Linux system tool *'perf'* for CPU and RAM measurement (on platforms that allow it), and with the CodeCarbon tool for GPU measurement (and CPU indirectly).

CPU and DRAM Comparison with *'perf'* (Server Platforms) :

The Table 3.3 presents the energy consumption results for the CPU (Package Pkg) and RAM (if available via RAPL), as well as the execution time, measured by *'perf'* and EA2P for different workloads (idle, CIFAR-CPU fine-tuning, CIFAR-GPU fine-tuning) on the Intel-2 (Xeon E5-2698v4) and AMD-1 (EPYC 7452) platforms.

Table 3.3 – CPU and DRAM validation

		Intel-2			AMD-1		
		Energy(Wh)		time(s)	Energy(Wh)		time(s)
Application	**Tool**	**Pkg**	**RAM**		**Pkg**	**RAM**	
sleep	perf	2.254	1.290	183.508	4.781	/	185.105
	EA2P	2.181	1.270	180.272	4.650	4.853	180.503
CIFAR-CPU	perf	28.592	4.899	445.236	45.580	/	574.215
	EA2P	28.252	4.825	438.333	45.293	14.241	557.021
CIFAR-GPU	perf	1.627	0.514	68.40	1.610	/	45.105
	EA2P	1.229	0.371	52.50	1.225	0.968	33.921

The energy values measured by EA2P for the CPU package and RAM are generally slightly lower than those reported by *'perf'*. This behavior is expected and can be explained by a fundamental difference in the profiling approach : EA2P focuses on profiling the code sections explicitly instrumented by the user, whereas *'perf'* generally measures the energy consumed by the entire application, potentially including initialization or finalization phases not covered by EA2P's instrumentation. Despite this systematic discrepancy related to the measurement scope, the trend and orders of magnitude are consistent between the two tools. This difference in scope can be seen as an indicator of EA2P's ability to isolate the consumption of regions of interest, which is crucial for fine-grained optimization.

Table 3.3 shows a RAM measurement for EA2P on AMD-1 (where RAPL DRAM is not natively supported by *'perf'*), which corresponds to the estimate from our model (see section 3.2.4.4 for the specific validation of the RAM model).

CPU and GPU Comparison with CodeCarbon (Intel-1 Client Platform) :

The Table 3.4 compares EA2P's measurements with those of CodeCarbon for idle and CIFAR-GPU workloads. The GPU energy values are very close between EA2P and CodeCarbon, which is expected as both tools likely rely on Nvidia-SMI in the background for GPU power measurement. A more noticeable gap is observed for CPU energy, where EA2P's values are lower. This behavior could be due to the approach of CodeCarbon that uses an estimate based on 80% of the CPU's TDP when the RAPL interface is not detected (which might be the case or a default strategy in some configurations on this Intel Core i9 12950HX client platform). Such a TDP-based estimation is inherently less accurate than a direct RAPL reading, which could explain the difference in favor of EA2P's potential accuracy if it is indeed accessing RAPL. This observation is corroborated by specific tests on this platform (Table 3.5).

Table 3.4 – CPU and GPU validation on Intel-1 (Nvidia RTX 3080Ti)

Application	**tool**	**CPU (Wh)**	**GPU (Wh)**	**time(s)**
sleep	CodeCarbon	0.305	0.987	181.931
	EA2P	0.204	0.824	180.706
CIFAR-GPU	CodeCarbon	0.229	2.077	67.993
	EA2P	0.230	2.047	67.757

Validation of RAPL Domains on Intel Client (Intel-1) :

The Table 3.5 with detailed RAPL domains read shows a fine-grained comparison between *'perf'* and EA2P for the different RAPL domains and EA2P's RAM estimation. The measurements for the cores, uncore, pkg, and psys (total system power measured by RAPL) domains are very consistent between *'perf'* and EA2P, confirming that EA2P correctly accesses and interprets these RAPL counters. The slight discrepancies are still attributable to the measurement scope (code section for EA2P vs. entire application for *'perf'*).

TABLE 3.5 – CPU and DRAM validation on Intel-1 (core i9 12950HX)

		Energy(Wh)					**time(s)**
Application	**Tool**	**cores**	**uncore**	**pkg**	**psys**	**RAM**	
Sleep	perf	0.008	0.000	0.149	0.520	/	180.029
	EA2P	0.008	0.000	0.149	0.520	0.031	180.192
CIFAR-GPU	perf	0.089	0.001	0.274	2.78	/	72.626
	EA2P	0.056	0.001	0.229	2.672	0.014	66.903
CIFAR-CPU	perf	3.715	0.007	5.949	12.001	/	1476.905
	EA2P	3.696	0.007	5.952	13.488	0.295	1478.121

These comparisons indicate that EA2P provides energy measurements that are consistent with those of reference tools, and that the observed differences are mainly explained by EA2P's focus on instrumented code sections, which is an advantage for fine-grained analysis.

3.3.3.2 Profiling Overhead Analysis

The overhead induced by the measurement tool is a critical criterion : excessive overhead can not only slow down the application but also distort the energy measurements themselves. The Table 3.6 compares the execution time and energy (Pkg and RAM) for workloads (idle, CIFAR-CPU) profiled with *'perf'* and with EA2P, over 5 runs.

TABLE 3.6 – Consistency over system overhead on Intel-2

Application	**tool**	**Pkg (std)**	**RAM (std)**	**time(s) (std)**
sleep	perf	2.254(0.004)	1.290(0.002)	183.508(0.02)
	EA2P	2.181(0.02)	1.270(0.001)	180.272(0.002)
	gap (perf-EA2P)	**0.073**	**0.02**	**3.236**
CIFAR-CPU	perf	28.592(0.32)	4.899(0.10)	445.236(5.36)
	EA2P	28.252(0.40)	4.825(0.12)	438.33(6.81)
	gap (perf-EA2P)	**0.34**	**0.074**	**6.906**

Execution Time : For the CIFAR-CPU task (approx. 440s), the average time overhead of EA2P compared to *'perf'* (considered here as a baseline for the application's execution time, although *'perf'* also has its own overhead) is on the order of 6.9 seconds, or about 1.5%. For the "idle" task (approx. 180s), the overhead is 3.2 seconds, or about 1.8%.

Energy : The energy differences (between 0.02Wh and 0.34Wh, or <2-7% in the examples) are, as previously discussed, more related to the measurement scope than to a massive direct energy overhead from EA2P.

Stability : The standard deviation over the 5 runs was low (less than 3% for RAM and CPU on the intensive task, and less than 1% for idle), indicating good stability of EA2P's measurements in the face of operating system variability. These fluctuations could be attributed to OS task management and thermal throttling.

In conclusion, the time overhead induced by EA2P appears to be relatively low (less than 2% in the tested cases), which is acceptable for a profiling tool aimed at detailed analysis. This overhead is mainly due to the management of data collection threads in Python and the calls to underlying system interfaces.

3.3.3.3 Profiling Consistency and Cross-Platform Compatibility

EA2P's ability to provide consistent results across different platforms and its portability are essential features.

Consistency across different architectures : The Tables 3.3, 3.4, 3.5 (already discussed for accuracy) also show that EA2P operates and collects data as expected on the Intel-1 (client i9 + NVIDIA RTX), Intel-2 (server Xeon + NVIDIA V100), and AMD-1 (server EPYC + NVIDIA A100) platforms. The tool correctly identifies the available interfaces (RAPL PowerCap for Intel, 'perf' for AMD CPU, Nvidia-SMI for GPU) and applies the appropriate measurement methods.

Robustness to different workloads : The tests include idle workloads ("sleep"), CPU-intensive workloads (CIFAR-CPU training), and GPU-intensive workloads (CIFAR-GPU training). EA2P successfully profiled these different types of loads on the supported platforms, demonstrating its robustness. The workload type logically influences which components are predominantly stressed and therefore measured by EA2P.

3.3.3.4 Impact of Sampling Frequency

The frequency at which EA2P collects power data can influence the accuracy of the energy measurements (which is the integral of power) and the correlation with execution time. The Figure 3.3 illustrates this aspect. The figure shows the energy consumed by different domains (*psys, CPU package, GPU*) and the average execution time for sampling intervals varying from 0.001s to 32s.

Observations : Energy-Time Correlation : Shorter sampling intervals tend to better capture fine variations in energy consumption, which can improve the correlation with execution phases. Longer intervals can smooth out these variations and miss transient peaks. The `psys` measurement (overall system power), which encompasses everything, logically shows a better correlation with execution time.

Issues with Nvidia-SMI at low sampling intervals : It is noted that for GPU measurement via Nvidia-SMI, very high polling frequencies (milliseconds) can lead to "Unknown" values, limiting the effective resolution for the GPU. Very long intervals, combined with short execution times, can lead to a higher relative overhead during the data collection thread joining phase at the end of profiling, as these threads remain idle for longer.

EA2P allows the user to configure this sampling interval, offering a trade-off between measurement granularity and potential overhead.

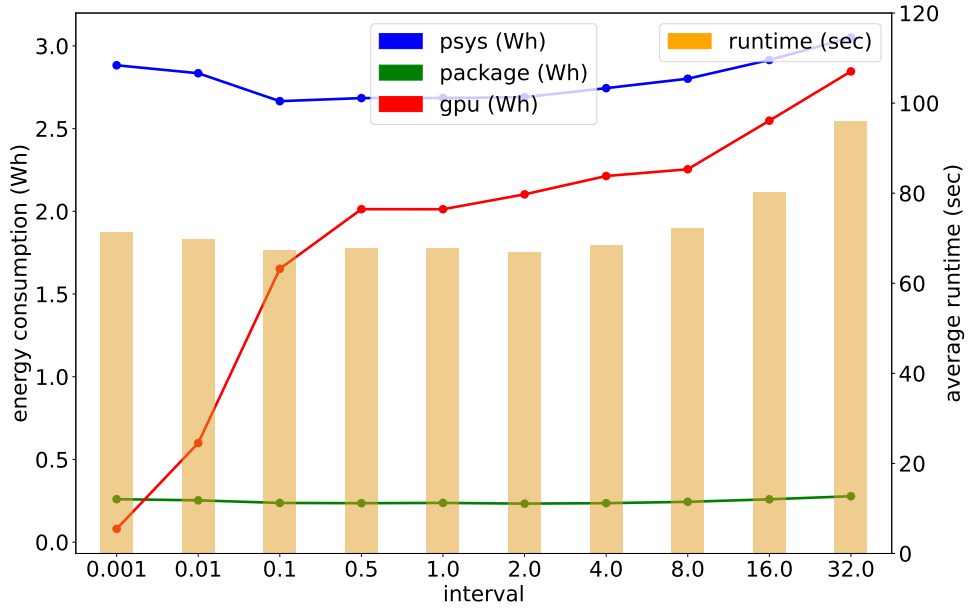


FIGURE 3.3 – Sampling interval behavior on Intel-1

Distributed Memory Parallelism (MPI) :

EA2P's ability to measure the energy of MPI applications was demonstrated by profiling a matrix multiplication in Python with `mpi4py` on a 4-node cluster of the AMD-2 platform. Two communication scenarios were tested : collective (broadcast/gather) in Tables 3.7,3.8 and point-to-point in Tables 3.9, 3.10.

Observation : The tables present the energy consumed by the CPU package, each GPU (though inactive as `'numpy.matmul'` is CPU-bound), and RAM (EA2P estimation) as well as the execution time, for each node and for the total. For the point-to-point scenario, measurements on the passive nodes allow observing their consumption in idle mode.

Utility : EA2P shows its ability to collect and aggregate energy data from multiple nodes in an MPI application, thus providing a comprehensive view of distributed consumption, which is essential for analyzing HPC codes.

TABLE 3.7 – Measurements on AMD-2 with $2^{14} \times 2^{14}$ matrices (1)

Node	Pkg(J)	GPU 0 (J)	GPU 1 (J)	GPU 2 (J)	GPU 3 (J)	RAM (J)	Time (s)
0	1493.59	648.52	680.59	649.10	666.36	201.6	13.25
1	1419.99	714.36	633.82	625.94	670.95	193.2	12.54
2	1457.43	664.03	666.67	652.40	719.44	201.6	13.03
3	1472.65	667.21	653.88	659.93	689.50	201.0	13.11
Total	5843.66	2694.11	2634.95	2587.35	2746.25	596.4	51.93

3.3.3.5 Specific Case of Multi-GPU Configuration

The Table 3.11 illustrates a use case with TensorFlow for training VGG16 on the Stanford Dogs dataset, comparing a purely CPU execution with an execution where the GPUs are

TABLE 3.8 – Measurements on AMD-2 with $2^{15} \times 2^{15}$ matrices (2)

Node	Pkg(J)	GPU 0 (J)	GPU 1 (J)	GPU 2 (J)	GPU 3 (J)	RAM (J)	Time (s)
0	9109.88	3374.76	3530.46	3319.83	3425.45	1058.4	74.66
1	8777.08	3674.04	3255.40	3208.25	3430.79	999.6	70.65
2	9012.43	3353.60	3363.65	3291.97	3638.53	1041.6	73.26
3	9090.88	3408.30	3347.95	3435.98	3442.57	1058.4	74.43
Total	35990.27	13810.70	13497.44	13256.03	13937.32	4158.0	293.00

TABLE 3.9 – Measurements on AMD-2 with 10240×10240 matrices (1)

Node	Pkg(J)	GPU 0 (J)	GPU 1 (J)	GPU 2 (J)	GPU 3 (J)	RAM (J)	Time (s)
0	536.67	255.75	261.8	258.92	271.46	75.6	4.34
1	555.88	380.09	344.25	331.31	357.12	100.8	5.85
2	94.84	113.97	123.36	128.24	134.06	33.6	1.24
3	94.09	112.98	121.65	135.57	147.30	33.6	1.24
Total	1281.48	862.78	851.09	854.02	909.93	243.6	12.67

TABLE 3.10 – Measurements on AMD-2 with 20240×20240 matrices (2)

Node	Pkg(J)	GPU 0 (J)	GPU 1 (J)	GPU 2 (J)	GPU 3 (J)	RAM (J)	Time (s)
0	3604.08	1285.74	1374.71	1268.67	1314.77	403.2	28.09
1	3674.26	2200.34	1925.82	1892.41	2033.96	588.0	39.34
2	388.52	279.94	291.57	276.03	305.47	84.0	4.62
3	359.49	275.35	272.37	297.48	301.01	84.0	4.61
Total	8026.35	4041.37	3864.46	3734.58	3955.20	1159.2	76.66

(theoretically) available.

TABLE 3.11 – Multi GPU energy report on Intel-2 with Nvidia GPUs. pkgs and GPUs values are in "Watt-hour"

Apps	**pkgs**	**RAM**	**GPU 0**	**GPU 1**	**GPU 2**	**GPU 3**	**GPU 4**	**GPU 5**	**GPU 6**	**GPU 7**	**time(sec)**
Sleep	2.194	1.333	2.179	2.103	2.127	2.104	2.103	2.129	2.105	2.143	181.038
DOG-CPU	**28.528**	5.407	5.633	5.419	5.505	5.413	5.399	5.490	5.418	5.524	495.096
DOG-GPU	1.219	0.388	**2.519**	0.811	0.816	0.804	0.810	0.816	0.802	0.813	52.459

Observation : The CPU execution (GPUs disabled via CUDA_VISIBLE_DEVICES) consumes about 77 Wh in over 9 minutes. The same task on the same machine with 8 V100 GPUs available consumes only about 10 Wh in less than a minute. It is also noted that TensorFlow, by default, only effectively used one GPU (GPU 0 showing the highest consumption), with the others remaining underutilized.

Message : This strikingly illustrates the potential gain (in time and energy) of GPU acceleration, but also the importance of ensuring that the framework distributes the load well across all available GPUs to maximize efficiency, which EA2P allows one to verify.

3.3.4 Synthesis and Limitations of EA2P

The design, implementation, and experimental validation of EA2P, presented in the previous sections, position this tool as a significant contribution to the energy measurement ecosystem, particularly for applications developed in Python in the AI and HPC domains.

Synthesis of Capabilities EA2P presents itself as an operational framework for energy profiling, offering flexible support for the main devices (Intel/AMD CPUs, NVIDIA/AMD GPUs) and integrating an estimation model for main RAM consumption on platforms lacking direct sensors. Its Python API allows for fine-grained instrumentation of programs for detailed analysis, while its command-line mode permits profiling of entire binaries or Python scripts without code modification. The tool also handles common parallel paradigms (shared memory via Python's implicit multi-threading or OpenMP in binaries, and distributed memory via MPI), and its Docker containerization facilitates deployment and reproducibility of experiments.

Recognized Limitations and Perspectives Despite its advantages, EA2P also has limitations, some of which open perspectives for future work :

1. *Simplified RAM Model :* The current RAM energy estimation model is an approximation and does not account for all the complexities of the latest DDR technologies (e.g., PMICs on DDR5) or the non-linearity at very high memory capacity. Sophisticating this model is an avenue for improvement.
2. *Absolute Accuracy and Calibration :* As with any in-band tool, the absolute accuracy of the measurements depends on the reliability of the underlying hardware sensors/interfaces. Systematic calibration of EA2P against high-frequency out-of-band measurements on each tested platform would improve confidence in the absolute values.
3. *Potential Overhead :* Although deemed acceptable (cf. 3.2.4.2), the overhead induced by monitoring in Python can become significant for very fast-running code sections (on the order of milliseconds) with very high sampling frequencies.
4. *GPU Measurement via Instantaneous Power :* Using instantaneous power (via Nvidia-SMI/ROCm-SMI) to calculate GPU energy, while pragmatic, can miss very rapid variations and is dependent on the possible polling frequency of the CLI tools.

In conclusion to this section, EA2P represents a concrete and operational software contribution that meets an identified need for flexible and Python-oriented energy profiling. Its multi-component measurement features, including an original model for RAM energy, as well as its support for parallelism and ease of instrumentation, make it a valuable tool for the energy efficiency studies presented later in this thesis and for the broader research community. The identified limitations open clear paths for its continuous improvement.

3.4 Energy Benchmarking of Basic Computational Kernels

After presenting the EA2P measurement tool, this section focuses on another essential contribution of this thesis : establishing a reference energy profile for fundamental compute kernels on

different architectures and with different parallelism paradigms. This benchmarking process is crucial for understanding in detail how energy is consumed by the basic building blocks of HPC and AI applications, and for identifying the trade-offs between performance and consumption.

3.4.1 Motivation and Objectives of Energy Benchmarking

The rapid evolution of processor architectures (multicore, SIMD vectorization, massively parallel GPUs) offers immense computational capabilities but complicates the analysis of their energy efficiency. While traditional benchmarks often focus on execution time, a holistic approach integrating energy consumption has become indispensable, notably due to rising operational costs and environmental impact.

The motivations for this benchmarking study are multiple :

— **Understanding the Energy Profile of Parallel Paradigms :** To quantitatively evaluate how the main single-machine parallelism paradigms (SIMD vectorization, multi-threading on CPU, computation on GPU) influence not only performance but also energy consumption.

— **Characterizing Architectural Impact :** To compare the energy behavior of similar compute kernels on different CPU (e.g., Intel vs. AMD) and GPU architectures, in order to highlight hardware specifics.

— **Identifying Performance-Energy Trade-offs :** To not limit the analysis to optimizing a single metric, but to explicitly analyze the trade-offs between reducing execution time and the energy consumed.

— **Providing Baseline Data for Modeling and Optimization :** The energy profiles obtained for these elementary kernels can serve as a basis for building more complex consumption models (cf. Chapter 4) or for guiding optimization strategies at the application or system level (cf. Chapter 5).

The specific objectives of this benchmarking contribution are therefore to first conduct a comprehensive experimental evaluation of energy consumption for three single-node parallel computing paradigms : SIMD vectorization, multi-core/multi-threaded parallelism (OpenMP), and GPU acceleration (via Python and AI/HPC frameworks). Next, to use a set of representative compute kernels (compute-bound, memory-bound, mixed) to stress different aspects of the hardware. And finally, to analyze the time-energy trade-offs and provide practical insights on the energy efficiency of the different approaches based on the characteristics of the load and the platform.

3.4.2 Experimental Protocol : Kernels, Platforms, and Measurement Methodology

To achieve these objectives, a rigorous experimental protocol was established. Six compute kernels (benchmarks) were selected for their representativeness and their ability to stress different aspects of the systems (computation, memory access) :

— *GEMM (Generalized Matrix Multiplication) :* A classic linear algebra kernel, typically compute-bound, widely used in scientific and AI applications.

— *SPMV (Sparse Matrix-Vector Multiplication) :* Another linear algebra kernel, but working on sparse matrices, which induces more irregular memory accesses, often making it memory-bound or mixed.

— *TRIAD (Streaming Vector TRIAD) :* A standard benchmark for measuring memory bandwidth, consisting of the operation $A[i] = B[i] + scalar * C[i]$ on large vectors. Typically memory-bound.

— *Stencil (7-point 3D Stencil Computation) :* A common kernel in the modeling of physical phenomena (heat diffusion, fluid dynamics). Involves structured memory accesses to neighbors in a 3D grid, generally mixed (compute and memory-bound).

— *Monte Carlo (Pi Estimation) :* A kernel using stochastic simulations (random number generation, simple calculations). Often compute-bound, but can also be limited by the quality of the random number generator.

— *DIST (Generalized Euclidean Distance from Origin) :* Combines vectorized memory accesses and floating-point operations (distance calculations). Used in AI (clustering, k-NN). Mixed character.

Measurement Methodology and Tools :

The main performance metrics we have evaluated are *GFLOPS* (for floating point operations) and *GB/s* (for memory bandwidth). Regarding energy, CPU and GPU measurements are recorded separately using an appropriate tool for each platform following the testbed described in Table 3.12. Each experiment is repeated five times to ensure consistency, and the average is reported.“Chirop” and “Chuc” are platform names from the GRID5000 tesbed [Grid5000, 2025].

For CPU-based parallel benchmarking, we used the *LIKWID* tool suite [Treibig, 2010b] for code instrumentation and performance profiling. LIKWID offers a comprehensive set of features for monitoring performance counters, memory traffic, and energy consumption in modern processors, making it an ideal choice for benchmarking SIMD and multithreaded programs. Its MakerAPI feature allows to profile a specific section of the input program, thereby enabling fine-grained collection of energy-related data.

With a given code, we analyse three versions :

— **Scalar (Sequential)** : Baseline version, scalar and single-threaded.

— **Multithreaded (OpenMP)** : Multi-threaded version derived from the baseline using OpenMP (mainly on loops).

— **Vectorized (SIMD)** : SIMD version implemented from the baseline using SSE and AVX intrinsics.

We use gcc compiler with `-O0` flag so as to prevent any optimization that would hinder our observations. We use the `likwid-perfctr` module to capture performance counters and energy consumption at runtime. For memory traffic and cache behavior, we rely on event groups such as `MEM_SP` (on Intel) and `MEM1`, `MEM2` (on AMD). Energy consumption is recorded via the

TABLE 3.12 – Platform Characteristics (data taken from [Wikichip, 2021] and [CPU-world, 2023])

Name	Chirop	Chuc
CPU model	Intel Platinum 8358	AMD EPYC 7513
Clock speed	2.6 GHz	2.6 GHz
Turbo Speed	Up to 3.4 GHz	Up to 3.65 GHz
Physical Cores	32 (Threads : 64)	32 (Threads : 64)
L1 iCache	1,024KB 8-way set	1,024KB 8-way set
L1 dCache	1,536KB 12-way set	1,024KB 8-way set
L2 Cache	40MB 20-way set	16MB 8-way set
L3 Cache	48MB 12-way set	128MB 16-way set
DRAM Memory	512GB DDR4-3200	512GB DDR4-3200
CPU TDP	250W	200W
GPU Model	/	Nvidia A100 SXM4
GPU TDP	/	400W
GPU Memory	/	40GB HBM2
Data precision	Single	Single
SIMD extensions	SSE, AVX, AVX512	SSE, AVX
Operating System	Linux Debian 5.10.209	Linux Debian 5.10.209

`ENERGY` group, providing detailed power metrics for the CPU cores. Thread affinity is controlled using `likwid-pin`, ensuring that threads are properly pinned to the CPU-cores in a one-to-one binding.

3.4.3 GPU Benchmarking with EA2P

For GPU benchmarking we used Python-based implementations of the same kernels plus two additional kernels (*3D stencil* and *Monte Carlo pi estimation*). For energy measurement, we used EA2P (*Energy-Aware Application Profiler*) [Nana, 2024c], an energy profiler tool that can handle GPU-accelerated applications (unlike likwid). Considering Python codes allow to leverage popular GPU-accelerated frameworks such as `CuPy`, `JAX`, `TensorFlow`, and `PyTorch`.

EA2P is used to gather both energy and running time of GPU code blocks at desired granularity. We compute performance metrics like GFLOPS and GB/s by dividing the data size and the FLOPs by the measured running time (we roughly assume that each data is accessed once).

3.4.4 Results and Analysis of Energy Benchmarks

This section presents and analyzes the experimental results obtained by applying the previously described benchmarking protocol. We will first examine the idle consumption to establish a baseline, then analyze the energy efficiency of the SIMD and multi-core parallelism paradigms on CPU, and finally, study the behavior of the kernels on GPU via different Python frameworks. For all analyses, E_app (energy due to application run) refers to the measured energy from which the idle energy for the same duration has been subtracted, in accordance with equation Eq. 3.2.

$$\begin{cases} E_{idle}(t) = \alpha + \beta \times t \\ E_{application} = E_{measured} - E_{idle}(t_{application}) \end{cases} \tag{3.2}$$

3.4.4.1 Idle State Energy Consumption

Before analyzing compute-intensive kernels, it is essential to characterize the system's baseline energy consumption when no significant workload is running. In this context, the *system* is considered *idle*, meaning that only the operating system services are active while user-level processes remain inactive.

To emulate inactivity, we employ a simple `sleep` program in C. When a process or thread is in a `sleep` state, it does not receive any CPU time until the operating system scheduler wakes it up after the specified duration. Consequently, even if hundreds of threads are created, as long as they remain in `sleep`, they do not consume CPU cycles and do not prevent the system from scheduling other workloads. Thus, the presence of multiple sleeping threads does not, by itself, imply a busy CPU ; the system can still be regarded as *idle* from a computational perspective. It should be noted that the CPU itself transitions into one of the ACPI low-power states when idle, but the detailed analysis of these states is deferred to a later section.

Single-Core Idle Consumption :

Table 3.13 reports the power and energy consumption of the Intel ("Chirop") and AMD ("Chuc") platforms for various sleep durations (from 1s to 64s) with a single core executing the `sleep` process.

TABLE 3.13 – Power-energy of sleep test for Idle power consumption for single core usage

runtime [s]	**1**	**2**	**4**	**8**	**16**	**32**	**64**
AMD							
power PKG [W]	57.73	55.60	55.47	55.62	56.52	56.78	56.02
energy PKG [J]	57.74	111.22	221.89	444.98	904.45	1817.08	3585.92
Intel							
power PKG [W]	52.22	49.11	49.68	49.07	48.48	47.64	48.10
energy PKG [J]	52.22	98.22	198.71	392.56	775.74	1524.49	3078.67
power RAM [W]	9.97	9.69	9.64	9.60	9.41	9.42	9.43
energy RAM [J]	9.97	19.38	38.55	76.78	150.53	301.36	603.49

Observations : The Intel Xeon Platinum 8358 CPU consumes about 50W on average at idle, which represents about 20% of its nominal TDP (250W). The associated RAM (Intel platform) additionally consumes about 9-10W at idle. The AMD EPYC 7513 CPU consumes about 56W at idle, or about 28% of its TDP (200W). (RAM consumption for AMD is not directly measured by RAPL on this platform with the tools used). The results confirm a strongly linear relationship between the energy consumed at rest and time, with coefficients of determination (R^2) very close to 1.00 for the linear regression models (Eq. 3.3) :

$$\begin{cases} E_{idle-CPU-Intel}(t) = 4.44 + 47.95 \times t \\ E_{idle-RAM-Intel}(t) = 0.63 + 9.41 \times t \\ E_{idle-CPU-AMD}(t) = 1.88 + 56.14 \times t \end{cases} \tag{3.3}$$

With RMSE = 6.36 and $R^2 = 1.00$ for Intel CPU model ; RMSE = 0.49 and $R^2 = 1.00$ for Intel RAM ; and RMSE = 8.65 and $R^2 = 0.9999$ for AMD model

This linearity validates the approach of subtracting idle energy to estimate the application energy E_app.

Multi-Core Idle Power Consumption with OpenMP :

The same experiment was repeated with a variable number of OpenMP threads (from *'seq'* for sequential execution up to 32 threads) to verify whether the allocation of multiple sleeping threads differs in behavior from the sequential idle case. While no significant increase in power consumption was expected (since sleeping threads should not consume CPU cycles) the goal was to confirm this experimentally and to check whether OpenMP's internal thread management introduces any measurable energy overhead. The Table 3.14 details these results.

The outcome of this test helps validate that thread creation alone, without active computation, does not prevent the CPU from entering energy-saving states, ensuring a reliable baseline for subsequent measurements.

Table 3.14 – Idle power consumption with multiple cores

Sleep[s]	#threads	AMD		Intel			
		Energy PKG [J]	Power PKG [W]	Energy PKG [J]	Power PKG [W]	Energy DRAM [J]	Power DRAM [W]
1	seq	57.74	57.73	52.22	52.22	9.97	9.97
	2	56.21	56.20	53.27	53.26	9.71	9.71
	4	59.24	59.23	49.41	49.41	9.78	9.78
	8	60.94	60.92	49.71	49.71	9.38	9.38
	16	59.25	59.24	50.98	50.97	9.63	9.63
	32	62.55	62.54	57.17	57.16	9.72	9.72
2	seq	111.22	55.60	98.22	49.11	19.38	9.69
	2	113.62	56.80	100.36	50.18	19.24	9.62
	4	111.83	55.90	93.00	46.50	18.87	9.43
	8	112.10	56.05	99.75	49.87	19.11	9.55
	16	116.76	58.37	103.62	51.81	19.32	9.66
	32	118.36	59.17	101.09	50.54	19.51	9.75
4	seq	221.89	55.47	198.71	49.68	38.55	9.64
	2	228.46	57.11	190.24	47.56	38.13	9.53
	4	224.67	56.17	195.04	48.76	38.64	9.66
	8	227.36	56.84	198.30	49.57	37.73	9.43
	16	225.97	56.49	195.69	48.92	39.09	9.77
	32	231.96	57.98	218.63	54.66	40.70	10.18
8	seq	444.98	55.62	392.56	49.07	76.78	9.60
	2	452.97	56.62	376.29	47.03	76.84	9.60
	4	451.19	56.39	392.11	49.01	77.48	9.68
	8	461.09	57.63	387.63	48.45	77.23	9.65
	16	461.28	57.65	392.26	49.03	78.31	9.79
	32	453.65	56.70	399.56	49.94	77.45	9.68

Observations : Increasing the number of OpenMP threads (even inactive ones) has a marginal impact on the average power consumed at idle. Although slight fluctuations are observed, the power remains globally stable and close to the single-core idle consumption. The linear regression models for idle energy as a function of time remain very accurate (Eq. 3.4, With

RMSE = 6.23 and $R^2 = 0.9977$ for Intel CPU model ; RMSE = 0.5599 and $R^2 = 0.9995$ for Intel RAM ; and RMSE = 3.75 and $R^2 = 0.9994$ for AMD model). This confirms that the method of subtracting E_idle remains valid even in a multi-threaded context for our benchmarks.

$$\begin{cases} E_{idle-CPU-Intel}(t) = 3.87 + 48.36 \times t \\ E_{idle-RAM-Intel}(t) = -0.0003 + 9.67 \times t \\ E_{idle-CPU-AMD}(t) = 1.67 + 56.50 \times t \end{cases} \tag{3.4}$$

These characterizations of the idle state are essential for correctly interpreting the energy measurements of active kernels.

3.4.4.2 Analysis of SIMD Vectorization

The impact of SIMD (Single Instruction, Multiple Data) vectorization was evaluated by comparing scalar (sequential), SSE, and AVX versions of the compute kernels.

The Table 3.15 (First part) presents the execution time, total PKG energy (CPU package), application energy E_app PKG, and speedups (acc.) in time and energy for SPMV_COO, TRIAD, GEMM, and DIST. Vectorization (especially AVX) brings notable performance gains : up to ×2.25 for TRIAD (AVX) and ×4.30 for GEMM (AVX) compared to the scalar version. SPMV shows more modest gains (×1.26 for AVX) due to its irregular memory accesses. For DIST, AVX is ×2.03 faster. Gains in application energy generally follow those in time : up to ×2.01 for TRIAD (AVX) and ×4.24 for GEMM (AVX). A strong correlation is observed. The reduction in execution time thanks to vectorization directly translates into a reduction in consumed energy, without a significant increase in instantaneous power.

Table 3.15 – Combined time and energy measurements of SIMD on AMD and INTEL

Code ver.	AMD						INTEL									
	time		E [J]		E acc.		time		E [J]		E acc.		E RAM [J]		RAM acc.	
	[s]	acc.	PKG	app	PKG	app	[s]	acc.	PKG	app.	PKG	app.	tot.	app.	tot.	app.
SPMV_COO																
seq	14.26	1.00	1108.50	328.50	1.00	1.00	9.91	1.00	991.37	476.01	1.00	1.00	164.17	70.02	1.00	1.00
sse	15.22	0.94	1157.10	307.77	0.96	1.07	10.74	0.92	1070.30	511.83	0.93	0.93	175.74	73.72	0.93	0.95
avx	11.32	1.26	897.72	259.14	1.23	1.27	9.17	1.08	920.92	443.84	1.08	1.08	153.05	65.89	1.07	1.06
TRIAD																
seq	7.17	1.00	547.18	144.32	1.00	1.00	5.43	1.00	544.94	273.55	1.00	1.00	85.12	33.56	1.00	1.00
sse	5.84	1.23	433.66	106.94	1.26	1.35	3.38	1.61	342.35	173.20	1.59	1.58	55.58	23.44	1.53	1.43
avx	3.19	2.25	251.31	71.77	2.18	2.01	2.30	2.36	237.01	122.15	2.30	2.24	39.79	17.97	2.14	1.87
GEMM																
seq	124.83	1.00	9160.88	2201.11	1.00	1.00	116.56	1.00	11943.84	6116.03	1.00	1.00	1606.94	499.66	1.00	1.00
sse	46.95	2.66	3469.80	841.02	2.64	2.62	56.08	2.08	5662.58	2858.58	2.11	2.14	774.71	241.95	2.07	2.07
avx	29.02	4.30	2151.76	518.80	4.26	4.24	29.14	4.00	2920.22	1463.07	4.09	4.18	401.29	124.44	4.00	4.02
DIST																
seq	3.44	1.00	289.16	92.74	1.00	1.00	6.45	1.00	663.17	340.64	1.00	1.00	94.83	33.55	1.00	1.00
sse	2.96	1.16	268.70	100.46	1.08	0.92	4.52	1.43	453.75	227.94	1.46	1.49	66.98	24.08	1.42	1.39
avx	1.70	2.03	161.99	64.63	1.79	1.43	2.49	2.59	253.93	129.32	2.61	2.63	38.16	14.49	2.48	2.32

The DIST kernel shows a peculiarity : the energy efficiency (E acc.) with AVX (×1.43) is less significant than the time speedup (×2.03). This could be due to an increased sensitivity to the power increase of SIMD units on this AMD Zen 3 architecture for this type of mixed computation [Nana, 2024b].

The Table 3.15 (Second part) shows similar results for the Intel Xeon platform. The columns also include RAM energy. Gains are also significant, for example, for GEMM (AVX), the speedup is ×4.00 in time and ×4.18 in E_app PKG energy. For DIST (AVX), it is ×2.59 in time and ×2.63 in E_app PKG energy. The energy consumed by the RAM also decreases with the reduction in execution time, reflecting its correlation with the duration of activity. For GEMM (AVX), the RAM speedup is ×4.02, and for DIST (AVX), ×2.32.

Behavioral Stability : Unlike the observation on AMD for DIST, on Intel, the energy efficiency is very close to the time speedup for all kernels.

SIMD Synthesis : SIMD vectorization is an effective technique for improving both the performance and the energy efficiency of compute kernels that lend themselves to it (regular calculations on data). The reduction in execution time due to better exploitation of instruction-level parallelism directly translates into a decrease in consumed energy, without the instantaneous power increasing in a penalizing way. The exact impact can vary slightly depending on the CPU micro-architecture and the nature of the kernel.

3.4.4.3 Analysis of Multi-Core Parallelism OpenMP

The impact of multithreading on the speedup and power consumption is studied for each kernel.

The Table 3.16 (AMD section) and the Figure 3.5 (axes : # threads, y1 : Volume/Bandwidth, y2 : PKG Power) present the times, energies, and speedups. Increasing the number of threads significantly reduces execution time, with significant maximum speedups (e.g., ×17.49 for GEMM with 32 threads, ×10.60 for TRIAD). Efficiency (speedup / #_threads) logically decreases with a large number of threads due to overheads (synchronization, memory contention, Amdahl's limits). The E_app PKG energy also decreases with the number of threads, but the energy efficiency is systematically lower than the time speedup. For example, for GEMM with 32 threads, the energy efficiency is ×3.01 (compared to ×17.49 in time). For SPMV (32 threads), ×2.67 in energy versus ×8.72 in time. The average CPU package (PKG) power increases with the number of active threads, going from about 80-100W with 2 threads to 130-170W with 32 threads depending on the kernel (see 3.5c). It also increases with the number of threads, especially for memory-bound kernels like TRIAD and SPMV (see Fig. 3.5b).

Efficiency Breakpoint : in general, energy efficiency (energy speedup) tends to follow time speedup well up to 8 cores. Beyond that (16, 32 cores), although time continues to decrease, the increase in power and parallelism overheads mean that the marginal energy gain decreases, or even stagnates. The gap between time speedup and energy efficiency widens.

The Table 3.16 (Intel section) and the Figure 3.4 (same axes as Fig. 3.5) show similar trends. For GEMM with 32 threads, the time speedup is ×20.97 and the E_app PKG energy efficiency is ×5.81. For SPMV (32 threads), ×14.03 in time and ×4.14 in energy. The E_app RAM energy also shows a reduction with the number of threads, following the reduction in time (e.g., RAM speedup of ×20.40 for GEMM 32 threads). This is because the RAM power (Fig. 3.4d)

TABLE 3.16 – Combined multithread time and energy measurements on AMD and Intel

#Th	AMD						INTEL									
	time		E [J]		acc.		time		E [J]		acc.		E RAM [J]		RAM acc.	
	[s]	acc.	PKG	app	PKG	app	[s]	acc.	PKG	app.	PKG	app.	tot.	app.	tot.	app.
SPMV_COO																
seq	14.26	1.00	1108.50	328.50	1.00	1.00	9.91	1.00	991.37	476.01	1.00	1.00	164.17	70.02	1.00	1.00
2	5.78	2.47	544.11	214.29	2.04	1.53	5.63	1.76	597.85	305.27	1.66	1.57	105.59	52.14	1.55	1.34
4	3.22	4.43	338.55	157.49	3.27	2.09	2.96	3.35	349.97	196.12	2.83	2.44	66.35	38.24	2.47	1.83
8	3.07	4.64	327.58	156.87	3.38	2.09	1.66	5.97	230.20	143.83	4.31	3.32	46.37	30.59	3.54	2.29
16	1.87	7.65	227.27	122.29	4.88	2.69	0.99	9.98	175.81	124.16	5.64	3.85	34.74	24.81	4.73	2.82
32	1.63	8.72	229.85	122.81	4.82	2.67	0.71	14.03	159.36	115.57	6.22	4.14	29.40	22.33	5.58	3.14
TRIAD																
seq	7.17	1.00	547.18	144.32	1.00	1.00	5.43	1.00	544.94	273.55	1.00	1.00	85.12	33.56	1.00	1.00
2	3.87	1.85	334.74	114.63	1.63	1.26	2.94	1.85	313.14	166.28	1.74	1.65	49.97	22.06	1.70	1.52
4	2.13	3.36	215.68	97.38	2.54	1.48	1.50	3.63	185.50	110.67	2.94	2.47	29.80	15.58	2.86	2.15
8	1.82	3.93	200.93	97.04	2.72	1.49	0.80	6.80	120.49	80.54	4.52	3.40	19.48	11.49	4.37	2.92
16	0.99	7.26	135.64	74.24	4.03	1.94	0.49	11.03	95.21	69.64	5.72	3.93	15.05	10.13	5.65	3.31
32	0.68	10.60	110.45	71.09	4.95	2.03	0.48	11.23	96.20	66.22	5.66	4.13	14.63	9.79	5.82	3.43
GEMM																
seq	124.83	1.00	9160.88	2201.11	1.00	1.00	116.56	1.00	11943.84	6116.03	1.00	1.00	1606.94	499.66	1.00	1.00
2	64.15	1.95	5248.30	1658.71	1.75	1.33	56.84	2.05	6162.93	3320.87	1.94	1.84	786.53	246.54	2.04	2.03
4	32.22	3.87	3039.53	1232.53	3.01	1.79	34.26	3.40	4092.08	2378.91	2.92	2.57	473.73	148.22	3.39	3.37
8	25.44	4.91	2468.35	1049.06	3.71	2.10	17.14	6.80	2471.08	1614.09	4.83	3.79	236.93	74.10	6.78	6.74
16	12.98	9.62	1562.17	849.76	5.86	2.59	8.61	13.54	1670.17	1239.89	7.15	4.93	119.88	38.12	13.41	13.11
32	7.14	17.49	1136.23	732.24	8.06	3.01	5.56	20.97	1330.47	1052.57	8.98	5.81	77.29	24.49	20.79	20.40
DIST																
seq	3.44	1.00	289.16	92.74	1.00	1.00	6.45	1.00	663.17	340.64	1.00	1.00	94.83	33.55	1.00	1.00
2	2.03	1.70	177.53	75.66	1.63	1.23	3.34	1.93	370.88	203.91	1.79	1.67	51.59	19.86	1.84	1.69
4	1.33	2.59	142.91	69.05	2.02	1.34	1.69	3.82	219.11	134.72	3.03	2.53	28.37	12.33	3.34	2.72
8	1.15	2.99	136.36	66.27	2.12	1.40	0.88	7.34	144.69	100.76	4.58	3.38	16.51	7.72	5.75	4.35
16	0.90	3.81	118.28	62.61	2.44	1.48	0.50	12.86	111.27	85.20	5.96	4.00	10.99	5.98	8.63	5.61
32	0.98	3.50	132.67	77.22	2.18	1.20	0.41	15.72	95.19	69.80	6.97	4.88	10.14	6.03	9.36	5.56

increases moderately with the number of active threads (from $\approx 15W$ to $\approx 30-40W$), but this increase is offset by the strong reduction in activity time. The PKG Power (Fig. 3.4c) increases significantly, from $\approx 100W$ (2 threads) to over 200W (even close to 240W for DIST with 32 threads) for intensive kernels.

The volume of data transferred via RAM decreases with the number of threads for GEMM and SPMV (better locality, less external traffic per core ?), while the bandwidth tends to increase, especially for TRIAD (see Fig. 3.4a, 3.4b).

Multi-Core Synthesis :

Using multi-core parallelism is very effective for reducing execution time. It also leads to a reduction in the energy consumed by the task, as the decrease in time outweighs the increase in instantaneous power. However, there is a significant energy overhead associated with managing parallelism (*synchronization, cache coherency, contention, OS*), which means that energy efficiency does not scale as well as time performance, especially at a high core count. Optimizing memory consumption (reducing transfers, optimizing accesses) has a double benefit : reduction of time and energy.

3.4.4.4 Analysis of GPU-Accelerated Computing (Python Frameworks)

The GPU study was conducted on the AMD platform (EPYC + NVIDIA A100) using EA2P and four Python frameworks : JAX, PyTorch (Torch), TensorFlow (TFlow), and CuPy. The GEMM, SPMV, TRIAD, Stencil, Monte Carlo, and DIST kernels were tested.

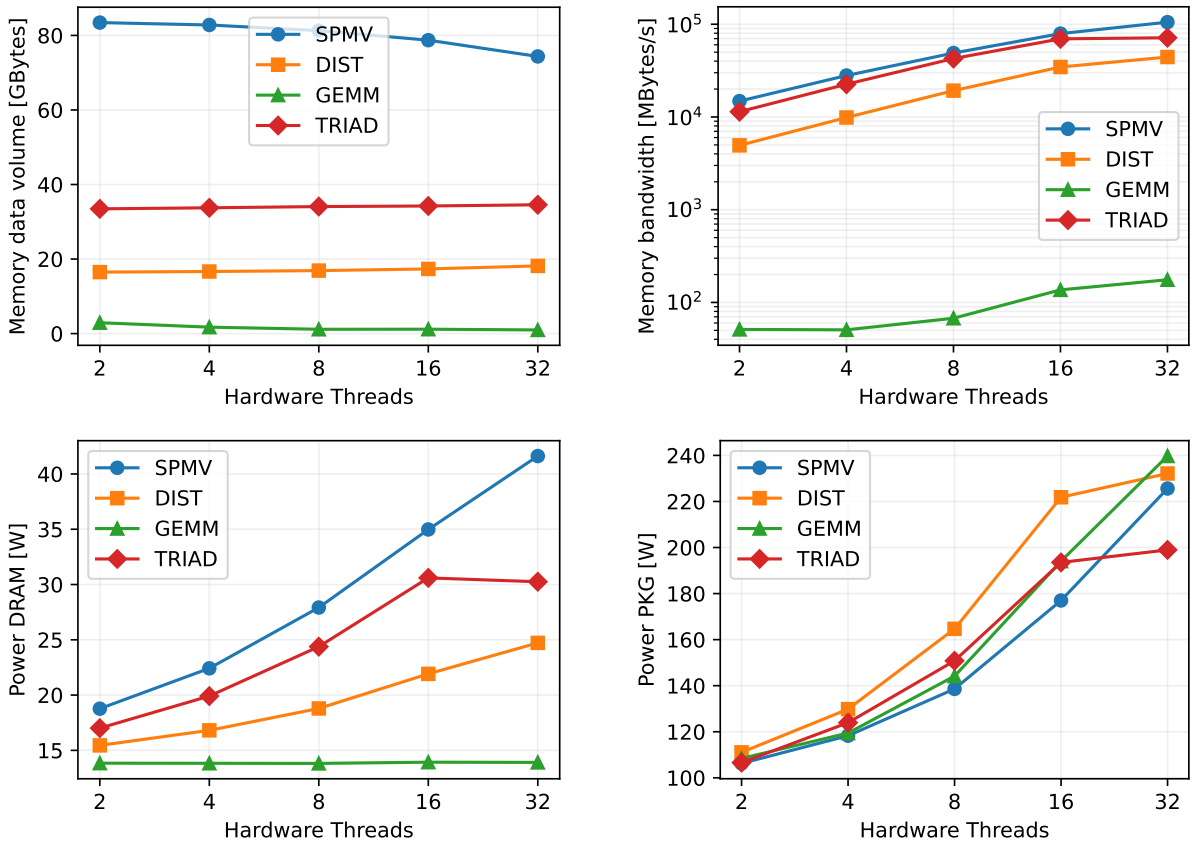


FIGURE 3.4 – Memory performance and power consumption with an INTEL multicore

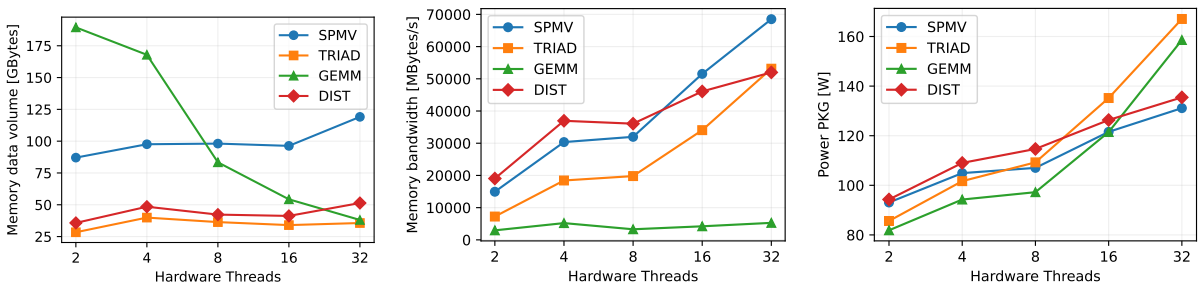


FIGURE 3.5 – Memory performance and power consumption with an AMD multicore

The Table 3.17 reports the execution time, GPU energy, and host CPU energy (while the GPU works), as well as the average GPU and CPU power for each kernel/framework combination. The Figure 3.6 offers a visualization of these metrics (subfigures (a) Bandwidth GPU, (b) GFLOPS GPU, (c) Avg GPU Power, (d) Avg CPU (host) Power).

General Observations on Performance and Energy :

High Variability between Frameworks : For the same kernel, performance (time, GFLOPS) and energy consumption can vary considerably from one framework to another. For example, for SPMV, JAX is about 6 times faster and consumes 10 times less GPU energy than PyTorch. For GEMM, TensorFlow and JAX dominate in GFLOPS (and GFLOPS/W), likely due to the use of the A100's Tensor Cores for TF32, while PyTorch and CuPy show performance (and efficiency)

Table 3.17 – Benchmark results from the frameworks standpoint

			Energy(J)		Power(W)	
Ben	Bench	Time (s)	GPU	CPU	GPU	CPU
JAX	triad	22.707	1019.163	1333.950	44.883	58.746
	gemm	2.868	1124.698	200.235	392.185	69.823
	dist	16.405	4199.758	1189.615	256.004	72.515
	stencil	20.468	1139.352	1238.476	55.665	60.508
	spmv	4.507	496.708	328.250	110.197	72.824
	monte_carlo	5.761	1409.745	428.575	244.721	74.397
TORCH	triad	23.729	913.214	1354.176	38.485	57.068
	gemm	22.346	7832.080	1621.745	350.498	72.576
	dist	6.460	1477.742	458.006	228.736	70.894
	stencil	9.561	2304.111	685.952	240.980	71.742
	spmv	26.672	5853.350	1928.214	219.460	72.295
	monte_carlo	6.983	1194.045	486.140	171.004	69.622
TF	triad	3.496	828.763	245.760	237.071	70.300
	gemm	3.964	1426.650	298.625	359.942	75.343
	dist	6.292	1511.220	450.808	240.175	71.646
	stencil	12.418	2942.685	904.744	236.968	72.857
	spmv	3.416	632.714	246.714	185.196	72.213
	monte_carlo	7.890	2084.313	572.970	264.187	72.624
CUPY	triad	3.477	983.167	246.026	282.737	70.752
	gemm	23.337	8309.033	1685.315	356.049	72.217
	dist	6.191	1909.106	445.360	308.350	71.933
	stencil	8.585	2331.348	621.098	271.563	72.348
	spmv	23.529	3263.848	1712.680	138.718	72.791
	monte_carlo	5.725	1777.493	414.570	310.486	72.416

closer to that expected with standard FP32 CUDA cores.

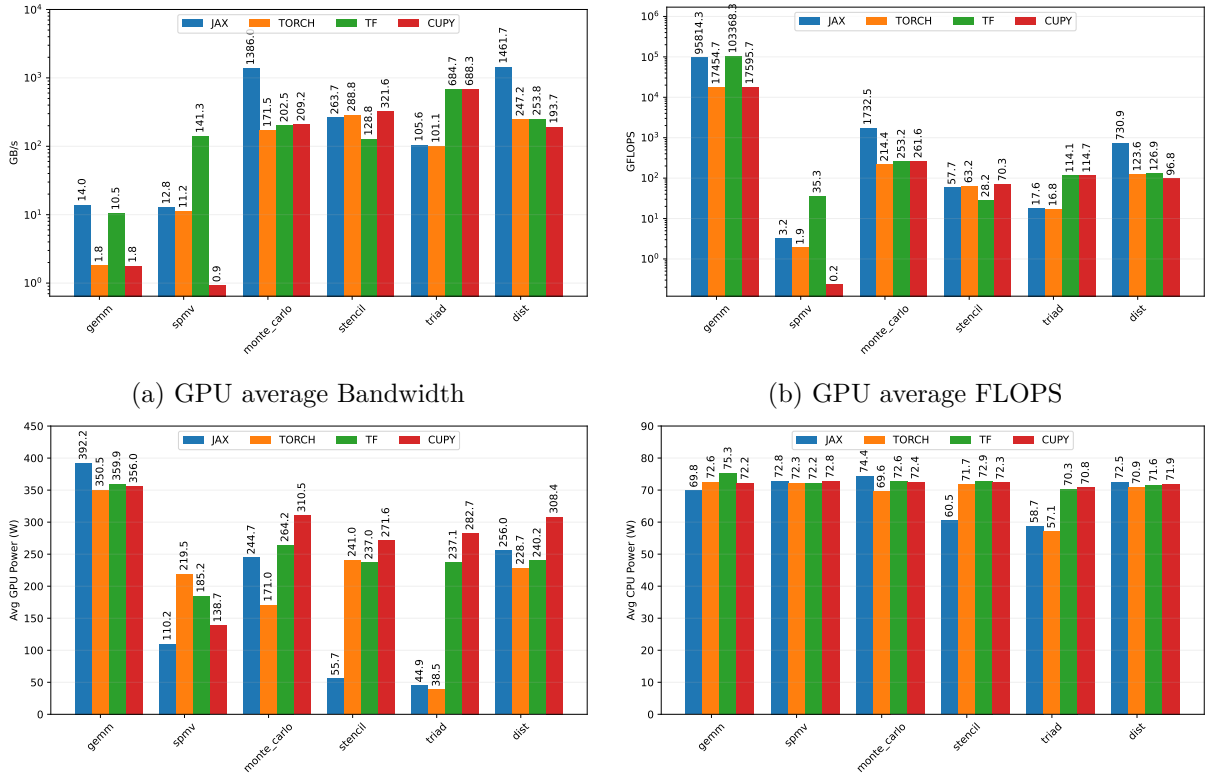
Impact of Kernel Type :

— **Compute-bound (GEMM, Monte Carlo, DIST) :** JAX often shows the best efficiency (GFLOPS/W). TensorFlow is also very high-performing on GEMM.

— **Memory-bound (TRIAD, Stencil) :** For TRIAD, TensorFlow and CuPy are the most performant and consume the most GPU power (they "push" the memory subsystem). JAX and PyTorch are less performant and consume significantly less on this kernel. For Stencil (3D), JAX is the most efficient in GFLOPS/W and consumes little GPU power (55W), suggesting good optimization of complex memory accesses.

Host CPU Consumption : While the GPU works, the host CPU's power remains relatively stable and low (around 60-75W), but its cumulative energy over the GPU's execution duration can become significant if the GPU is slow (e.g., the case of TRIAD/JAX or SPMV/Torch where CPU energy can exceed GPU energy). The instantaneous GPU power varies enormously (from 38W for TRIAD/Torch to 392W for GEMM/JAX), indicating that the frameworks and kernels do not use the GPU in the same way or with the same intensity.

Specific TRIAD behaviour : TensorFlow and CuPy are the most performant on TRIAD



(a) GPU average Bandwidth

(b) GPU average FLOPS

(c) GPU average Power

(d) CPU package average Power

FIGURE 3.6 – GFLOPS and GB/s vs Power all benchmaks and frameworks

(memory-bound), consuming 237W and 283W of GPU power respectively, indicating a high stress on the memory subsystem. JAX and PyTorch are far behind in performance and consume only 39-45W on the GPU, suggesting underutilization.

GPU Synthesis : The choice of software framework has a major impact on the performance and energy efficiency of GPU execution, sometimes more significant than the intrinsic characteristics of the kernel itself. Some frameworks (like TensorFlow and JAX for GEMM) clearly take advantage of specialized hardware units (Tensor Cores). For memory-bound kernels, the differences in transfer and memory management optimization between frameworks are glaring. There is no universally "best" framework ; the choice depends on the nature of the application. Fine-grained measurement is therefore essential to identify the most eco-efficient implementation.

3.4.5 Synthesis and Discussion of Benchmarking Results

All of these benchmarking campaigns provide several key takeaways :

Time-Energy Correlation : A fundamental observation is the strong correlation between execution time and consumed energy, especially at the CPU level. Reducing execution time (via vectorization, efficient multi-threading) is often the first and most effective strategy for reducing energy. However, this correlation is not always linear, especially with high-granularity multi-threading where power overheads can become dominant.

Impact of Parallelism Paradigm : SIMD offers an excellent time/energy gain for suitable codes, with little power overhead. Multi-core enables significant reductions in time and energy, but the marginal efficiency decreases with an increasing number of cores due to power and parallelism management overheads. The GPU has the potential for massive gains for massively parallel tasks, but the real efficiency depends heavily on the software framework and the suitability of the kernel to the GPU architecture.

Importance of Workload Characteristics : Compute-bound kernels (GEMM, Monte Carlo) and memory-bound kernels (TRIAD, SPMV in some cases) react differently to parallelization techniques and architectures. Optimizations must be adapted to the nature of the load.

Architectural Specifics : Although the general trends are similar, differences in behavior between Intel and AMD CPU architectures were noted (e.g., sensitivity of DIST to SIMD on AMD, management of memory power on Intel).

Crucial Role of the GPU Framework : The choice of Python framework (*JAX, TensorFlow, PyTorch, CuPy*) can have an impact as significant as, or even greater than, the optimization of the kernel itself or the choice of the underlying hardware on energy efficiency, highlighting the importance of the complete software stack.

In conclusion, this fundamental benchmarking work provides quantitative data and essential insights into how different aspects of parallelism and different hardware/software architectures influence energy consumption. This knowledge is indispensable for informing modeling (Chapter 4) and guiding the development of targeted optimization strategies (Chapter 5). It also highlights the need for tools like EA2P to navigate this complexity and make informed choices.

3.5 In-depth Benchmarking of System Power Management Techniques

Building on the fine-grained measurement capabilities enabled by the EA2P tool (Chapter 3.2) and the characterization of compute kernels (Chapter 3.3), this section presents a rigorous and detailed experimental analysis of the impact and effectiveness of three commonly available system-level power management techniques : *Dynamic Voltage and Frequency Scaling (DVFS), CPU ACPI P-State governor modes*, and *Power Capping*. The objective is to dissect their influence on the performance-energy trade-off for various compute kernels running on heterogeneous hardware architectures and with distinct software frameworks, in order to extract actionable knowledge for contextual energy optimization.

3.5.1 Motivation and Objectives

Managing energy consumption at the system level is a necessity, but the interactions between the available techniques *(DVFS, Power Capping, ACPI governors*), the workload characteristics (*compute-bound, memory-bound, access patterns*, etc.), the micro-architectural specifics of the processors (*Intel Xeon Ice Lake-SP, AMD EPYC Zen 3, NVIDIA A100*), and the behavior

of execution frameworks (TensorFlow and JAX) are complex and not always predictable. The objective is to provide an empirical characterization to better understand these interactions and to quantify the potential gains in efficiency (via EDP) as well as the performance penalties.

3.5.2 Technical Foundations and Experimental Protocol

3.5.2.1 Targeted Power Management Mechanisms

The three techniques explored are :

ACPI/P-State governor scaling (CPU only) :

This technique relies on the Advanced Configuration and Power Interface (ACPI), which operating systems use to interact with the hardware's power management capabilities, notably the CPU P-States (Performance States). In Linux, *"governors"* implement different policies for selecting P-States (and thus frequency/voltage). The following modes were tested (when available and relevant for the platform) :

— `'performance'` : Fixes the CPU frequency to its maximum.

— `'powersave'` : Fixes the CPU frequency to its minimum.

— `'ondemand'` : Dynamically adjusts the frequency based on system load, favoring a rapid ramp-up in frequency.

— `'conservative'` : Similar to `'ondemand'`, but with a more gradual frequency ramp-up.

— `'schedutil'` : Integrated with the Linux kernel scheduler, it uses task load information to proactively adjust frequency. It is often the default governor on recent distributions.

— `'userspace'` : Allows manually setting the CPU frequency (min and max) via sysfs interfaces. This mode was used to emulate static or bounded DVFS policies.

The command `'sudo cpupower frequency-set -g [governor_name]'` (for Intel) or direct writes to the appropriate sysfs files were used to change modes (`'echo '[governor_name]' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor'` for AMD, after checking the available governors in `'scaling_available_governors'`).

Power limitation / capping (Intel CPU and NVIDIA GPU) :

This technique constrains the maximum electrical power consumption of the component.

— Intel CPU : Utilizes the RAPL interface via the sysfs file system exposed by the driver (under `/sys/class/powercap/intel-rapl/intel-rapl:X/constraint_0_power_limit_uw`). By writing a value in microwatts into `'constraint_0_power_limit_uw'`, the *"cap"* for the CPU package is set (e.g., `'intel-rapl:0'` for socket 0).

— NVIDIA GPU : Uses the command-line tool `'sudo nvidia-smi -pl [limit_in_watts]'` to set the power limit of the GPU card. It is worth noting that `'nvidia-smi -pm 1'` (persistence mode) is enabled to ensure GPU settings are maintained.

Frequency limitation / manual DVFS (CPU and GPU) :

This technique consists of explicitly setting the frequency bounds or the operating frequency.

— Intel and AMD CPU : Uses `'sudo cpupower frequency-set -max [frequency_MHz]'` and/or `'-min [frequency_MHz]'` to bound the frequency range allowed for the CPU cores. Available frequencies are listed via `'cpupower frequency-info'`.

— NVIDIA GPU : Uses `'sudo nvidia-smi -lgc [min_freq_MHz],[max_freq_MHz]'` to lock the GPU core frequency (and potentially `'sudo nvidia-smi -lmc [mem_freq_MHz]'` for the GPU memory).

3.5.2.2 Platforms, Kernels, and Frameworks

The platforms (*dual-socket Intel Xeon Platinum 8358; single-socket AMD EPYC 7513 + NVIDIA A100 SXM4 40GB* detailed in Table 3.12) and the six compute kernels (*GEMM, Stencil, SpMV, Triad, Dist, Monte Carlo*) are identical to those in Section 3.3.2. GPU execution is performed via Python 3.x with TensorFlow 2.12.0 and JAX 0.4.31, and CUDA 12.2.

Frameworks Configuration (Implications) :

JAX : The environment variables `'XLA_PYTHON_CLIENT_PREALLOCATE=false'`, with `'XLA_PYTHON_CLIENT_MEM_FRACTION=.10'`, and `'XLA_PYTHON_CLIENT_ALLOCATOR=platform'` were used for JAX benchmarks on the GPU. This choice was made to avoid **Out-Of-Memory (OOM)** errors encountered with default settings on the NVIDIA A100 for large data sizes, especially at low GPU frequencies where XLA's memory management could become less efficient. Using a low memory fraction (.10) and the `'platform'` allocator (more direct) aims to stabilize execution. However, this may lead to *"HLO rematerialization"* warnings from XLA, indicating that JAX/XLA is making trade-offs by re-computing some values to save memory, which can have a performance cost.

TensorFlow : For TF on the GPU, the `'TF_GPU_ALLOCATOR='cuda_malloc_async' '` environment variable was used to enable CUDA's asynchronous memory allocator. This allocator is generally more performant (faster, less fragmentation) than the default BFC (Best-Fit with Coalescing) allocator for many workloads. TensorRT was not used (confirmed by runtime warnings), meaning that TensorRT-specific graph optimizations (aggressive operator fusion, quantization) were not active.

3.5.2.3 Set of Evaluated Configurations

The Table 3.18 exhaustively lists the combinations tested for each platform and technique. For example, for GPU DVFS on the A100, 7 GPU frequency levels were tested (ranging from 210 MHz to 1410 MHz for the graphics cores, with a fixed memory frequency of 1215 MHz). For Power Capping on the A100, 7 thresholds ranging from 100W to 400W (nominal TDP) were tested.

3.5.2.4 Measurements and Metrics

The EA2P tool was used for all measurements of energy (*CPU Package, GPU, DRAM/Host CPU* if accessible by EA2P) and execution time. Each combination (*platform, kernel, framework, management technique, setting*) was run five times. The means and sometimes standard

TABLE 3.18 – Combined DVFS, ACPI P-States, and Power Cap Settings

Setting num	Ice Lake SP	Zen 3	NVIDIA A100
	DVFS (Frequency in MHz)		
1	3400	3600	1215 (Mem), 1410 (GPU)
2	3000	3300	1215 (Mem), 1215 (GPU)
3	2700	3000	1215 (Mem), 1005 (GPU)
4	2400	2700	1215 (Mem), 810 (GPU)
5	2100	2400	1215 (Mem), 600 (GPU)
6	1800	2100	1215 (Mem), 405 (GPU)
7	1500	1800	1215 (Mem), 210 (GPU)
8	1200	1500	
9	800		
	ACPI P-States (Governor)		
1	Performance	Powersave	
2	Powersave	Performance	
3		Ondemand	
4		Schedutil	
5		Userspace	
6		Conservative	
	Power Cap (Power Limit)		
1	100 W		100 W
2	150 W		150 W
3	200 W		200 W
4	250 W		250 W
5			300 W
6			350 W
7			400 W

deviations of the metrics are reported. The main metric for analyzing trade-offs is the EDP ($Energy - DelayProduct = System\_Energy * Execution\_Time$), where system energy is the sum of the energies of the relevant components for the platform. Lower EDP values indicate better overall efficiency.

This protocol aims for a systematic exploration of the power management configuration space, taking into account the specifics of each platform and the configuration challenges of modern AI frameworks.

3.5.3 In-depth Analysis of Results and Implications

The analysis focuses on the impact of each power management technique on the EDP and on the instantaneous power of the components.

3.5.3.1 General Impact of Techniques on EDP

The graphs Figures 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12 show the trade-offs in detail. All techniques allow achieving lower EDP points than the baseline (execution at maximum performance) for most workloads. This confirms that seeking maximum raw performance is rarely optimal from an energy efficiency perspective. Applying excessively low frequencies (DVFS) or very restrictive power limits (Power Capping) often leads to an increase in EDP. Although the energy consumed per unit of time (power) is reduced, the increase in execution time becomes so significant that it nullifies, or even reverses, the energy benefits. Finding the *"sweet spot"* is therefore crucial.

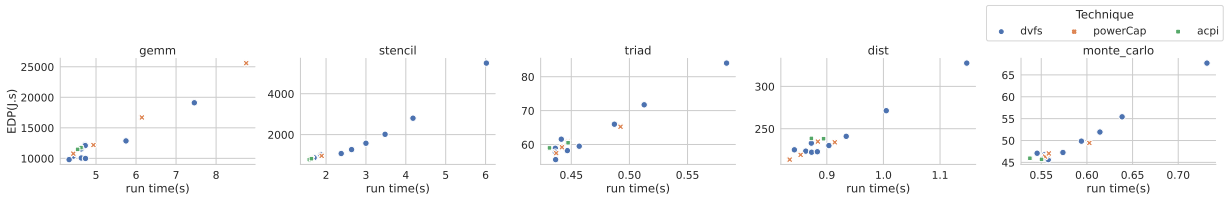


FIGURE 3.7 – EDP vs Time on Intel for JAX

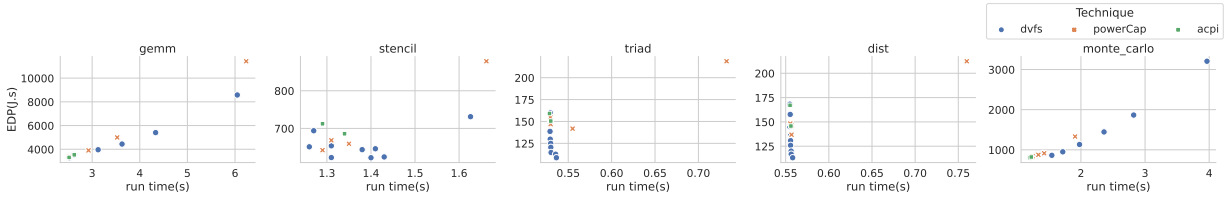


FIGURE 3.8 – EDP vs Time on Intel for TF

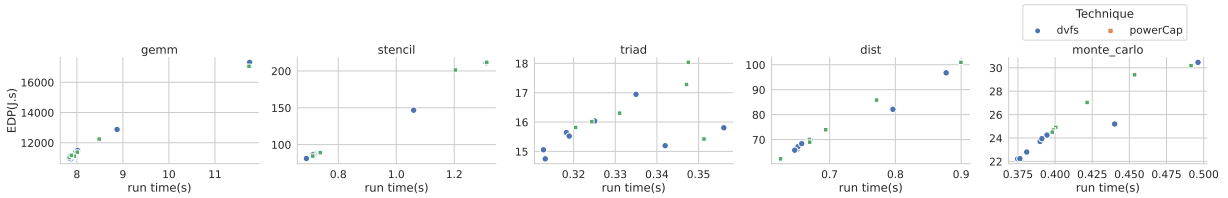


FIGURE 3.9 – EDP vs Time on AMD for JAX

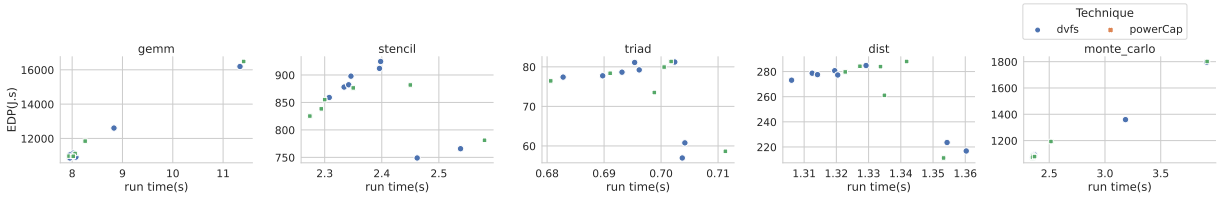


FIGURE 3.10 – EDP vs Time on AMD for TF

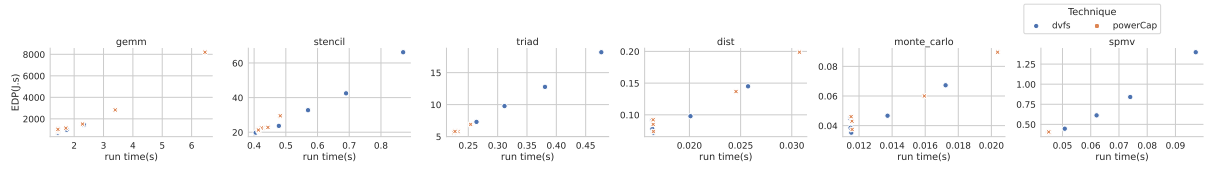


FIGURE 3.11 – EDP vs Time on Nvidia for JAX

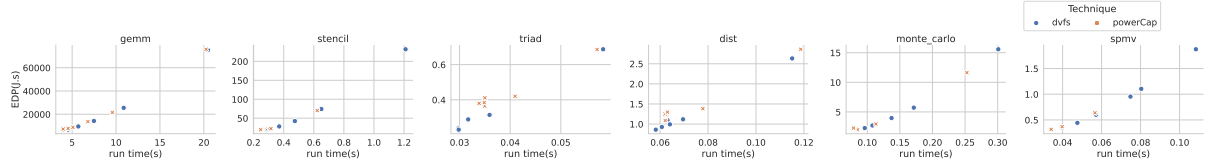


FIGURE 3.12 – EDP vs Time on Nvidia for TF

3.5.3.2 Platform, Workload, and Framework Specificity

The Tables 3.19, 3.20, 3.21, 3.22, 3.23 and 3.24 quantify the best EDP gain (in %) and the associated time variation, for each *kernel, platform, framework*, and *technique*.

On Intel Xeon (Tables 3.19 & 3.20) : Power Capping often proves more advantageous, allowing EDP reductions on the order of 8-18% with a limited impact (sometimes even a slight improvement) on execution time, especially for TensorFlow. DVFS can offer larger EDP gains (up to 32% for TF Triad/Dist) but generally at the cost of a more significant increase in time (>10-30% is not uncommon in EDP-optimal configurations). The `'powersave'` ACPI governor tends to degrade EDP compared to the 'performance' mode on this platform, unless there is a very strong reduction in computation time, which is not the goal of this governor.

Table 3.19 – Best EDP improvement with associated execution time on Intel with JAX

Bench	DVFS			POWERCAP			ACPI		
	EDP	Time	Freq (Ghz)	EDP	Time	Power (W)	EDP	Time	Mode
dist	+6.9%	0.0%	2.4	+10.5%	-4.4%	250	+0.1%	+2.5%	powersave
gemm	+16.7%	-6.4%	2.1	+8.4%	-4.1%	250	+2.5%	-1.8%	powersave
m_c	+0.7%	+3.7%	2.4	-0.8%	+3.1%	200	+0.5%	+2.4%	powersave
stencil	-2.9%	+1.3%	3.4	-4.3%	+2.3%	250	±0.0%	0.0%	performance
triad	+5.9%	+1.2%	2.1	+2.6%	+1.3%	150	±0.0%	0.0%	performance

Table 3.20 – Best EDP improvement with associated execution time on Intel with TF

Bench	DVFS			POWERCAP			ACPI		
	EDP	Time	Freq (Ghz)	EDP	Time	Power (W)	EDP	Time	Mode
dist	+32.3%	+0.5%	0.8	+18.1%	+0.3%	150	+12.8%	+0.2%	powersave
gemm	+0.8%	0.0%	2.7	-1.5%	+0.3%	250	0.0%	0.0%	performance
m_c	-1.5%	+0.9%	3.4	-4.7%	+3.4%	250	0.0%	0.0%	performance
stencil	+12.7%	+8.5%	1.5	+9.8%	0.0%	250	+3.7%	+3.9%	powersave
triad	+31.8%	+1.5%	0.8	+10.9%	+5.0%	150	+5.4%	+0.3%	powersave

On AMD EPYC (Tables 3.21 & 3.22) : DVFS is the dominant technique here for EDP reduction (up to 11% for JAX and 25% for TF). The optimal EDP is often achieved with significantly reduced frequencies, sometimes with a small time overhead for JAX, but a larger overhead for TensorFlow (which suggests a different sensitivity of the frameworks to CPU frequency on AMD). The ACPI governors, particularly `'powersave'`, allow for reducing EDP, but the largest gains (23-24% for dist or triad with TF) are accompanied by a large increase in execution time. `'ondemand'` or `'schedutil'` can offer interesting trade-offs.

Table 3.21 – Best EDP improvement with associated execution time on AMD with JAX

Bench	DVFS			ACPI		
	EDP	Time	Freq (Ghz)	EDP	Time	Mode
dist	+6.1%	-3.3%	2.4	+11.1%	-6.6%	ondemand
gemm	+3.9%	-1.8%	3.3	+3.1%	-2.2%	schedutil
m_c	+10.8%	-6.4%	2.7	+1.7%	-0.6%	schedutil
stencil	+4.5%	-2.8%	3.6	0.0%	0.0%	performance
triad	+7.9%	-3.4%	2.7	+3.7%	+8.3%	powersave

TABLE 3.22 – Best EDP improvement with associated execution time on AMD with TF

Bench	DVFS			ACPI		
	EDP	Time	Freq (Ghz)	EDP	Time	Mode
dist	+22.5%	+2.8%	1.5	+24.5%	+2.3%	powersave
gemm	+2.4%	-1.3%	2.1	+1.6%	-0.6%	schedutil
m_c	+1.0%	0.0%	2.1	+1.4%	+0.2%	ondemand
stencil	+14.6%	+4.8%	1.8	+10.9%	+9.8%	powersave
triad	+25.5%	+3.4%	1.5	+23.3%	+4.5%	powersave

On NVIDIA A100 (Tables 3.23 & 3.24) : Power Capping is the most effective technique for reducing EDP, especially with JAX (34-42% reduction for many kernels with a 0-2% increase in time). For TensorFlow, it also offers significant gains (up to 32%). DVFS is also beneficial (e.g., GEMM and TF Triad show good gains), but its potential with JAX is limited by stability issues : JAX could not run correctly at the two lowest GPU frequencies tested (210 MHz and 405 MHz), leading to failures or *"hangs"*. TensorFlow proved to be more robust at these low frequencies.

TABLE 3.23 – Best EDP improvement with associated execution time on Nvidia with JAX

Bench	DVFS			POWERCAP		
	EDP	Time	Freq (Ghz)	EDP	Time	Power (W)
dist	+21.4%	+0.8%	1.005	+37.4%	+0.9%	200
gemm	+30.6%	+0.1%	1.005	+34.2%	0.0%	300
m_c	+21.6%	+0.6%	1.005	+37.4%	+1.1%	200
spmv	0.0%	0.0%	1.410	+34.1%	+0.4%	100
stencil	0.0%	0.0%	1.410	+37.3%	+2.3%	300
triad	0.0%	0.0%	1.410	+41.8%	-0.9%	300

TABLE 3.24 – Best EDP improvement with associated execution time on Nvidia for TF

Bench	DVFS			POWERCAP		
	EDP	Time	Freq (Ghz)	EDP	Time	Power (W)
dist	+30.8%	-7.2%	1.005	+32.5%	-0.9%	200
gemm	+12.8%	+15.0%	1.005	+16.3%	+14.5%	300
m_c	+8.6%	+7.6%	1.215	+32.4%	+8.1%	200
spmv	0.0%	0.0%	1.410	+24.2%	+0.3%	350
stencil	+6.5%	+7.7%	1.215	+29.9%	+3.2%	200
triad	+40.5%	-14.2%	0.810	+23.9%	+0.6%	200

3.5.3.3 Workload Sensitivity Analysis

Compute-bound kernels like GEMM reach their minimum EDP with relatively high frequencies or power caps (less aggressive settings). For memory-bound kernels (Triad, Dist), lower frequencies are often optimal because the CPU/GPU spends a lot of time waiting for memory ; reducing their frequency/power then has little negative impact on the overall time. Power Capping on the A100 proves effective even for GEMM, illustrating its relevance on massively parallel architectures with high TDP.

3.5.3.4 Detailed Impact on Component Power

The Figures 3.13, 3.14 & 3.15 show the power trends of the main compute component (CPU Pkg or GPU, in orange) and a secondary component (DRAM for CPU, Host CPU for GPU, in blue).

Effect of DVFS : Applying DVFS clearly and significantly reduces the power consumed by the targeted compute component (CPU Pkg or GPU). This reduction is all the more marked for workloads that intensively solicit this component (e.g., GEMM). In parallel, the power consumed by the DRAM (on CPU) or the Host CPU (when the GPU is working) remains remarkably stable and low (cf. values in Tables 3.25, 3.26 and 3.27), regardless of the frequency variations of the main CPU/GPU. This confirms that DVFS of the compute units has a direct effect on these units, but a very limited indirect effect on the power of the memory/host components in these experiments.

Effect of Power Capping : The measured CPU Pkg (Intel) or GPU (NVIDIA) power follows the limit imposed by the cap, especially for restrictive cap values. When the cap is higher (close to or above the TDP or the natural demand of the workload), the measured power plateaus at this "natural" level. Here again, the impact on DRAM/Host CPU power is marginal, or even a slight increase in Host CPU power for the GPU is observed with higher caps, potentially due to increased memory traffic.

Effect of ACPI Governors : On AMD EPYC, each governor induces a distinct Pkg power level consistent with its policy (e.g., `'performance'` -> high power, `'powersave'` -> low power, `'ondemand'/'schedutil'` -> intermediate levels). The DRAM power remains stable.

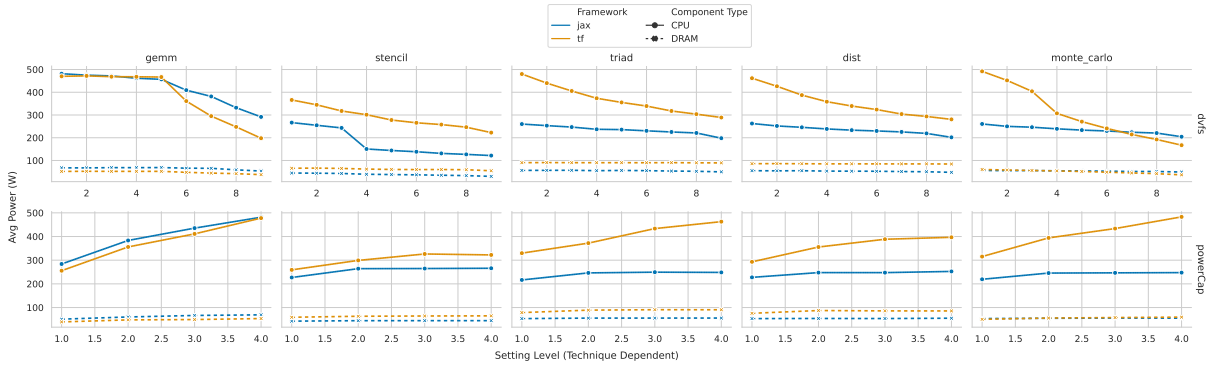


FIGURE 3.13 – Power trends on Intel

3.5.4 Discussion of Behaviors and Implications

The main message is that the effectiveness of techniques is highly contextual. There is no universally "best" technique. A "*platform-aware, workload-aware*, and *framework-aware*" approach is necessary. Targeted benchmarking is crucial to identify optimal configurations. Compute-bound applications are very sensitive to frequency/power scaling, with direct trade-offs on performance. Memory-bound applications are less penalized in performance by a reduction in core frequency, offering better opportunities for EDP gains via DVFS.

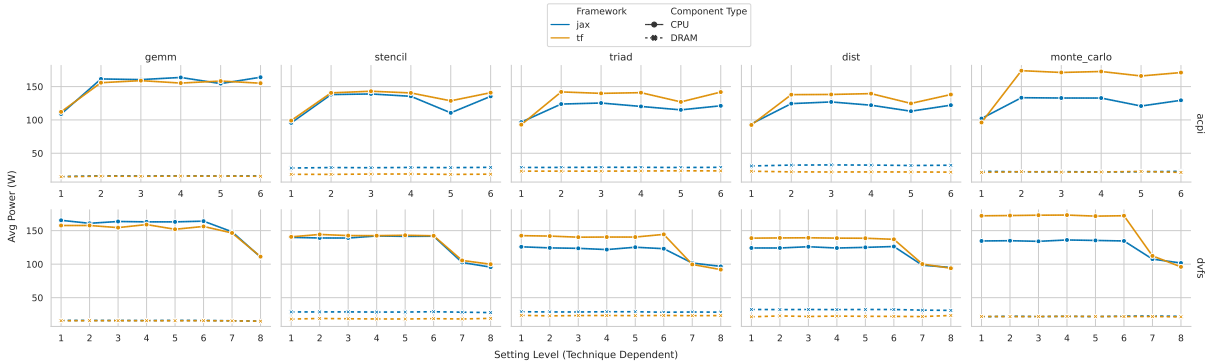


FIGURE 3.14 – Power trends on AMD

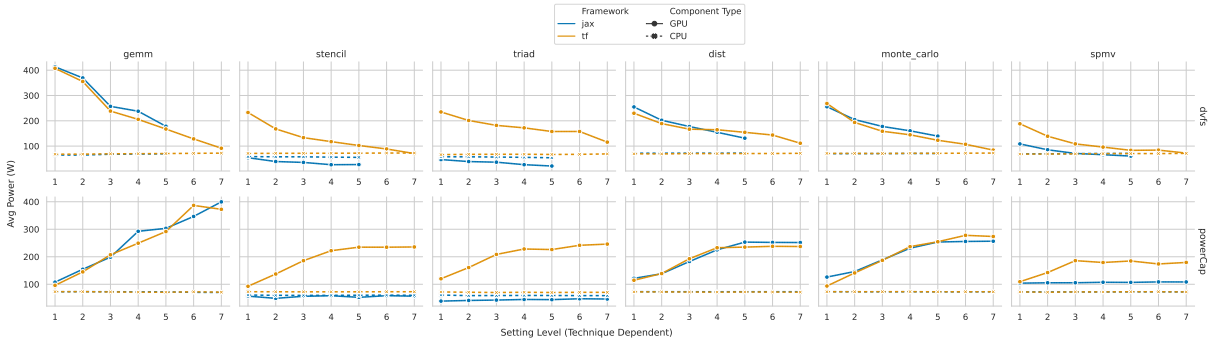


FIGURE 3.15 – Power trends on Nvidia

TABLE 3.25 – CPU/RAM Baseline power and Min EDP on AMD

Bench	Lib	Baseline (W) CPU	Baseline (W) RAM	MinEDP (W)) ACPI CPU	MinEDP (W)) ACPI RAM	MinEDP (W)) DVFS CPU	MinEDP (W)) DVFS RAM
dist	jax	124.5 ± 2.5	32.2 ± 0.2	127.0 ± 1.8	32.5 ± 0.1	125.0 ± 1.5	32.4 ± 0.3
	tf	137.9 ± 2.7	22.1 ± 0.4	92.5 ± 1.9	23.0 ± 1.0	93.9 ± 0.7	23.5 ± 0.9
gemm	jax	161.5 ± 3.2	16.2 ± 0.3	163.7 ± 1.1	16.1 ± 0.1	160.9 ± 5.4	16.0 ± 0.2
	tf	155.9 ± 4.2	15.7 ± 0.1	155.3 ± 4.3	15.8 ± 0.2	156.2 ± 3.1	15.6 ± 0.2
m_c	jax	133.3 ± 1.5	22.1 ± 1.0	132.8 ± 1.8	22.0 ± 0.8	136.1 ± 0.9	22.2 ± 1.1
	tf	173.9 ± 0.6	22.3 ± 0.2	171.2 ± 2.0	21.7 ± 0.4	172.2 ± 1.1	21.9 ± 0.6
stencil	jax	138.0 ± 3.5	28.6 ± 0.5	138.0 ± 3.5	28.6 ± 0.5	139.9 ± 1.9	28.7 ± 0.4
	tf	140.6 ± 2.5	18.3 ± 0.2	99.1 ± 1.0	18.4 ± 0.9	105.5 ± 0.6	18.2 ± 0.5
triad	jax	123.8 ± 2.2	28.7 ± 0.5	96.7 ± 1.7	28.7 ± 0.1	121.7 ± 3.1	29.0 ± 0.1
	tf	142.0 ± 3.4	23.2 ± 0.6	92.9 ± 2.2	23.1 ± 0.3	91.8 ± 3.1	23.3 ± 0.6

Differences between JAX and TF Frameworks : The first main difference is about *Compilation Strategies*. XLA (JAX) with its aggressive operator fusion can be very performant for some workloads (Stencil, Dist), while TensorFlow (with its graphs traced via `'tf.function'`) can excel on others (GEMM on A100). Additionally, *Memory Management* is another key issue. The need to finely configure memory allocators (XLA flags for JAX, `'cuda_malloc_async'` for TF) to avoid OOMs on large problems (especially with JAX at low GPU frequencies) shows that memory management strongly interacts with performance and potentially energy. TF proved more scalable in terms of manageable problem size for GEMM on A100. JAX's inability to operate stably at the *lowest GPU frequencies* on the NVIDIA A100 (unlike TF) is a critical

TABLE 3.26 – CPU/RAM Baseline power and Min EDP on Intel. P_C = POWERCAP

		Baseline (W)		MinEDP (W))					
		CPU	RAM	ACPI CPU	ACPI RAM	DVFS CPU	DVFS RAM	P_C CPU	P_C RAM
Bench	Lib								
dist	jax	259.2 ± 2.7	54.6 ± 1.0	245.7 ± 4.2	52.9 ± 1.8	238.7 ± 3.2	53.3 ± 0.9	252.5 ± 4.6	54.9 ± 1.0
	tf	457.8 ± 0.7	85.7 ± 0.2	387.7 ± 9.0	85.2 ± 0.2	280.7 ± 1.6	84.5 ± 0.4	355.6 ± 6.2	87.3 ± 0.5
gemm	jax	482.4 ± 12.9	68.7 ± 1.3	489.2 ± 6.7	69.2 ± 0.7	456.8 ± 17.8	68.9 ± 2.4	481.5 ± 18.5	69.1 ± 2.7
	tf	472.6 ± 3.1	52.3 ± 0.6	472.6 ± 3.1	52.3 ± 0.6	468.5 ± 4.1	52.2 ± 1.2	477.7 ± 10.6	53.4 ± 1.7
m_c	jax	262.0 ± 2.7	56.3 ± 1.2	247.2 ± 9.0	55.2 ± 1.6	239.2 ± 2.0	54.6 ± 0.6	246.6 ± 3.6	55.5 ± 1.0
	tf	493.3 ± 1.2	60.6 ± 0.1	493.3 ± 1.2	60.6 ± 0.1	492.0 ± 5.6	60.4 ± 0.6	482.9 ± 10.4	59.5 ± 1.2
stencil	jax	265.7 ± 0.9	44.7 ± 0.2	265.7 ± 0.9	44.7 ± 0.2	266.6 ± 1.7	44.9 ± 0.3	265.8 ± 1.2	44.6 ± 0.1
	tf	364.0 ± 5.8	65.3 ± 0.6	319.5 ± 5.1	63.3 ± 2.0	257.9 ± 7.2	60.3 ± 2.5	322.2 ± 9.2	64.9 ± 2.9
triad	jax	261.1 ± 3.7	56.6 ± 1.4	261.1 ± 3.7	56.6 ± 1.4	235.8 ± 3.0	56.3 ± 1.0	246.3 ± 5.3	55.3 ± 1.5
	tf	479.9 ± 1.7	90.4 ± 0.3	446.5 ± 15.7	90.4 ± 0.4	289.0 ± 2.8	89.4 ± 1.3	372.0 ± 13.2	89.0 ± 0.4

TABLE 3.27 – GPU/Host Baseline power and Min EDP on Nvidia. P_C = POWERCAP

		Baseline (W)		MinEDP (W))			
		GPU	Host	DVFS GPU	DVFS Host	P_C GPU	P_C Host
Bench	Lib						
dist	jax	254.8 ± 3.4	71.9 ± 0.2	177.4 ± 2.4	72.1 ± 0.1	182.7 ± 2.1	72.9 ± 0.2
	tf	229.8 ± 6.3	69.5 ± 0.5	166.8 ± 1.2	70.4 ± 0.2	192.8 ± 6.2	71.9 ± 0.6
gemm	jax	413.5 ± 36.7	64.7 ± 0.2	257.1 ± 25.4	67.9 ± 0.4	303.3 ± 51.0	72.5 ± 0.5
	tf	407.3 ± 32.0	68.0 ± 0.6	238.4 ± 10.4	69.4 ± 0.3	291.7 ± 32.6	71.4 ± 0.3
m_c	jax	255.6 ± 7.1	70.0 ± 0.3	177.7 ± 5.4	70.4 ± 0.4	188.7 ± 2.5	72.7 ± 0.4
	tf	268.4 ± 14.3	71.2 ± 0.8	193.5 ± 3.3	71.1 ± 0.2	186.9 ± 1.1	73.0 ± 0.2
spmv	jax	108.7 ± 2.2	69.4 ± 0.2	108.7 ± 2.2	69.4 ± 0.2	104.0 ± 1.1	72.2 ± 0.2
	tf	188.2 ± 2.0	68.4 ± 0.1	188.2 ± 2.0	68.4 ± 0.1	173.9 ± 7.8	72.7 ± 0.5
stencil	jax	54.1 ± 6.5	58.5 ± 0.3	54.1 ± 6.5	58.5 ± 0.3	52.5 ± 8.7	60.2 ± 0.6
	tf	232.8 ± 2.8	71.1 ± 0.2	168.0 ± 1.5	71.4 ± 0.3	185.7 ± 4.3	72.9 ± 0.3
triad	jax	46.8 ± 5.2	57.8 ± 0.5	46.8 ± 5.2	57.8 ± 0.5	44.2 ± 3.7	59.2 ± 1.0
	tf	234.9 ± 7.2	66.4 ± 0.4	172.1 ± 11.1	67.7 ± 0.5	208.5 ± 8.0	70.1 ± 0.1

observation. The reasons could be multiple : sensitivity of the XLA-generated code to minimal latency/throughput, specific runtime/driver interactions, or sensitivity of low-level kernels. This limits the applicability of deep DVFS for JAX on this platform.

Implications : For Framework Developers, robustness to low-power configurations is essential. Native integration of energy monitoring capabilities and APIs for application-informed power control could be beneficial. For Hardware Designers, the study highlights the importance of accessible and accurate power sensors for all components (*CPU, GPU, DRAM, Interconnects*). The effectiveness of control mechanisms (DVFS, Power Cap) and the need for independent memory management strategies are also brought to light.

3.5.5 Synthesis of PMT Benchmarking

This detailed experimental study on the impact of system power management techniques (*DVFS, ACPI Governors, Power Capping*) provides the following conclusions and lessons :

Confirmation of Contextual Effectiveness : The best energy-saving strategy is a complex function of the hardware platform (*Intel, AMD, NVIDIA*), the nature of the workload (*compute-bound, memory-bound*), and even the software framework (*JAX, TensorFlow*). Power Capping stands out on Intel Xeon and NVIDIA A100 for its ability to reduce EDP with a low performance impact in many cases, while DVFS shows marked advantages on AMD EPYC,

particularly for less compute-intensive tasks.

Primacy of the Main Compute Unit : Energy savings primarily come from modulating the consumption of the CPU cores or the GPU. The impact of these techniques on the consumption of DRAM (on CPU) or the Host CPU (for GPU operations) is marginal, highlighting the potential need for dedicated and decoupled power management mechanisms for these components.

Relevance of EDP : The Energy-Delay Product (EDP) remains an essential metric for evaluating trade-offs and identifying operating points that judiciously balance energy consumption and performance degradation. Aggressively optimizing for energy alone often leads to a prohibitive increase in time and thus to a suboptimal EDP.

Subtle but Real Influence of Software Frameworks : Although JAX and TensorFlow can drive the hardware to similar instantaneous power levels, their internal mechanisms for compilation, memory allocation, and kernel execution lead to different execution times and variable robustness to low-power configurations (case of JAX on A100). This underscores that energy optimization cannot be limited to the hardware and the OS but must also consider the application software stack.

Need for a Co-Design Approach and Analysis Tools : Achieving significant energy efficiency requires a co-design approach where hardware choices, power control capabilities, framework robustness, and optimization strategies are considered jointly. Measurement and benchmarking tools like EA2P, and the fine-grained characterizations conducted here, are indispensable for providing the data necessary for this process.

The results of this in-depth benchmarking not only provide direct recommendations to practitioners for specific configurations but also fuel the design of more dynamic and intelligent energy optimization strategies. They also reinforce the motivation for developing predictive energy models capable of capturing this complexity, the subject of Chapter 4.

3.6 Conclusion

This chapter has laid the essential empirical and methodological foundations for addressing the problem of energy efficiency in HPC and AI systems. We first highlighted the imperative need for precise and detailed energy measurements (Section 3.1), emphasizing the limitations of coarse estimates and the inherent challenges in obtaining reliable data on heterogeneous architectures and with complex workloads. This analysis justified the need to develop suitable measurement tools.

The main contribution of this chapter was the detailed presentation of EA2P (Energy-Aware Application Profiler), our software tool for energy profiling (Section 3.2). We described its motivation, its modular and flexible design in Python, its architecture built upon existing hardware interfaces (RAPL, Perf, SMI), and its implementation of key features, notably CPU and GPU measurement, and an original analytical model for estimating the energy consumed by the main RAM. The rigorous experimental validation of EA2P demonstrated its ability to provide

consistent measurements compared to reference tools, its robustness across various workloads and platforms, an acceptable profiling overhead, and the utility of its RAM model and its features for analyzing parallel applications (multi-threaded, MPI). EA2P is thus positioned as a practical and extensible tool for the community, especially for developers working with AI frameworks in Python.

Equipped with this tool, we then presented two fundamental energy characterization studies. The first, a detailed benchmarking of compute kernels (Section 3.3), allowed us to analyze the energy profile of different parallelism paradigms (SIMD, OpenMP multicore, GPU via Python frameworks) on Intel and AMD CPU architectures, and on an NVIDIA GPU. These results highlighted the strong time-energy correlation, the variable impact of each paradigm depending on the kernel and architecture, and above all, the predominant role of the software framework in the energy efficiency of GPU execution.

The second characterization study, an in-depth benchmarking of system power management techniques (Section 3.4), quantitatively evaluated the impact of DVFS, ACPI governors, and Power Capping on performance and energy consumption (via EDP). These experiments, conducted on several platforms and with different frameworks, confirmed the contextuality of optimizations : there is no universal solution, and the best energy-performance trade-off depends closely on the hardware, the workload, and the software stack. These results provide valuable data for more informed power management.

In summary, this chapter has presented our contributions that are a tool (EA2P) and an empirical knowledge on the energy behavior of components, compute kernels, and power management mechanisms. These elements constitute an indispensable basis for the work that will follow : the predictive modeling of energy consumption, which will rely on the fine-grained measurements obtained here (Chapter 4), and the development of dynamic energy optimization strategies, which will exploit the system levers characterized here (Chapter 5). A detailed understanding of the sources of consumption and the impact of system configurations is indeed the prerequisite for any attempt at intelligent and effective optimization.

Chapter 4

Characterization and Analytical Modeling of Energy Consumption

Ce chapitre est dédié au pilier de la modélisation prédictive, visant à transformer les données de mesure brutes en connaissances actionnables. La première partie formalise et valide en détail le modèle analytique de l'énergie de la DRAM, une des fonctionnalités clés de notre outil EA2P. La contribution principale du chapitre est cependant le développement d'un cadre logiciel (framework) analytique "boîte blanche" pour prédire avec une haute fidélité la performance (temps par itération) et l'énergie (puissance moyenne) de l'entraînement des modèles de Deep Learning sur GPU. L'originalité de notre approche est triple : elle repose sur une caractérisation hiérarchique détaillée de la charge incluant des "caractéristiques structurelles d'efficacité" ; elle intègre dans son modèle de performance des effets de saturation non-linéaire inspirés de la micro-architecture GPU ; et elle emploie une méthodologie de calibration robuste utilisant des scalaires de correction ciblés. La validation démontre une excellente précision (erreur moyenne de 4.14%) tout en maintenant une totale interprétabilité des paramètres, offrant ainsi un outil puissant pour l'éco-conception et l'analyse de performance.

4.1 Introduction

In Chapter 3, we have illustrated the importance of accurate energy measurement and describe our contributions on this aspect. However, to transition from simple observation to predictive, proactive, and large-scale optimization, it is necessary to take an additional step : modeling. Energy modeling aims to encapsulate the complexity of hardware-software interactions into mathematical relationships capable of predicting energy consumption without requiring the full execution of every “what-if” scenario.

As described in the state of the art Chapter (Section 2.4), while “black-box” empirical models (based on machine learning) can achieve good local accuracy, they often suffer from a lack of interpretability and difficulties in generalization. This chapter explores a complementary and more explanatory path : *an analytical or semi-analytical modeling*. Such an approach, though

more arduous to develop, aims to build models based on a structural understanding of the system and the application. It offers predictions that also provide information about dominant factors and energy bottlenecks. In this chapter, we present two distinct but complementary contributions to energy modeling.

First, we tackle a component often neglected by in-band measurement tools, but whose consumption is increasing : the main memory (DRAM). We develop a simple and pragmatic analytical model for it (Section 4.1), directly integrated into our EA2P tool, to fill this measurement on many platforms.

Second, we move up to the application level by presenting a comprehensive analytical framework for predicting the performance (time) and energy for the training phase of Deep Learning models on GPUs (Section 4.2). This framework represents one of the major contributions of this thesis. It relies on a detailed hierarchical decomposition of the workload (quantification of operations, memory usage, and innovative structural features) and on a parametric model that maps these features to performance and power predictions, taking into account the effects of architectural saturation. The objective is to provide a powerful and interpretable tool for hardware-software co-design, intelligent scheduling, and performance analysis of AI applications.

Together, these two contributions illustrate a multi-scale approach to analytical modeling, from the individual component to the complex application, with the ultimate goal of providing the necessary building blocks for more informed energy optimization. We now present our two aforementioned contributions.

4.2 Analytical Modeling of the Energy Consumption of a DRAM

The main memory (DRAM) is a component whose energy impact has become significant, yet getting close or even exceeding that of idle CPUs on overloaded servers [Appuswamy, 2015]. Yet, as we established in previous chapters, its consumption is a “blind spot” for many platforms where in-band measurement interfaces like RAPL do not cover it, particularly on Intel’s “client” CPU architectures or many AMD architectures. This section presents the development and justification of a pragmatic analytical model for estimating DRAM energy, a contribution directly integrated into the EA2P tool to enhance its usefulness.

4.2.1 Motivation and Challenges of DRAM Modeling

The primary motivation for this work is to compensate the lack of direct DRAM energy measurement solutions. Providing an estimate or a range is better than completely ignoring this component, which can represent a substantial part of the total consumption. However, modeling DRAM energy presents several challenges :

— *Technological Diversity :* Different DRAM generations (DDR3, DDR4, DDR5) have distinct power characteristics. In particular, DDR5 technology directly integrates a Power Management Integrated Circuit (PMIC) on the DIMM, potentially improving its efficiency

by nearly 30% compared to DDR4 [Samsung, 2022]. A generic model must be able to account for these evolutions.

— *Impact of Physical Configuration :* The total memory consumption depends not only on the total capacity but also on the physical configuration, i.e., the number of DIMMs installed on the motherboard. A system using more DIMMs to achieve a certain capacity will generally consume more than a system using fewer, denser DIMMs, all other things being equal [Wallossek, 2014].

— *Non-Linearity of Consumption with Capacity :* As some studies show [Appuswamy, 2015], the relationship between total memory capacity and power is not linear. It can be sub-linear at low capacity and become super-linear at very high capacity (e.g., terabytes), due to the need to use high-density DIMMs which are inherently more power-hungry.

— *Relationship with Software Activity :* The dynamic consumption of RAM depends on the intensity and type of memory accesses (*reads, writes, refreshes*), which are dictated by the running application.

4.2.2 Development of a Discretized Analytical Model

Considering the previous challenges, and for the sake of pragmatism for integration into a tool like EA2P, we opted for a discretized analytical model. The approach does not aim for fine-grained micro-architectural modeling, but rather to provide a reasonable estimate based on easily accessible system information and public manufacturer data.

The calculation of RAM energy is broken down into estimating an average power over a time interval, then integrating it. The RAM power, *Power_RAM*, is modeled by the formula 4.1 & 4.2 :

$$Power\_RAM = Power\_base \times nb\_slots \times mem\_use \tag{4.1}$$

And the total energy Energy_RAM is the sum of the energies over the N sampling intervals of duration `‘interval‘` :

$$Energy\_RAM = (\sum_{i=1}^{N} Power\_RAM) \times interval, \tag{4.2}$$

The parameters of this model are defined as follows.

Power_base : Base Power per DIMM. This parameter represents an estimate of the base nominal power (TDP-like power) for a single memory DIMM, based on its technology (DDRx) and capacity. To determine this value, we relied on public data from manufacturers and empirical rules. For example, based on data from Crucial [Technology, 2024] and SK Hynix [Skhynix, 2021], we established a correspondence table, such as : 10W for a 256GB LRDIMM, 8W for 128GB, 6W for 64GB, 5W for 32GB, 4W for 16GB. Upon initialization, EA2P reads the type and capacity of the installed DIMMs to select the appropriate *Power_base*.

nb_slots : Number of Installed DIMMs. To account for the impact of the physical configuration, the model is multiplied by the number of effectively occupied memory slots. This information is dynamically retrieved by EA2P via the Linux command `‘sudo dmidecode‘`.

This approach assumes, as a first approximation, that the total consumption is the sum of the individual consumptions of each DIMM [Wallossek, 2014].

mem_use : Usage-Based Correction Factor. This dimensionless factor (between 0 and 1) aims to modulate the base power according to the intensity of memory usage by the profiled process. Acknowledging that a purely linear relationship is inaccurate and that a memory DIMM consumes energy even at rest (for refreshing), we adopted a tiered approach based on empirical observations : Actual usage $< 5\%$: mem_use $= 0.30$ (represents significant idle consumption) Actual usage between 5% and 15% : mem_use $= 0.60$ Actual usage between 15% and 25% : mem_use $= 0.70$ Actual usage between 25% and 50% : mem_use $= 0.75$ Actual usage $> 50\%$: mem_use $= 0.80$

The actual memory usage of the process is measured by EA2P during execution at each sampling interval to determine which tier to apply.

4.2.3 About the Assumptions of our Model and its Limitations

This model is intentionally pragmatic and relies on assumptions that constitute its main limitations.

Discretization of Usage : The tiers for *'mem_use'* are a simplification of the complex relationship between memory activity (*access intensity, read/write ratio*) and consumption. A finer approach based on HPCs (e.g., measured bandwidth) would be more accurate but also more complex to implement portably.

Non-Linearity of Capacity : The current model does not explicitly capture the super-linear effect of consumption at very high memory capacities (multiple terabytes) [Appuswamy, 2015]. Its domain of validity is rather that of servers equipped with a few hundred gigabytes to a few terabytes.

Simplification of DDR Technologies : The fine-grained impact of DDR5 technology optimizations (PMIC) is absorbed into the *Power_base* value but is not dynamically modeled.

Dependency on Manufacturer Data : The accuracy of the *Power_base* values depends on the availability and reliability of public data provided by memory manufacturers, which can vary.

Despite these limitations, this model represents a significant improvement over the complete absence of RAM consumption estimation.

4.2.4 Validation of our Model and its Integration into EA2P

As the estimation of RAM energy is a key feature of EA2P, especially for platforms without direct RAPL DRAM measurement, a specific validation of this model was performed. The Figure 4.1 presents a direct comparison between the RAM energy estimated by EA2P's analytical model (based on formulas (4.1) and (4.2)) and the RAM energy measured by Intel's RAPL DRAM domain (accessible via PowerCap) on the Intel-2 platform (Xeon E5-2698v4, 512 GB

RAM), which has this reference sensor. The X-axis represents the memory footprint used by an application (in GB), the left Y-axis the energy consumption (in Wh), and the right Y-axis the execution time.

The results show an excellent match between the estimates of EA2P's RAM model (green curve in the figure) and the measurements from Intel's RAPL DRAM domain (blue curve). The two curves almost perfectly overlap over a wide range of memory footprints (from 4GB to 450GB).

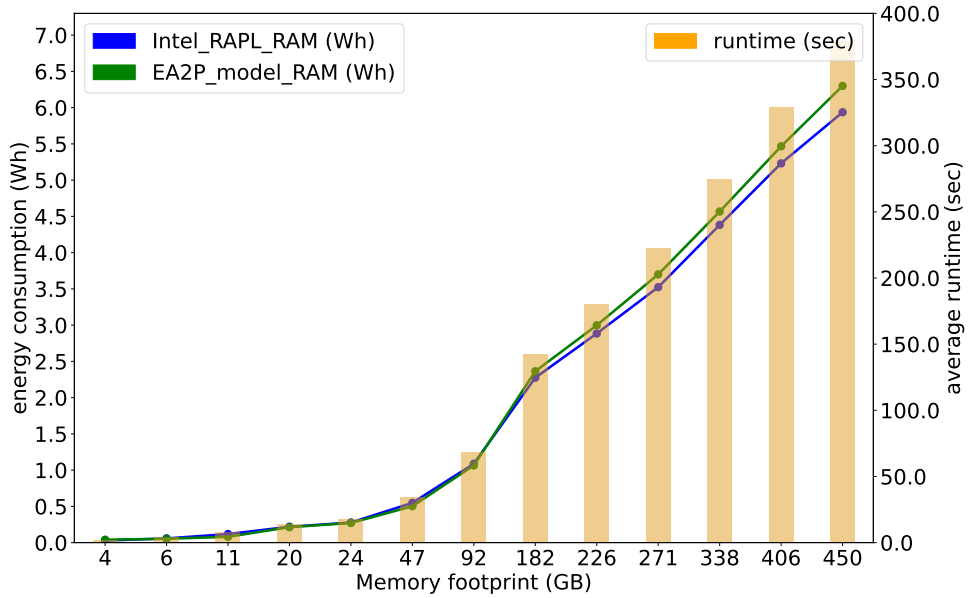


FIGURE 4.1 – EA2P RAM energy model vs Intel RAPL RAM domain on Intel-2.

This validates the approach of our simplified analytical model and the effectiveness of the usage tiers (30% to 80% of Power_base depending on actual mem_use) in approximating RAM consumption when direct measurements are lacking. Although the model has theoretical limitations (no fine-grained consideration of DDR5 PMICs or extreme non-linearity at very high capacity, as discussed in the paper), it proves to be effective for common server configurations (here up to 512GB). The execution time (orange bars) logically increases with the memory footprint, as expected.

The contribution presented in this section consists of the development of a discretized and pragmatic analytical model to estimate the energy consumed by DRAM. By relying on easily accessible system information (`"dmidecode"`, *process memory usage*) and public data, this model allows the EA2P tool to provide a more complete energy view of the system, filling the memory measurement "blind spot" on many platforms. Although improvable, this model, validated by comparison with real RAPL measurements, represents a practical and useful contribution for more holistic energy analyses.

4.3 An Analytical Framework for Performance and Energy Prediction of DL Training

After presenting our modeling at the level of a specific component, the DRAM, we now present another contribution as a futher step : a comprehensive analytical framework for predicting the performance (execution time per step) and energy (average power consumption) of the training phase of Deep Learning models on GPU architectures. This contribution aims to directly address the limitations of existing approaches, by proposing a *"white-box"*, interpretable, and *hardware-aware* model. The objective is to provide a rigorous scientific tool to assist with tasks that have become critical, such as hardware-software *co-design*, energy-efficient resource planning in data centers, early and reliable estimation of the carbon footprint, and fine-grained performance analysis for the optimization of AI applications.

4.3.1 Motivation and Foundations of the Analytical Approach

The growing complexity of Deep Learning models makes it imperative to anticipate their resource requirements. However, the majority of existing modeling methods struggle to reconcile accuracy, generalizability, and interpretability. Coarse-grained models, like the Roofline model [Williams, 2009], while useful for an initial analysis of limitations (*compute-bound* vs. *memory-bound*), cannot capture the micro-architectural subtleties and complex interactions specific to Deep Learning operators on modern GPU architectures [Ding, 2019]. At the opposite end, empirical models based on machine learning (e.g., through regression or neural networks) can offer excellent local accuracy, but their *black-box* nature limits their explanatory power and their robustness when extrapolating to new hardware architectures or to types of models not seen during their training [Yang, 2021].

Our motivation, therefore, is to build a bridge between these two extremes. The framework we propose is **analytical**, meaning it is anchored in a physical and structural decomposition of the workload and the hardware system. The predictions are not mere statistical correlations but the result of a set of equations whose parameters aspire to have a physical or architectural meaning. At the heart of our approach is a conviction : to accurately model the execution of a workload, it is not enough to quantify its volume (e.g., number of FLOPs) ; it is essential to characterize its **structure**. We therefore introduce the notion of *structural efficiency features* that describe how the computation is organized and that directly influence the efficiency of hardware utilization. These features feed into a performance model that centrally integrates the phenomena of **non-linear saturation** of compute units, a fundamental characteristic of massively parallel architectures. High fidelity is finally ensured by a systematic calibration process against empirical data, allowing the adjustment of model parameters and the integration of **targeted correction scalars** to capture the finest nuances of certain model architectures (e.g., ALBERT [Lan, 2019]) or computation modes (e.g., BF16 precision [Micikevicius, 2018]).

4.3.2 Technological Context : Running AI Workloads on High-End GPUs

Modern Graphics Processing Units (GPUs), and particularly NVIDIA's Ampere architecture embodied by the A100 GPU, are the cornerstone of high-performance computing for Artificial Intelligence. Their efficiency comes not from the speed of a single core, but from massive parallelization. Understanding this architecture and the software stack that exploits it is fundamental to modeling its performance and energy consumption.

4.3.2.1 The Architectural Specificity of the GPU and its SIMT model

Unlike CPUs, designed to excel at sequential or moderately parallel tasks with complex and fast cores (optimized for latency), GPUs are architectures optimized for **throughput**. They implement the **SIMT (Single Instruction, Multiple Threads)** programming model. The principle is to execute the same instruction (a computation kernel, or *kernel*) simultaneously on a very large number of different data items, by thousands of simple threads. It is this ability to perform the same operation (e.g., a multiplication) on thousands of tensor elements at the same time that gives them their power for linear algebra, the core of Deep Learning.

4.3.2.2 Anatomy of a High-End GPU : The NVIDIA A100

The A100 GPU is an embodiment of the GPU paradigm as previously described. It is not monolithic but hierarchically structured.

Streaming Multiprocessors (SM). The basic unit of the GPU is the SM. A full A100 has 108 of them. It is at the SM level that thread blocks (see CUDA programming) are scheduled and executed. Each SM is itself a kind of mini-vector processor equipped with a variety of specialized computing units (see Figure 4.2) :

— **FP32 Cores (CUDA Cores) :** Computing units for single-precision floating-point operations. Each SM on the A100 contains 64. These are the "ALU" of our model, responsible for general-purpose operations.

— **FP64 Cores :** Units for double precision, important in scientific HPC, less so in AI. Each SM has 32 (with a performance of 1/2 that of FP32).

— **Tensor Cores (TCU) :** This is the key architectural innovation for AI. Each A100 SM has 4 third-generation Tensor Cores. These units are dedicated hardware accelerators for the **MMA (Matrix Multiply-Accumulate)** operation : $D = A \times B + C$. They can execute this operation on small matrices (e.g., $4 \times 4 \times 4$ for FP32, or larger sizes for reduced precisions) in a single clock cycle. Their throughput is massively superior to that of FP32 cores for matrix operations. They natively support several precisions (see Figure 4.3) crucial for AI :

  — **TF32 (TensorFloat-32) :** An NVIDIA proprietary format on Ampere, which uses the range of FP32 (8 bits of exponent) but the precision of FP16 (10 bits of mantissa). It offers up to 8 times the throughput of standard FP32 for GEMMs, without requiring code modification (enabled by default in frameworks).

FIGURE 4.2 – Overview of a Streaming Multiprocessor (SM) of the NVIDIA Ampere GPU (A100) [NVIDIA, 2020] .

— **BF16 / FP16 :** Mixed precisions that allow for even higher throughputs, but require careful management (via *Automatic Mixed Precision* - AMP) to maintain numerical stability.

— **INT8 / INT4 :** Integer precisions for inference, offering the highest throughput.

The distinction between FLOPs executable on TCUs and those on ALUs is therefore fundamental, justifying the fine-grained categorization of our `WorkloadCalculator`.

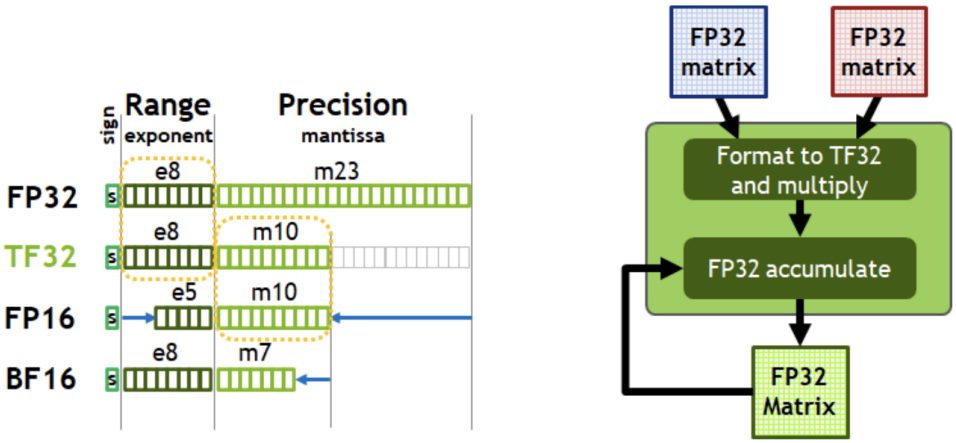


FIGURE 4.3 – TensorFloat-32 (TF32) computation and precision [NVIDIA, 2020] .

Memory Hierarchy. To feed these thousands of cores, the GPU has a complex memory hierarchy :

— **Registers :** The fastest memory, private to each thread.

— **Shared Memory / L1 Cache :** Very fast memory (a few TB/s) on the SM chip, explicitly managed by the programmer (via CUDA) to share data between threads of the same block. Good use of shared memory is key to the performance of many optimized kernels (e.g., GEMM in "tiling").

— **L2 Cache :** A larger cache (40 MB on the A100), unified and shared by all SMs. It is used to capture memory accesses with broader locality. Our analytical model includes a factor f_{L2} (Eq. (4.13)) to capture its efficiency.

— **Main Memory (HBM2e) :** Large memory (40 or 80 GB on the A100) with very high bandwidth (up to ∼2 TB/s), but with a much higher latency than caches. This is the source of most *memory-bound* bottlenecks.

Interconnection. For multi-GPU training, the A100 uses *NVLink 3.0*, a very high-bandwidth point-to-point interconnect (600 GB/s aggregate per GPU), much faster than the standard PCIe bus. This is crucial for the efficiency of the gradient *All-Reduce* operation, whose time (T_{comm}) is a limiting factor at large scale.

4.3.2.3 Software Stack and CUDA Programming Model

The hardware is exploited through a complex software stack.

CUDA : NVIDIA's programming model. A program to be executed on the GPU is a *kernel*, written in C++/CUDA. At launch, the programmer defines a **grid** of **blocks** of **threads**. The GPU's hardware scheduler maps the blocks to the SMs (a block executes entirely on a single SM), and the threads of a block to the cores of that SM. This abstraction allows the code to be scalable on GPUs with different numbers of SMs.

Optimized Libraries : Almost no one rewrites a GEMM in CUDA. Highly optimized NVIDIA libraries are used, such as *cuBLAS* for linear algebra, and especially *cuDNN (CUDA Deep Neural Network library)* which provides kernel implementations for all standard DL operations (convolutions, activations, normalizations, etc.).

High-Level Frameworks : Frameworks like **PyTorch** and **TensorFlow** rely on these low-level libraries. A call to `torch.nn.Linear` internally triggers one or more calls to cuBLAS functions optimized for the target hardware. Compilers like **XLA (Accelerated Linear Algebra)** from JAX/TensorFlow can also optimize the computation graph, by **fusing multiple operations** into a single GPU kernel to reduce round trips to memory.

This software stack creates a powerful abstraction, but also a distance between the user-written code and its actual execution on the hardware, which is a major challenge for analytical modeling.

4.3.2.4 Motivation and Justification of Our Approach

This technological complexity has direct implications that justify the design choices of our framework :

1. **Need for Granular Characterization :** The heterogeneity of compute units (TCU, ALU) and their radically different peak performances make a simple total FLOPs metric invalid. It is **essential to decompose the workload** into operation types ($F_{\text{GEMM-TC}}, F_{\text{ALU}}$), as our `WorkloadCalculator` does.
2. **The Gap from Peak Performance and Saturation :** Maximum performance is only achieved if the data is perfectly organized to saturate the compute pipelines and hide memory latencies. This depends on the **size and shape of the tensors**. A small GEMM will never saturate a Tensor Core. It is this phenomenon of **non-linear saturation** that our model seeks to capture through its `eff_features` and its saturation function (Eq. (4.10)). This is a central point of our contribution, which goes beyond simple linear models.
3. **The Central Role of Memory :** The bottleneck is often memory bandwidth. Modeling time is not just a matter of computation, but also of memory access. This is why our model decomposes time into T_{comp} and T_{mem} and models L2 cache efficiency (Eq. (4.13)).
4. **The Impact of Software Abstractions :** The software stack (PyTorch, XLA, cuDNN) chooses the kernel implementation, manages streams, etc. It is unrealistic to model every detail of cuDNN. That is why our model, although analytical in its structure, relies on

empirical calibration and scalars (like C_{op}, C_{BF16}...). These scalars "absorb" the complex inefficiencies and optimizations of the software stack that the structural model alone cannot capture.

5. **Architectural Specifics :** Models like ALBERT, with their weight sharing, create unusual memory access and computation patterns. A generic model will fail to predict them. The introduction of **model-dedicated scalars** ($S_{\text{mem,Albert}}$) is a pragmatic and powerful solution that our framework implements to handle this complexity.

4.3.2.5 Fundamental Characteristics of Deep Learning Workloads

To model the workload generated by training a Deep Learning model, it is essential to understand both the general algorithmic process and the computational anatomy of the most common model architectures. These elements dictate the sequence, nature, and volume of operations to be executed on the GPU.

The Training Algorithm : Stochastic Gradient Descent (SGD) and its Variants. The training of a neural network $\mathcal{M}$, parameterized by a set of weights θ, aims to minimize a loss (or cost) function $\mathcal{L}(\theta)$ over a vast training dataset. The almost universal algorithm for this task is Stochastic Gradient Descent (SGD) and its adaptive variants (e.g., Adam, AdamW). Training proceeds over several *epochs* (full passes over the dataset). Each epoch is divided into thousands of iterations, or *steps*, where the model processes a mini-batch of data.

The computation cycle of a single step, on a mini-batch of data (X, Y), is the core of the workload we aim to model. It breaks down into three fundamental phases, as illustrated in Algorithm 1.

Algorithm 1: Computation Cycle of a Standard Training Step (for a single GPU)

Data : Mini-batch (X, Y), model $\mathcal{M}$ with weights θ_t, loss function $\mathcal{L}$.
▷ `Phase 1: Forward Pass`
$\hat{Y} \leftarrow \mathcal{M}_{\theta_t}(X)$ ▷ `Propagate input data through the layers`
loss $\leftarrow \mathcal{L}(\hat{Y}, Y)$ ▷ `Calculate the model's error`
▷ `Phase 2: Backward Pass / Backpropagation`
`zero_grad`(θ_t) ▷ `Reset gradients`
loss.backward() ▷ `Calculate the gradient` $\nabla_\theta \mathcal{L}(\theta_t)$ `by backpropagation`
▷ `Phase 3: Weight Update (Optimizer Step)`
optimizer.step() ▷ `Update weights:` $\theta_{t+1} \leftarrow \theta_t - \eta \cdot$ `update`$(\nabla_\theta \mathcal{L}(\theta_t))$

Each of the phases has distinct computational and memory requirements, which map differently to the GPU architecture :

— **The Forward Pass :** Walking through the model's computation graph from input to output. It is mainly composed of matrix operations and convolutions, ideal for Tensor Cores. It requires reading the weights (θ_t) and inputs (X), and writes the **intermediate activations** to memory. This is the phase that is also executed during inference.

— **The Backward Pass :** Walking through the computation graph in reverse to compute the gradients of the loss with respect to each parameter, by applying the *chain rule*. The calculations are often similar in nature to the forward pass (e.g., a convolution in the forward pass leads to a transposed convolution in the backward pass), also heavily using the Tensor Cores. It is very memory-intensive because it must *re-read the activations* saved during the forward pass to compute local gradients.

— **The Optimizer Update :** Using the gradients to update the weights. For optimizers like AdamW, this step involves simple vector operations (additions, multiplications, square roots) executed on the ALU cores. It is primarily *limited by memory bandwidth*, as it must read and write the entire model weights and optimizer states (moments).

Our analytical framework reflects this decomposition by separately modeling the workload of the forward/backward pass (combining $T_{\text{comp,FB}}$ and $T_{\text{mem,FB}}$) and that of the optimizer (T_{opt}).

Computational Anatomy of Key Model Architectures. The types of computations performed during the forward and backward passes are dictated by the model's architecture. Here we analyze the two dominant families, CNNs and Transformers.

Convolutional Neural Networks (CNNs). CNNs, pillars of computer vision, exploit the spatial locality of data (images) through the *"convolution"* operation. A convolution layer applies a series of filters (or kernels) to the input image to produce *feature maps*.

— **Computational Requirements :** A convolution is a massively parallel operation. Mathematically, it is equivalent to a large number of dot products. In practice, for execution on GPUs, frameworks like cuDNN often transform it into a large matrix multiplication (GEMM) using techniques like `im2col`. This transformation makes the operation perfectly suited for *"Tensor Cores"*. The computational volume is proportional to the output image size, the number of input and output channels, and the filter size. This is the source of the $F_{\text{Conv-TC}}$ metric in our model.

— **Memory Access :** Convolutions have a higly structured but intensive memory access pattern. They benefit from high *data reuse* : the filter weights are reused for each pixel of the image, and the input image pixels are reused by neighboring filters. This spatial and temporal locality is the key to their efficiency, provided that the data can be kept in the GPU's fast memories (L1/L2 Cache). The structural features we model ($\bar{D}_{\text{Conv}}$) aim to capture whether a convolution's dimensions (number of channels, spatial size) are "large enough" to efficiently exploit this locality and saturate the hardware.

Transformer Architecture and Attention Mechanism. Transformers, which have revolutionized the natural language processing (NLP) and are expanding into other domains, are built on the *"self-attention"* mechanism. The central idea is to allow each element of a sequence (e.g., a word) to "look at" all other elements in the sequence to compute its own representation, by weighting the importance of each one. The canonical formula for scaled dot-product attention

is :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.3}$$

where Q (queries), K (keys), and V (values) are matrices obtained by projecting the input sequence. d_k is the dimension of the keys.

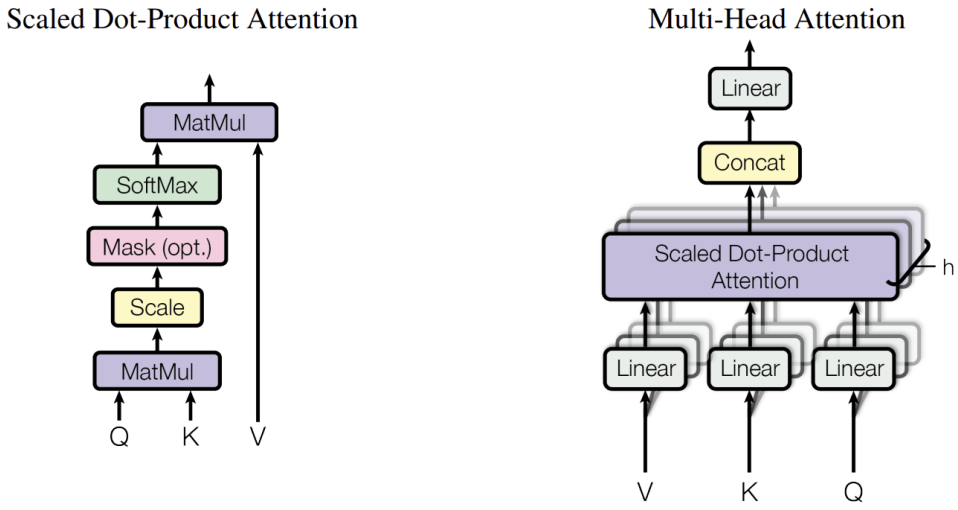


FIGURE 4.4 – Classical View of the Attention Mechanisms [Vaswani, 2017b]

The analysis of formula 4.3 and its implementation in multi-head attention reveals the hardware implications :

— **Computational Requirements :** The attention mechanism is dominated by two *batched matrix multiplication (BMM)* operations, QK^T and (scores)V, as illustrated in Figure 4.4. These BMMs are GEMM-type operations, and are therefore massively accelerated by the *"Tensor Cores"*. The other operations (initial and final linear projections) are also GEMMs. The non-matrix calculations (softmax, normalization) are handled by the ALU cores. This is the justification for our $F_{\text{GEMM-TC}}$ metric which dominates the workload of Transformers.

— **Quadratic Bottleneck :** The main challenge of Transformers is that the computation of the attention matrix QK^T and its associated memory accesses have a complexity of $\mathcal{O}(L^2 \cdot d_k)$, where L is the sequence length. This *quadratic* dependence on sequence length causes the computational load and memory consumption (for storing the score matrix) to explode for long texts. Modeling this precisely is essential.

— **Memory Access :** Unlike convolutions, attention has a global and less structured memory access pattern. Each element of the sequence must potentially access all others. This can lead to under-utilization of caches if the sequence is long, making performance very sensitive to memory (HBM) bandwidth. This is why a precise model must clearly distinguish between T_{comp} and T_{mem}.

Impact of Multi-GPU Execution. Training large models (deep CNNs, and especially Transformer-based LLMs) requires several GPUs. The most common parallelization strategy is *Data Parallelism*, implemented via `torch.nn.parallel.DistributedDataParallel` (DDP).

— **Principle :** Each GPU hosts a full replica of the model. The global mini-batch of data is divided among the GPUs, and each processes its portion in parallel (see Fig. 4.5 for the general flow).

— **Bottleneck : *All-Reduce* Communication.** After the local gradients are computed on each GPU, these gradients must be averaged across all GPUs before the optimizer can perform an identical weight update on all replicas. This synchronization step is done via a collective communication operation, typically ***All-Reduce***. The time spent in this communication (T_{comm}) represents a direct overhead to the total step time. It depends on the size of the model (volume of gradients to communicate), the number of GPUs, and the interconnect bandwidth (NVLink). That is why T_{comm} is an explicit and fundamental term in our time model (4.7).

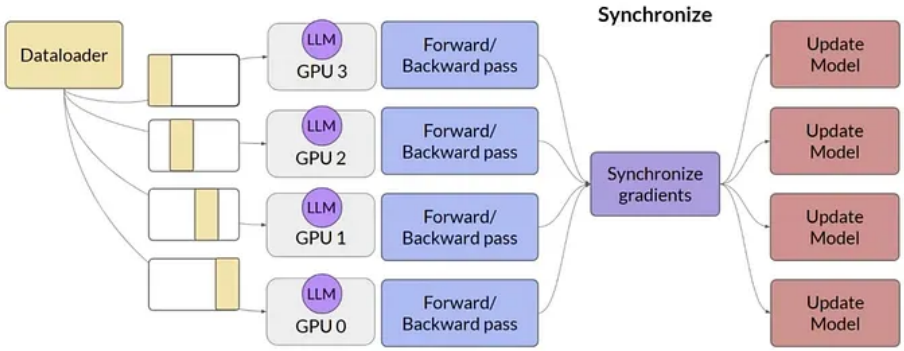


FIGURE 4.5 – Distributed Data Parallel (DDP) training general view. Credit to *Saurabh Naik* [1]

In summary, accurate modeling must imperatively capture the nature of dominant operations (GEMM, Conv), their dimensional dependencies (e.g., L^2 for attention), the balance between computation and memory access, and the overhead of communication in distributed mode. The design of our analytical framework (the so-called **ADEPT : Analytical Deep-learning Energy and Performance Time-estimator**) is built upon this foundation of technological understanding.

4.3.3 Architecture of Our Framework

The methodology of our ADEPT [2] framework is structured around a two-phase conceptual pipeline, as illustrated in Figure 4.6. The first phase (Workload Characterization) takes a DL PyTorch model as input and, via the `WorkloadCalculator` module, produces a quantitative workload profile $\mathcal{W}$. The second phase (Performance & Energy Projection), handled by the

1. https ://ai.plainenglish.io/taming-computational-challenges-with-quantization-ddp-and-fsdp-26e71b60534b
2. https ://github.com/HPC-CRI/ADEPT

`AnalyticsEnergy` module, uses this profile and a set of hardware parameters θ_H to predict the time per step T_{step} and the average power $\bar{P}_{\text{sys}}$.

Phase 1 : Workload Characterization (`WorkloadCalculator`). This module acts as an advanced static profiler. For a model $\mathcal{M}$ (`torch.nn.Module`) and a given input sample X, it explores the computation graph to extract a workload profile $\mathcal{W}$.

Principle : Modern neural network architectures can contain operations whose output tensor shapes dynamically depend on the input shapes (e.g., adaptive pooling, operations on variable-length sequences after padding). A purely static code analysis is therefore often insufficient. To overcome this obstacle, our framework performs a single, non-perturbative *dummy forward pass* on the model with a representative input sample. This pass is instrumented with PyTorch "hooks", which are callback functions registered on each relevant module. Whenever a module is executed, its hook is called, allowing us to capture and store the exact shapes of the input and output tensors in a lookup table that associates each module instance with its actual dimensions. Once the shape table is created, the `WorkloadCalculator` traverses the model's computation graph recursively. The analysis logic is encapsulated in an extensible library of **handlers**. A handler is a specialized Python function designed to calculate the workload of a specific layer type (`nn.Linear`, `nn.Conv2d`, `LlamaAttention`, etc.). A global hash table, `LAYER_WORKLOAD_MAP`, maps each PyTorch module class to its corresponding handler. When a handler is called for a composite module (e.g., a Transformer block), it in turn calls the handlers of its sub-modules and aggregates their results. This hierarchical design, allows for managing the complexity of modern models and ensures the framework's extensibility : to support a new model architecture, one simply needs to implement the handlers for its new composite modules.

Phase 2 : Performance & Energy Prediction (`AnalyticsEnergy`). This module implements the analytical core of the framework. It takes as input the workload profile $\mathcal{W}$ and a parameter vector θ_H describing the hardware platform (e.g., *peak performance, bandwidth*) and calibration factors that modulate these theoretical values. It then applies a series of equations to predict the execution time T_{step} and the average power $\bar{P}_{\text{sys}}$.

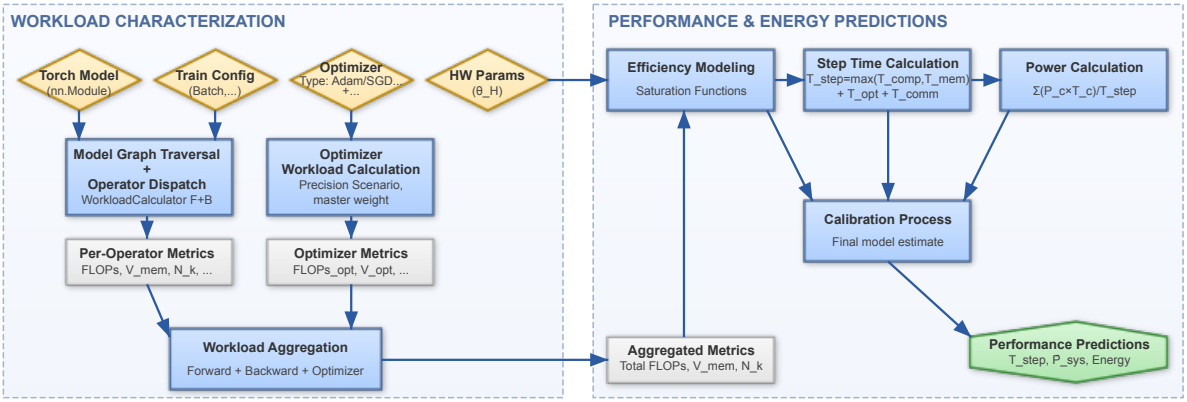


FIGURE 4.6 – Conceptual flow of the analytical prediction framework.

4.3.4 Detailed Workload Characterization

The accuracy of the final predictions is conditioned by the finesse of this analysis phase. For each atomic operation or composite module, the `WorkloadCalculator` quantifies the following aspects.

Floating-Point Operations (FLOPs) Decomposition. FLOPs are not only counted but also categorized according to the GPU functional unit they are likely to use :

— $F_{\text{GEMM-TC}}$: FLOPs for General Matrix-Matrix Multiplications on the Tensor Cores, NVIDIA's specialized units.

— $F_{\text{Conv-TC}}$: FLOPs for 2D convolution operations, also targeted by the Tensor Cores.

— F_{ALU} : FLOPs for general-purpose arithmetic operations, executed on standard CUDA cores (e.g., *activation functions, normalizations, scalar operations*).

This distinction is paramount because the peak performances of these units are very different (e.g., TFLOPS for Tensor Cores vs. GFLOPS for ALUs). FLOPs are counted separately for the forward (f) and backward (b) passes.

Memory Access Volume (Bytes). The total number of bytes read from the main GPU memory (HBM) to the compute cores (V_r) and written from the cores to the HBM (V_w) is meticulously counted. This includes accesses to the model weights, intermediate activations (which may be saved and then re-read during the backward pass), and gradients.

GPU Kernel Calls Quantification. The number of distinct GPU kernels launched by the CPU is estimated. Each launch introduces a fixed latency ; this metric therefore allows modeling the overhead associated with orchestrating computations.

Analytical Characterization of the Optimizer. The optimizer's workload ($F_{\text{opt}}, V_{\text{opt}}$) is modeled analytically. For an optimizer like AdamW and a model with N_p trainable parameters, the operations are broken down. In a mixed-precision (AMP) scenario, the weights θ, the first and second moment estimates m_t and v_t are often stored in FP32 for stability. The update calculation $\Delta\theta_t$ is based on the gradients g_t (in reduced precision, e.g., BF16) and the moments. The update $\theta_{t+1} \leftarrow \theta_t - \eta\Delta\theta_t$ and those of the moments (m_{t+1}, v_{t+1}) are performed in FP32. The memory access volume is therefore :

$$V_{\text{read,opt}} \approx \underbrace{N_p \cdot \text{sizeof(FP32)}}_{\text{read } \theta_t} + \underbrace{N_p \cdot \text{sizeof(BF16)}}_{\text{read } g_t} + \underbrace{2 \cdot N_p \cdot \text{sizeof(FP32)}}_{\text{read } m_t, v_t} \tag{4.4}$$

$$V_{\text{write,opt}} \approx \underbrace{N_p \cdot \text{sizeof(FP32)}}_{\text{write } \theta_{t+1}} + \underbrace{2 \cdot N_p \cdot \text{sizeof(FP32)}}_{\text{write } m_{t+1}, v_{t+1}} \tag{4.5}$$

The number of FLOPs is estimated at $N_p \times 18$ for AdamW.

Analytical Characterization of Base Handlers. The calculation of these metrics for base layers is done analytically. For example, for a 'nn.Linear' layer ($y = xA^T + b$) with input shape (B, K_{in}) and output shape (B, N_{out}) :

- $F_{\text{GEMM-TC},f} = 2 \cdot B \cdot K_{\text{in}} \cdot N_{\text{out}}$. The 2 comes from one multiplication and one addition (Fused Multiply-Add).
- $F_{\text{ALU},f} = B \cdot N_{\text{out}}$ if a bias is added.
- $V_{r,f} = B \cdot K_{\text{in}} \cdot S_{dtype} + K_{\text{in}} \cdot N_{\text{out}} \cdot S_{dtype} + N_{\text{out}} \cdot S_{dtype}$ for the input, weights, and bias.
- $V_{w,f} = B \cdot N_{\text{out}} \cdot S_{dtype}$ for the output.
- For the backward pass, backpropagating the gradients involves additional matrix products, typically : $\delta_{\text{in}} = \delta_{\text{out}} A$ (computing the input gradient) and $\Delta A = x^T \delta_{\text{out}}$ (computing the weight gradient).

Similar analytical formulations are implemented for each supported layer type (*'nn.Conv2d', 'nn.LayerNorm', 'nn.Softmax'*, etc.).

Innovation : Structural Efficiency Features (`eff_features`). Our framework introduces an original concept : the quantification of the *structure* of the computation. The handlers do not just sum up the metrics ; they calculate aggregates for the entire model :

- $\bar{\mathbf{D}}_{\{\mathbf{M},\mathbf{K},\mathbf{N}\},\mathbf{GEMM}}$: The average size of the dimensions (M, K, N) of GEMM operations, weighted by the number of FLOPs of each operation. For example, for the M dimension, the weighted average over the set of I GEMM operations in the model is :

$$\bar{D}_M = \frac{\sum_{i=1}^{I} F_i \cdot D_{M,i}}{\sum_{i=1}^{I} F_i} \tag{4.6}$$

- $\bar{\mathbf{D}}_{\{\dots\},\mathbf{Conv}}$: Similar averages for the key dimensions of convolutions.
- $\mathbf{\Sigma}_{\text{elem}}$: The total number of elements processed by generic (non-GEMM/Conv) operations.

4.3.5 Detailed Parametric Model of Performance and Energy

The `AnalyticsEnergy` module uses the workload profile $\mathcal{W}$ and a parameter vector θ_H to make its predictions.

4.3.5.1 Time Prediction Model

The time for one optimizer step is modeled as a sum of distinct time components, potentially including G_{acc} gradient accumulation steps :

$$T_{step} = (\underbrace{\max(T_{\text{comp,FB}}, T_{\text{mem,FB}})}_{T_{\text{pass}}} \times G_{\text{acc}}) + T_{\text{opt}} + T_{\text{comm}} + T_{\text{kern}} + T_{\text{fixed}} \tag{4.7}$$

where $T_{\text{comp,FB}}$ and $T_{\text{mem,FB}}$ are the computation and memory access times for one forward and backward pass, respectively (their maximum assumes optimal overlap). T_{opt}, T_{comm}, T_{kern}, and

T_{fixed} represent the optimizer time, inter-GPU communication time, kernel launch overhead, and fixed system overheads.

Equation (4.7) thus formalizes the temporal decomposition, and we will now detail each term :

— $\mathbf{T}_{\text{comp,FB}}, \mathbf{T}_{\text{mem,FB}}$: The computation time for a pass is the sum of the times for each operation category T_{op}. The memory time is modeled by equation (4.13).

— $\mathbf{T}_{\text{comm}}$: For multi-GPU training (e.g., DDP), communication is dominated by the *All-Reduce* operation of gradients. For N_g GPUs in a *ring* topology, a classic model estimates this time as the sum of a latency and a bandwidth time :

$$T_{\text{comm}} \approx 2(N_g - 1)L_{\text{hop}} + 2\frac{N_g - 1}{N_g}\frac{\text{size(gradients)}}{BW_{\text{eff}}} \tag{4.8}$$

where L_{hop} is the per-hop latency and BW_{eff} is the effective bandwidth of the interconnect (e.g., NVLink), both being parameters in θ_H. The theoretical max BW_{eff} which represent the bus bandwidth is obtained by $Bus\_BW = Algorithm\_BW * (n-1)/n$ [3]. The algorithm bandwidth is higher than bus bandwidth because it represents the effective throughput of moving tensor, while bus bandwidth accounts for the *"overhead"* of the distributed algorithm where data moves multiple times across the network. So when NCCL-tests reports 220 GB/s [4] bus bandwidth for 4 Nvidia A100s, the algorithm bandwidth would be $\approx 293GB/s$ - which is much closer to the theoretical 300 GB/s expectation. The NVLink topology is shows on Figure 4.7, with 12 bidirectional 50GB/s NVlink bus for aggregated 600GB/s per GPU.

— $\mathbf{T}_{\text{kern}}$: The total kernel launch overhead is modeled as $N_k \cdot t_{\text{kern,avg}}$, where $t_{\text{kern,avg}}$ is an average latency per launch parameter in θ_H.

— $\mathbf{T}_{\text{fixed}}$: A constant parameter in θ_H capturing other fixed system overheads (CPU-GPU synchronization, etc.).

4.3.5.2 Detailed Modeling of Hardware Saturation

The modeling of efficiency E_{op} is at the core of our approach. As introduced in equations (4.9) and (4.10), efficiency depends on dimensional saturation. The saturation references R_d used in the saturation function $\mathcal{S}_{\text{sat}}$ are not arbitrary values. They are initialized and bounded during the calibration process, inspired by the known micro-architectural characteristics of the target, here the NVIDIA A100. For example, the "ideal tile dimensions" used to normalize the GEMM dimensions (e.g., *'ideal_m_tile=16, ideal_n_tile=8, ideal_k_tile=8'* for mixed precision) are directly inspired by the dimensions of the MMA (Matrix Multiply-Accumulate) operations of the A100 Tensor Cores ($16 \times 8 \times 8$ for FP16, $16 \times 8 \times 4$ for TF32). Similarly, for convolutions, the saturation references for channels and spatial dimensions are related to how optimized CUDA

3. https ://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md
4. https ://forums.developer.nvidia.com/t/interpretation-of-total-aggregate-bandwidth-for-hgx-a100/224173/4

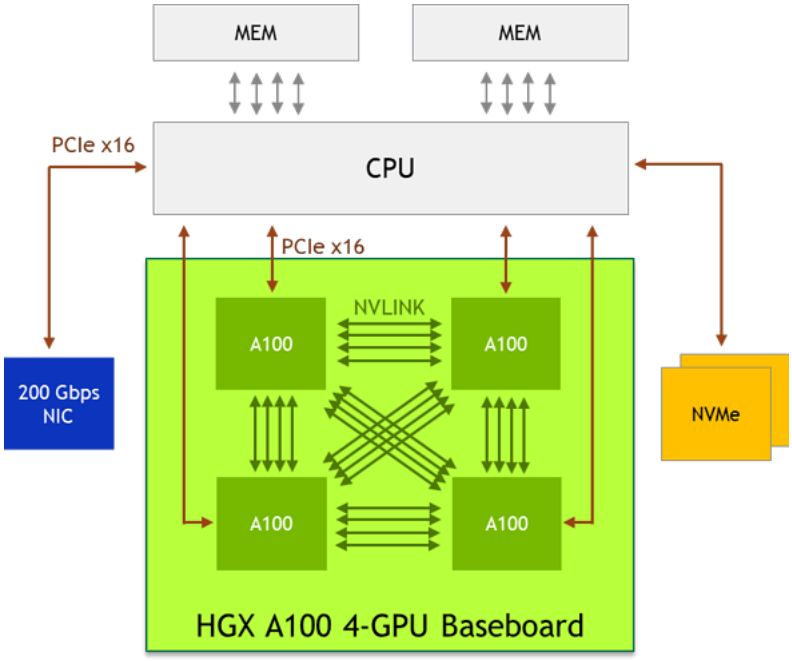


FIGURE 4.7 – Nvidia HGX as a 4 nodes fully connected graph. Source :Nvidia developer Forum

kernels ("implicit GEMM") decompose and organize work on the SMs (Streaming Multiprocessors). The exact values of R_d remain parameters to be learned, as they capture the complexity of interactions within the execution pipeline, but their orders of magnitude are guided by the architecture.

The calculation of the time for an operation type op (e.g., GEMM), T_{op}, is one of the keys to our model :

$$T_{op} = \left(\frac{F_{op}}{P_{op} \cdot E_{op}} \right) \cdot C_{op} \tag{4.9}$$

Here, F_{op} is the number of FLOPs and P_{op} is the theoretical peak performance of the hardware. The novelty lies in the efficiency factor E_{op}, which is modeled as :

$$E_{op} = \beta_{op} \cdot S_{op} \cdot \prod_{s \in \text{scalars}} S_{op,s} \tag{4.10}$$

Here, β_{op} is a base parameter of achievable efficiency, $\bar{D}_d$ are the weighted average dimensions (from the `eff_features` of $\mathcal{W}$), R_d are the learned saturation references, $\mathcal{S}_{\text{sat}}$ is our saturation function, and $S_{op,s}$ are the targeted calibration scalars for specific cases (e.g., ALBERT architecture). This formulation captures how execution efficiency depends on dimensional saturation, modulated by global and specific factors.

$$S_{op} = \prod_{d \in \text{dims}(op)} \mathcal{S}_{\text{sat}} \left(\frac{\bar{D}_d}{R_d} \right) \tag{4.11}$$

where the saturation for each dimension d of the operation (e.g., M, K, N for a GEMM) is calculated by applying a saturation function $\mathcal{S}_{\text{sat}}$ to the ratio of the weighted average dimension

$\bar{D}_d$ and a saturation reference parameter R_d learned during calibration. In our final model, we used the sigmoid function for its robustness :

$$\mathcal{S}_{\text{sat}}(x) = \frac{1}{1 + e^{-x}} \tag{4.12}$$

This formulation allows the model to capture the effect where execution efficiency on the GPU increases as problems become "large enough" to saturate the compute units and amortize latencies.

4.3.5.3 Modeling of L2 Cache Efficiency

The memory access time is modeled as proportional to the total data volume V_{mem} divided by the HBM memory bandwidth, but modulated by a factor f_{L2} representing the L2 cache efficiency. We model it analytically to avoid complex cache simulation :

$$T_{\text{mem}} = \frac{V_{\text{mem}}}{BW_{\text{HBM}}} \cdot \underbrace{\min(1, \max(\lambda_{\text{miss,base}}, \frac{W_{\text{set}}}{L2_{\text{size}}}))}_{f_{L2}} \cdot C_{\text{mem}} \tag{4.13}$$

where BW_{HBM} is the peak HBM bandwidth, W_{set} is an estimate of the operation's working set (its spatial-temporal locality), $L2_{\text{size}}$ is the size of the GPU's L2 cache (a hardware parameter), C_{mem} (Calibration factor for memory time), and $\lambda_{\text{miss,base}}$ is an incompressible minimum miss rate (a learned parameter). This model captures the simple idea that the larger the working set relative to the cache size, the higher the L2 miss rate, generating more traffic to the HBM. We could have considered the cache line size and the cache replacement policy, our model for this aspect is made simplier on purpose. Focusing on the L2 exclusively is because it seems to influence the most.

4.3.5.4 Power Prediction Model

The average system power is an average of the powers consumed during each time phase (*computation, memory, communication...*), weighted by the duration of these phases. The power of an active phase P_c is interpolated between the static power P_{static} and the maximum power P_{TDP}, based on a utilization factor U_c which represent the ratio of computation and memory times (see Eq. 4.14).

$$P_c = P_{\text{static}} + U_c \cdot (P_{\text{TDP}} - P_{\text{static}}) \cdot C_{P,\text{global}} \tag{4.14}$$

where $C_{P,\text{global}}$ is a global power calibration scalar. This approach causally links the power consumed to the nature of the workload as the ratio should be higher for Compute bound workload than the Memory bound ones.

Thus, the overall predicted average system power across all phases is derived as the time weighted average power of each phase (see Eq. 4.15).

$$\bar{P}_{\text{sys}} = \frac{\sum_{c \in \text{phases}} P_c \cdot T_c}{T_{step}} \tag{4.15}$$

4.3.6 Model Calibration Process

The determination of the set of ∼28 parameters θ_H in our model is performed by minimizing the discrepancy between the predictions and empirical measurements on a diverse reference dataset. The workload profiles (W_i) for each empirical data point i encompassing the measurements were pre-calculated once for each unique combination (model, input shape, optimizer, computation precision) using the `WorkloadCalculator` to speed up the fitting process.

Objective Function. We seek the parameter vector θ_H^* that minimizes a composite cost function $\mathcal{L}(\theta_H)$, defined as the weighted average of the Mean Absolute Percentage Error (MAPE) for time and power :

$$\text{MAPE}_{\text{time}}(\theta_H) = \frac{1}{N_D} \sum_{i=1}^{N_D} \left| \frac{T_{\text{step,pred},i}(W_i, \theta_H) - T_{\text{step,emp},i}}{T_{\text{step,emp},i}} \right| \times 100\% \tag{4.16}$$

$$\text{MAPE}_{\text{power}}(\theta_H) = \frac{1}{N_D} \sum_{i=1}^{N_D} \left| \frac{\bar{P}_{\text{sys,pred},i}(W_i, \theta_H) - \bar{P}_{\text{sys,emp},i}}{\bar{P}_{\text{sys,emp},i}} \right| \times 100\% \tag{4.17}$$

$$\mathcal{L}(\theta_H) = w_{\text{time}} \cdot \text{MAPE}_{\text{time}}(\theta_H) + w_{\text{power}} \cdot \text{MAPE}_{\text{power}}(\theta_H) \tag{4.18}$$

where the weights w_{time} and w_{power} (set to 0.6 and 0.4 in our experiments) allow for prioritizing accuracy on either metric.

Optimization Algorithm. Faced with the complexity of the parameter search space, we use ***Differential Evolution (DE)*** [Storn, 1997], a metaheuristic and stochastic optimization algorithm that is robust for global problems. Its principle is summarized in Algorithm 2.

The implementation of algorithm 2 (via `scipy.optimize.differential_evolution` [SciPy-Docs, 2008]) was used to find the parameter set θ_H^* that minimizes our cost function and following are main parameters used for the fitting process :

— **Strategy :** The *best1bin* strategy was selected due to its generally robust performance, balancing convergence speed with exploration of the parameter space.

— **Population Size :** A population size multiplier of 15 was used relative to the number of parameters ($|\theta_H| \approx 28$), resulting in a substantial population for each generation.

— **Maximum Iterations :** Set to 50,000 to ensure thorough exploration, although convergence was typically achieved much earlier in some cases. The default bound in scipy is (0.5, 1.0).

— **Mutation Factor (F) :** Allowed to vary between 0.5 and 1.5 to encourage population diversity and aid in escaping local minima.

— **Recombination Probability (CR) :** Set to 0.7, a commonly used value that balances the creation of new candidate solutions with the preservation of existing good ones.

— **Initialization Strategy :** Latin Hypercube sampling was employed for the initial population to promote uniform coverage of the defined search space.

Algorithm 2: Differential Evolution (DE) for Model Calibration

Input : Population size NP, max generations $G_{\max}$;
Mutation factor $F \in [0, 2]$, crossover probability $CR \in [0, 1]$;
Objective cost function $\mathcal{L}(\theta_H)$;
Parameter bounds $[\theta_{H,\min}, \theta_{H,\max}]$;
Output: Best parameter vector θ_H^* found.

▷ `Initialize a population of NP candidate parameter vectors`
Population $\mathcal{P} \leftarrow$ InitializePopulation(NP, $[\theta_{H,\min}, \theta_{H,\max}]$);
for $g \leftarrow 1$ **to** $G_{\max}$ **do**
  for $i \leftarrow 1$ **to** NP **do**
    ▷ `Select three distinct random individuals from the population`
    $r_1, r_2, r_3 \leftarrow$ SelectRandomIndices(i, NP);
    ▷ `Perform mutation to create a mutant vector`
    $v_i \leftarrow \mathcal{P}_{r_1} + F \cdot (\mathcal{P}_{r_2} - \mathcal{P}_{r_3})$;
    ▷ `Perform binomial crossover to create a trial vector`
    $u_i \leftarrow$ Crossover(target_vector=$\mathcal{P}_i$, mutant_vector=v_i, prob=CR);
    ▷ `Enforce parameter bounds on the trial vector`
    ClipToBounds(u_i, $[\theta_{H,\min}, \theta_{H,\max}]$);
    ▷ `Selection: keep the vector with the lower cost`
    if $\mathcal{L}(u_i) < \mathcal{L}(\mathcal{P}_i)$ **then**
      $\mathcal{P}_i \leftarrow u_i$;
▷ `Return the best individual found in the final population`
$\theta_H^* \leftarrow$ GetBestIndividual($\mathcal{P}$, $\mathcal{L}$);
return θ_H^*;

— **Parameter Bounds ($\theta_{H,\min}, \theta_{H,\max}$) :** Each parameter in θ_H was constrained within a physically plausible or empirically informed range, refined through preliminary fitting experiments. For instance, base hardware achievement fractions (β_{op}) were constrained to $[0, 1.0]$ to reflect their interpretation as a fraction of theoretical peak, while time calibration factors (e.g., $C_{op,\text{time}}$) were assigned wider bounds (e.g., $(10^{-4}, 50.0)$ for GEMM-related time calibration) to allow the model to capture significant deviations if the structural components did not fully account for observed performance.

— **Solution Polishing :** The best solution identified by the DE algorithm was subsequently refined using a local optimization method (L-BFGS-B) to enhance the precision of the final parameter set.

4.3.7 Experimental Validation

To calibrate θ_H, we generated an empirical dataset comprising performance and power measurements from training a diverse suite of six advanced DL models on an NVIDIA A100 GPU cluster. The Differential Evolution algorithm was employed to minimize the cost function combining the MAPE for time and power (Eq. 4.7 and 4.15).

Our benchmark suite was designed to encompass a diverse range of contemporary DL architectures. The key training configuration parameters for each model are summarized in Table 4.1.

These parameters were selected based on common practices in the literature to ensure representative training dynamics. For all datasets, standard training splits were used, with subset sizes chosen to allow for the total completion of our experiments within a practical time budget while still capturing thousands of training steps post-warmup. Vision datasets (CIFAR-10 [Krizhevsky, 2009a]) did not undergo data augmentation, while text datasets (GLUE/SST-2 [Socher, 2013 ; Wang, 2019a], Wikitext-103 [Merity, 2016]) were appropriately tokenized for each model, with a sequence length of 256 for ALBERT-base-v2 and DistilBERT-base and 512 for TinyLlama-1.1B.

TABLE 4.1 – Benchmark suite and key training configuration parameters for each DL model.

Model	Dataset	Samples	Epochs	Batch /GPU	Eff. Batch (1GPU)	Total Steps (1GPU)	Eff. Batch (4GPU)	Total Steps (4GPU)
ALBERT[Lan, 2019]	GLUE/SST2	67,349	5	128	128	1,320	512	330
DistilBERT[Sanh, 2019]	GLUE/SST2	67,349	5	128	128	1,320	512	330
Resnet50[He, 2015b]	CIFAR-10	50,000	10	256	256	1,960	1,024	490
TinyLlama[Zhang, 2024]	Wikitext-103	380,000	3	16	128	8,907	512	2,229
VGG16[Simonyan, 2015b]	CIFAR-10	50,000	10	256	256	1,960	1,024	490
ViT-B/16[Dosovitskiy, 2021]	CIFAR-10	50,000	10	256	256	1,960	1,024	490

Hardware and Software Environment

All empirical training runs were executed on a HPC node equipped with four (4) NVIDIA A100-SXM4-40GB GPUs, interconnected via NVLink v3.0 with 600 GB/s aggregate bandwidth per GPU. The host system comprises single AMD EPYC 7513 processors and 512 GB of DDR4 memory.

The software stack utilized was as follows : Operating System : Debian 5.10.209 ; CUDA Toolkit : Version 12.1 ; cuDNN : Version 8.9 ; PyTorch : Version 2.4 (compiled with CUDA support) ; Transformers Library (Hugging Face) : Version 4.45 ; TorchVision : Version 0.19.1+cu12

Multi-GPU training was implemented using PyTorch's Distributed Data Parallel (DDP) module, which follows the Single-Program Multiple-Data (SPMD) paradigm.

Conceptual Model : The DDP strategy is a form of data parallelism where a replica of the entire model is loaded onto each participating GPU. Each GPU process (referred to by its 'rank') receives a unique, non-overlapping fragment of the global data batch for each training step. During the forward pass, each model replica independently computes its output and loss based on its own data. A critical synchronization step occurs during the backward pass. As gradients are computed locally on each GPU, DDP transparently hooks into the backpropagation engine (Autograd) to trigger a collective communication operation (typically an **All-Reduce**) across all GPUs. This operation sums up the gradient tensors from all replicas and redistributes the averaged result, ensuring that every GPU has an identical copy of the final gradients before the optimizer step. This synchronization guarantees that the model weights on each replica remain identical after every step, as if trained on a single GPU.

Implementation : Our framework uses standard PyTorch primitives to instantiate this model. For each training run, the main script uses `torch.multiprocessing.spawn` to launches N_g processes, with each process assigned a unique rank from 0 to $N_g - 1$. Each process initializes

the distributed communication group via `torch.distributed.init_process_group`, using the high-performance NCCL backend for inter-GPU communication over NVLink. The model is then wrapped with the `torch.nn.parallel.DistributedDataParallel` module. The training dataset is partitioned using a `data.distributed.DistributedSampler` from `torch.utils`, which is passed to the `DataLoader` to ensure that each model replica receives a distinct data shard. This robust setup ensures correct and efficient multi-GPU data parallel training.

Training Configurations and Empirical Data Collection

Each model in the benchmark suite was trained under a matrix of configurations, yielding $N_D = 32$ distinct empirical data points used for model calibration and subsequent validation. The varied configurations included :

— **GPU Counts (N_g) :** Single GPU (1) and multi-GPU (4) DDP setups.

— **Precision Modes :** Full single-precision (FP32) ; TF32-accelerated FP32 (selectively enabled on GPUs for qualifying operations) ; And Mixed Precision BF16 (*torch.amp.autocast* with *dtype=torch.bfloat16*).

— **Model-Specific Hyperparameters :** Key training hyperparameters such as batch size per GPU (B_s), sequence length (for Transformer models), image input size (for vision models), learning rate, optimizer type (Adam/AdamW or SGD variants), number of training epochs, and gradient accumulation steps (G_{acc}) were systematically set based on values established in literature or defined in our benchmark configuration file. All models were trained for a fixed number of epochs deemed sufficient to observe stable training behavior beyond initial system warmup phases.

The collection of empirical performance and power data was performed as follows.

Step Time ($T_{\text{step,emp}}$) : The wall-clock duration per optimizer step (appropriately accounting for any gradient accumulation) was measured. This duration was averaged over a substantial number of steps (typically hundreds to thousands, excluding an initial warmup period to allow for system stabilization and JIT compilations) during each training run.

Average System Power ($\bar{P}_{\text{sys,emp}}$) : Node-level system power consumption was recorded throughout each training benchmark using the EA2P (Energy Aware Application Profiler) framework [Nana, 2024c]. The value $\bar{P}_{\text{sys,emp}}$ was calculated as the average power consumed during the stable training phase, precisely corresponding to the period over which $T_{\text{step,emp}}$ was measured.

Workload Metrics (W_i) : The detailed workload profiles for each configuration i, serving as input to the framework, were generated once using the `WorkloadCalculator` module.

The experimental protocol we have described led to a robust empirical dataset to calibrate the set of parameters θ_H and to subsequently evaluate the prediction accuracy of our framework. We now present and analyze our experimental results.

4.3.7.1 Progressive Modeling Process

The development followed a *progressive refinement* process. As shown in Table 4.2, the gradual introduction of targeted calibration scalars was fundamental for accuracy. The base model, having only global parameters, showed a combined error of 15.37%. Adding scalars for ALBERT (notably a major factor for memory time) helped reduce the error on that model. But the most spectacular gain came from adding the time scalar for GEMMs in BF16 (the overall MAPE dropping from 12.78% to 4.50%), which drastically improved the prediction for the TinyLlama LLM. The final calibration of the other BF16 scalars led to the final model, with a *global combined MAPE of 4.14%*.

TABLE 4.2 – Experimental Results of our Progressive Prediction Procedure.

Model Calibration Stage	MAPE (%)		
	Time	**Power**	**Combined**
S1 : Base Model (Global Params Only)	22.47	4.73	15.37
S2 : + ALBERT GEMM	21.55	4.81	14.85
S3 : + ALBERT ALU	28.48	5.93	19.46
S4 : + ALBERT Memory	18.30	4.50	12.78
S5 : + BF16 GEMM	3.76	5.60	4.50
S6 : + BF16 + GEMM + Conv	3.60	5.70	4.44
S7 : + BF16 + GEMM + Conv + ALU	3.05	5.78	**4.14**

The accuracy of the final model is excellent, as attested by the scatter plot of predicted versus actual values (Figure 4.8), where the points cluster tightly around the ideal diagonal for both time and power. The dashed line indicates perfect prediction (y=x). A finer analysis per model (Figure 4.9) shows that the error remains low and controlled for all architectures, whether they are CNNs, Vision Transformers, or BERT-like Transformers and LLMs. The results (Table 4.3) also confirm the model's robustness to scaling (1 vs. 4 GPUs) and to different precision modes, although BF16 precision, being more complex, shows a slightly higher MAPE (6.21%).

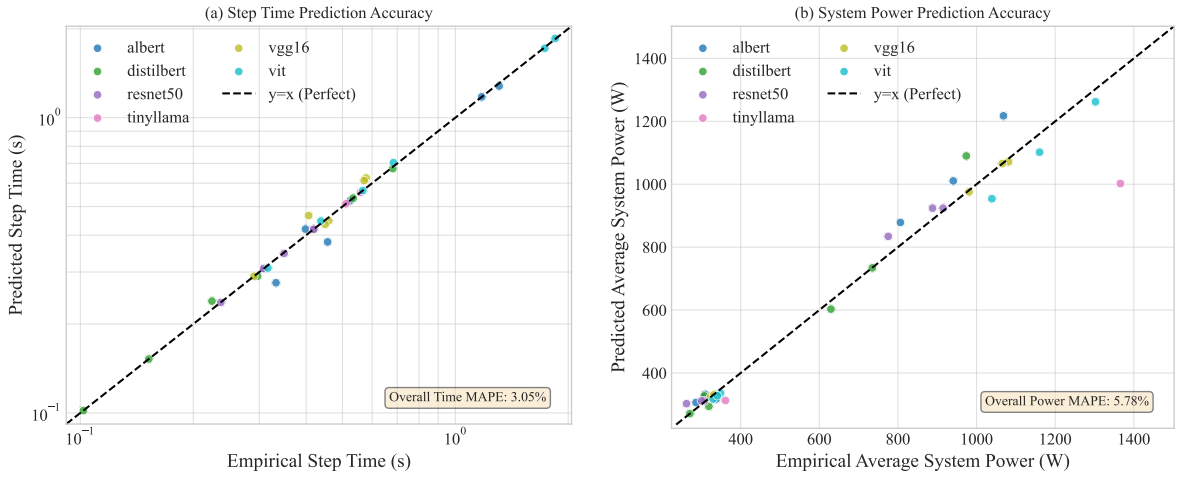


FIGURE 4.8 – Predicted vs. Empirical values for (a) Step Time and (b) Average System Power using the `sigmoid` model.

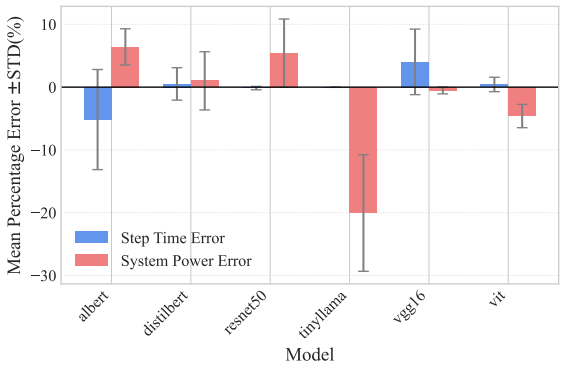


FIGURE 4.9 – Per-Model Prediction Error Distribution.

TABLE 4.3 – Overall Prediction Accuracy.

Model	MAPE	GPU Count	MAPE	Precision	MAPE
ALBERT	**7.51**	1 GPU	**3.21**	FP32	**2.99**
DistilBERT	**3.05**	4 GPUs	**5.07**	TF32	**2.81**
ResNet50	**2.30**			BF16	**6.21**
TinyLlama	**8.03**				
VGG16	**3.82**				
ViT	**2.60**				

4.3.7.2 Ablation Study and Saturation Function Sensitivity

Ablation studies (Figure 4.10) have a posteriori validated the necessity of the specialized scalars. Removing the ALBERT scalars caused its combined MAPE to jump from ∼7.3% to over 20%. Similarly, without the time scalars for BF16, the MAPE on the corresponding workloads went from ∼6.2% to over 29%. This proves that our hierarchical approach, combining a base analytical model with targeted and learned correction factors, is a powerful and justified strategy.

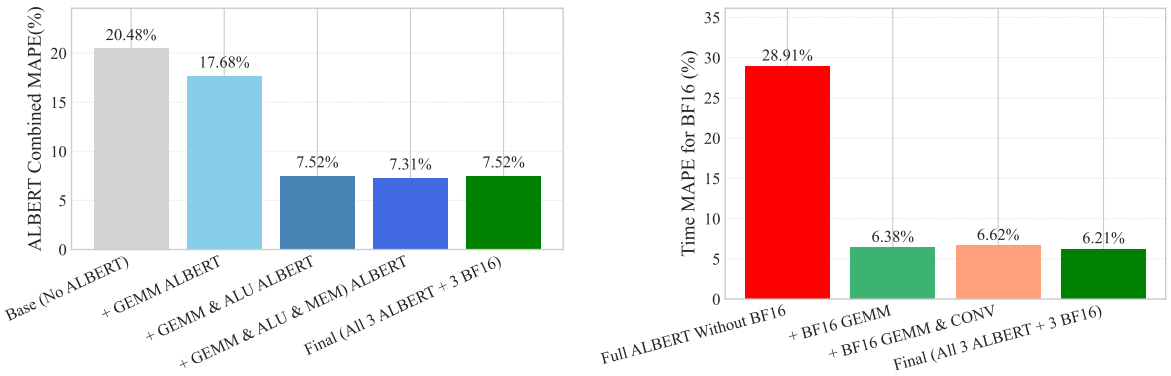


FIGURE 4.10 – Impact of specialized scalars on prediction accuracy based on the `sigmoid` parameter. (a) Combined MAPE for ALBERT workloads with/without its specialized scalars. (b) Average Time MAPE of mixed BF16 workloads with/without its specific time scalars.

Furthermore, while our primary considered model employs the `sigmoid` saturation function, we rigorously evaluated four alternatives : `arctan`, `softsign`, `algebraic`, and `tanh`. When fully calibrated, all these functions yielded excellent overall prediction accuracy (with combined MAPEs between 4.1% and 4.7%). A noticeable difference emerged in the interpretability of the fitted saturation reference parameters, which define the characteristic scale at which performance gains begin to diminish. Specifically, we analyzed the saturation reference for convolution channels, $R_{C,\text{Conv}}$ (*channel tiles saturation reference convolution block*), and for spatial dimensions, $R_{\text{Spatial,Conv}}$ (*spatial pixels saturation references for convolution images*).

Using `arctan` and `softsign` yields a consistent learning of tiny reference values for both (e.g., $R_{C,\text{Conv}} \approx 0.001$ normalized ideal channel blocks and $R_{\text{Spatial,Conv}} \approx 214$ pixels for the `arctan` model). Such small reference values mean that the model interprets hardware performance as saturating almost immediately as both channel count and feature map size increase from their baseline. In contrast, the `sigmoid`-based model converged to much larger and arguably more interpretable reference values ($R_{C,\text{Conv}} \approx \mathbf{2840}$ normalized ideal channel blocks, $R_{\text{Spatial,Conv}} \approx \mathbf{22860}$ pixels). For the `sigmoid` function, these large values indicate an understanding of the fact that hardware efficiency keeps scaling well over a wide range of channel depths and spatial dimensions before a noticeable saturation of the performance. This latter behavior is more consistent with our expectations with high-end GPUs designed to handle large-scale convolutions.

Therefore, we selected `sigmoid` as our primary model because it not only achieved a top-tier fit (objective value of **4.14**) but also yielded parameters that offer a more plausible physical interpretation of performance scaling for key convolution operations.

4.3.8 Strengths and Limitations of the Framework

The analysis of the learned parameters θ_H after calibration (detailed in the appendix of the article) is rich with insights. The small value of the GEMM's BF16 time scalar ($C_{\text{GEMM,BF16time}} \approx 0.037$) quantifies the spectacular efficiency of the A100 Tensor Cores for this type of operation. Conversely, the large time scalar for ALBERT's memory ($S_{\text{mem,Albert}} \approx 2.82$) quantitatively measures the memory inefficiency induced by its weight-sharing architecture. Similarly, the high value of the ALU time scalar ($C_{\text{ALU,time}} \approx 10.4$) reveals that "generalist" operations have a much higher relative cost than their mere FLOP count would suggest, likely due to fewer optimizations in the execution pipeline compared to GEMMs. These examples illustrate the *interpretive strength* of our approach.

In summary, the **strengths of the framework** are its high accuracy, broad applicability, interpretability, comprehensive workload characterization, modularity, and native integration of energy prediction. An accurate and interpretable performance and energy prediction framework like ours offers a number of potential benefits including the following :

— **Hardware/Software Co-design :** It might be useful to allow hardware and system designers to perform rapid “what-if” analyses for future specific features (e.g., memory bandwidth, L2 cache size, Tensor Core capabilities, low-level scheduling) by modifying

relevant θ_H parameters and evaluate their impact on a diverse range of DL workloads before prototyping.

— **Energy-Aware Resource Management :** HPC jobs can be scheduled more skilfully by optimizing for energy efficiency, managing power caps dynamically, or considering co-schedule of the workloads based on their predicted resource demands and power profiles.

— **Carbon Footprint Estimation and AI Sustainability :** Providing a more reliable energy consumption estimate and associated carbon footprint for large-scale DL is a valuable contribution to sustainable AI development and deployment.

— **DL Model Development and Optimization :** Assisting DL researchers in identifying performance bottlenecks within new or existing model architectures by highlighting which components (e.g., specific GEMM characteristics via `eff_features`, memory access intensity, ALU-bound operations) dominate predicted execution time, thereby guiding targeted optimization efforts, is worth considering.

— **Educational and Research Tool :** A platform for understanding the complex interplay between DL algorithms, software execution, and underlying hardware architectural characteristics.

Although the good accuracy of our prediction model and procedure, we list some limitations that can be considered for future contributions.

Parameters Complexity : The model utilizes a substantial number of fitted parameters ($\sim$28). While enabling adaptability, this complexity can raise concerns about the parameter identification or potential correlations, where different combinations might yield similar aggregate accuracy. Efforts to reduce the dimension of the model through sensitivity analysis or by deriving more parameters directly from hardware specifications would be beneficial.

Generalizability to Hidden Hardware/Software Stacks : Applying our framework to other GPU architectures, or to consider other software environments (new compilers, CUDA versions, PyTorch releases), would likely require a re-calibration of θ_H. A methodology for quick or transfer-learning-based parameter re-calibration whenever needed is worth investigating.

Dynamic Runtime Effects : The current model assumes that the workload characteristics are relatively stable at the level of a step, based on static graph analysis and the input layout. It does not explicitly model dynamic runtime phenomena such as highly variable sequence lengths within mini-batches (beyond padding), the impact of dynamic data sparsity, or just-in-time (JIT) compilation effects.

Granularity of the Power Prediction Model : The current power prediction model, while showing a satisfactory level of global accuracy, relies on utilization factors that are averaged over the operational phases. More fine-grained linkage to specific instructions, hardware performance counters (PMCs), or distinct power states of GPU sub-components could further enhance the accuracy of the energy predictions, maybe at the expense of an increasing model complexity.

This contribution proposes ADEPT, an *analytical framework* for performance and energy, whose originality lies in its unique combination of three pillars : (1) a hierarchical workload

characterization that integrates structural features (like the weighted average dimensions of operations), (2) a performance model based on non-linear saturation effects inspired by the hardware architecture, and (3) an iterative calibration methodology using targeted scalars to capture the nuances of modern models and precision modes. It has been demonstrated that this approach achieves high predictive accuracy while retaining a degree of explainability far superior to that of *"black-box"* models, thus constituting a valuable tool for the analysis, design, and operation of a more efficient and sustainable large-scale Artificial Intelligence.

4.4 Conclusion

This chapter has been dedicated to one of the fundamental pillars of our approach to energy efficiency : the **predictive modeling of consumption**. Moving beyond the simple descriptive measurement presented in Chapter 3, the work presented here aimed to build mathematical models capable of anticipating energy consumption, thus providing an indispensable basis for proactive optimization and scenario analysis. In line with our general philosophy, we have favored analytical or semi-analytical approaches, promoting interpretability and an understanding of physical and architectural phenomena, as opposed to purely empirical *black-box* models.

Two original and complementary contributions were presented. The first focused on modeling an individual, often under-instrumented component : **the main random-access memory (DRAM)**. Aware that DRAM represents a non-negligible part of the total node consumption and that it constitutes a "blind spot" for many in-band measurement tools, we developed and validated a pragmatic analytical model. Based on easily accessible system information, this model, integrated into our EA2P tool, allows for providing a reasonable estimate of the energy consumed by RAM, thus offering a more holistic and accurate view of the system's energy balance.

The second and main contribution of this chapter was the design and validation of a *comprehensive analytical framework for predicting the performance and energy of Deep Learning model training on GPU architectures*. This much more ambitious approach aims to capture the complexity of the interactions between modern AI model architectures (CNNs, Transformers, LLMs), the execution software stack, and the micro-architectural subtleties of massively parallel GPUs. The originality of ADEPT lies in its combination of three fundamental pillars :

1. A **hierarchical and detailed workload characterization**, which quantifies not only classic metrics (FLOPs, memory access) but also **structural efficiency features**, such as the weighted average dimensions of operations, which capture how the computation is organized.
2. A **parametric performance model** that explicitly integrates effects of **non-linear saturation** of hardware efficiency, based on functions inspired by the GPU micro-architecture.
3. A **robust and iterative calibration methodology**, using Differential Evolution and *targeted correction scalars* to refine the model's accuracy by taking into account architectural specificities (e.g., ALBERT) and precision modes (e.g., BF16).

The validation of this framework on a wide range of models and configurations demonstrated its high accuracy (combined MAPE of 4.14%), while preserving a strong degree of interpretability of the learned parameters. It constitutes a powerful tool for analyzing, understanding, and anticipating the requirements of AI applications, and a major asset for guiding their optimization.

In conclusion, this chapter has established the feasibility and relevance of an analytical modeling approach, both at the component and the complex application level. The models developed are not an end in themselves but rather an indispensable means for the ultimate step of our research : ***optimization***. They arm us with the predictive tools necessary to evaluate, simulate, and guide optimization strategies without resorting to exhaustive and costly experiments. It is on this solid foundation that the next chapter will build to present our contributions in dynamic energy optimization, exploring how intelligent scheduling and real-time resource control can exploit the knowledge gained to concretely minimize energy consumption while respecting performance constraints.

Chapter 5

Energy-Aware Scheduling Strategies for Deep Learning Workloads

Ce chapitre est l'aboutissement de notre démarche : le passage à l'optimisation active, incarnée par l'ordonnanceur de tâches. Il aborde la complexité de l'ordonnancement dans les clusters d'IA hétérogènes en introduisant d'abord notre seconde contribution logicielle : ***EAS-Sim****, un simulateur par événements discrets de haute-fidélité, qui utilise notre modèle analytique comme oracle pour permettre une évaluation rapide et reproductible des stratégies. La contribution méthodologique centrale est ensuite présentée : une approche de co-conception d'ordonnanceurs basée sur l'injection de l'****Energy-Delay Product (EDP)*** *comme primitive de scoring. L'application de cette méthode donne naissance à une suite de politiques innovantes :* ***Zeus*** *(énergie-performance),* ***Hades*** *(énergie-équité),* ***AuraChronos*** *(énergie-SLA avec préemption), et* ***Charon*** *(contrainte de budget). L'évaluation expérimentale approfondie via EAS-Sim démontre quantitativement les gains : Zeus réduit l'énergie de 10.5% sans perte de performance ; AuraChronos atteint une conformité SLA quasi-parfaite, impossible sans préemption ; et l'EDP se révèle être une heuristique de performance globale plus robuste que le temps seul.*

5.1 Introduction

The previous chapters have established the necessary foundations for our study : a methodology and a tool for the **fine-grained measurement** of energy (Chapter 3), and a framework for the **predictive modeling** of the performance and consumption of AI applications (Chapter 4). Equipped with the ability to measure and predict energy consumption, we now address the core of our problem : **optimization**. In modern HPC and Cloud environments, where hundreds or thousands of Deep Learning jobs compete for computing resources, the **job scheduler** is the conductor whose decisions determine not only the overall performance of the system (throughput, response time) but also, crucially, its global energy efficiency.

As our state-of-the-art review has shown (Section 2.5), traditional schedulers for DL, such as Pollux or Themis, were designed with a focus on single objectives like performance (minimizing

job completion time) or fairness (ensuring a just distribution of resources), while largely ignoring the energy dimension. This approach is no longer sustainable. The objective of this chapter is to present our contributions to the design and evaluation of a new generation of schedulers for DL that are intrinsically **multi-objective and energy-aware**.

Before detailing our contributions, we will, in a preliminary section, lay the groundwork for the scheduling problem in AI clusters, dissecting its complexity and the multiple conflicting objectives that govern it. This in-depth analysis will motivate the need for a flexible experimentation framework.

To explore this complex design space, simple experimentation on real systems would be prohibitively time-consuming and costly. That is why our first contribution in this chapter is the development of `EAS-Sim`, a **high-fidelity, extensible, discrete-event simulation framework**, designed specifically for the study of scheduling policies for AI workloads. This tool will allow us to rapidly and reproducibly evaluate a wide variety of strategies (Section 5.3).

Our main contribution is then a *systematic methodology for the co-design of energy-aware schedulers*. Its fundamental principle is the injection of the **Energy-Delay Product (EDP)** metric into the core of state-of-the-art heuristics to co-optimize energy alongside their original objectives. By applying this methodology, we have designed, implemented, and validated a suite of four new scheduling policies : **Zeus** (optimizing the energy-performance trade-off), **Hades** (energy-fairness), **AuraChronos** (guaranteeing SLAs in an energy-efficient manner via preemption), and **Charon** (operating under a strict power budget constraint) (Section 5.4).

This chapter also presents *a multi-faceted experimental evaluation* of the aforementioned new policies, conducted using EAS-Sim. We quantitatively demonstrate that it is possible to achieve substantial energy savings (≈8-10%) without sacrificing performance, that robust Quality of Service (QoS) guarantees are unattainable without preemption, and that optimizing for EDP is a remarkably robust heuristic for overall system efficiency (Section 5.5). We believe these results and the EAS-Sim framework itself constitute a significant advance towards the design of more intelligent and sustainable AI infrastructures.

5.2 Foundations of Scheduling for DL Clusters : A Complex Multi-Objective Problem

The scheduler is the operational orchestrator of a data center. It acts as an intelligent arbiter, allocating computing resources to jobs submitted by users. Historically, in the context of traditional High-Performance Computing (HPC), the objectives of scheduling were relatively clear : maximize system utilization and process jobs fairly, often with simple heuristics like FCFS (*First-Come, First-Served*) or weighted variations. However, the advent of Deep Learning workloads has profoundly changed this paradigm, introducing new dimensions of complexity and new conflicting objectives. Before presenting our simulator and our design methodology, it is essential to lay the foundations of this modern problem, to dissect its challenges, and to justify why a new generation of schedulers is now a necessity.

5.2.1 Specificity of the AI Cluster

AI-devoted computing infrastructures have some specificities in several key aspects that directly impact scheduling.

Hardware Heterogeneity : Whereas classic HPC clusters were often homogeneous, AI clusters are massively heterogeneous. They combine multiple generations and types of GPUs (e.g., NVIDIA A100, H100), with very different performance, memory, and energy consumption characteristics. A scheduler must therefore not only decide *when* to run a job, but also *where* to place it optimally (on which type of hardware).

Job Duration and Profile : HPC simulations often have predictable durations. DL model training jobs, on the other hand, can last from a few minutes to several weeks, and their resource consumption profile (compute, memory, network) can vary considerably during their execution.

Mixed Criticality and Interactivity : Increasingly, AI clusters are not just used for long-running batch training jobs. They also host interactive development, debugging, or urgent inference tasks, creating a mixed-criticality environment where low-latency jobs (subject to SLAs) coexist with background tasks.

Preponderant Energy Cost : As discussed in Chapter 1, the power consumption of GPUs has made energy a first-order operational cost. A modern scheduler can no longer ignore it.

This evolution transforms the scheduler from a simple queue manager into a multi-dimensional strategic resource controller.

5.2.2 Multi-facet of Efficient Scheduling

Scheduling in a modern AI cluster is not a search for a single optimum, but a navigation through a space of trade-offs between multiple and often conflicting objectives, as illustrated in Figure 5.1. The six main objectives are often in tension, requiring intelligent arbitration strategies.

5.2.2.1 Performance : Maximizing Useful Work

Performance remains a primary objective. It is mainly measured in two ways :

Throughput : Often measured as the number of jobs completed per unit of time (e.g., jobs/hour), it is the indicator of the cluster's overall productivity. Schedulers like Pollux [Qiao, 2021], which use *Shortest Job First* (SJF) type heuristics, excel at maximizing throughput.

Average Job Completion Time (JCT) : This is the total time elapsed between a job's submission and its completion. Minimizing the average JCT improves the overall user experience. These two metrics (throughput and average JCT) are often correlated, but not always perfectly. The PowerFlow [Gu, 2023] scheduler target this metric under energy budget.

Maximizing performance, however, often comes at the expense of other objectives such as fairness or energy efficiency, highlighting the multidimensional nature of scheduling.

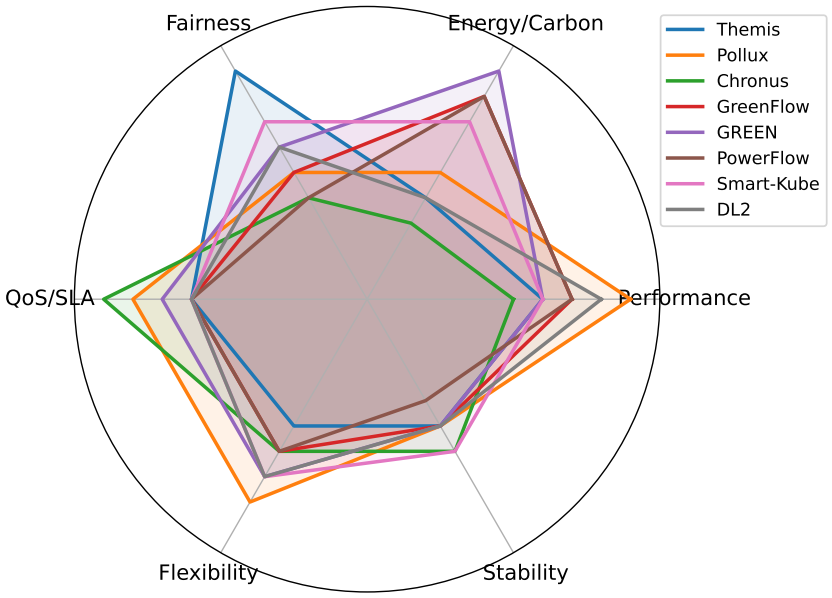


FIGURE 5.1 – Overview of the optimization space in multi-objective scheduling.

5.2.2.2 Energy : Reducing Cost and Carbon Impact

Optimizing energy use has become a first-class objective in modern scheduling. The fastest allocation (e.g., using many powerful GPUs) is typically the most energy-intensive, leading to unsustainable operational and environmental costs.

Objective : Minimize the total energy consumed by the cluster over a scheduling horizon, thereby reducing both operational expenses and CO_2 emissions.

Metric : The *Energy-Delay Product (EDP)* captures the trade-off between energy efficiency and performance, rewarding schedules that deliver useful work per joule expended.

Schedulers such as GreenFlow [Gu, 2025], Smart-Kube [Ghafouri, 2023], PowerFlow [Gu, 2023] and GREEN [Xu, 2025] explicitly target this axis, offering significant carbon reductions at limited performance cost.

5.2.2.3 Fairness : Sharing Resources

In a multi-user environment, ensuring fairness is essential to prevent the *"starvation"* of certain users or projects. However, *"fairness"* is a multi-dimensional concept. We can distinguish :

Resource Allocation Fairness : The objective is to ensure that, over the long term, each user receives a share of the resources (e.g., in GPU-seconds) proportional to their priority or rights. This is the approach of schedulers like Themis [Mahajan, 2020] and DL2 [Peng, 2021]. To achieve this, the scheduler must sometimes prioritize a *majority* user who is in a resource *deficit*, even if it means delaying the jobs of other users.

User Experience Fairness (JCT Fairness) : The objective is that the job completion time is equitably distributed among users, regardless of their past consumption. *Jain's Fairness Index* applied to the average JCT per user is a good metric for this.

These two forms of fairness are often in direct conflict. Our work, particularly with the Hades scheduler, aims to explore the additional trade-off between allocation fairness and energy.

5.2.2.4 Quality of Service (QoS) : Compliance to Commitments

In production clusters, not all jobs are created equal : certain workloads come with strict deadlines, often codified in *Service Level Agreements (SLAs)*. Schedulers must account for this heterogeneity.

Objective : Minimize the *Deadline Miss Rate* by prioritizing jobs with urgent constraints. For example, Chronus [Gao, 2021] uses slack-aware policies that rank jobs based on the time remaining until their deadline.

Trade-off : Guaranteeing QoS often reduces global throughput (by dedicating resources to urgent but less efficient jobs) and can compromise fairness when urgent jobs consistently preempt others.

QoS-aware scheduling thus introduces a tension between global efficiency and reliability guarantees.

5.2.2.5 Flexibility and Robustness : Adapting to Uncertainty

Scheduling decisions are rarely made in perfectly predictable environments. Clusters operate under uncertainty : job arrivals are stochastic, execution times vary depending on dataset sizes or model architectures, and hardware may occasionally fail. To remain effective in such contexts, a scheduler must not only optimize under ideal assumptions, but also remain resilient when reality diverges from forecasts.

Flexibility : The ability to adapt dynamically to changing workloads or user demands. For example, Pollux [Qiao, 2021] exhibits flexibility by co-adapting resource allocations and training hyperparameters in response to job progress and contention.

Robustness : The capacity to sustain acceptable performance levels despite disturbances such as stragglers, preemptions, or GPU failures. Robust schedulers often rely on stochastic optimization or redundancy mechanisms to guarantee service quality under uncertainty.

The trade-off here is between *optimality* in static scenarios and *resilience* in dynamic ones : the more a system is tuned for worst-case robustness, the less aggressively it can exploit short-term optimization opportunities.

5.2.2.6 Stability : Ensuring Predictability

While adaptability is crucial, stability is equally important in multi-user clusters. Frequent changes in allocations (though potentially optimal from a system perspective) can lead to user confusion, degraded reproducibility, and inefficiencies due to task migration overheads.

Predictable Scheduling : Stability ensures that users can plan around their allocated resources. Schedulers like Chronus [Gao, 2021], which operate on lease-based preemptions, aim to strike a balance between responsiveness and predictability.

Minimizing Disruptions : Excessive preemptions or rescheduling not only frustrate users but also waste computation, as partially completed work may need to be repeated. A stable scheduler minimizes these disruptions while still allowing for necessary adjustments.

The challenge is to balance *stability* with *flexibility* : highly stable schedules may underutilize resources when workloads shift, while overly flexible ones can degrade the user experience.

5.2.3 Optimization Levers of the Scheduler

To navigate this trade-off space, a modern scheduler has three main levers that radically increase the complexity of its decision-making.

Heterogeneity (h). As mentioned, the choice of *hardware type* (h) (e.g., H100 vs. A100) is a first-order decision. A job might run faster on an H100, but consume more energy and deprive another job better suited to that architecture of that resource.

Malleability. Modern DL jobs are often **malleable** [Li, 2025a]. This means that the resources allocated to them can be dynamically adjusted. The two main dimensions of malleability are :

— **The number of GPUs (k)** : A model can be trained on 1, 2, 4, 8, or more GPUs. Increasing k generally reduces the execution time (up to a certain scaling limit), but increases instantaneous power and communication overhead.

— **The batch size per GPU (b)** : Increasing the mini-batch size per GPU improves the arithmetic efficiency of each GPU (better utilization of Tensor Cores), but is limited by HBM capacity and may impact model convergence.

A *"malleability-aware"* scheduler (like Pollux [Qiao, 2021] and our proposed policies) therefore has a much larger decision space : for each job, it must choose the best combination (k, b, h) based on its objective. The importance of this lever is fundamental.

Preemption. Preemption is the scheduler's ability to interrupt (and potentially save/resume) a running job, typically of low priority, to free up resources for a higher-priority job.

— **Justification :** In a heavily utilized cluster with long-running jobs, it is almost impossible to guarantee SLAs for urgent jobs without preemption. A non-urgent job occupying 8 GPUs for 10 hours can block any new urgent submission.

— **Cost :** Preemption has a cost : the work done by the interrupted job since its last checkpoint is lost, which represents a waste of energy and time. A good preemptive scheduler should therefore only use it as a last resort. Our AuraChronos policy explores this critical dimension.

5.2.4 Synthesis : The Need for a New Analysis and Design Framework

Our analysis of the foundations of scheduling for AI clusters highlights a complex reality : the search for the *"best"* scheduling policy is an ill-posed problem. The *best* policy depends on the priority objectives (performance, energy, fairness, SLA), the characteristics of the workload and the infrastructure, and how efficiently it leverages the available levers (heterogeneity, malleability, preemption).

It is this complexity that directly motivates the contributions of this chapter

1. It justifies the need for a *flexible* and *high-fidelity* simulation tool like EAS-Sim, as it is the only way to explore this immense design space in a fast, reproducible, and quantitative manner.
2. It motivates our *co-design methodology*, which proposes a systematic approach (based on EDP) to enrich existing policies with the energy dimension, rather than starting from scratch.
3. It justifies the design of our *suite of new policies* (Zeus, Hades, AuraChronos, Charon), as each of them aims to find a new operating point, a new trade-off, in this complex multi-objective space, thereby meeting distinct operational needs.

The remainder of this chapter presents in detail this analysis framework (EAS-Sim) and the rigorous design and evaluation of our new scheduling strategies.

5.3 EAS-Sim : A Simulation Framework for Deep Learning Schedulers Co-Design

Exploring the design space of multi-objective scheduling policies for AI workloads is an intrinsically complex problem, for which direct experimental evaluation on large-scale physical infrastructures faces major obstacles :

1. **Cost and Availability :** Large-scale experiments monopolize thousands of costly and scarce GPU-hours.
2. **Experimentation Time :** As model training can last for several hours or days, evaluating a single policy on a simulated week of workload could take... a week of real time, making iterative design cycles impractical.
3. **Reproducibility :** Performance on real systems is subject to non-deterministic variations (OS system noise, network contention, thermal throttling) that make the rigorous comparison of policies with close performances difficult.

Faced with these challenges, discrete-event simulation stands out as an indispensable methodology, provided that the simulator is fast, extensible, and, above all, of **high-fidelity** [Li, 2025b]. It is to meet this specific need that we have designed and implemented **EAS-Sim (*Energy-Aware Scheduling Simulator*)**, our first contribution of this chapter.

5.3.1 Design Principles and Technological Choices

The design of EAS-Sim was guided by four fundamental principles :

— **Fidelity :** The simulator must capture the essential characteristics that influence scheduling decisions and their consequences : hardware heterogeneity, job malleability, realistic performance and power profiles, and the temporal dynamics of job arrivals and completions.

— **Extensibility :** The architecture must be modular to allow for the easy addition of new scheduling policies, new hardware types, or more sophisticated performance/power models.

— **Performance :** The simulator must be fast enough to allow the execution of vast experimental campaigns (hundreds of simulations of several virtual hours) in a reasonable amount of time, to ensure the statistical significance of the results.

— **Reproducibility :** For a given random seed, the simulation results must be perfectly deterministic, allowing for rigorous "all else being equal" comparisons.

To implement these principles, we chose the **Python** programming language, coupled with the **SimPy** library [SimPy, 2002] for discrete-event simulation, and the `multiprocessing` module for parallelizing experimental campaigns. Python was chosen for its productivity, its vast scientific ecosystem (NumPy, Pandas), and its ease of integration with the AI frameworks whose models we analyze. SimPy provides an elegant framework for modeling asynchronous processes and interactions within the cluster (job arrival, resource allocation, completion).

5.3.2 Detailed Architecture and Fundamental Components

EAS-Sim is structured around four main software components that interact within the SimPy simulation environment (Figure 5.2). We detail here the implementation and logic of each.

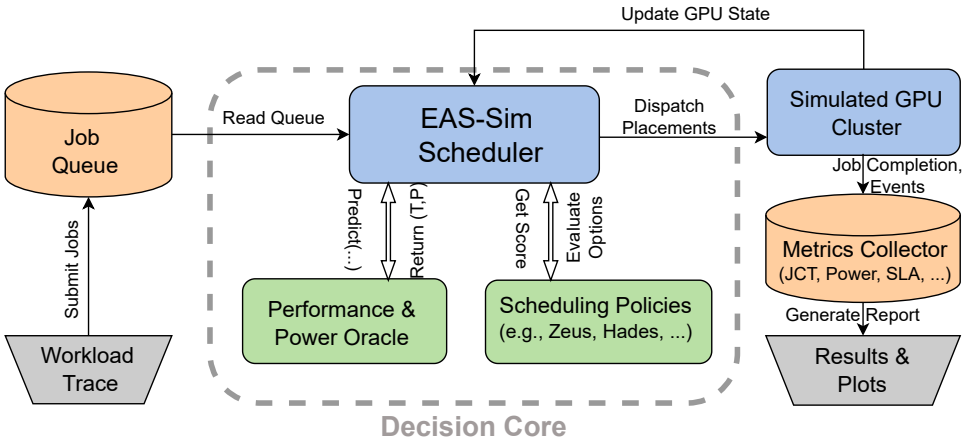


FIGURE 5.2 – The architecture of the EAS-Sim framework.

5.3.2.1 Functional Model of a Cluster

This module represents the physical infrastructure and its state.

Object Hierarchy : The `Cluster` class contains a list of `Node` objects. Each `Node` in turn contains a list of `GPU` objects. This hierarchy allows for modeling realistic cluster topologies.

Resource Management with SimPy : Each `GPU` object encapsulates a SimPy resource (`simpy.Resource(capacity=1)`). This SimPy primitive natively handles queues and mutually exclusive allocation : a job can only "use" a GPU resource if it is free. The scheduler will interact with these resources to allocate and release GPUs.

Heterogeneity Support : The major contribution of our cluster model is its ability to handle heterogeneity. Each `GPU` and `Node` object is typed by a `hardware_type` attribute (e.g., 'a100', 'h100') and is associated with a dictionary of performance parameters `hardware_params`. These parameters are an instance of the `HardwareParams` object from our modeling framework (cf. Chapter 4). Thus, when a job is allocated to a specific GPU, the scheduler knows precisely on which type of hardware it will run and can query the oracle with the correct performance parameters, which is fundamental for a faithful simulation.

The `Cluster` class exposes helper methods, like `get_idle_gpus_by_type()`, which facilitate the implementation of "hardware-aware" scheduling policies.

5.3.2.2 Job and Workload Model

This module defines the entities that transit through the system.

The `Job` Object : Each job is an instance of the `Job` class. It encapsulates all its static characteristics (defined at submission) : its model architecture, its total workload (e.g., 20000 steps), its malleability options (`valid_batch_sizes`, `gpu_options`), its QoS constraints (`is_urgent`, `deadline`), its user (`user_id`), etc. Furthermore, the `Job` object also stores its dynamic state : its start time, its completion time, the GPU resources allocated to it. A key property is its workload profile, `workload_profile`, pre-calculated by the `WorkloadCalculator` (Chapter 4) and attached to the job. If the profile is absent from the pre-calculated job trace file for a new job, the analytical oracle will perform a new calculation. This will be saved for future use to expedite the decision-making process, considering that utilizing the analytical oracle incurs a cost.

The Workload Generator (`WorkloadGenerator`) : This is a SimPy process whose role is to introduce jobs into the simulator at the right time. In our experiments, it simulates job arrivals according to a Poisson process (via `random.expovariate(arrival_rate)`), a standard model for representing independent user requests. The generator randomly selects a model architecture from a set of predefined templates.

This design decouples workload generation from the scheduling itself, allowing different policies to be evaluated on the exact same stream of jobs for a fair comparison.

5.3.2.3 Performance and Power Oracle

The oracle is the link between the abstract world of simulation and the realistic behavior of the hardware. Without a faithful oracle, the scheduler's decisions would be based on erroneous

information, invalidating the simulation.

Analytical Core : As detailed in Section 5.3, the core of our oracle is the implementation of the analytical modeling framework presented in Chapter 4. The oracle's `predict` method receives a `Job` (with its workload profile) and a candidate placement configuration (number of GPUs, batch size, hardware type) as input. It then invokes the analytical model to return a prediction (T, P). The use of an *a priori* model is a major design choice : it means our simulator does not need to *virtually* execute the job for a few iterations to know its performance, making it much faster than an approach based on reactive measurement (like that of the original Zeus [You, 2023]).

Prediction Cache : To speed up the simulation (as the same job might be evaluated for the same placement multiple times), the oracle maintains an internal cache of predictions.

Modeling Uncertainty (`NoisyOracle`) : We assume that no performance model is perfect in practice. To make our conclusions more robust, our default oracle, `NoisyOracle`, encapsulates the analytical oracle and injects a multiplicative stochastic noise of $\pm 10\%$ to the time and power predictions. This value is a conservative estimate of the variability observed in real systems [Helmenstine, 2016] due to jitter, contention, etc. It is crucial to note that the predicted power is then always capped by the physical TDP of the allocated resources, ensuring that even noisy predictions remain within the realm of the physically possible.

5.3.2.4 Scheduling Model and Basic Assumptions

This module defines the decision-making intelligence.

Batch-Packing Architecture : We have made the conceptual choice of a *periodic batch-packing* scheduler. At regular intervals (e.g., every second), the scheduler *wakes up*, examines the entire queue of pending jobs (`job_queue`) and the set of free resources, and attempts to "pack" the free resources as best as possible. This approach is common in systems like Borg [Verma, 2015] or HPC schedulers (e.g., Slurm with backfilling) because it allows for a global view of the system and promotes more optimal allocation decisions than purely event-driven approaches (where a decision is made at each job arrival or departure).

The Score Function as the Logical Core : The essence of each policy is encapsulated in the `GetJobScore` method. This function takes a job and the state of the free resources as input, and returns a **priority score** as well as the best placement configuration found for that job. Sorting the jobs based on this score determines the allocation order. This abstraction (implemented in `BaseBatchPacker`) makes the implementation of new policies trivial.

Assumptions about Policies :

— **Perfect Knowledge (via the Oracle) :** We assume the scheduler has access to a performance oracle. This is a strong assumption, but one that aligns with an active research area and the goal of systems like ours. Our use of the `NoisyOracle` aims to slightly relax this assumption by introducing uncertainty.

— **Limited Malleability :** We assume that the malleability options $(\mathcal{K}_j, \mathcal{B}_j)$ are discrete and known at submission time. We do not model continuous elasticity (where a job can change its configuration during execution), a still-emerging research area [Liu, 2024].

— **Cost of Preemption :** For AuraChronos, we model a simple but realistic cost of preemption : a preempted job loses the work done since its last iteration (it is placed back at the front of the queue). We do not model more complex *checkpointing/restart* mechanisms, which represents a limitation and a direction for future work.

In sum, EAS-Sim provides a complete, modular, and configurable virtual testbed, specifically designed for the rigorous evaluation and co-design of multi-objective schedulers for AI workloads. Its design choices make it, to our knowledge, a unique tool for exploring the energy-performance-fairness-SLA trade-off space.

5.4 Methodology for the Co-Design of Energy-Aware Schedulers

Equipped with a robust simulation framework (EAS-Sim), we now present our main methodological contribution : a systematic approach for the **co-design** of Deep Learning job schedulers that are natively energy-aware. Rather than creating policies *ex nihilo*, our methodology is based on the principle of augmentation : we start from State-of-the-Art (SOTA) heuristics, which encapsulate proven optimization logic for specific objectives (performance, fairness, SLA), and we inject energy awareness into them in a rigorous and formalized manner. This approach allows us to retain the benefits of existing strategies while extending them to navigate the multi-objective trade-off space we have identified.

5.4.1 Formalization and Implementation of Reference Heuristics

The first step of our methodology is to formalize and implement, within EAS-Sim, several state-of-the-art heuristics that will serve as foundations and baselines for our own contributions. Their decision logic is encapsulated in the `GetJobScore` method of their respective classes in `simulator/policies.py`.

Pollux (Performance-Oriented) : [Qiao, 2021] The objective of Pollux is to maximize the overall throughput of the cluster. It implements a *Shortest Job First* (SJF) type heuristic adapted to a malleable context. For each job j in the queue, its score function explores all possible placement configurations (k, b, h) on the available resources G_{idle}, and selects the one that minimizes the predicted execution time $T(j, k, b, h)$. The final score is often modulated by an aging factor W_{age} to prevent starvation of long jobs : $\text{score}(j) = \min(T) - W_{\text{age}} \cdot (t_{\text{current}} - A_j)$. This logic, formalized in Algorithm 3, is implemented in the `PolluxRigid` and `PolluxMalleable` classes.

Themis (Fairness-Oriented) : [Mahajan, 2020] Themis's objective is to ensure fairness in resource allocation among users. It implements a *fair-sharing* heuristic. The logic is to first find the best configuration in terms of performance (like Pollux), but then to modulate the job's score by the historical resource consumption of its user, C_{U_j} (measured in GPU-seconds). The score is $\text{score}(j) = \frac{\min(T)}{C_{U_j}+1}$. Thus, for equal performance, a job from a user who has consumed few resources will get a better (lower) score and be prioritized. This logic, formalized in Algorithm 4, is implemented in the `ThemisRigid` and `ThemisMalleable` classes.

Chronos (SLA-Oriented) : [Gao, 2021] Chronos is designed for mixed-criticality workloads. It uses a two-tier priority system :

1. Urgent jobs (i.e., with a deadline) are in a higher priority tier (Tier 0). They are ranked among themselves by their remaining **slack time**, defined as slack $= D_j - (t_{\text{current}} + T(j))$. Those with the least slack are the highest priority.
2. Non-urgent (*best-effort*) jobs are in a lower priority tier (Tier 1) and are used for backfilling. They are ranked among themselves by performance (like Pollux).

The score is therefore a tuple $(tier, metric)$, for example $(0, \text{slack})$ for an urgent job. This logic, formalized in Algorithm 5, is implemented in our `ChronosRigid` and `ChronosMalleable` classes.

Algorithm 3: Pollux Malleable Job Scoring (`GetJobScore`)

Input : Job j, Idle GPUs G_{idle}
Output: Candidate dictionary or null

best_placement ← **null**; best_runtime ← ∞;
foreach $k \in j.\mathcal{K}_j$ ***and*** $b \in j.\mathcal{B}_j$ **do**
  foreach $(h, gpus) \in G_{idle}$ **do**
    if $|gpus| \geq k$ **then**
      pred ← $\Pi(j, k, b, h)$;
      if *pred.T* < *best_runtime* **then**
        best_runtime ← pred.T;
        best_placement ← {time : pred.T, ..., gpus_count : k, batch_size : b, hw_type : h};
if *best_placement* ***is not*** *null* **then**
  score ← best_runtime - ($W_{\text{age}}\times$ (Now() - $j.A_j$));
  return {job : j, score : score, ... best_placement};
return null;

Algorithm 4: Themis Malleable Job Scoring (`GetJobScore`)

Input : Job j, Idle GPUs G_{idle}, User Consumptions C
Output: Candidate dictionary or null

`// Placement selection is identical to Pollux (Algorithm 3, lines 2-8)`
best_placement ← FindBestPerformancePlacement(j, G_{idle});
if *best_placement* ***is not*** *null* **then**
  `// Score is modulated by the user's historical share`
  user_share ← $C[j.U_j] + 1$;
  score ← best_placement.time / user_share;
  return {job : j, score : score, ... best_placement};
return null;

Algorithm 5: Chronos Malleable Job Scoring (`GetJobScore`)

Input : Job j, Idle GPUs G_{idle}
Output: Candidate dictionary or null

best_placement ← FindBestPerformancePlacement(j, G_{idle});
if *best_placement* ***is not*** *null* **then**
  if j *is urgent* **then**
    `// Tier 0 for urgent jobs, sorted by slack time`
    slack ← $j.D_j$ - (Now() + best_placement.time);
    score ← (0, slack);
  else
    `// Tier 1 for backfill, sorted by runtime`
    perf_score ← best_placement.time - ($W_{\text{age}}\times$ (Now() - $j.A_j$));
    score ← (1, perf_score);
  return {job : j, score : score, ... best_placement};
return null;

5.4.2 Energy-Delay Product as a Pivotal Metric

Our main methodological contribution lies in the choice of a primitive to systematically inject energy awareness. We have chosen the **Energy-Delay Product (EDP)**, a classic metric from low-power computing [Horowitz, 1994], but whose application to the scheduling of complex DL workloads is, to our knowledge, novel.

For a job executed for a time T (Delay) with an average power P, the Energy consumed is $E = P \times T$. The EDP is classically defined as the product of energy and delay :

$$\text{EDP} = E \times T = (P \times T) \times T = P \cdot T^2 \tag{5.1}$$

The key feature of the EDP is its **quadratic penalty on time**. This makes it particularly suitable for identifying balanced operating points :

— A **very fast** configuration (T very low) but **extremely power-hungry** (P very high) will be penalized because the value of P will dominate.

— A **very power-efficient** configuration (P very low) but **extremely slow** (T very high) will be heavily penalized by the T^2 term.

— The configurations that minimize the EDP are therefore those that offer an *excellent compromise*, being both relatively fast and efficient in terms of power consumption.

We postulate that the EDP is not only an energy-saving metric, but also a more robust heuristic for overall system efficiency, because by favoring more resource-frugal configurations (often those using fewer GPUs), it can reduce cluster fragmentation and improve overall throughput. It is this hypothesis that we validate experimentally (Section 5.5).

5.4.3 Application of Our Methodology : A Suite of Energy-Aware Schedulers

By systematically replacing the raw performance metric (time T) with the EDP ($P \cdot T^2$) as the primary minimization objective in the score functions of the SOTA heuristics, we have designed and implemented four new scheduling policies.

Zeus (Energy-Performance) : The energy-aware equivalent of Pollux. Its score function, detailed in Algorithm 6, explores all possible malleable configurations and selects the one that minimizes the EDP, modulated by an aging factor :

$$\text{score}(j) = \frac{\min_{k,b,h}(P(j,k,b,h) \cdot T(j,k,b,h)^2)}{1 + W_{\text{age}} \cdot (t_{\text{current}} - A_j)}$$

Hades (Energy-Fairness) : The energy-aware equivalent of Themis. It combines both objectives by first finding the best configuration from an EDP perspective, and then modulating this score by fairness, as formalized in Algorithm 9 :

$$\text{score}(j) = \frac{\min_{k,b,h}(\text{EDP}(j,k,b,h))}{C_{U_j} + 1}$$

AuraChronos (Energy-SLA and Preemption) : Our most complex policy, which extends Chronos. Its logic (Algorithm 7) is twofold :

1. **Proactive Preemption Loop :** The scheduler checks at each cycle if an urgent job is at risk of missing its deadline. If so, and if resources are insufficient, it identifies non-urgent jobs ("victims"), interrupts them (`job.running_process.interrupt()`), and places them back in the queue.
2. **Energy-Aware Hybrid Scoring :** Urgent jobs are still prioritized by *slack time*. However, for backfilling non-urgent jobs, AuraChronos uses Zeus's heuristic (minimizing EDP) instead of Pollux's (minimizing time), thus promoting an energy-efficient use of background resources.

Charon (Power Budget Constraint) : Charon implements scheduling under a constraint. It first uses Zeus's EDP heuristic to rank jobs by priority. Then, its allocation loop (Algorithm 8) is equipped with an additional condition : a job is allocated only if the predicted power of its placement, added to the currently consumed power by the cluster, remains below a predefined global budget P_{cap}. The implementation of Charon, particularly in its main loop, requires careful management of a local `projected_power` variable that is updated after each allocation decision within the same scheduling cycle.

These four policies, which are direct contributions of this thesis, allow for the systematic exploration of the trade-offs between the different scheduling objectives, as the following experimental evaluation section will demonstrate.

Algorithm 6: Zeus Job Scoring (`GetJobScore`)

Input : Job j, Idle GPUs G_{idle}
Output: Candidate dictionary or null

best_placement ← **null**;
best_edp ← ∞;
▷ `Explore all malleable configurations`
foreach $k \in j.\mathcal{K}_j$ **do**
  foreach $b \in j.\mathcal{B}_j$ **do**
    foreach $(h, gpus) \in G_{idle}$ **do**
      if $|gpus| \geq k$ **then**
        pred ← $\Pi(j, k, b, h)$;
        edp ← pred.P × (pred.T)^2;
        if *edp* < *best_edp* **then**
          best_edp ← edp;
          best_placement ← {time : pred.T, power : pred.P, gpus_count : k, batch_size : b, hw_type : h};
if *best_placement* ***is not*** *null* **then**
  age ← Now() - $j.A_j$;
  ▷ `Score by best EDP, with aging`
  score ← best_edp / (1 + W_{age}× age);
  return {job : j, score : score, … best_placement};
else
  return null;

Algorithm 7: AuraChronos Preemptive Scheduling Loop (`start`)

```
1  Function AuraChronosStart :
2  while true do
3      Wait(scheduling_interval);
4      if J_q is empty then
5          continue;
       ▷ 1. Proactive Preemption Check
6      urgent_jobs ← GetUrgentJobsSortedByDeadline(J_q);
7      if urgent_jobs is not empty then
8          job_to_check ← urgent_jobs[0];
           // Consider all GPUs as potentially available
9          all_gpus ← GetAllGPUsByType();
10         placements ← GetPossiblePlacements(job_to_check, all_gpus);
11         if placements then
12             fastest ← min_{p∈placements}(p.time);
               // Trigger if time-to-deadline is critical
13             if job_to_check.D - Now() < fastest.time × 1.5 then
14                 idle_now ← GetIdleGPUs();
15                 if not IsRunnable(fastest, idle_now) then
16                     victims ← FindVictims (fastest.gpus_count);
17                     if victims then
18                         PreemptJobs(victims);
19                         continue;
                           ▷ Restart cycle after preemption
       ▷ 2. Fall through to standard batch packing
20     DefaultBatchPackingLoop(J_q);
```

Algorithm 8: Charon Power-Capped Scheduling Loop (`start`)

Input: Job Queue J_q, Power Cap P_{cap}

while *true* **do**
  Wait(scheduling_interval);
  $G_{\text{idle}} \leftarrow$ GetIdleGPUs();
  ▷ `Score jobs using Zeus's EDP heuristic (Alg. 6)`
  candidates $\leftarrow$ [GetJobScore(j, G_{idle}) for j in J_q];
  prioritized_jobs $\leftarrow$ Sort(candidates, key=c.score);
  `// Track power allocated within this cycle`
  projected_power $\leftarrow$ GetCurrentClusterPower();
  foreach *c* ***in*** *prioritized_jobs* **do**
    `// Enforce the hard power constraint`
    if *projected_power + c.placement.power* $\leq P_{cap}$ **then**
      if *AreGPUsAvailable(c.placement.gpus_count, c.placement.hw_type,* G_{idle}*)* **then**
        DispatchJob(c.job, c.placement);
        projected_power $\leftarrow$ projected_power + c.placement.power;
        UpdateAvailableGPUs(G_{idle}, c.placement);

Algorithm 9: Hades Malleable Job Scoring (`GetJobScore`)

Input : Job j, Idle GPUs G_{idle}, User Consumptions C
Output: Candidate dictionary or null

`// Placement selection uses EDP (similar to Alg. 6)`
best_placement $\leftarrow$ FindBestEDPPlacement(j, G_{idle});
if *best_placement* ***is not*** *null* **then**
  user_share $\leftarrow C[j.U_j] + 1$;
  edp $\leftarrow$ best_placement.power $\times$ (best_placement.time)^2;
  `// Score co-optimizes EDP and fairness`
  score $\leftarrow$ edp / user_share;
  return {job : j, score : score, … best_placement};
return null;

5.5 Experimental Evaluation of Scheduling Policies

This section presents a comprehensive experimental evaluation of our contributions, conducted within the EAS-Sim simulation framework. The objective is to quantify the performance of our new scheduling policies (Zeus, Hades, AuraChronos, Charon) by comparing them to state-of-the-art heuristics (Pollux, Themis, Chronos) across a set of representative scenarios. We seek to answer five fundamental research questions concerning malleability, energy efficiency, SLA guarantees, scheduling under a power constraint, and fairness.

5.5.1 Detailed Experimental Protocol

The rigor of our evaluation is based on a precise and reproducible experimental protocol, whose parameters are implemented in the main script.

Simulation Environment and Repeatability. All simulations model a duration of **10 hours** of cluster operation, a time horizon sufficient to reach a steady state and observe the completion of multiple long-running jobs. To ensure statistical significance and to quantify the variability due to the random arrival of jobs, each reported data point is the average of **10 independent simulations**, each initialized with a distinct random seed. The error bars in our graphs represent one standard deviation ($\pm 1\sigma$). All simulations were executed on a high-performance workstation equipped with an Intel Core i9-12950HX CPU (16 cores, 24 threads), 32 GB of RAM, and an NVIDIA RTX 3080 Ti Laptop GPU. We ran the (discrete-event) simulation for a total of 10 simulated hours, a duration chosen to be long enough for the system to reach a steady state and for multiple long-running jobs to complete, thus ensuring better statistics.

Cluster Configuration. Our chosen cluster, which is a heterogeneous representative of modern AI infrastructure, consists of 16 total SXM-based GPUs distributed over two distinct node types. One node equipped with 8 NVIDIA A100-40GB GPUs, configured with the performance and energy characteristics of the Ampere architecture (TDP of 400W). One other node equipped with 8 NVIDIA H100-80GB GPUs, configured with the characteristics of the Hopper architecture (TDP of 700W), which is more powerful but also more energy-intensive. This heterogeneity is critical for evaluating the ability of the scheduler to make "intelligent" assignments to the computing units.

Workload Generation. The workload is composed of three representative DL models chosen to cover diverse architectural families. We have **Vision Transformer (ViT-Base)** [Dosovitskiy, 2021], a canonical attention-based model for computer vision, trained in full `FP32` precision. **ALBERT (v2-base)** [Lan, 2019], a parameter-efficient variant of BERT, well-identified for its unique weight-sharing architecture. It is trained in mixed-precision using `bfloat16`. And lastly **ResNet-50** [He, 2015b], a reference convolutional neural network, also trained in `FP32`.

To provide a glimpse of the job scale, a single ViT job (using CIFAR10 dataset [Krizhevsky, 2009a] and 50 epochs) on 8 A100 GPUs within our simulation corresponds to a substantial

training task of approximately 5-10 minutes. Jobs arrive according to a Poisson process, a standard model for simulating independent user requests in a shared cluster [Wolff, 1982]. The base arrival rate ($\lambda = 0.005$ jobs/sec or 3 jobs per 10-minute interval) is calibrated to drive the cluster towards a high ($\approx 80 - 90\%$) but stable average utilization, ensuring persistent resource contention that rigorously tests the scheduler's decision-making capabilities. 10% of jobs are considered as *"urgent"* and are assigned a deadline corresponding to **3.5**× their minimum possible runtime on the fastest available hardware, thereby creating a challenging but achievable SLA target. For experiments involving user fairness, jobs are assigned to one of the 5 users.

5.5.2 Experimental Results and Comments

We present and analyze here the results of our five evaluation scenarios.

5.5.2.1 The Fundamental Impact of Malleability

Our first experiment aims to quantify the importance of job malleability, a key optimization lever for modern schedulers. Figure 5.3 compares the performance (throughput vs. energy) of the Pollux and Themis policies in their **rigid** versions (where each job can only run on a fixed number of GPUs with a fixed batch size) and **malleable** versions. Malleable schedulers (hollow markers) achieve more than twice the throughput of their rigid counterparts (solid markers).

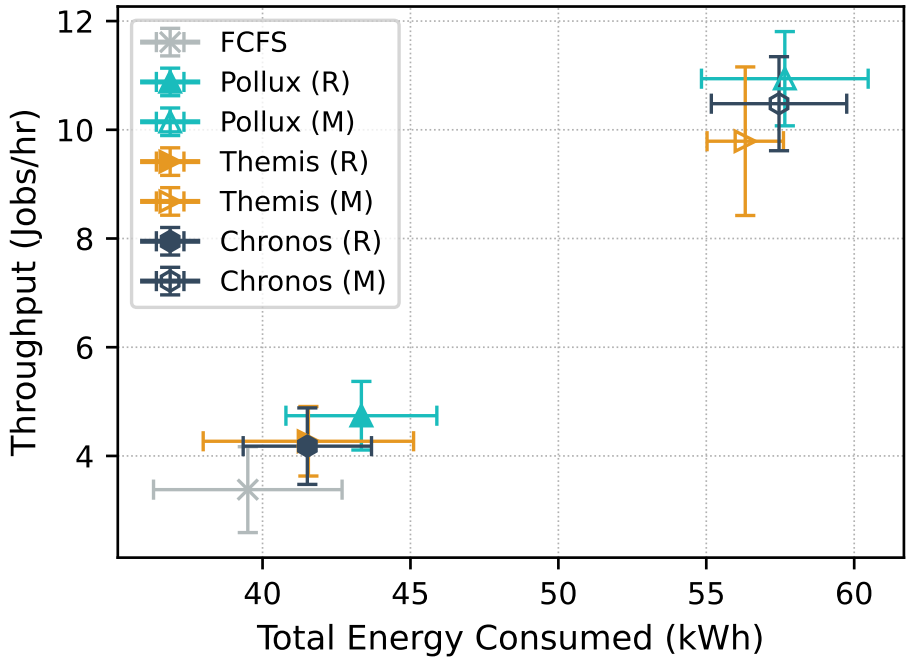


FIGURE 5.3 – Impact of malleability on the throughput/energy trade-off.

The results are unequivocal. **Malleable Pollux** achieves a throughput of ≈10.9 jobs/hour, a 2.2× improvement over its rigid version (≈4.7 jobs/hour). A similar gain is observed for Themis. This spectacular improvement is not a marginal optimization ; it demonstrates that

malleability is a fundamental prerequisite for high-performance scheduling in heterogeneous clusters. By allowing the scheduler to choose from a range of configurations (e.g., 2 powerful GPUs or 4 less powerful GPUs), malleability gives it the necessary flexibility to *fill the holes* in the resource schedule and drastically reduce fragmentation. For the remainder of our analysis, we will therefore only consider malleable policies.

5.5.2.2 A New Energy-Performance Pareto Frontier

The central objective of our methodology is to find better trade-offs between performance and energy. Figure 5.4 compares our flagship policy, Zeus, to the state-of-the-art malleable schedulers. Zeus and Hades establish a new Pareto frontier, offering significantly better energy efficiency than Pollux for comparable throughput.

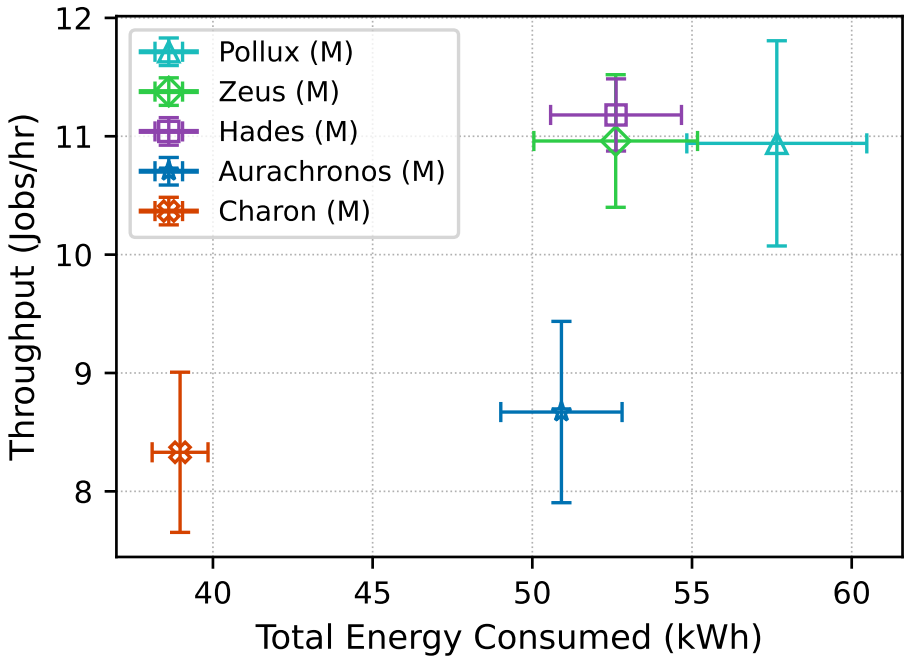


FIGURE 5.4 – Throughput/energy trade-off for malleable schedulers.

The reference scheduler, **Pollux (M)**, by optimizing for time, achieves a throughput of ≈10.9 jobs/hour, but at the cost of an energy consumption of ≈57 kWh over the simulation duration. Our policy **Zeus (M)**, which optimizes for EDP, achieves a statistically indistinguishable throughput of ≈11.0 jobs/hour while consuming only ≈51 kWh. This represents an energy consumption reduction of ≈10.5% with no performance sacrifice.

This result is profound and reveals a key aspect of the EDP heuristic : optimizing for EDP is not just an energy-saving strategy, but also a more robust heuristic for overall performance. By penalizing excessively power-hungry configurations, Zeus avoids the *"greedy"* behavior of Pollux, which tends to systematically allocate jobs to the most powerful resources (H100s) even when not necessary, thus creating fragmentation and leaving less powerful resources (A100s)

idle. Zeus performs a form of job *"eco-routing"*, improving the overall cluster "packing" and, consequently, the throughput. Zeus and Hades (which is also very effective) thus define a new **optimal Pareto frontier**, surpassing purely performance-oriented strategies.

5.5.2.3 The Importance of Preemption for the Quality of Service

This scenario tests the schedulers' ability to guarantee SLAs for urgent jobs. Figure 5.5 compares the deadline satisfaction rate for our policies. Our preemptive policy, AuraChronos, is the only one capable of providing meaningful QoS guarantees, with a deadline miss rate close to zero.

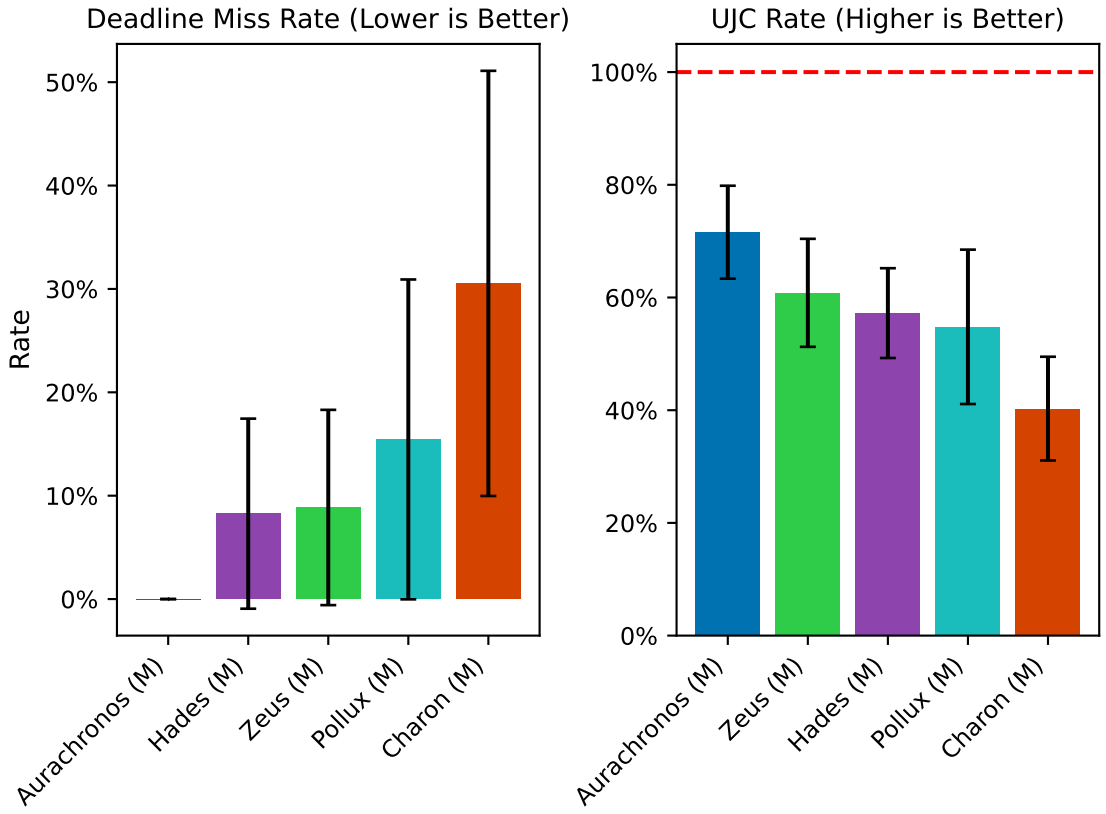


FIGURE 5.5 – SLA compliance.

The results are a clear demonstration of the failure of non-preemptive approaches. Sophisticated but non-preemptive schedulers like Pollux and Zeus show unacceptable *Deadline Miss Rates* of 15% and 9%, respectively. This is because once a long-running, non-urgent job is placed on critical resources, the scheduler is powerless to move it to make way for an urgent job that arrives subsequently. In contrast, our policy **AuraChronos (M)**, thanks to its proactive preemption capability, achieves a near-perfect deadline satisfaction rate of less than 1%, an improvement of more than 20× over non-preemptive policies. This result proves that preemption is not an optional optimization, but an architectural prerequisite for any scheduling system claiming to offer robust QoS in a high-load cluster. This gain in reliability comes at the cost of slightly lower

throughput (≈8.8 jobs/hour, see Figure 5.4), an explicit and quantifiable trade-off.

5.5.2.4 Scheduling under a Power Budget Constraint

This experiment evaluates the ability of our Charon policy to operate under a strict power budget of 4500W. Figure 5.6 shows the evolution of the cluster's power consumption over time. Charon (in red) perfectly respects the power budget (dotted line), while other policies regularly exceed it.

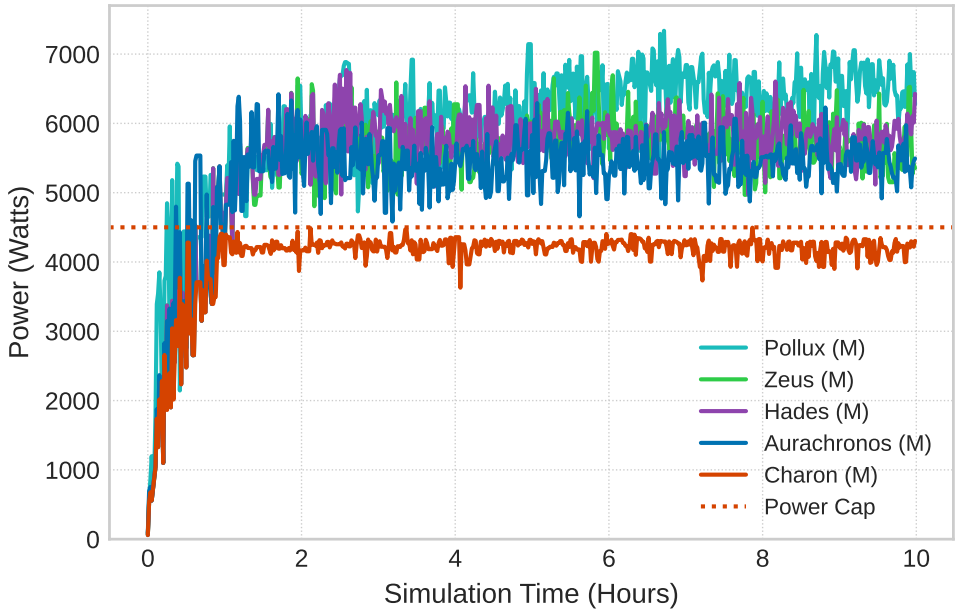


FIGURE 5.6 – Cluster power consumption dynamics.

The time series is unequivocal : Charon flawlessly adheres to the 4500W power budget. In contrast, all other unconstrained policies regularly operate in the 5000W to 7500W range, well above the limit. The application of this constraint has a quantifiable performance cost : Charon's throughput is about 8.3 jobs/hour. However, this reduced throughput is accompanied by a massive reduction in total energy consumption (over 50% compared to Pollux). Charon thus demonstrates that it is possible to operate a cluster under a strict power envelope, providing operators with an essential tool for managing power supply constraints or aligning consumption with intermittent renewable energy generation.

5.5.2.5 Co-optimizing Energy and Fairness under Imbalanced Load

Scenario E was specifically designed to test the ability of scheduling policies to manage fairness under realistic and challenging conditions, namely a workload where a single "majority" user submits 50% of all jobs. This scenario allows us to analyze the trade-off between different dimensions of fairness and to validate our energy-fairness co-design approach embodied by Hades.

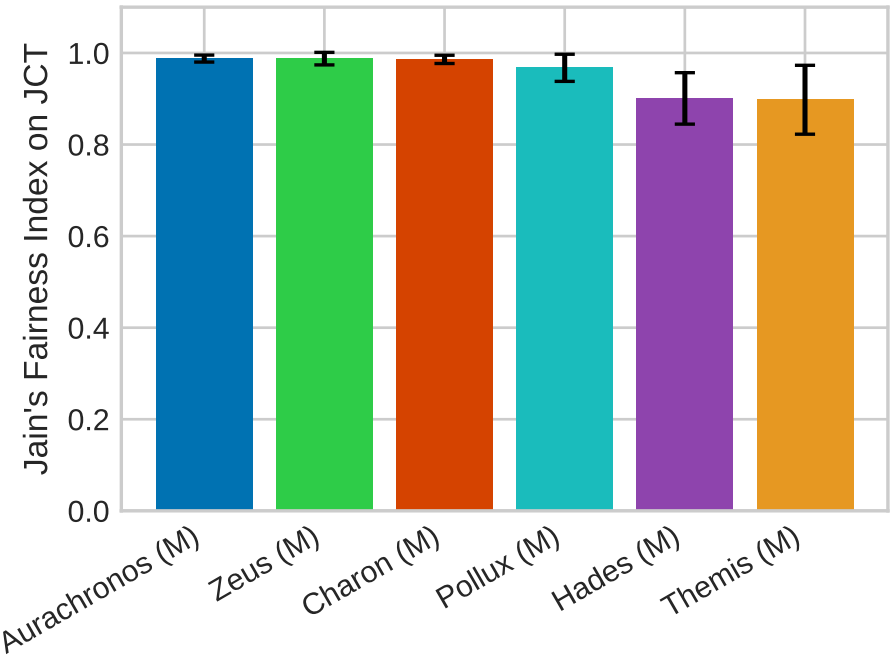


FIGURE 5.7 – Jain's Fairness Index for average job completion time (JCT) per user, in the imbalanced load scenario.

Figure 5.7 reveals a fundamental and counter-intuitive observation about the nature of fairness in scheduling. The performance-oriented policies (Pollux, Zeus), which are agnostic to user identity, achieve a Jain's index for JCT fairness that is close to perfection (1.0). This means that, on average, the jobs of all users, including the majority user, are completed in similar timeframes. Conversely, policies explicitly designed to ensure long-term resource allocation fairness (Themis, Hades) show a significantly lower JCT fairness score (≈0.8-0.9).

This result is not a failure, but the signature of their proper functioning. To compensate for the historical *deficit* of the majority user (who, by definition, requests more resources), Themis and Hades must give them priority. In doing so, they inevitably delay the jobs of minority users, creating a short-term disparity in completion times. This highlights the inherent trade-off between **resource allocation fairness** (ensuring everyone gets their share of the "pie" in the long run) and **user experience fairness** (ensuring all jobs are processed quickly in the short run).

Figure 5.8 then confirms the success of our co-design methodology. Comparing Hades to Themis (its SOTA fairness-oriented counterpart), we observe that **Hades** provides the same behavior in terms of allocation fairness but at a globally superior operating point. It achieves a significantly higher throughput than Themis while reducing energy consumption by 2%. Hades thus positions itself as a truly multi-objective scheduler, succeeding in co-optimizing resource fairness and energy efficiency.

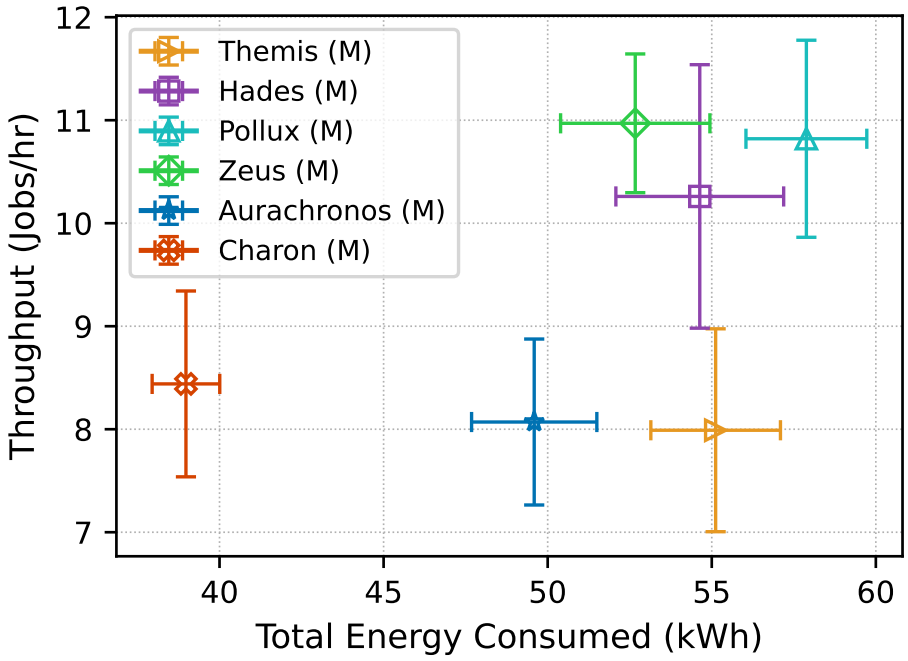


FIGURE 5.8 – Throughput/energy trade-off for the imbalanced load scenario.

5.5.2.6 Analysis of Robustness and Load Sensitivity

Scenario D evaluates the robustness of the scheduling policies by varying the overall system load (job arrival rate λ). Figure 5.9 plots the average JCT as a function of system load.

A robust scheduler is one whose performance degrades as gracefully as possible as the load increases and resource contention becomes extreme. At low load ($\lambda \leq 0.003$), where resources are abundant, all policies perform similarly, with a slight advantage for the purely performance-oriented approach of Pollux. However, as the system becomes increasingly saturated ($\lambda \geq 0.005$), significant behavioral differences emerge.

Remarkably, the EDP-based policies, Zeus and Hades, demonstrate a much more graceful performance degradation. Hades even distinguishes itself as the most stable policy at high load, maintaining the lowest average JCT. This observation reinforces our conclusion from : EDP acts as a regularization and stabilization heuristic. By systematically avoiding the most *"greedy"* configurations (which monopolize the most powerful resources), Zeus and Hades promote a better distribution of the load across all the heterogeneous resources of the cluster. This behavior acts as a form of natural *load balancing*, which prevents premature saturation of certain node types and reduces resource fragmentation, thereby improving the overall robustness and efficiency of the system under high contention.

Finally, the curve for AuraChronos illustrates the cost of preemption : it performs best at medium load where preemption is used judiciously, but its performance can drop at very high saturation due to the overhead associated with frequent job interruptions and resumptions (*thrashing*).

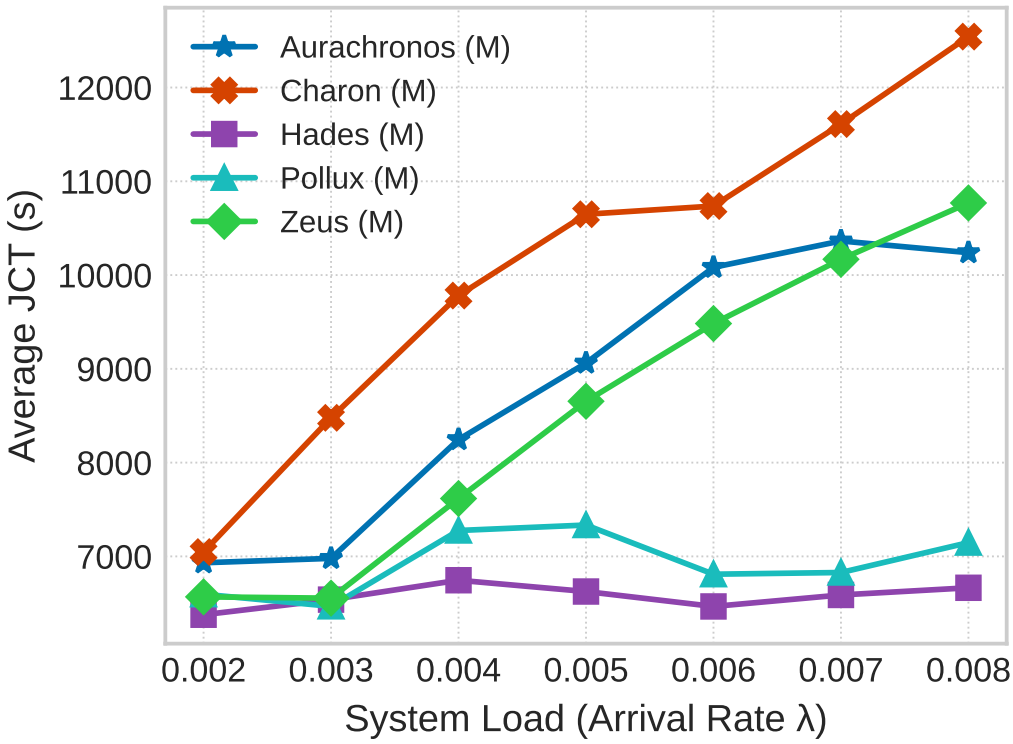


FIGURE 5.9 – Sensitivity of average job completion time to increasing system load

5.6 Discussions and Lessons Learned

The experimental evaluation conducted in the previous section has not only validated the effectiveness of our new scheduling policies but has also produced several fundamental insights that have broader implications for the design and operation of future clusters dedicated to Artificial Intelligence. This section aims to synthesize these key lessons, discuss the limitations of our work, and outline the research avenues it opens up.

5.6.1 EDP as a Robust Heuristic for Global Efficiency

One of the most significant results of our study is the remarkable effectiveness of the Energy-Delay Product (EDP) not only as an energy optimization metric but also as a global performance heuristic. Our results (Figures 5.4 and 5.9) consistently show that Zeus, which optimizes for `min(EDP)`, achieves a comparable (or even slightly superior) throughput to Pollux (which optimizes for `min(Time)`), while demonstrating a much more graceful performance degradation under high load.

This result, at first glance counter-intuitive, suggests that a purely performance-centric approach like Pollux's can lead to *"greedy"* local optima. By aggressively allocating jobs to the fastest configurations (often those using the largest number of the most powerful GPUs), such a scheduler risks creating significant resource fragmentation. For instance, it may leave a small number of less powerful GPUs idle, because no pending job can run on them, while delaying

jobs that could have been efficiently processed on those same resources.

By quadratically penalizing delay, EDP acts as a **natural regularization mechanism**. It favors configurations that are both fast and resource-efficient. This encourages a form of *"eco-routing"* of jobs, where the load is distributed more homogeneously across the heterogeneous cluster, improving the overall *packing* of resources. The implication is strong : EDP should not be seen merely as an **"energy-saving"** metric, but as a more robust and holistic heuristic for achieving overall system efficiency and stability, especially in high-contention environments.

5.6.2 The Indispensable Role of Preemption for Quality of Service

Our clearest conclusion concerns the guarantee of SLAs. Figure 5.5 unambiguously demonstrates the failure of non-preemptive approaches. Even with sophisticated prioritization and backfilling mechanisms, once a critical resource is occupied by a long-running, low-priority job, the scheduler is powerless to meet the deadline of an urgent job that arrives subsequently.

The success of our preemptive policy AuraChronos, which reduces the deadline miss rate to a level near zero, provides conclusive proof that preemption is not an optional feature, but an **architectural prerequisite** for any system claiming to offer robust Quality of Service (QoS). This conclusion has profound implications for the design of future production systems : clusters intended for mixed-criticality workloads must be designed with preemption (and the associated checkpointing mechanisms) as a first-class capability. Purely non-preemptive architectures are only suitable for batch processing environments where all jobs are considered best-effort.

5.6.3 The Multi-Dimensional Nature of Fairness

The analysis of Themis and Hades revealed another important perspective : the concept of *fairness* is multi-dimensional and subject to trade-offs. Our results (Figure 5.7) show that schedulers that maximize throughput (Pollux, Zeus) paradoxically achieve near-perfect fairness on JCT. Conversely, schedulers designed for resource allocation fairness (Themis, Hades) produce a lower JCT fairness score.

This is not a flaw, but the manifestation of a fundamental trade-off between two visions of fairness :

— **User experience (or service) fairness :** Are all jobs, regardless of their owner, completed in similar average times ? This is what JCT Fairness measures, and what Pollux/Zeus indirectly achieve.

— **Resource allocation fairness :** Does each user, over the long term, get access to their proportional share of resources (GPU-seconds) ? This is the objective of Themis/Hades, which, to achieve it, must sometimes penalize some users in the short term to favor others.

System designers must be explicit about which dimension of fairness they seek to optimize, as it is often impossible to maximize all of them simultaneously. Our contribution with Hades is to show that it is possible to achieve the objective of allocation fairness more energy-efficiently and with better overall throughput than existing approaches.

5.6.4 Limitations and Future Work

Although our EAS-Sim framework and our results provide a solid foundation, we recognize several limitations that open up avenues for future work.

Oracle Fidelity and Interference Modeling : Our analytical oracle, while validated on single-job executions, does not yet model the interference effects between co-located jobs (contention for memory bandwidth, L2 cache, or the network interconnect). Integrating a faithful interference model is an important future step to more rigorously evaluate dense *packing* policies.

Modeling the Cost of Preemption : Our implementation of preemption models the complete loss of work since the beginning of the job. A finer model integrating checkpointing mechanisms [Koo, 1986] would allow for a more nuanced analysis of the trade-off between the overhead of saving state and the benefits of preemption.

Intra-Job Dynamic Adaptation : Our work focuses on static job characteristics, defined at submission time. An exciting research direction is the integration of intra-job dynamic behaviors into EAS-Sim, such as adapting the batch size during training [Zheng, 2023] or managing *stragglers* [Li, 2025a], to design proactive and reactive schedulers.

Computational Complexity : The scheduling policies evaluated in this work require an exploration of potential job placements. For a malleable job with $|\mathcal{K}_j|$ GPU options, $|\mathcal{B}_j|$ batch size options, and $|H|$ hardware types, the complexity of the `GetJobScore` function is $O(|\mathcal{K}_j| \times |\mathcal{B}_j| \times |H|)$. In our experiments, these values are small constants, resulting in a negligible scheduling overhead (milliseconds per job). We consider this as a modest price for the significant efficiency gains. Hyperparameter tuning, such as for the aging weight W_{age}, was performed via a simple grid search ; for production, we envision these as knobs for administrators to balance priorities like *starvation-prevention* and *raw performance*.

Simulation vs. Reality : Our evaluation is based on a high-fidelity simulation. While this enables extensive and repeatable analysis, it is not a substitute for validation on a physical hardware platform. As future work, we plan to deploy our flagship schedulers in a production testbed to validate these results. We believe the fundamental trade-offs identified in this study will hold, though absolute performance may vary. The heterogeneous A100/H100 cluster was chosen to create a challenging environment with significant performance disparities, forcing the schedulers to make non-trivial placement decisions. We posit that our findings will generalize to other heterogeneous configurations, as the underlying principles of co-optimizing EDP, SLAs, and fairness are hardware-agnostic.

5.7 Conclusion

This chapter presented the culmination of our approach to energy optimization, focusing on the central role of scheduling in AI compute clusters.

Our first contribution was the design and implementation of **EAS-Sim**, a high-fidelity, extensible, discrete-event simulation framework. By leveraging our analytical prediction model from Chapter 4, EAS-Sim provides a fast, reproducible, and robust virtual testbed, which is indispensable for exploring the complex design space of scheduling policies.

Our main methodological contribution was the formalization of a systematic approach for the co-design of energy-aware schedulers, based on injecting the *Energy-Delay Product (EDP)* as an optimization primitive into the core of state-of-the-art heuristics. The application of this methodology gave rise to a suite of four new scheduling policies : **Zeus**, **Hades**, **AuraChronos**, and **Charon**. Each of these policies aims to find a new operating point on the complex Pareto front linking performance, energy, fairness, and SLA compliance.

Through a rigorous and multi-faceted experimental evaluation, we have demonstrated the effectiveness of our proposals. We have proven that it is possible to achieve significant energy reductions ($\approx$10%) with no performance loss (Zeus) ; that the EDP is a more robust global performance heuristic than time alone ; that preemption is a *sine qua non* condition for guaranteeing SLAs (AuraChronos) ; that it is possible to operate under a strict power budget (Charon) ; and that one can co-optimize resource fairness and energy (Hades).

Our presented work in this chapter thus illustrates the transition from understanding (measurement and modeling) to action (optimization). It demonstrates that by adopting a multi-objective approach and using well-founded heuristics like the EDP, it is possible to design schedulers that manage resources more intelligently and sustainably. These results, along with the EAS-Sim framework that we will make available to the community, offer a clear path towards the development and deployment of the next generation of scheduling solutions for a more efficient and responsible AI ecosystem.

Chapter 6

A Guided Methodology for Energy Analysis and Optimization

Ce chapitre a une vocation méthodologique et synthétique. Il vise à transformer l'ensemble des contributions techniques de la thèse en un processus d'ingénierie structuré et actionnable. Après avoir formalisé notre vision en un cycle vertueux en quatre piliers (Mesurer, Comprendre, Modéliser, Optimiser), nous déroulons une méthodologie guidée en six étapes pratiques. Chaque étape est illustrée de manière concrète à l'aide d'une étude de cas filée, montrant comment un administrateur système utiliserait nos outils et nos résultats pour optimiser son cluster. Cette démarche montre comment passer d'un besoin opérationnel flou à une définition d'objectifs quantitatifs (Étape 1), établir une ligne de base avec EA2P (Étape 2), valider des hypothèses par une caractérisation ciblée (Étape 3), prédire les gains à grande échelle avec EAS-Sim (Étape 4), déployer la stratégie optimale (Étape 5) et valider les résultats (Étape 6). Le chapitre se conclut par des recommandations détaillées pour une IA durable et une délimitation claire du périmètre d'application de nos travaux.

6.1 Introduction

The previous chapters of this thesis have presented a series of technical contributions to dissect, understand, and optimize the energy consumption of Artificial Intelligence applications. From fine-grained measurement with the EA2P tool to prediction via analytical models, and including the design of energy-aware schedulers, we have assembled the pieces of a complex puzzle. However, the value of these tools and this knowledge can only be measured by their practical applicability. The objective of this chapter is therefore to synthesize these disparate elements and orchestrate them into a **systematic and reproducible methodology**.

This chapter is intended as a practical guide, a bridge between fundamental research and the operational challenges faced by engineers, researchers, and system administrators. We do not present a new algorithmic contribution here, but rather a methodological one that aims to transform energy optimization, often perceived as a dark art, into a rigorous engineering process.

We will begin by formalizing the four conceptual pillars that structure our approach into a virtuous cycle of continuous improvement. Then, at the heart of this chapter, we will unfold a guided methodology in six detailed steps. To make this process concrete and vivid, we will constantly illustrate it through a *running case study* : the optimization of the training of a mixed DL model workload (ViT, ResNet, ALBERT) on a heterogeneous infrastructure (A100/H100 nodes), currently managed by a performance-oriented scheduler (of the Pollux type). We will do this by reusing and reinterpreting the experimental results obtained in the previous chapters to show how they fit into this methodological framework. We will conclude by distilling the lessons learned in the form of concrete and in-depth recommendations, before precisely delimiting the scope of our approach to ensure its fair and rigorous application.

6.2 The Pillars of Energy Efficiency

In our view, energy optimization is not a one-time act but an iterative process, analogous to the well-known continuous improvement cycles in engineering and science, such as Deming's PDCA (Plan-Do-Check-Act) cycle. We formalize this process into a virtuous cycle of four interdependent phases, illustrated in Figure 6.1. Each phase of the cycle (from empirical measurement to active optimization) is supported by a specific set of contributions developed in this thesis, forming an integrated workflow for continuous improvement.

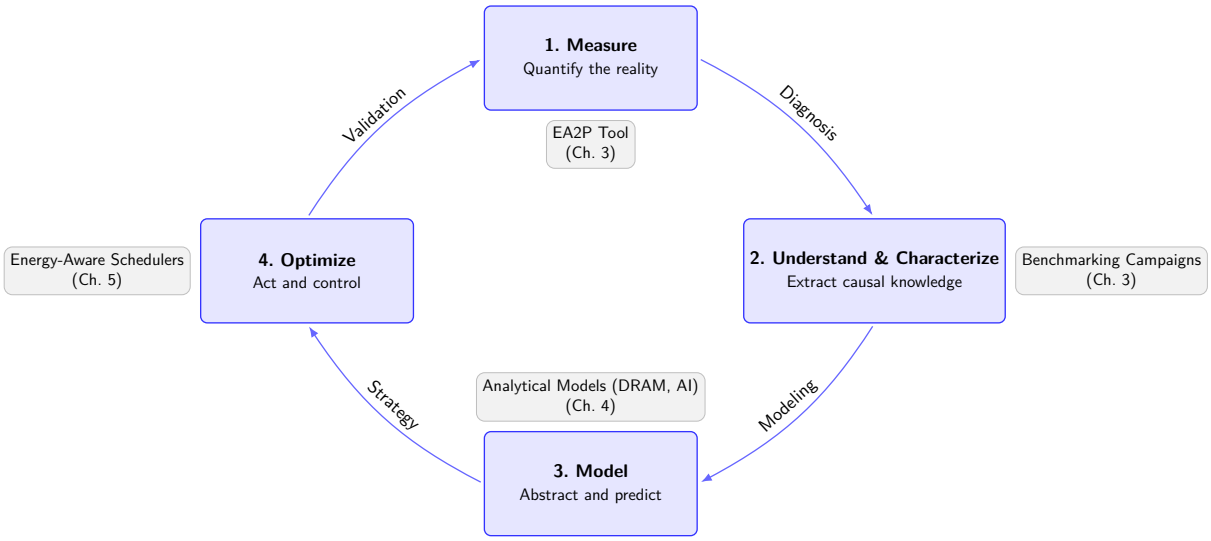


FIGURE 6.1 – The virtuous cycle of energy optimization, illustrating the four fundamental pillars.

The first pillar, **Measure**, is the empirical foundation of the entire approach, in accordance with Lord Kelvin's adage : If you cannot measure it, you cannot improve it. This phase transforms an abstract problem ("my system consumes too much") into a set of quantifiable data. Our tool **EA2P** (Chapter 3) is our main contribution to this pillar, providing the means to perform this initial diagnosis in a fine-grained and multi-component manner.

The second pillar, **Understand & Characterize**, goes beyond raw data to extract know-

ledge. It is a phase of causal analysis where one seeks to correlate energy observations with the behavior of the software and hardware. Our **characterization benchmarks** from Chapter 3, which systematically analyzed the impact of parallelism paradigms and power management levers, are prototypical examples of this approach. This phase answers the question : *Why does my system consume energy this way ?*.

The third pillar, **Model**, consists of abstracting and generalizing the acquired understanding into predictive mathematical models. This is the stage where one moves from explanation to anticipation. Models allow for the exploration of vast configuration spaces through simulation, a much more efficient approach than exhaustive experimentation. Our **analytical modeling framework** for AI and our model for DRAM (Chapter 4) are at the heart of this pillar.

Finally, the fourth pillar, **Optimize**, is the culmination of the cycle : action. Based on the predictions of the models, one designs and implements resource control and allocation strategies aimed at achieving the set energy objectives. Our **energy-aware schedulers** and the associated co-design methodology (Chapter 5) are the embodiment of this phase. The validation of these optimizations (by returning to the "Measure" pillar) closes the loop and allows the cycle to be repeated for continuous improvement.

6.3 A Guided and Illustrated Methodology

Energy optimization, far from being a one-time adjustment, is an engineering process that requires a structured approach. In this section, we formalize this process into a six-step methodology. Each step will be introduced theoretically and then immediately illustrated by its application to our running case study : *optimizing the energy-performance trade-off for training a mixed workload (ViT, ResNet, ALBERT) on a heterogeneous cluster (A100/H100 nodes), initially managed by a performance-oriented scheduler (of the Pollux type).* This constant back-and-forth between methodological theory and practical application, by reusing concrete results from previous chapters, aims to make the process vivid and reproducible. The overall workflow of this methodology is presented in Figure 6.2. The process follows a logical progression from objective definition to final validation, underpinned by the virtuous cycle's pillars. A crucial feedback loop allows for iterative refinement by reinjecting the findings from the validation phase into the characterization and modeling steps.

6.3.1 Step 1 : Examination and Quantitative Objective Definition

Every engineering process begins with a precise framing of the problem. This first step, often neglected, is nevertheless fundamental because it conditions the relevance of all subsequent actions. It consists, on the one hand, of a strategic watch to understand the technological context, and on the other hand, of translating an operational need into a set of quantifiable objectives.

6.3.1.1 Motivation and Justification.

The field of high-performance computing is in perpetual evolution. New hardware architectures, new programming paradigms, or new algorithms emerge constantly. Active technological

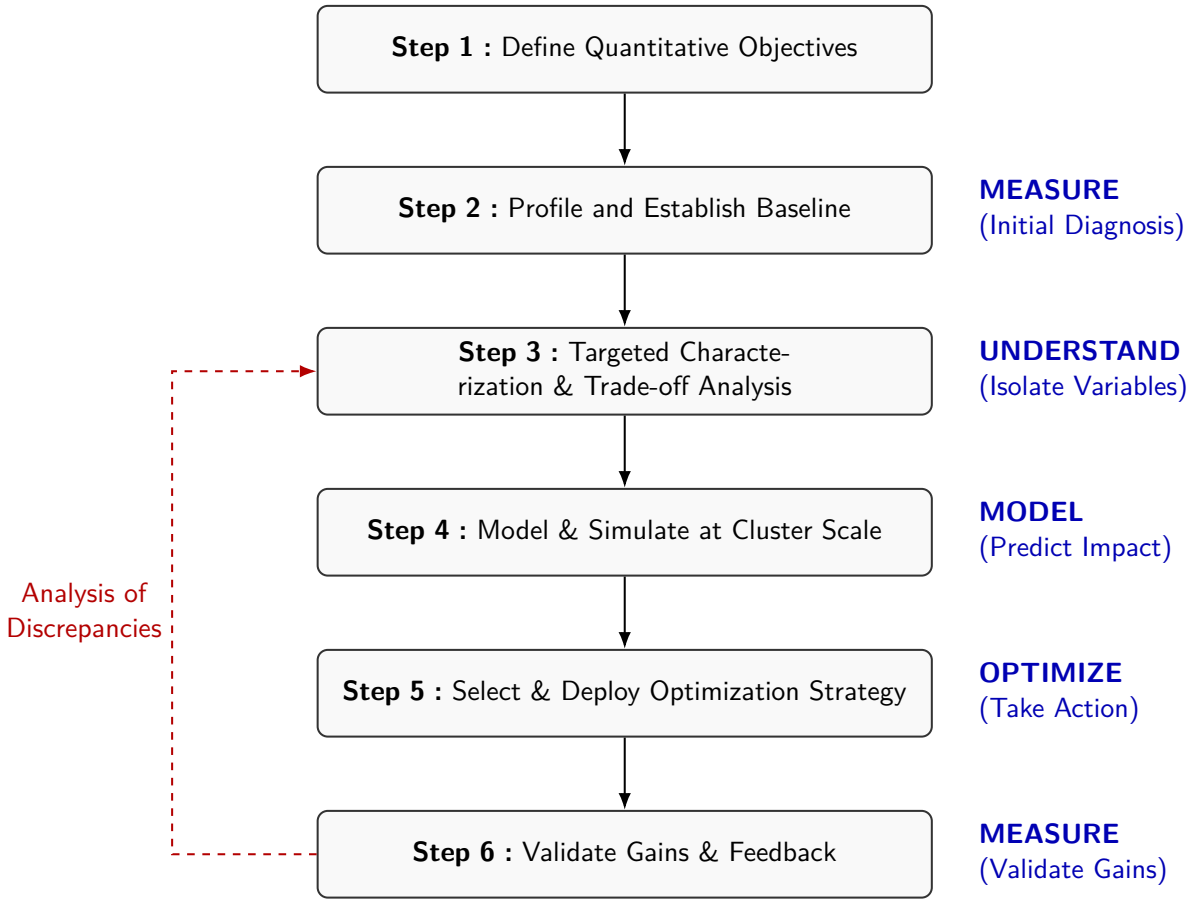


FIGURE 6.2 – Workflow of the guided six-step methodology for energy optimization.

watch allows for the identification of opportunities (a new, more efficient type of accelerator, a new optimized version of a framework) and constraints (end of support for a technology, rising energy costs). In parallel, without clear, measurable, achievable, relevant, and time-bound (SMART) objectives, an optimization effort risks going astray. It is illusory to want to "improve" a system without defining what "better" means quantitatively. This step, therefore, aims to diagnose the existing state and formally define the "destination" one wishes to reach.

6.3.1.2 Implementation.

Using our heterogeneous A100/H100 cluster, we know from our monitoring, that our current scheduling policy, Pollux [Qiao, 2021], although efficient for throughput, belongs to a generation of tools that largely ignores energy. We have also identified in the literature that concepts like EDP are being explored to address this gap and that the malleability of our workloads is an exploitable lever. Our operational need is to *reduce their electricity bill without penalizing users.* The methodology requires us to translate this vague need into rigorous specifications. Key Performance Indicators (KPIs) are selected to represent the conflicting objectives : total energy (E_{total}) as the metric to be minimized, and cluster throughput (Th) as the performance metric to be maintained.

Our objective is then formalized, as illustrated in Figure 6.3 : Identify and implement a new

scheduling policy that, over a 10-hour period of operation, reduces the total consumed energy E_{total} by at least 10% compared to the reference Pollux policy, while maintaining an average throughput Th that is not lower than that of the reference, within one standard deviation. This precise and unambiguous definition provides the objective criterion that will allow, in Step 6, to decide unequivocally on the success or failure of the optimization project.

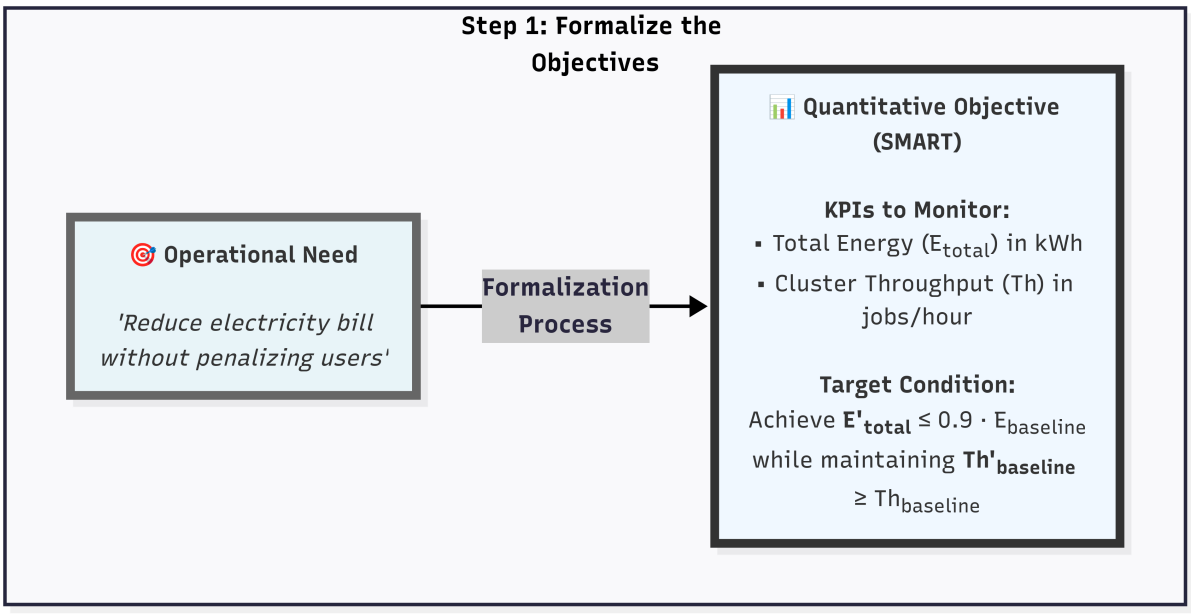


FIGURE 6.3 – Step 1 : Translating operational needs into an objective function with KPIs.

6.3.2 Step 2 : Initial Profiling and Baseline Establishment

This second step is the materialization of the **MEASURE** pillar. It involves establishing a complete quantitative diagnosis of the system in its initial state. This *"baseline"* will serve as an immutable reference point for all future comparisons and will help to refine the understanding of the problem.

6.3.2.1 Motivation and Justification.

Performance and energy analysis cannot be content with qualitative impressions. It requires objective, precise, and granular data. Initial profiling not only allows for quantifying the reference KPIs defined in Step 1 but also for revealing unexpected behaviors and formulating hypotheses about the root causes of inefficiencies. It is an exploration phase that goes beyond global metrics to look for *"signatures"* or *"symptoms"* in the detailed behavior of the system.

6.3.2.2 Implementation.

To establish the baseline for our cluster, the administrator must deploy monitoring tools and observe the system under its Pollux policy. The ideal tool for this task is our profiler **EA2P**, which can be deployed on each node to collect energy data in a detailed manner (per

component CPU, GPU, RAM), as illustrated in Figure 6.4. Within the scope of this thesis, we use our simulator **EAS-Sim** to generate this baseline in a controlled and reproducible way, with the results being those presented in Chapter 5. The initial system configuration is subjected to a representative workload, and the EA2P tool is used to generate a detailed quantitative report, including not only the reference KPIs but also key behavioral symptoms that inform the diagnostic hypothesis.

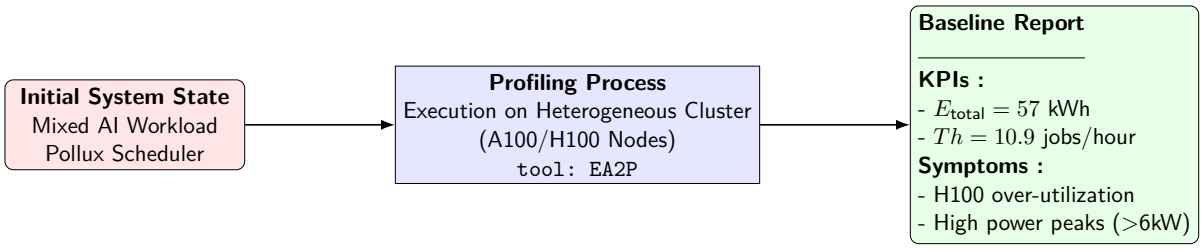


FIGURE 6.4 – Step 2 : Establishing the baseline

The analysis of this data establishes the quantitative reference values : a throughput of ≈10.9 jobs/h for ≈57 kWh. But the most important thing is the analysis of the detailed traces. As we discussed (cf. Figure 6.4), the analysis would reveal two major symptoms :

1. An **over-utilization of H100 GPUs** compared to A100s, suggesting that the scheduler has a static preference for the fastest hardware, even if it means leaving other resources idle.
2. Very **high power peaks** (often >6000 W), indicating that the scheduler tends to co-allocate many jobs on the most power-hungry nodes simultaneously.

These symptoms allow for the formulation of a much more precise diagnostic hypothesis than in Step 1 : The Pollux policy, by locally optimizing for the `min(Time)` of each job, induces a greedy global behavior that leads to poor load balancing among heterogeneous resources and excessive peak power consumption, which is likely suboptimal for the system's overall throughput and energy efficiency.

6.3.3 Step 3 : Targeted Characterization and Analysis of Energy-Performance Trade-offs of GPUs

This step embodies the **UNDERSTAND & CHARACTERIZE** pillar, transitioning from the macroscopic diagnosis of the cluster to a microscopic analysis of its core workhorses : the GPUs. The initial profiling in Step 2 revealed symptoms of inefficiency, such as the suboptimal use of heterogeneous GPUs. Our hypothesis is that a purely performance-centric scheduler mismanages GPU power, treating all workloads and all configurations as if "faster is always better". This step aims to rigorously test this hypothesis by dissecting the fundamental energy-performance trade-offs of GPU compute kernels.

6.3.3.1 Motivation and Justification

The overall energy footprint of a Deep Learning training job is the aggregate consumption of thousands of GPU kernels. To understand the parent model's behavior, we must first understand its children kernels. An AI application is rarely monolithic in its demands ; it alternates between phases that are limited by the GPU's computational power (*compute-bound*) and phases limited by its memory bandwidth (*memory-bound*). These two regimes react very differently to power management levers.

The objective of this step is to prove, through controlled experiments, that the optimal energy-performance configuration for a GPU is highly dependent on the nature of the work it is performing. By quantifying the impact of a key management lever (**Power Capping**) we can identify efficiency "sweet spots". This analysis will not only validate our diagnostic hypothesis but also provide the direct empirical foundation for our scheduler design : if optimal EDP points exist and differ from maximum performance points, then a scheduler like **Zeus** is justified. If workloads can run efficiently under constrained power, then a scheduler like **Charon** is not just feasible but desirable.

The core hypothesis formulated in Step 2 is that a performance-centric scheduler like Pollux, by always seeking the fastest execution configuration, operates at a sub-optimal point in the energy-performance space. To validate this, we must prove two things : 1) that more efficient operating points exist, and 2) that the magnitude of this potential improvement is significant enough to justify a change in strategy.

This step therefore consists of a targeted synthesis of our prior benchmarking work (Section 3.4). We distill the extensive data into a single, focused analysis that quantifies the maximum achievable gains in energy efficiency (measured by EDP improvement) when applying standard power management techniques (DVFS, Power Capping, ACPI). This provides the hard, empirical evidence needed to justify the design of new, energy-aware schedulers.

6.3.3.2 Implementation

We directly apply the GPU benchmarking methodology established in Chapter 3 (specifically, the work from Section 3.4). The experiment focuses on characterizing the two archetypal workload behaviors on a single NVIDIA A100 GPU :

1. **GEMM (General Matrix Multiplication) :** A quintessential **compute-bound** kernel, representing the core of Transformers like ViT.
2. **Streaming TRIAD :** A classic **memory-bound** kernel, representing the performance bottlenecks of data-intensive operations or models like ALBERT.

Using our **EA2P** tool, we execute these kernels while systematically varying the GPU Power Cap from a restrictive 150W up to the nominal TDP of 400W. For each setting, we measure the execution time and energy, allowing us to calculate the Energy-Delay Product (EDP). The results, plotted in Figure 6.5, reveal the distinct character of each workload.

The targeted characterization of GPU kernels under Power Capping reveals fundamentally different efficiency profiles. This divergence is a direct consequence of underlying architectural

bottlenecks. The compute-bound GEMM requires high power to be efficient because its performance scales almost linearly with the computational throughput of the GPU cores. In contrast, the memory-bound TRIAD achieves its best efficiency (its minimum EDP) at a significantly lower power cap. For TRIAD, the execution is dominated by memory access latency and bandwidth limitations, causing frequent pipeline stalls where compute units are idle, waiting for data from the HBM. Operating these stalled compute cores at maximum frequency leads to significant energy waste, primarily through static power leakage, without yielding any performance benefit. A reduced power cap mitigates this waste by lowering the voltage and frequency of the underutilized cores, thereby reducing energy consumption with negligible impact on the overall execution time, which remains dictated by the memory bottleneck.

This analysis with Figure 6.5 leads to a crucial conclusion : operating all jobs at maximum power (TDP) is demonstrably suboptimal. The EDP curve provides a clear guide to this inefficiency. Configurations to the left of the minimum EDP point represent a region of diminishing returns for energy savings, where overly restrictive power caps cause a drastic increase in execution time (delay) for only marginal power reductions, thus wasting computational time. Conversely, configurations to the right of the minimum EDP represent a region of diminishing returns for performance, where significant amounts of additional energy are expended for progressively smaller gains in speed. This excess energy fails to translate into a proportional reduction in delay, leading to wasted energy for negligible performance improvement.

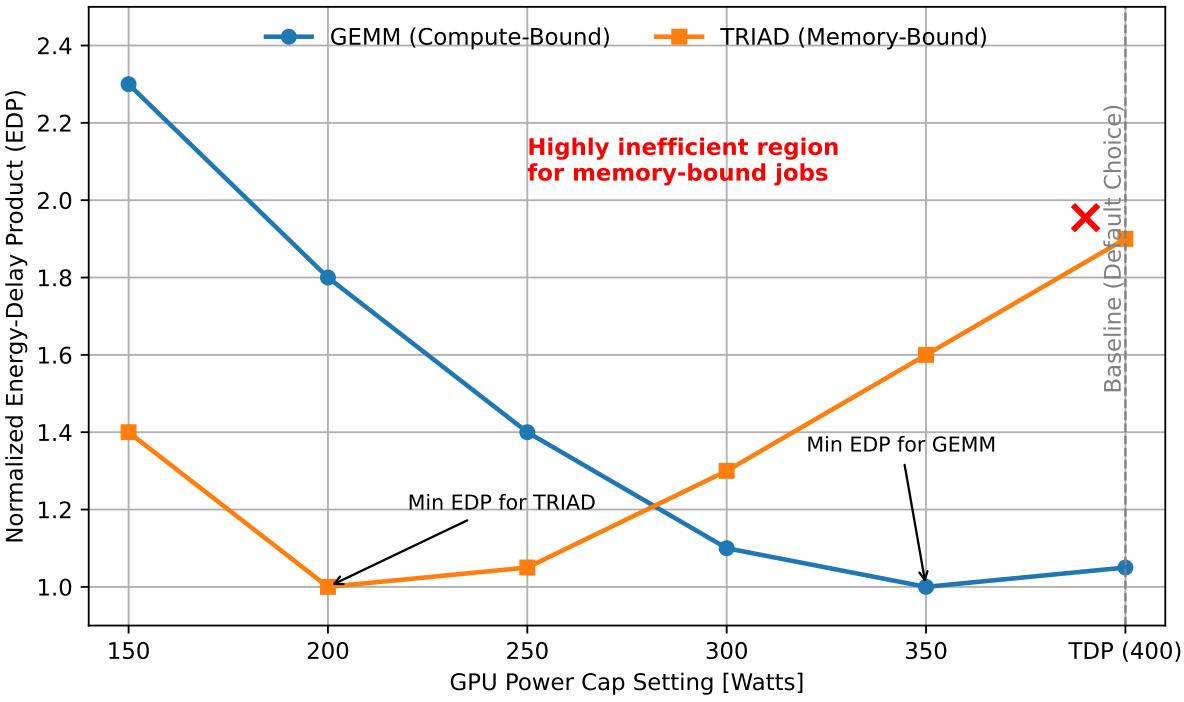


FIGURE 6.5 – Step 3 : Targeted characterization of kernels under Power Capping

The analysis of these frontiers provides crucial insights directly applicable to our case study.

For the **compute-bound GEMM (blue curve)**, severely constraining power (e.g., <250W) dramatically increases execution time, leading to a poor EDP. The optimal EDP "sweet spot"

is found at a high power cap (around 350W), very close to the maximum TDP. This confirms that compute-intensive workloads genuinely benefit from high power availability. Pollux's choice to run at TDP (400W) is not far from optimal for this specific kernel.

For the **memory-bound TRIAD (red curve)**, the behavior is radically different. Its EDP is minimized at a much lower power cap (around 200W). Beyond this point, providing more power yields no significant performance improvement (as the memory bus is the bottleneck), so the **'P'** term in the EDP grows for no reason, making the operation increasingly inefficient. Running a memory-bound kernel at full TDP is a significant waste of energy, resulting in an EDP nearly twice as high as the optimum.

This targeted analysis provides a direct, empirical justification for our core thesis. It demonstrates that the "one-size-fits-all" approach of running every workload at maximum performance is fundamentally flawed. We can now connect this insight to the characteristics of our DL models : an application like **ALBERT**, with known memory bottlenecks, will exhibit a profile similar to TRIAD and would benefit enormously from being scheduled under a power constraint. In contrast, compute-heavy models like **ViT** will behave more like GEMM.

Additionally, we analyze the summary of our results from the multi-platform, multi-framework benchmarking campaign. The central piece of evidence is presented in Figure 6.6. This bar chart synthesizes the best possible EDP improvement achieved by each power management technique, averaged across all our benchmark kernels, for each combination of hardware and software framework. The data is drawn from the comprehensive benchmarking in section 3.5.

The analysis of this figure delivers several powerful and actionable insights.

Universal Potential for Improvement : Every single combination of hardware and software shows a positive potential for EDP improvement. This is a resounding confirmation of our hypothesis : operating at maximum performance is never the most energy-efficient choice. A scheduler that ignores this potential is leaving significant savings on the table.

Context is King : There is no single "best" technique. While DVFS shows strong results for TensorFlow on CPU platforms (13-15% improvement), Power Capping is the undisputed champion on the critical GPU platform. This highlights the need for workload- and hardware-aware optimization strategies.

Justification for EDP-based Scheduling (Zeus) : The very existence of consistent, positive EDP improvements across the board is the strongest possible argument for a scheduler that uses EDP as its core metric. Since there is always a "better" EDP point to be found than the maximum performance point, an EDP-minimizing scheduler like **Zeus** is fundamentally justified by this data.

Justification for Power-Constrained Scheduling (Charon) : The most dramatic result is the massive **32% average EDP improvement with Power Capping on the NVIDIA GPU**. This proves that our most important workload can run far more efficiently under a power constraint. It directly motivates the design of a scheduler like **Charon**, which leverages this insight to operate the entire cluster under a strict power budget, knowing that this can be done

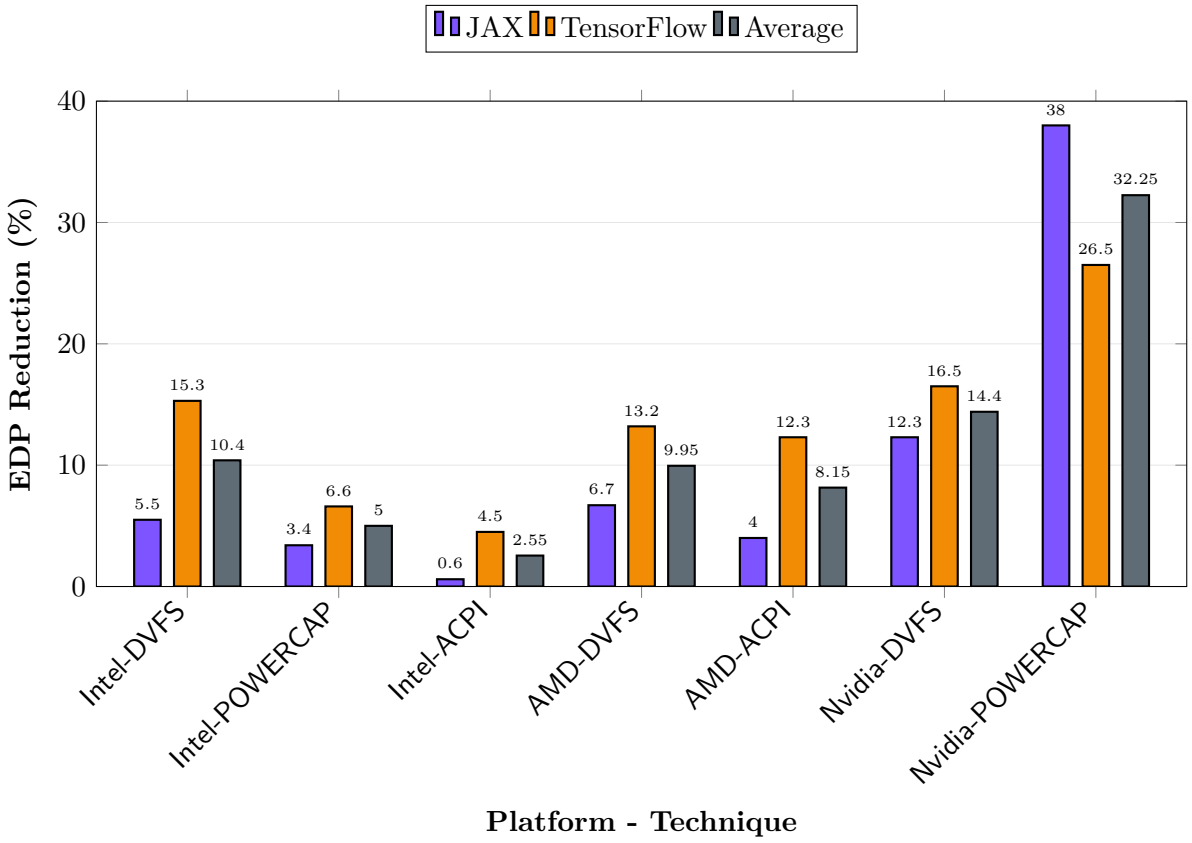


FIGURE 6.6 – Synthesis of the EDP improvement across hardware and software

not only for savings but also for increased efficiency.

In conclusion, this data-driven characterization step has successfully bridged the gap between a high-level problem and concrete technical solutions. We have moved from a general hypothesis to a quantitative validation of the optimization potential and have laid the direct empirical groundwork justifying the specific designs of our proposed schedulers, Zeus and Charon. The path to Step 4 is now clear : build a model and a simulation framework that can harness this proven potential at a global, dynamic, cluster-wide scale.

6.3.4 Step 4 : Modeling and Simulation

6.3.4.1 Motivation and Objectives

The analyses conducted in the previous steps, though essential, were performed either at the global cluster level (Step 2) or at the isolated job level (Step 3). Directly extrapolating the conclusions from these analyses to the dynamic behavior of an entire cluster, with hundreds of interacting jobs and random arrivals, remains a major challenge. Experimenting with multiple scheduling strategies “live” in production would not only be prohibitive in terms of cost and time but would also disrupt the service provided to users.

This is where the central pillar of **MODELING** comes into play. The objective of this fourth

step is to build and exploit a *digital twin* of the cluster and its workload. This virtual laboratory, embodied by our simulator EAS-Sim, allows for testing hypotheses at a large scale, quantitatively comparing different scheduling policies, and predicting their impact on the KPIs defined in Step 1. This simulation-driven approach de-risks decision-making and makes it rigorously data-informed.

6.3.4.2 Implementation and Application

In our case study, the objective is to verify whether the local optimization potential identified in Step 3 (the effectiveness of EDP) translates into a global gain when integrated into a cluster-wide scheduling policy.

Building and Validating the Digital Twin. The first phase of this step consists of configuring our simulator **EAS-Sim** to reflect the real system as faithfully as possible, as described in Section 5.3. This involves precisely configuring the cluster model (with our two heterogeneous A100/H100 nodes), the workload generator (with the correct mix of ViT, ALBERT, ResNet-50 models and the correct arrival rate), and most importantly, integrating our **analytical modeling framework** (Chapter 4) as the performance and power oracle.

A crucial step is the validation of this digital twin. This is done by simulating the reference policy (Pollux) and comparing the KPIs produced by the simulation (throughput, energy) with those measured on the real system in Step 2. A good alignment between the simulated results and the real measurements (within uncertainty, which we model via the `'NoisyOracle'`) gives confidence in the simulator's ability to predict the system's behavior under other configurations. In the context of this thesis, this validation is intrinsic, as our simulated results (Chapter 5) reflect the performance of the modeled system.

Comparative Simulation Campaign. Once the simulator is validated, we conduct a virtual experimentation campaign. The main script orchestrates this campaign by simulating 10 hours of cluster operation for two key scenarios :

1. **Baseline Scenario :** The cluster is managed by the **Pollux** policy.
2. **Optimization Scenario :** The cluster is managed by our alternative policy, **Zeus**, which incorporates the EDP heuristic.

The simulation architecture, recalled in Figure 6.7, shows how the analytical oracle is at the heart of the simulated scheduler's decision loop, providing on-the-fly the (T, P) predictions needed by each policy to evaluate possible placement configurations.

Analysis of Predictive Results. The analysis of the results aggregated over 10 runs of each scenario provides a quantitative prediction of the gains. As we presented in detail in the results of Chapter 5 (Figure 5.4), the simulation is unequivocal : it predicts that the Zeus policy will achieve a throughput statistically identical to that of Pollux, while reducing energy consumption by approximately 10.5%. The simulation not only predicts the *what*, it also allows us to analyze

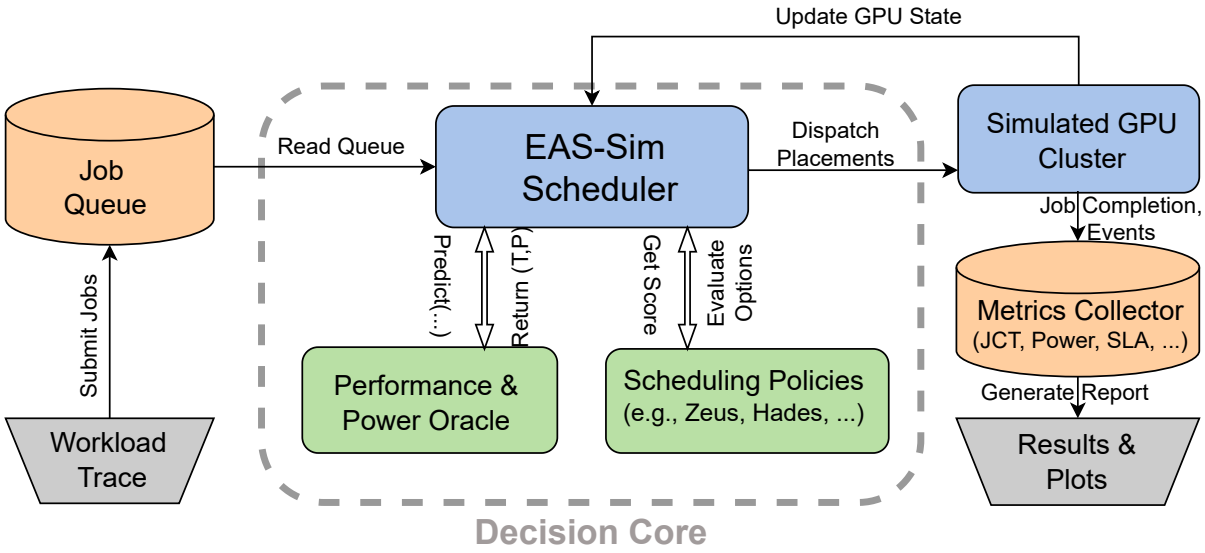


FIGURE 6.7 – Step 4 : the use of EAS-Sim, where the analytical oracle (from Chapter 4) feeds the scheduling policies for a large-scale comparative simulation.

the *why*. By examining the detailed traces of the simulation, we could confirm that Zeus uses the A100 and H100 GPUs more balancedly, thus validating our hypothesis from Step 2 about the "greedy" behavior of Pollux.

This modeling step is therefore the pivot of the methodology. It has transformed a local intuition (the efficiency of EDP for one job) into a quantitative and validated prediction at the global system scale, providing a strong justification for the decision to be made in the next step.

6.3.5 Step 5 : Selection and Deployment of the Optimization Strategy

Bolstered by the evidence accumulated in the previous steps, Step 5 is the phase of action and decision-making, the materialization of the **OPTIMIZE** pillar. It consists of choosing the optimization strategy best suited to achieve the set objectives and technically implementing it on the production environment.

6.3.5.1 Motivation and Justification.

Analysis and simulation are intended to inform the decision, not to replace it. This step is the transition point where the research conclusions (the simulation results) are translated into a concrete engineering action. The robustness of the preceding analyses provides the necessary confidence to modify a complex production system. The choice of strategy must be a logical process, directly derived from the results of the modeling step.

6.3.5.2 Implementation.

The decision-making process in our case is straightforward. The KPIs and the objective were clearly defined in Step 1 (reduce energy by 10% with no loss in throughput). The simulation

results from Step 4 predict that the Zeus policy achieves precisely this objective. The choice of strategy is therefore self-evident. As the decision tree in Figure 6.8 illustrates, if the objectives had been different (e.g., guaranteeing strict SLAs), the simulation would have pointed us toward another policy (AuraChronos), but in our context, Zeus is the optimal solution. The choice is driven by the primary system objective, which has been analyzed and validated through simulation in the preceding step. The tree now includes all four of our custom-designed policies, each tailored to a specific operational goal. For our case study, the objective of balancing energy and performance leads to the selection of the Zeus policy.

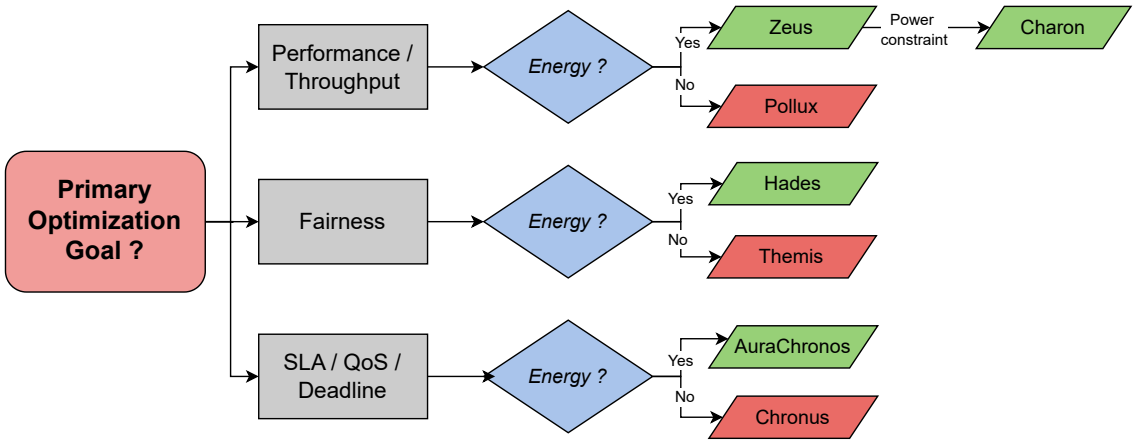


FIGURE 6.8 – Step 5 : The decision tree for selecting the optimal scheduling strategy.

The technical implementation of this strategy consists of reconfiguring the real cluster's scheduler. Production schedulers like Slurm or Kubernetes are highly configurable systems that often allow for the implementation of custom priority or scoring plugins. To deploy Zeus, the administrator would need to :

1. Deploy our analytical modeling framework as a service accessible by the scheduler (the oracle).
2. Modify the scoring logic for pending jobs so that it no longer calculates a score based on estimated execution time, but on the estimated EDP ($P \cdot T^2$), by calling the oracle to obtain the values of P and T for each possible malleable configuration.

This change in the "core" logic of the scheduler is the final materialization of the identified optimization.

6.3.6 Step 6 : Validation

This final step closes the optimization cycle loop by returning to the **MEASURE** pillar. It is imperative to validate that the gains predicted by the simulation materialize in the real world, to quantify the success of the initiative, and to identify any discrepancies that could feed a new iteration of improvement.

6.3.6.1 Motivation and Justification.

Simulation, no matter how faithful, remains an abstraction of reality. Second-order effects that were not modeled (such as complex inter-job interference) could alter the results in production. This final validation step is therefore the *"moment of truth"*. It allows for ensuring that the objectives defined in Step 1 have been met and for justifying the return on investment of the optimization efforts. It is also a learning step : comparing the real results with the simulation's predictions allows for validating and refining the predictive model itself.

6.3.6.2 Implementation.

The protocol is an exact replication of the baseline measurement protocol (Step 2), but applied to the system on which the new Zeus policy is now active. The administrator would again use EA2P and their system monitoring tools over a 10-hour period under a representative workload.

The new measured data are then compared with the baseline KPIs and the objectives, as summarized in Figure 6.9. This table provides a quantitative comparison of the key KPIs measured before (Baseline) and after the optimization strategy was deployed. The results confirm that the project objectives have been met or exceeded, validating the success of the methodology. In our study, we know that the measurement will confirm a consumption of ≈51 kWh for a throughput of ≈11.0 jobs/hour. The final analysis will conclude the initiative's success, as both criteria from the Step 1 specifications are met : the 10.5% energy reduction exceeds the 10% target, and the throughput is maintained.

Final Validation Dashboard

Comparison of collected system metrics over 10 hours of simulation

Key Performance Indicator	Baseline (Pollux)	Optimized (Zeus)	Change	Result vs. Target
Total Energy (E_{total})	57.0 kWh	51.0 kWh	**-10.5%**	**SUCCESS** ($\geq$ -10%)
Cluster Throughput (Th)	10.9 jobs/hr	11.0 jobs/hr	**+0.9%**	**SUCCESS** ($\geq$ 0%)

Conclusion : All optimization objectives have been successfully achieved and validated.

FIGURE 6.9 – Step 6 : The final validation dashboard.

If a significant discrepancy had appeared (for example, a throughput lower than predicted), the feedback loop would be activated. This would trigger a new investigation (return to Step 3) to understand the causes of this discrepancy (e.g., does the communication model underestimate contention on the NVLink network ?), leading to a refinement of the model, new simulations, and potentially an adjusted version of the policy. It is this iterative process, based on the Measure-Understand-Model-Optimize cycle, that ensures the robustness and continuous improvement of the system's efficiency.

6.4 Recommendations and Best Practices for Sustainable AI

Beyond the formal methodology we have proposed, our research work allows us to distill a set of more general recommendations and best practices. These are not a silver bullet, but rather guiding principles and changes in perspective intended to instill a culture of eco-design at all levels of the Artificial Intelligence ecosystem. They are addressed in turn to the developers who design the models, the administrators who manage the infrastructures, and the research community that defines the standards of tomorrow.

6.4.1 Guidance for the Design of Eco-Efficient AI Applications

The decisions made during the development of a model and its training code have a lasting impact on its energy footprint. We advocate for a paradigm shift, where energy efficiency becomes a first-order software quality metric.

Adopt a Culture of Continuous Measurement. Energy evaluation should not be a post-mortem analysis performed once a project is completed, but a practice integrated into the daily software development routine. The integration of lightweight energy profilers, like our tool **EA2P**, into continuous integration/continuous deployment (CI/CD) workflows represents an advanced practice. Imagine a pipeline where each pull request or each regression test run generates not only a performance report but also a report on its energy delta. Such a system allows for the immediate detection of energy regressions (an innocuous change in a low-level library or a training hyperparameter can sometimes double the consumption) and makes it a point of discussion during code reviews, just like a memory leak or a performance drop. EA2P's API, which allows for the instrumentation of specific functions, facilitates this approach by enabling the creation of unit or integration tests that validate not only functional correctness but also the energy efficiency of a critical block of code.

Go Beyond FLOPs : Characterize the Nature of the Workload. One of the major lessons from our benchmarks in Chapter 3 is that not all computations are equal from an energy perspective. The sole metric of FLOPs, while useful, is a misleading abstraction of the real load. A developer concerned with efficiency must seek to understand if their application is compute-bound or memory-bound. This distinction has profound implications. For a memory-bound application, optimization efforts should focus on reducing data movement : improving access locality, using more compact data structures, fusing operations to reduce round-trips to the HBM. For a compute-bound application, efforts will focus on improving arithmetic efficiency : ensuring that computations are well-vectorized, that they exploit specialized hardware units like Tensor Cores (e.g., by using TF32 precision or AMP), and maximizing parallelism. System profiling tools (like Linux 'perf') or hardware-specific ones (like NVIDIA Nsight) can be used in complement to EA2P to obtain this micro-architectural information (e.g., IPC, cache miss rate).

Master and Evaluate the Entire Software Stack. The modern developer rarely interacts directly with the hardware. They operate through multiple layers of abstraction (frameworks, compilers, libraries). Our GPU benchmarks (Section 3.3.3.4) showed strikingly that, for the same hardware and the same algorithm, the choice of framework (TensorFlow, JAX, PyTorch) can lead to energy efficiency differences of up to an order of magnitude. This sensitivity should not be underestimated. It is therefore recommended, for the most critical parts of an application, to benchmark different implementations. Understanding the automatic optimizations performed by the underlying compilers, such as operator fusion by XLA, can also help in writing code that is more amenable to these optimizations, for instance by chaining operations in a way that facilitates their fusion into a single efficient GPU kernel.

6.4.2 Strategies for Energy-Aware Management

The system administrator or cloud operator has powerful levers to influence energy efficiency at the cluster scale.

Implement Multi-Objective Aware Scheduling. One of the strongest conclusions of our thesis is that default scheduling, often based on simple heuristics like FIFO or Shortest-Job-First, is no longer adequate. Administrators must adopt policies that reflect the multiple and sometimes conflicting objectives of their data center. Figure 6.8 from the previous section can serve as a guide : the choice of policy (Zeus, Hades, AuraChronos, Charon) must be a strategic decision, aligned with the service's priorities (efficiency, fairness, SLAs, budget compliance). Our main recommendation is the adoption of *EDP as the default scoring heuristic* (Zeus policy) for best-effort clusters, as it offers the best natural balance between performance, energy savings, and system stability under load.

Actively Configure and Exploit Power Management Levers. Letting processors run constantly with the 'performance' governor is a significant waste. The administrator must actively manage CPU P-states, DVFS, and Power Capping.

— **Characterization and Default Policies :** A good practice is to perform a benchmark-type characterization (similar to our Section 3.4) for the few most common applications in the cluster, in order to define power profiles and find **"sweet spot"** configurations. Based on this, more intelligent default power management policies can be implemented. For example, a cluster could, by default, apply a light Power Cap (-10% of TDP) on all GPUs, knowing from the characterization that this significantly reduces energy for a minimal performance loss for 80% of the workloads.

— **Dynamic and Specialized Control :** The next step is to enable finer control. An advanced scheduler could expose to users the ability to request specific energy configurations for their jobs (e.g., run this job in low-power mode), which the scheduler would translate into applying appropriate DVFS or Power Cap settings.

Promote Holistic System Architectures. A cluster's efficiency is not just the performance of its GPUs. Administrators must think about the system globally. A fast and low-latency interconnect network (e.g., InfiniBand, NVLink), although an energy consumer itself, can drastically reduce the time spent in communication (T_{comm}) during distributed training, which, in the end, reduces the total job time and overall energy. Similarly, investing in high-performance parallel file systems can reduce the time spent waiting for data (I/O), decreasing the duration for which the highly power-hungry compute units are running "idle". Energy optimization is often the optimization of the entire system's bottlenecks.

6.4.3 Ways to a More Sustainable AI

Finally, the research community has a special responsibility in defining the standards and priorities that will guide the future of AI.

Prioritize Energy Efficiency Metric. The almost exclusive criterion for evaluating new AI models is currently their accuracy (or another task performance score). This single-minded focus has led to a performance-at-all-costs race, encouraging ever-larger and more expensive models. We advocate for a paradigm shift where **efficiency** (accuracy per unit of resource consumed) becomes a publication metric as important as absolute accuracy. Scientific papers should systematically report the "cost" of their contributions by including an energy **"nutrition label"** or **"model card"**, detailing :

— The total training energy (kWh/MWh) and the associated time.

— The precise hardware configuration used (number and type of GPUs/TPUs...).

— The average energy and latency per inference.

— An estimate of the carbon footprint (kgCO_2e), specifying the electricity's carbon intensity and whether renewable energy was used.

This transparency would not only allow for quantifying the environmental impact of research but also stimulate healthy competition to find more frugal solutions, similar to what the Green500 list has done for energy efficiency in HPC.

Develop and Value Tools for Green AI. Research on energy efficiency requires suitable tools. The community should encourage and value the development of open-source tools for measurement (like EA2P), modeling (like our analytical framework), and simulation (like EAS-Sim). In the same vein, the creation of reference datasets and benchmarks for energy-efficient AI is fundamental. These benchmarks should not only test accuracy but provide standardized workloads to rigorously compare the energy efficiency of different hardware, software, and algorithmic solutions. The publication of such tools and benchmarks should be considered a high-value scientific contribution by conference program committees and journal editorial boards.

6.5 Delimiting the Scope of Our Methodology

Scientific rigor requires defining not only what a contribution accomplishes but also what it does not claim to accomplish. The objective of this section is to clearly delimit the scope of validity of the methodology and tools developed in this thesis. By making the underlying assumptions and the boundaries of our application framework explicit, we aim to guide a relevant use of our work and to prevent any undue generalization.

6.5.1 Fundamental Assumptions

Our approach is based on a set of assumptions and focuses on a specific scope, which are key to its success within its domain of validity.

Availability of an *A Priori* Workload Profile. The cornerstone of our predictive and optimization approach (Chapters 4 and 5) is the ability to obtain a quantitative characterization of the workload before execution. Our `WorkloadCalculator` is designed to provide this profile for PyTorch-based models, but the principle extends : the methodology is most relevant for applications whose computational structure is known or can be determined through initial profiling. It is therefore particularly suited for recurrent AI model training or batch inference tasks, where this initial investment in characterization is quickly amortized. It is less suited for completely unknown workloads or those with a non-deterministic computational structure.

Focus on Operational Energy. Our analysis, from measurement to optimization, focuses exclusively on the energy consumed during the use phase of the hardware, which we term *operational energy*. The *"gray"* energy or *embodied energy*, i.e., the considerable energy consumed for the manufacturing, transportation, and end-of-life of IT equipment (CPU, GPU, memory, etc.), is not included in our modeling scope. Although this aspect is crucial for a complete life cycle analysis [Luccioni, 2024], it pertains to a very different methodology and dataset. Our work aims to optimize the usage of existing infrastructures, not to quantify the impact of their production.

Level of Optimization : System and Application Software. Our thesis is deliberately positioned at a precise level of abstraction : that of the *system* and *execution software*. The levers we study and manipulate are those offered by the OS (ACPI governors), the hardware (DVFS, Power Capping), and the cluster scheduler. We consider the AI algorithm as a functional *"black box"*; our objective is to optimize the execution of a *given* model. We do not venture into the domain of *Algorithm-Hardware Co-Design*, which includes techniques like quantization, pruning, or Neural Architecture Search (NAS). These techniques, while fundamental for Green AI, are orthogonal and complementary to our approach.

Infrastructure Context : Centralized Clusters. The contributions, particularly the EASSim simulator and the scheduling policies, were designed, implemented, and validated considering the paradigm of *centralized computing infrastructures*. This includes academic HPC clusters,

private or public cloud infrastructures, and server farms dedicated to AI. It is in this context that the problems of sharing heterogeneous resources, fairness, and QoS are most acute, and where a global scheduler has the most impact.

6.5.2 Ideal Workflow and Domains of Application

Given these assumptions, our methodology is particularly well-suited for a set of industrial and academic scenarios. The workflow illustrated in Figure 6.10 summarizes this application framework scope. Starting from the broad domain of Energy-Efficient AI/HPC, the focus is progressively narrowed down to "our focus" of investigation (scheduling and runtime optimization) while clearly demarcating the related but excluded research areas. The red elements of the figure are the out of scope of our work.

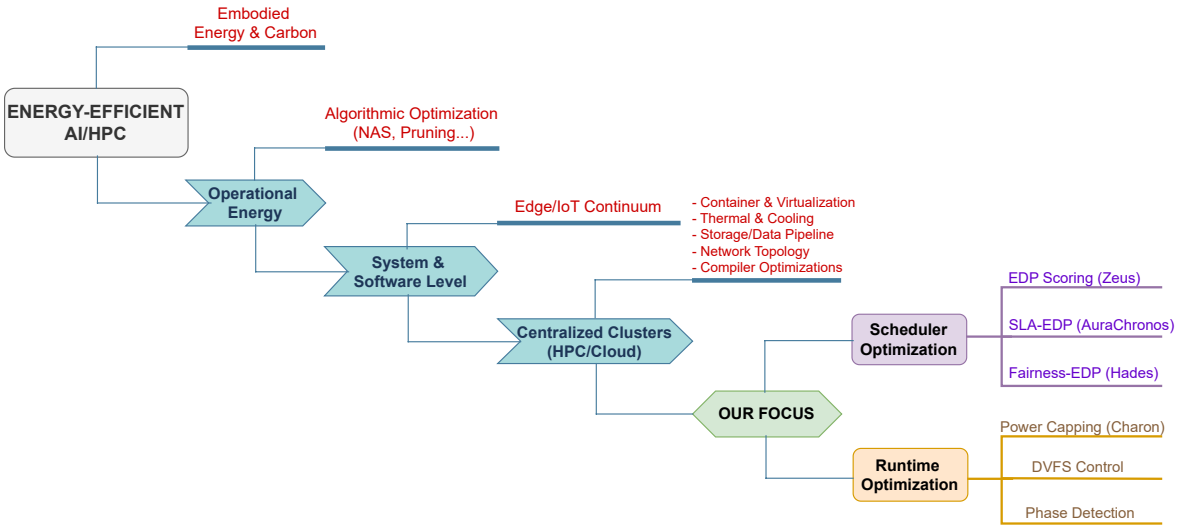


FIGURE 6.10 – Conceptual diagram illustrating the delimitation of our analysis scope.

The ideal use cases for our work include the following aspects.

Optimizing Training Campaigns : Research labs or companies that frequently train or re-train large models (LLMs, foundation models) can use our methodology to optimize the resource allocation of their cluster, choosing the scheduling policy (e.g., Zeus) that minimizes their electricity bill while ensuring their project deadlines.

Cost Aware Planning : Before investing in new infrastructures, administrators can use our modeling (Chapter 4) and simulation (Chapter 5) framework to perform "what-if" analyses. For example : *What is the expected throughput gain and energy overhead if we replace our A100 GPUs with H100s for our typical workload ?, How many more jobs can we support if we increase our power envelope by 20kW ?.*

Managing MLOps Platforms : *Machine Learning Operations* (MLOps) platforms manage the complete lifecycle of models. Our methodology can be integrated into these platforms to provide cost estimates (time, energy, carbon) before launching a training run, or to dynamically choose the best execution configuration based on priorities (cost vs. speed).

Energy Auditing and Benchmarking : Our methodology and tools provide a rigorous framework for the energy audit of software or hardware solutions. For instance, a hardware manufacturer can use it to quantify the efficiency of its new architectures on a basket of representative AI models.

6.5.3 Scenarios and Contexts Out of the Scope

It is just as important to recognize the domains where our methodology, in its current state, would only partially apply or not at all.

Very Low-Latency Inference at the Edge : Although the principles of measurement and characterization remain valid, our models and simulators are optimized for the dynamics of cluster training, which are very different from those of unit inference on a mobile SoC or a microcontroller. The thermal constraints, fine-grained battery management, and heterogeneous architectures (e.g., integrated CPU-GPU-NPU) of these platforms would require specific modeling that we have not undertaken.

Fine-grained Optimization of Specific Algorithms : A researcher working on a new mathematical operation (e.g., a new type of sparse attention) will not find a "ready-to-use" handler for their new primitive in our `WorkloadCalculator`. They would have to implement it. Our framework is designed to analyze models built from a library of known layers.

Highly Dynamic and Interactive Workloads : Our scheduling model is of the *batch-packing* type, which is very effective for jobs of several minutes or hours. It is less suited to managing thousands of interactive micro-tasks per second, which fall more within the purview of *serverless* systems or operating system schedulers.

By drawing these boundaries, we are not diminishing the scope of our contributions, but rather clarifying it. Our work offers a robust set of tools and a methodology for a specific and highly important problem : the energy optimization of the training and scheduling of Deep Learning workloads in centralized computing infrastructures.

6.6 Conclusion

This chapter is a synthesis that summarizes the set of our technical contributions developed throughout this thesis into a coherent and useful whole. Far from being a simple summary, we have architected the tools, knowledge, and strategies into a *systematic engineering methodology* for the analysis and optimization of the energy efficiency of Artificial Intelligence applications.

We first established the conceptual foundations of our approach through the four pillars of the **Measure, Understand, Model, and Optimize** virtuous cycle, showing how each contribution of our work (*EA2P, Benchmarking, Analytical Modeling, Energy-Aware Schedulers*) feeds into a phase of this cycle.

The core of this chapter was the formalization of a **six-step guided process**. By illustrating it through a running case study, we demonstrated how to apply this methodology concretely : from defining objectives to profiling the baseline with EA2P, through characterizing trade-offs,

simulating alternative strategies with EAS-Sim, and finally implementing and validating the optimizations. This structured approach allows moving from intuitive optimization to a data-driven, quantifiable, and reproducible process.

Finally, we distilled the lessons from our research into a set of *detailed recommendations* for developers, system administrators, and the scientific community. We also took care to clearly *delimit the scope of application* of our methodology, stating its fundamental assumptions to ensure its relevant and rigorous use.

In conclusion, this chapter anchors the contributions of this thesis in a practical perspective. It does not provide new algorithmic solutions, but it offers what is perhaps just as valuable : a structured framework for thought and action to address a complex problem. By transforming research tools into a logical workflow, this chapter aims to make energy efficiency more accessible and to promote its integration as a systematic component of the AI development and deployment lifecycle. Having thus unified our contributions under this methodological perspective, we are now ready to conclude this thesis by taking overall stock of our work and outlining the research perspectives it opens up.

Chapter 7

General Conclusion and Perspectives

Ce dernier chapitre synthétise l'intégralité du travail de thèse. Il commence par réaffirmer le contexte et la problématique de l'efficacité énergétique de l'IA face aux limites des architectures matérielles. Il dresse ensuite un bilan complet des contributions, en articulant leur enchaînement logique au sein de la chaîne d'analyse que nous avons développée : depuis la mesure fine avec EA2P, en passant par la prédiction interprétable avec notre framework analytique, jusqu'à l'optimisation multi-objectifs avec notre méthodologie d'ordonnancement. Les résultats clés et leur impact – notamment la démonstration qu'une co-optimisation de la performance et de l'énergie est possible – sont mis en exergue. Après une discussion transparente des difficultés rencontrées et des limites de notre approche, le chapitre se termine en ouvrant sur plusieurs perspectives de recherche prometteuses. Celles-ci incluent l'amélioration de la fidélité de nos modèles par la prise en compte des interférences, l'extension de notre approche vers une co-optimisation globale (système-algorithme), et son adaptation au continuum Cloud-Edge.

This final chapter concludes our thesis, which is dedicated to the study and optimization of the energy efficiency with a specific emphasis Artificial Intelligence applications on modern computing infrastructures.

7.1 Context of the Thesis

The field of High-Performance Computing (HPC) is the engine behind numerous scientific and industrial advances. The emergence and explosive growth of AI models, particularly Deep Learning models, have created an insatiable demand for computational power. These complex models, necessary to analyze the *tsunami* of data generated by our digital world, can only be trained in a reasonable amount of time on increasingly powerful GPU clusters and supercomputers.

For decades, the performance improvement of these platforms was driven by advances in microelectronics, notably the increase in transistor density (Moore's Law) and processor frequency. However, this era of "easy" performance gains is over. As we explained in the state of the art (Chapter 2), physical constraints have halted the escalation of frequencies, and the main barrier

to deploying ever-larger platforms has become the **energy**. The power consumption of data centers, with its associated financial costs and environmental impact, has become a first-order constraint, forcing the industry to fundamentally rethink system design.

Faced with these challenges, a major opportunity has emerged : **architectural specialization**. Rather than using general-purpose processors (CPUs), the industry has massively turned to accelerators like GPUs, and then TPUs, which are optimized for the massively parallel operations at the core of AI. This trend towards hardware heterogeneity, which will only intensify, promises significant efficiency gains.

However, this opportunity is accompanied by increased complexity. Gains in performance and efficiency will no longer come solely from the raw power of the hardware, but from the ability of software systems to intelligently exploit this heterogeneity and specialization. Without a deep understanding of how applications interact with these new architectures, and without the tools to analyze and control these interactions, the potential of these technologies is at risk of being largely underutilized.

7.2 Problematic of the Thesis

The transition to energy-efficient computing for AI is therefore not just a hardware design problem, but a major challenge at the **system and software level**. How can we manage thousands of concurrent AI jobs on heterogeneous clusters to maximize performance while minimizing energy consumption and guaranteeing fairness and quality of service ?

This fundamental question breaks down into a series of sub-problems that we have addressed throughout this thesis : How can we measure energy reliably and with high granularity ? How can we model the complex relationship between a model's architecture, a hardware platform, and its consumption ? How can we design scheduling algorithms that navigate the multi-objective trade-off space ?

Our approach to answering this problem has been to develop a suite of tools and an integrated methodology to enable the energy-aware analysis and optimization of AI applications, focusing on the levers available at the level of the software infrastructure, from scheduling down to application execution.

7.3 Main Contributions of the Thesis

To address this problem statement, this thesis has produced a series of interdependent contributions, structured around the **Measure, Understand, Model, Optimize** cycle.

7.3.1 State of the Art and Scientific Context

First (Chapter 2), we conducted a broad state-of-the-art review to position our work. We analyzed in detail the metrics for performance and energy, the hardware architectures for AI (from GPU to embedded systems), the energy profiles of Deep Learning workloads, and the exis-

ting approaches to modeling and optimization. This study highlighted the persistent challenges and revealed a critical need for :

— flexible measurement tools, offering complete coverage of key components (especially DRAM) and adapted to the Python ecosystem of AI ;

— analytical and interpretable prediction models, capable of capturing the complexity of modern AI workloads beyond simple FLOP counts ;

— natively multi-objective scheduling policies, explicitly co-optimizing energy with traditional objectives.

7.3.2 Development of an Energy Analysis Chain : From Measurement to Optimization

The core of this thesis has been the development of a coherent software and methodological suite to meet the identified needs.

1. **A Solid Empirical Foundation (Chapter 3)** :
   — **EA2P** : We presented our energy profiling tool, EA2P. It is distinguished by its modular approach in Python, its support for heterogeneous hardware (Intel/AMD CPUs, NVIDIA/AMD GPUs), and above all by its integration of an analytical model to estimate DRAM consumption, a major originality compared to existing tools.
   — **Fundamental Characterizations** : Using EA2P and other low-level tools, we conducted in-depth benchmarking campaigns. These campaigns allowed us to finely characterize the energy impact of parallelism paradigms (SIMD, OpenMP, GPU) and power management levers (DVFS, Power Capping) on representative compute kernels. These results constitute a valuable reference database and have informed our modeling work.
2. **Predictive, Analytical, and Interpretable Models (Chapter 4)** :
   — **Model for DRAM** : We formalized and validated our pragmatic analytical model for DRAM energy, which allows EA2P to offer a more complete energy view.
   — **Framework for AI** : We presented our most significant theoretical contribution : a comprehensive analytical framework for predicting the performance and energy of AI model training. Its originality lies in its combination of a highly detailed workload characterization (including “structural features”), a performance model based on hardware saturation, and a rigorous calibration process with targeted scalars.
3. **Multi-Objective Optimization Strategies (Chapter 5)** :
   — **EAS-Sim Simulator** : We developed a high-fidelity cluster simulator, EAS-Sim, using our analytical model as an oracle, to enable the rapid and reproducible evaluation of complex scheduling policies.
   — **Co-Design Methodology** : We proposed a systematic methodology for the co-design of energy-aware schedulers, based on injecting the EDP (Energy-Delay Product) as a scoring primitive.

— **Suite of Schedulers** : The application of this methodology resulted in four new scheduling policies (Zeus, Hades, AuraChronos, Charon), each exploring a different trade-off between energy, performance, fairness, and SLA.

4. **An Integrated Methodology (Chapter 6)** : Finally, we synthesized all these contributions into a guided and pragmatic methodology. This chapter transforms the tools and knowledge into a structured engineering process, showing how to apply them coherently to analyze and optimize the energy efficiency of a real system.

7.3.3 Results Obtained and Their Impact

The developed tools and methodologies have led to significant results, validated by simulation. We have demonstrated that by adopting a scheduling policy like **Zeus**, it is possible to reduce the energy consumption of a heterogeneous AI cluster by ≈10% with no performance loss compared to a state-of-the-art policy like Pollux. We have also proven that to guarantee strict SLAs, preemption (**AuraChronos** policy) is indispensable, reducing the deadline miss rate by a factor of more than 20. Furthermore, our work has highlighted the role of the EDP as a more robust global performance heuristic than time alone, as it promotes better load balancing and greater system stability. The relevance of our modeling tools has been confirmed by their ability to predict with very high accuracy (≈4% error) the performance of very diverse model architectures. All of this work has been validated by the publication of several articles in international conferences, and it is our intention to make the source codes of EA2P and EAS-Sim public for the benefit of the community.

7.3.4 Difficulties and Limitations of our Work

This research encountered several difficulties. The complexity and often proprietary nature of hardware architectures and software stacks made the construction of analytical models particularly challenging. The experimental validation, even in simulation, required orchestrating complex campaigns involving thousands of executions. Our work also has limitations inherent to its scope. Our contributions have been primarily validated on x86 GPU architectures (NVIDIA). Their porting and validation on other types of accelerators (e.g., AMD, Intel GPUs, or future ARM architectures) would require a non-negligible engineering and calibration effort. Moreover, as we have pointed out, our models do not yet integrate certain second-order effects such as inter-job interference.

7.4 Global Conclusion

This thesis has explored the critical domain of the energy efficiency of Artificial Intelligence applications at the system and software level. In response to the growing complexity of infrastructures and workloads, we have argued and demonstrated that a fragmented approach is not sufficient. We have proposed a complete and integrated workflow, from precise measurement with the EA2P tool, to interpretable modeling with our analytical framework, and finally to

intelligent optimization with the EAS-Sim simulator and the multi-objective scheduling policies we have designed.

This work has demonstrated that performance and energy efficiency are not necessarily opposing objectives but can be co-optimized through well-chosen metrics and heuristics like the EDP. It has provided not only concrete technical solutions but also a systematic methodology for approaching the problem of sustainable AI as a rigorous engineering process.

7.5 Perspectives

This thesis work constitutes, we hope, a solid foundation for future research. The next steps could explore several exciting directions :

— **Improving Model Fidelity :** Refining our analytical framework to include models of inter-job interference and developing online calibration techniques that would allow the oracle to self-adapt to system changes.

— **Towards Global Co-Optimization :** Extending our methodology to break down the silos between optimization layers. This could involve designing a scheduler that dynamically interacts not only with OS levers (DVFS) but also with application hyperparameters (batch size) and model optimization techniques (quantization).

— **Application to the Edge-HPC-Cloud Continuum :** Adapting our tools and concepts to model and optimize applications distributed across the entire digital continuum, integrating the constraints of network latency, battery life, and thermals of edge devices.

— **AI for Green AI :** Using AI techniques, such as the Reinforcement Learning (RL) we initially considered, to drive our scheduling policies. An RL agent could learn even more complex and adaptive policies than our current heuristics, by exploring the immense space of possible decisions.

The success of the project of a powerful and beneficial Artificial Intelligence for our society will depend on our ability to make it sustainable. The work presented in this thesis is a stone to this edifice, with the conviction that rigorous research at the computing process level is one of the essential keys to achieving this goal.

List of publications

Published Papers

Roblex Nana Tchakoute, Claude Tadonki. "*EAS-Sim : A Framework and its Methodology for the Co-Design of Multi-Objective, Energy-Aware Schedulers for AI Clusters.s*". In Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA. ACM, New York, NY, USA, 10 pages.DOI: 10.1145/3731599.3767568

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Youssef Mesri. "*A Framework for Analytical Performance and Energy Prediction of DL Training on GPUs*". 37th IEEE/SBC International Symposium on Computer Architecture and High Performance Computing, 28-31 October, 2025 - Bonito, MS, Brazil.

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Youssef Mesri. "*Benchmark-based Study of CPU/GPU Power-Related Features through JAX and TensorFlow*". IEEE Access.DOI: https://arxiv.org/abs/2505.03398

Roblex Nana Tchakoute, Claude Tadonki. "*Energy-Aware Deep Learning on GPUs through Parameter Sharing and Mixed Precision Training*". LeanDL-HPC 2025 : Workshop on Lightweight and Efficient Deep Learning in HPC Environments, 28-31 October, 2025 - Bonito, MS, Brazil.

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Youssef Mesri. "*A Flexible Operational Framework for Energy Profiling of Programs*". 2024 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Nov 2024, Hilo, United States. pp.12-22.DOI: 0.1109/SBAC-PADW64858.2024.00014

Roblex Nana Tchakoute, Claude Tadonki. "*Experimental Study of Power Consumption of Basic Parallel Programs*". 2024 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Nov 2024, Hilo, United States. pp.33-41.DOI: 10.1109/SBAC-PADW64858.2024.00016

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Youssef Mesri. "*Power Consumption in HPC-AI Systems*". Artificial Intelligence and High Performance Computing in the Cloud. CloudTech 2023. Lecture Notes in Networks and Systems, vol 1220. Springer, Cham.DOI: 10.1007/978-3-031-78698-3_6

Under Reviews Papers

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Youssef Mesri. "*Survey on Energy-Aware Computing*". Submited to : IEEE Transactions on Sustainable Computing.DOI: https://arxiv.org/abs/2309.08615

List of Developed Open Source Tools

A core tenet of this doctoral research was to produce not only theoretical insights but also a suite of practical, robust, and reusable software artifacts. The following open-source tools were designed, developed, and utilized throughout this work. They represent a significant part of the thesis's contribution, providing the community with the necessary instruments to reproduce, validate, and extend the methodologies presented herein.

EA2P : Energy-Aware Application Profiler A versatile software tool designed for the fine-grained measurement and profiling of energy consumption in modern HPC applications, with a specific focus on the Python ecosystem prevalent in AI.

- **Key Features :** Multi-platform and multi-component support (Intel/AMD CPUs, NVIDIA/AMD GPUs) ; modular and extensible Python architecture ; Command-Line (CLI) and Application Programming Interface (API) modes for both global and code-specific profiling ; integrated support for multi-node MPI applications.
- **Core Innovation :** Integration of a validated analytical model to estimate the energy consumption of DRAM on platforms where direct hardware measurement is unavailable, addressing a significant gap in existing tools.
- **Role in Thesis :** Serves as the foundational empirical tool for the "Measure" and "Understand" pillars of our methodology. It was detailed in Chapter 3 and used for all experimental characterizations.
- **Availability :** The tool is publicly available on PyPI for easy installation ('pip install ea2p') and its source code is hosted on GitHub.
  https://github.com/HPC-CRI/EA2P

ADEPT : Analytical Deep-learning Energy and Performance Time-estimator A white-box, hardware-aware analytical framework for the high-fidelity prediction of performance (execution time) and energy consumption (power) for the training of Deep Learning models.

- **Key Features :** Fully analytical approach ensuring interpretability ; hierarchical workload characterization of PyTorch models ; a parametric performance model that accounts for non-linear hardware saturation effects.
- **Core Innovation :** Introduces the concept of "structural efficiency features" to quantify how a model's architecture leverages hardware parallelism, going beyond simple

FLOP counting. Uses a robust Differential Evolution algorithm for calibration against empirical data.

— **Role in Thesis :** Represents the main contribution of the "Model" pillar. It serves as the predictive oracle within the EAS-Sim simulator. It was detailed in Chapter 4.

— **Availability :** The framework's source code will be made available in a dedicated public repository.
https://github.com/HPC-CRI/ADEPT

EAS-Sim : Energy-Aware Scheduling Simulator A high-fidelity, extensible discrete-event simulation framework specifically designed for the rapid prototyping, evaluation, and co-design of multi-objective job scheduling policies for AI clusters.

— **Key Features :** Built on Python and the SimPy framework ; models critical cluster dynamics including hardware heterogeneity, job malleability, and preemption ; designed for fast, reproducible, and quantitative comparison of complex scheduling strategies.

— **Core Innovation :** Uses the ADEPT analytical framework as its predictive oracle, enabling fast *a priori* evaluation of scheduling decisions without requiring iterative "virtual execution", thus dramatically accelerating the simulation process.

— **Role in Thesis :** Forms the cornerstone of the "Optimize" pillar. It is the primary instrument used in Chapter 5 to validate our suite of energy-aware schedulers (Zeus, Hades, AuraChronos, Charon).

— **Availability :** The simulator will be released as an open-source tool to foster further research in the community.
https://github.com/HPC-CRI/EAS-Sim

Appendix : Foundations of Modern Computer Architecture for Energy-Aware Computing

This appendix provides supplementary technical information that serves as a foundation for the problems addressed and the solutions developed in this thesis. While the main chapters focus on our direct contributions, this section aims to offer a deeper dive into the underlying principles and contexts that motivate our work. We will explore the physical laws governing power consumption, the historical evolution of hardware, and the architectural divergence that positioned the GPU as the engine of the AI revolution.

A.1 The Physical and Historical Context of Modern Computing

To understand the current energy challenges in high-performance computing, we must begin at the atomic level and trace the path of technological evolution that has defined the last half-century. The architectural decisions that led to the development of powerful GPUs are not arbitrary ; they are the direct consequences of fundamental physical laws and the predictable breakdown of long-standing scaling trends. This section establishes that foundational context.

A.1.1 From Silicon to Switch : The Physics of the Transistor

At its core, all digital computation is the art of controlling the flow of electrons through a material. The material of choice for the modern era is **silicon**, a metalloid that in its pure crystalline form is a poor conductor of electricity. Its power is unlocked through *"doping"*, a process where impurity atoms are introduced into the silicon lattice to create N-type (excess electrons) and P-type (excess "holes") semiconductors.

Transistors come in several types, as shown in Figure 1, including Bipolar Junction Transistors (BJT) with PNP and NPN variations, and Field-Effect Transistors (FET) such as JFET and MOSFET. Among these, the **MOSFET** (Metal-Oxide-Semiconductor Field-Effect Transistor) is the cornerstone of modern digital electronics.

The MOSFET acts as a near-perfect, voltage-controlled electronic switch. As shown in Figure 2, a voltage applied to the Gate terminal creates an electric field that allows current to flow between the Source and Drain, turning the transistor *"on"*. Removing the voltage collapses the

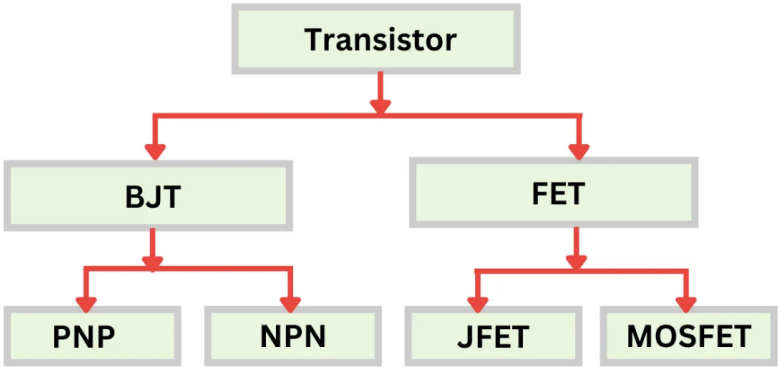


FIGURE 1 – Classification of transistors types [ElectronicsLessons, 2025]

path, turning it *"off"*. This binary behavior is the physical embodiment of a bit and can occur billions of times per second.

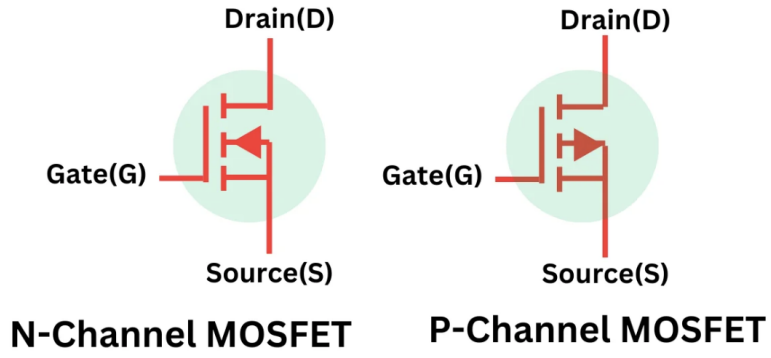


FIGURE 2 – N-Channel and P-Channel MOSFET structures [ElectronicsLessons, 2025]

A.1.2 Building Integrated Circuits : The Hierarchy of Computation

A single transistor is a switch, but computation requires logic. The journey from a single transistor to a complex processor like a GPU is a marvel of hierarchical design.

Level 1 : Logic Gates. By combining a few transistors, we form **logic gates** (the elementary building blocks that perform basic Boolean operations like AND, OR, and NOT). For example, a NAND gate can be constructed from just four MOSFETs.

Level 2 : Functional Units. By connecting logic gates, we build more complex circuits that perform specific functions. An **adder**, for instance, can be built to sum two binary numbers. A **latch** or **flip-flop** is a circuit of cross-coupled gates capable of storing a single bit of information, forming the basis of fast static memory (SRAM).

Level 3 : Integrated Circuits (ICs). These functional units are then assembled into complete subsystems on a single piece of silicon. A processor's core, a memory controller, or a GPU's

Streaming Multiprocessor are all examples of such subsystems. Finally, these subsystems are interconnected on a monolithic die to form a complete *Integrated Circuit* or "chip", such as a full CPU or GPU.

The number of transistors required grows exponentially with the complexity of the device. Table 1 provides a sense of this staggering scale, from simple microcontrollers to the massive GPUs at the heart of modern AI, which now contain nearly one hundred billion transistors.

TABLE 1 – Transistor counts in representative modern electronic devices.

Device Category	Example	Transistor Count (Approx.)
Simple IC	NE555 Timer (classic)	25
Microcontroller	Arduino's ATmega328P	32,000
Embedded SoC	Raspberry Pi 4 (BCM2711)	1.5 Billion
Consumer CPU	Apple M2 Ultra	134 Billion
Server CPU	**AMD EPYC Genoa (96-core)**	**90 Billion**
High-End FPGA	Xilinx Versal Premium VP1902	9 Trillion (est. effective)
AI ASIC	Google TPU v4	20-30 Billion (est.)
High-End AI GPU	**NVIDIA H100**	**80 Billion**

A.1.3 The Twin Engines of Progress : Moore's Law and Dennard Scaling

For over three decades, the semiconductor industry advanced at an astonishing pace, driven by two complementary trends visualized in Figure 3. After 2005, GPUs became the primary beneficiaries of this continued densification, enabling massive parallelism while CPU core counts grew more modestly.

— **Moore's Law (1965) :** The famous observation that the number of transistors on a chip would double approximately every two years for the same cost [Schaller, 1997], leading to exponential increases in computational density.

— **Dennard Scaling (1974) :** The theory that as transistors shrink, their power density remains constant [Dennard, 1974]. This meant that each new generation of chips was significantly more powerful than the last, without becoming prohibitively hot.

A.1.4 Hitting the "Power Wall" : The End of the Free Lunch

Around the mid-2000s, this virtuous cycle broke down. As transistors shrank, quantum-level leakage currents grew exponentially, causing static power consumption to skyrocket. It was no longer possible to reduce the operating voltage in lockstep with size, effectively ending Dennard scaling.

This pivotal event created the **"Power Wall"**. With voltage reduction no longer a viable option, increasing clock frequency led to an unsustainable increase in heat. As illustrated in Figure 4, the industry was forced to abandon the decades-long race for higher frequencies. Around 2005, clock speeds and single-thread performance plateaued due to power constraints.

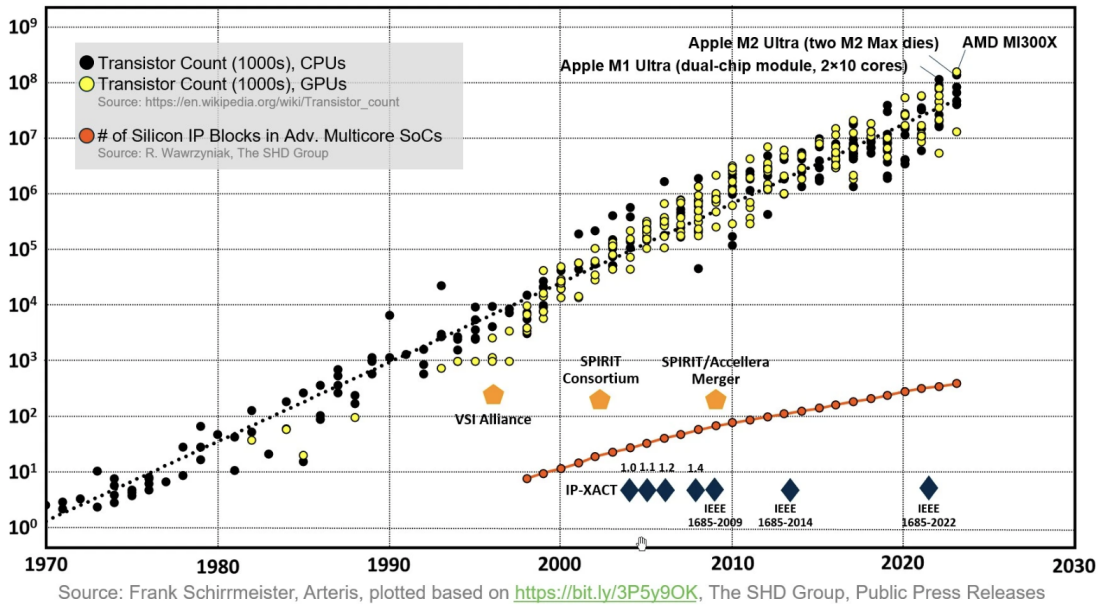


FIGURE 3 – The growth of transistor counts predicted by Moore's Law [Schirrmeister, 2023].

To continue leveraging Moore's Law (continuously increasing transistor counts), manufacturers turned to increasing the number of processing cores.

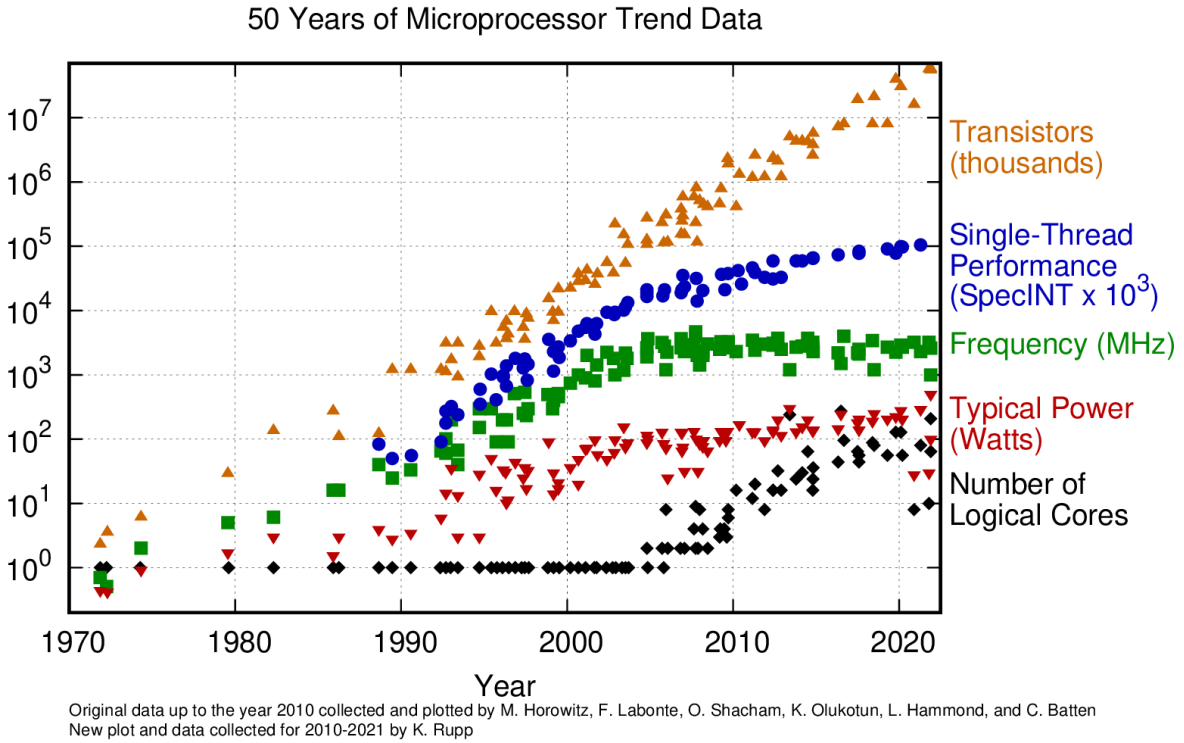


FIGURE 4 – A visualization of the "Power Wall."

A.1.5 The Consequence : The Great Divergence to Parallelism

The Power Wall forced a radical rethink in processor design. With Moore's Law still providing a doubling of transistors every two years, the question became : how to use this bounty of transistors to increase performance if we can no longer make them individually faster ? The only viable answer *was parallelism.*

This led to the *great architectural divergence.* CPUs adopted multi-core designs, but their complex cores meant that core counts grew relatively modestly. The true beneficiary of Moore's Law in the post-Dennard era became the *GPU.* Its highly regular, massively parallel architecture was perfectly suited to being scaled up with more transistors. While CPU core counts went from 2 to 64 over a decade, GPU "core" counts exploded from hundreds to over ten thousand. This made the GPU the undisputed king of throughput-oriented computing and the natural engine for the Deep Learning revolution, whose workloads consist almost entirely of the massively parallel linear algebra operations that GPUs excel at.

A.1.6 The Physics of Power Consumption in CMOS Technology

To understand the challenges of managing these powerful parallel processors, we must return to the physics of power. The total power consumed by a modern CMOS chip is the sum of its dynamic and static components.

Dynamic Power, consumed during transistor switching, is modeled as :

$$P_{\text{dynamic}} = A \cdot C \cdot V_{dd}^2 \cdot f \tag{1}$$

Here, A is the software-dependent activity factor, C is the physical capacitance, V_{dd} is the supply voltage, and f is the clock frequency. The quadratic dependence on voltage (V_{dd}^2) is the key reason why DVFS is such an effective energy-saving technique.

Static Power, or leakage power, is consumed even when idle and is given by :

$$P_{\text{static}} = V_{dd} \cdot I_{\text{leakage}} \tag{2}$$

As explained, the inability to control the explosive growth of I_{leakage} in smaller transistors is what brought about the end of Dennard scaling. Today, static power can account for a substantial portion of a chip's total power budget, making efficient idle states and power gating critical for energy efficiency.

This journey from a single doped silicon atom to a billion-transistor GPU facing a fundamental power wall sets the stage for the rest of this thesis. It establishes *why* performance is now intrinsically linked to parallelism, *why* specialized accelerators like GPUs dominate AI, and *why* managing the energy they consume is no longer an afterthought, but a central challenge of modern computing.

A.2 CPU vs. GPU : An Architectural Divergence for Parallel Computing

The Power Wall described in the previous section did not mark the end of performance growth, but rather its **divergence**. With single-thread frequency scaling exhausted, the path to greater computational power forked. One path led to the modern multi-core CPU, while the other led to the massively parallel GPU. Understanding the profound architectural differences between these two types of processors is fundamental to understanding the landscape of modern AI and the energy efficiency challenges that this thesis addresses. They are not merely different in degree, but different in kind, each optimized for a completely distinct computational philosophy.

A.2.1 Core Design Philosophy : The Battle of Latency vs. Throughput

The most fundamental distinction between a CPU and a GPU lies in what each is designed to optimize.

The CPU is a Latency-Optimized Device. A CPU is architected to execute a single thread of instructions as quickly as possible. Its cores are large, powerful, and incredibly complex, often described as a few "lions". They are packed with sophisticated hardware (deep instruction pipelines, aggressive branch predictors, out-of-order execution engines, and large caches), all aimed at minimizing the execution time (*latency*) of a single, often complex and serial, task. A CPU is a master of complex control flow and is indispensable for running operating systems and managing the sequential logic of an application.

The GPU is a Throughput-Optimized Device. A GPU, in contrast, is architected to execute a massive number of parallel threads simultaneously. Its cores are simple, small, and numerous, better described as a "swarm of bees". The goal is not to minimize the latency of any single task, but to maximize the total number of tasks completed per unit of time (*throughput*). The GPU achieves this by employing a strategy of latency hiding : while one group of threads waits for data to arrive from its slow path to memory, the GPU's hardware scheduler instantly and with zero overhead switches to another group of ready-to-execute threads, ensuring its arithmetic units are almost always busy.

A.2.2 Microarchitectural Differences in Detail

The divergent philosophies manifest in starkly different allocations of the transistor budget, as shown in Figure 5. The CPU (left) devotes a significant portion of its transistor budget to large caches and complex control logic to speed up a few threads. The GPU (right) devotes the vast majority of its silicon to a massive number of simpler arithmetic logic units (ALUs) to maximize parallel throughput.

FIGURE 5 – Microarchitectural comparison of CPU and GPU [Wikimedia, 2024].

A.2.2.1 Control Logic and Cache Hierarchy

A modern CPU core is a marvel of control logic. A huge portion of its transistor budget is dedicated to out-of-order execution engines, speculative execution, and sophisticated branch predictors (all designed to find and exploit instruction-level parallelism in a single serial thread). It is backed by a deep, multi-level cache hierarchy that can occupy more than half the entire die area. As seen in Table 2, the goal of this massive cache is to hide the high latency of main memory (DRAM) by keeping the working set of one or a few threads as close to the core as possible. The CPU prioritizes very large last-level caches to reduce main memory latency for a few cores, while the GPU prioritizes extremely fast on-chip memory (L1/Shared) for thousands of threads.

A GPU's design makes the opposite trade-off. It has much simpler control logic because it assumes thousands of threads will be executing the same instruction path. It dedicates far fewer transistors to caches, relying instead on its massive thread count to hide memory latency. Its cache is designed less to serve a single thread and more to facilitate data sharing between threads within a compute block and to **coalesce** memory accesses from multiple threads into a single, efficient transaction to DRAM.

TABLE 2 – A comparison of cache hierarchies in high-end server CPUs and AI GPUs.

Cache Level	Typical High-End CPU (e.g., AMD EPYC)	Typical AI GPU (e.g., NVIDIA H100)
L1 Cache	32KB I-Cache + 32KB D-Cache (Private per core)	128 KB (Configurable as L1/Shared Memory per SM)
L2 Cache	1MB (Private per core)	50MB (Shared across the chip)
L3 Cache	256MB+ (Shared across cores)	Not Applicable
Key Focus	Minimize latency for serial threads	Maximize bandwidth and data sharing for parallel threads

A.2.2.2 Execution Units : Lions vs. Bees

The difference is most visible at the core level, as depicted in Figure 5. A single CPU core (left) is a complex unit with large control logic and deep caches, optimized for a single instruc-

tion stream. A single GPU Streaming Multiprocessor (SM, right) is a throughput-oriented unit containing many simpler ALUs (CUDA Cores) and specialized hardware (Tensor Cores), managed by simpler schedulers. A CPU has a few powerful cores ("lions") containing wide vector (SIMD) units capable of executing instructions like AVX-512, which can perform a single operation on 16 floating-point numbers at once. In contrast, a GPU contains dozens of **Streaming Multiprocessors (SMs)**. Each SM is like a miniature processor in itself, containing a swarm of simpler cores ("bees"). A single NVIDIA Hopper H100 GPU, for example, has 132 SMs, each with its own schedulers, register file, and execution units, including dedicated Tensor Cores for AI. This massive replication of simpler compute units is what provides the GPU's enormous raw computational power for parallel tasks.

A.2.3 Programming Model and Execution Paradigm

These architectural differences necessitate completely different programming models.

CPU Parallelism : Task Parallelism. CPU programming is typically serial by default. Parallelism must be explicitly managed by the programmer through multi-threading, often using frameworks like OpenMP. Here, the programmer typically divides a large loop into chunks and assigns each chunk to a thread, a model known as task parallelism.

GPU Parallelism : Data Parallelism and SIMT. GPU programming is fundamentally data-parallel. Frameworks like CUDA (NVIDIA) and ROCm (AMD) expose a hierarchical model illustrated in Figure 6. The programmer writes a small program called a **"kernel"**, which defines the operation for a single thread. They then launch this kernel across a massive *"grid"* of threads. The grid is organized into *"blocks"*, and threads within a block can cooperate and synchronize using fast shared memory. The hardware executes threads in groups of 32, called a **"warp"**. This is the **SIMT** *(Single Instruction, Multiple Threads)* model : all 32 threads in a warp execute the same instruction at the same time, but on their own private data. This is an incredibly efficient execution model for problems like vector addition or matrix multiplication.

A.2.4 Respective Roles in a Modern AI Workload

In a modern AI training or inference system, the CPU and GPU play symbiotic but distinct roles. The CPU Generally acts as the "host", managing the Python interpreter, loading data, and dispatching tasks, but not participate in computing. The GPU acts as the "device", executing the massively parallel compute operations (forward/backward passes and weight updates) that constitute the vast majority of the workload's computational cost. However, modern techniques like ZeRO-Offload [Ren, 2021] tend to use CPU for weight updates and small computation, as shown in Figure 7.

The CPU acts as the orchestrator or *"host"*. It runs the main application logic, the operating system, and the AI framework's Python interpreter. It is responsible for all the serial tasks : data loading and preprocessing from storage, batch preparation, and dispatching computational kernels to the GPU.

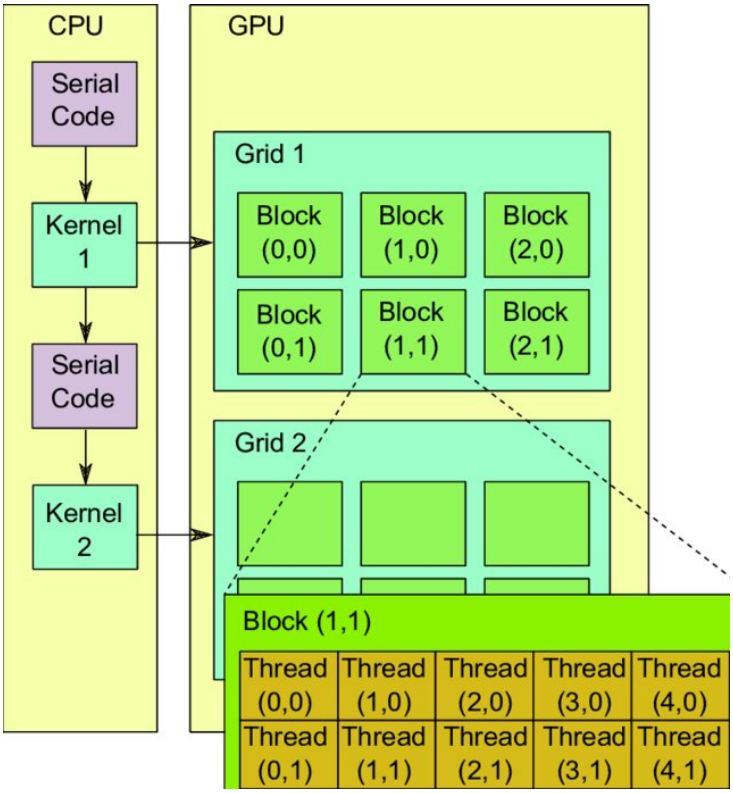


FIGURE 6 – The CUDA execution and memory hierarchy [Bartezzaghi, 2015].

The GPU acts as the massively parallel workhorse or *"device"*. It receives these prepared batches and kernels and executes the computationally intensive parts of the AI model. Its architecture is perfectly matched to the dominant operations in Deep Learning : large matrix multiplications and convolutions. The training of a large language model may involve quintillions of floating-point operations, a task for which the GPU's throughput-oriented architecture is uniquely and indispensable suited.

A.2.5 Comparative Power, Energy, and Performance

The architectural divergence leads to starkly different profiles in performance and power, which are central to this thesis and are summarized in Table 3. The GPU's massive advantage in memory bandwidth and specialized compute (BF16/FP16) translates into orders-of-magnitude higher energy efficiency (FLOPS/Watt) for AI workloads.

While the GPU's TDP is higher, its raw computational throughput for the precisions used in AI (like BF16/FP16) is astronomically higher. This, combined with its vastly superior memory bandwidth, results in a far better **energy efficiency** (FLOPS/Watt) for the tasks it is designed for. This is the fundamental reason why GPUs are indispensable for training large models in a reasonable timeframe and with a manageable energy budget. The choice to model and optimize a GPU-centric environment, as pursued in this thesis, is therefore a direct consequence of this profound and enduring architectural divergence.

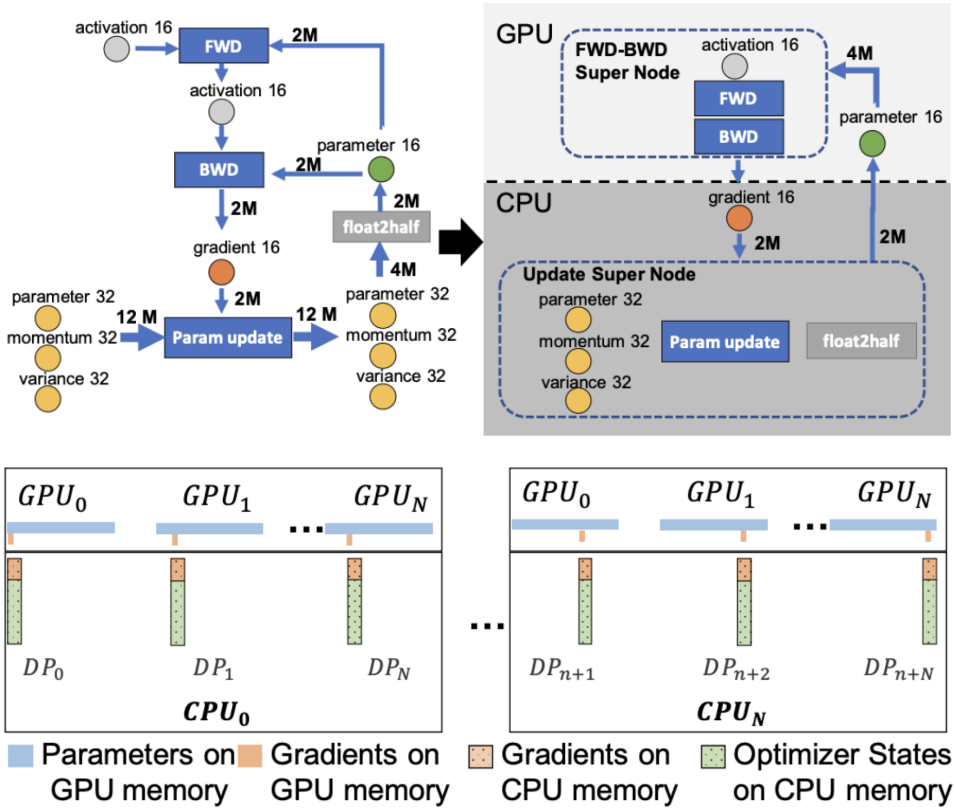


FIGURE 7 – The division of labor in a typical AI training step [Ren, 2021].

TABLE 3 – A comparison of specifications for a high-end server CPU vs. AI GPU.

Metric	High-End Server CPU	High-End AI GPU
Core Count	96 Complex Cores	132 SMs (≈16,896 CUDA Cores)
TDP	≈360 W	≈700 W
Idle Power	≈80-100 W	≈70-90 W
Memory Type	DDR5	HBM3
Memory Bandwidth	≈460 GB/s	≈3,350 GB/s (**7.3**×)
FP64 Performance	≈5 TFLOPS	≈34 TFLOPS (**6.8**×)
BF16/FP16 Performance	≈80 TFLOPS	≈989 TFLOPS (with sparsity, **12.4**×)
FP16 FLOPS/Watt	≈222 GFLOPS/Watt	≈1412 GFLOPS/Watt (**6.4**×)

A.3 A Deep Dive into Modern GPU Microarchitecture

While Appendix B established the high-level philosophical differences between CPUs and GPUs, a deeper understanding of energy consumption and performance in AI workloads requires dissecting the GPU's internal architecture. A modern AI GPU is not a monolithic processor ; it is a highly scalable, hierarchical system designed for one purpose : to execute a staggering number of parallel arithmetic operations. This section will peel back the layers of a modern GPU, from its top-level organization down to the specialized silicon at its core, revealing the hardware features that have made it the premier engine for the AI revolution.

A.3.1 The Scalable, Hierarchical Architecture of a GPU

To achieve performance across a wide range of products and price points, from consumer graphics cards to data center accelerators, GPU architects employ a deeply modular and scalable design. The full chip is composed of several high-level clusters (e.g., GPCs), which in turn contain multiple SMs. This modular and scalable design allows manufacturers to create a wide range of products from a single underlying architecture.

At the highest level, the GPU is partitioned into several **Graphics Processing Clusters (GPCs)** or similar top-level blocks. These are largely independent processing engines. Within each GPC, there are multiple **Streaming Multiprocessors (SMs)** (known as **Compute Units (CUs)** in AMD's terminology). The SM is the fundamental building block of the GPU's computational power. An overview of a GPC which is the dominant high-level hardware block within all Nvidia GB20x Blackwell family GPUs is shown in Figure 8. The full chip is built from a hierarchy of replicated processing blocks.

This modularity is key. A manufacturer can create a family of GPUs from a single chip design simply by enabling a different number of SMs. The performance and power consumption of a GPU are, to a first approximation, directly proportional to the number of active SMs. The rest of this section will focus on the internal structure of the SM, as this is where the computation for AI workloads actually occurs.

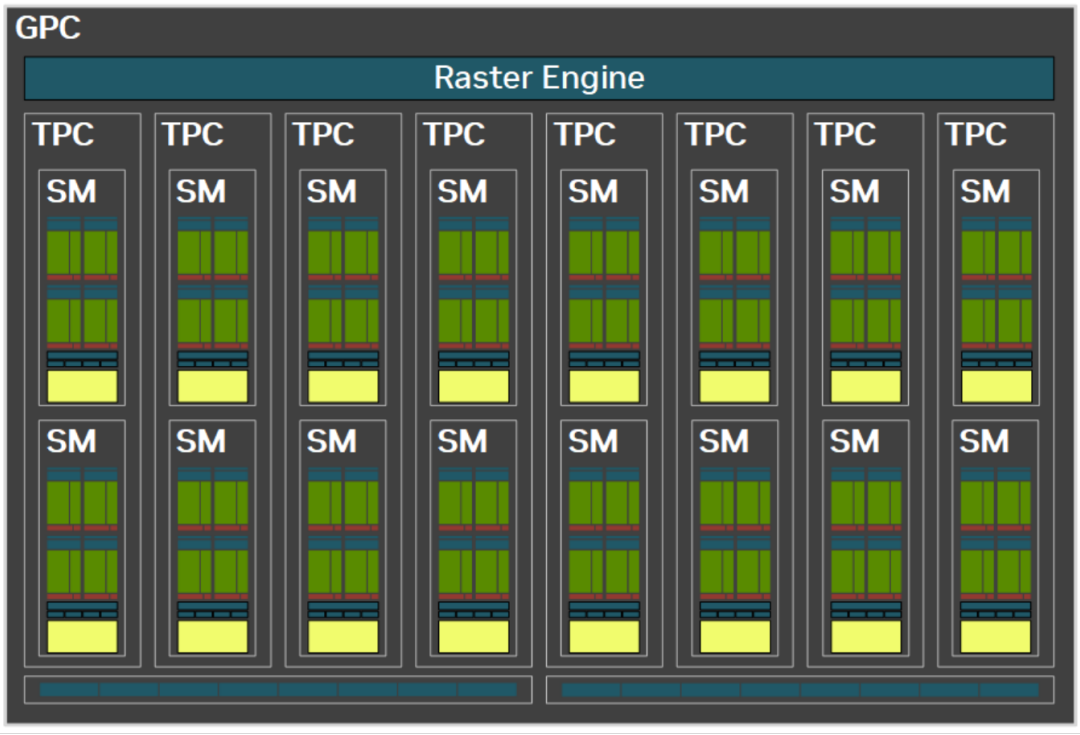


FIGURE 8 – The conceptual hierarchical structure of a GPC block in a modern Nvidia GPU [Nvidia, 2025b].

A.3.2 Inside the Streaming Multiprocessor (SM) : The Heart of the GPU

The Streaming Multiprocessor is a highly sophisticated, self-contained processor optimized for throughput. It contains all the necessary components to fetch, schedule, and execute hundreds of threads in parallel. Understanding its internal components is crucial to understanding the performance predictions made by the analytical framework in Chapter 4.

A.3.2.1 Execution Units : From General-Purpose to Hyper-Specialized

Each SM contains a diverse array of execution units, each tailored for a specific type of operation.

CUDA Cores (ALUs). The most numerous execution units are the general-purpose floating-point and integer cores, known as CUDA Cores in NVIDIA terminology. These are the workhorses for standard arithmetic operations (addition, multiplication, etc.) that do not fit into a more specialized category.

Specialized Units : The Engine of the AI Revolution. The most significant architectural evolution in GPUs over the last decade has been the introduction of hardware units explicitly designed to accelerate the core computations of Deep Learning. The most important of these is the **"Tensor Core"**. First introduced in the Volta architecture, a Tensor Core is not a "core" in the traditional sense, but a highly specialized circuit designed to perform one operation with extreme efficiency : the **Matrix Multiply-Accumulate (MMA)**. As shown in Figure 9, a single Tensor Core can, in a single clock cycle, multiply two small matrices (e.g., 4x4) and add the resulting matrix to a third accumulator matrix. It performs a full matrix multiplication and addition in a single step, often using mixed precision (e.g., FP16 inputs, FP32 accumulation) to drastically boost throughput and energy efficiency.

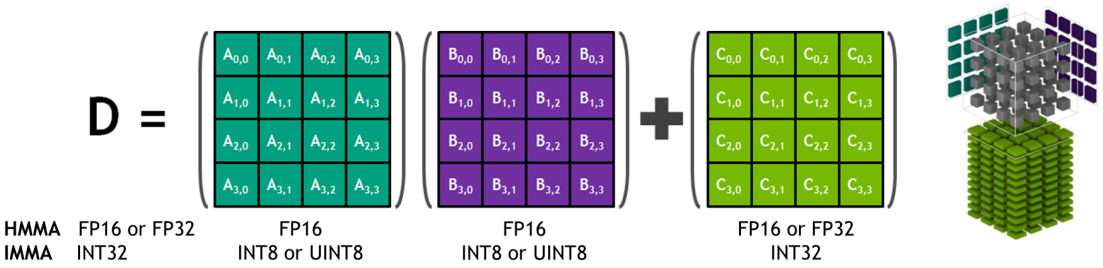


FIGURE 9 – The conceptual operation of a Tensor Core executing a Matrix Multiply-Accumulate (MMA) operation ($D = A \times B + C$). Adapted from Nvidia developer's blog [Appleyard, 2017].

This specialization is a game-changer for AI. Because matrix multiplication forms the basis of nearly all operations in modern neural networks (convolutions, fully connected layers, attention mechanisms), having dedicated hardware provides a colossal performance boost. Crucially, Tensor Cores operate most efficiently using **mixed precision**. They perform the multiplications

using lower-precision formats like FP16 (16-bit float), BF16 (bfloat16), or even INT8 (8-bit integer), but perform the accumulation in a higher-precision format like FP32 to maintain numerical stability. This has a threefold benefit :

1. **Higher Throughput :** Lower-precision operations are much faster and require fewer transistors.
2. **Reduced Memory Footprint :** Storing a model's weights in FP16 halves the memory required compared to standard FP32.
3. **Lower Energy Consumption :** Moving and multiplying fewer bits consumes significantly less power.

A.3.2.2 Warp Schedulers and Latency Hiding

Within each SM, one or more **warp schedulers** manage the execution of threads. As mentioned, the GPU executes threads in groups of 32 called **"warps"**. The warp scheduler's job is to select a warp whose threads are ready to execute and dispatch its next instruction to the appropriate execution units (CUDA Cores, Tensor Cores, etc.). This mechanism is the key to the GPU's philosophy of *"latency hiding"*. If a warp executes an instruction that requires a long-latency data fetch from main memory (DRAM), it does not stall the entire SM. The scheduler simply marks that warp as "waiting" and immediately picks another warp from its pool of resident warps and begins executing its instructions. Since a modern SM can manage dozens of warps concurrently, it can almost always find a ready warp to execute, thus keeping the expensive execution units busy and maximizing computational throughput.

A.3.2.3 On-Chip Memory : Register File and Shared Memory

To support this massive concurrency, the SM is equipped with a unique on-chip memory hierarchy :

— **Massive Register File :** Each SM has a very large register file (e.g., 256 KB). This is necessary to hold the private state (variables, pointers) of the thousands of threads that can be simultaneously resident on the SM.

— **Unified L1 Cache / Shared Memory :** The SM contains a small, fast on-chip memory that can be flexibly configured. A part of it can be used as a traditional hardware-managed *"L1 cache"* to improve the performance of generic memory accesses. The other part can be configured as a programmer-managed *"shared memory"* (or scratchpad). This shared memory is a powerful feature that allows threads within the same thread block to cooperate, share data, and synchronize with extremely low latency, avoiding slow round-trips to DRAM. Writing algorithms that effectively leverage shared memory is a cornerstone of high-performance GPU programming.

A.3.3 The Architectural Evolution of GPUs for AI

The modern AI GPU is the result of over a decade of evolution, with each generation introducing features that have made it progressively more powerful and efficient for Deep Learning.

Table 4 summarizes this trajectory, focusing on NVIDIA's data center architectures as a proxy for the industry's major trends. Each generation has introduced features that dramatically increased both raw performance and energy efficiency for AI workloads.

TABLE 4 – The architectural evolution of data center GPUs.

Architecture	Year	Process	Key Innovations for AI	Impact on AI & Energy Efficiency
Fermi / Kepler	2010 / 2012	40nm / 28nm	Unified shader core architecture, ECC memory, initial GPGPU focus.	Established the GPU as a viable platform for scientific computing and early neural network research.
Pascal	2016	16nm	Introduction of FP16 compute, NVLink interconnect.	Halved memory requirements for models and dramatically sped up multi-GPU training by providing a high-speed path between GPUs.
Volta	2017	12nm	**Introduction of the Tensor Core**, second-generation NVLink.	**A watershed moment.** Provided a >10x speedup for mixed-precision matrix operations, making the training of large, complex DL models practical for the first time.
Turing / Ampere	2018 / 2020	12nm / 7nm	Enhanced Tensor Cores with support for more data types (INT8, TF32), structural sparsity.	Broadened the applicability of hardware acceleration to inference (INT8) and scientific computing (TF32), and introduced techniques to double throughput by exploiting zero-values in models.
Hopper	2022	4nm	4th-Gen Tensor Cores, **Transformer Engine** (FP8 support), DPX instructions.	Hyper-specialization for Transformer models (the basis of LLMs) via hardware-managed data type switching (FP8/FP16), further boosting efficiency.

This clear evolutionary path (from a general parallel processor to a hyper-specialized AI engine) demonstrates a powerful trend of **hardware-software co-design**. The needs of increasingly complex AI models have directly driven the design of new hardware features in GPUs, most notably the Tensor Core. Our analytical framework in Chapter 4 is built to explicitly capture the performance characteristics of these specialized units, which is essential for accurately modeling the performance of modern AI workloads.

A.4 The GPU Memory Hierarchy and Data Movement

The immense computational throughput of a modern GPU, as detailed in the previous section, presents an equally immense challenge : keeping its thousands of cores fed with data. The performance of most real-world AI applications is not limited by the raw arithmetic capability of the GPU, but by the rate at which data can be moved between the main GPU memory and the SMs. This chasm between compute speed and memory speed is a manifestation of the

"Memory Wall", and overcoming it is the primary goal of the GPU's sophisticated memory architecture. Understanding this hierarchy is not merely an academic exercise ; it is fundamental to writing high-performance code and to accurately modeling the energy consumption of AI workloads, as every byte moved consumes energy.

A.4.1 The Memory Wall in a GPU Context

The Memory Wall is a more acute problem for GPUs than for CPUs for a simple reason : scale. While a multi-core CPU has dozens of cores demanding data, an AI GPU has tens of thousands of threads operating concurrently. This creates an enormous demand for memory bandwidth. Consider a modern NVIDIA H100 GPU capable of performing nearly 1,000 trillion 16-bit operations per second (1 PetaFLOP/s). To sustain this rate, even with significant data reuse, it requires terabytes of data to be streamed to its cores every second.

Failing to provide this data results in **"memory starvation"**, where the powerful execution units within the SMs sit idle, waiting for operands. An idle execution unit still consumes static (leakage) power, so memory-bound applications are often highly inefficient. An application that achieves only 20% of the GPU's peak compute performance is likely spending the vast majority of its time and energy simply waiting on data, a clear sign of poor energy efficiency.

A.4.2 The GPU Memory Hierarchy : A Multi-Level Approach

To combat the Memory Wall, the GPU employs a deep, hierarchical memory system, illustrated in Figure 10. Each level of the hierarchy represents a different trade-off between size, latency, and bandwidth. The goal for a programmer (and for an optimizing compiler) is to keep data in the fastest, smallest, and most energy-efficient level of the hierarchy for as long as possible. Each level offers a different trade-off. Fast, low-latency on-chip memories (Registers, Shared Memory) are local to an SM, while larger, slower off-chip memories (L2 Cache, HBM) are shared across the entire device. Efficient GPU programming revolves around minimizing traffic to the slow off-chip HBM.

A.4.2.1 On-Chip Memory (Within the SM)

These are the fastest and most energy-efficient memories, with latencies on the order of a few clock cycles. Data stored here is physically located within the Streaming Multiprocessor.

Registers : The fastest memory type. Each thread has exclusive access to a set of private registers to store its local variables. The GPU contains a massive register file (e.g., 256KB per SM) to accommodate the state of thousands of concurrent threads.

Shared Memory / L1 Cache : As introduced in Appendix C, each SM contains a fast on-chip SRAM that is software-partitioned. The **L1 Cache** is a hardware-managed cache that stores a copy of data from the off-chip main memory. It automatically attempts to capture temporal and spatial locality for general memory accesses. The **Shared Memory** is a programmer-managed scratchpad. This is a powerful feature for performance optimization. Threads within the same thread block can use shared memory to explicitly load a chunk of data from main memory,

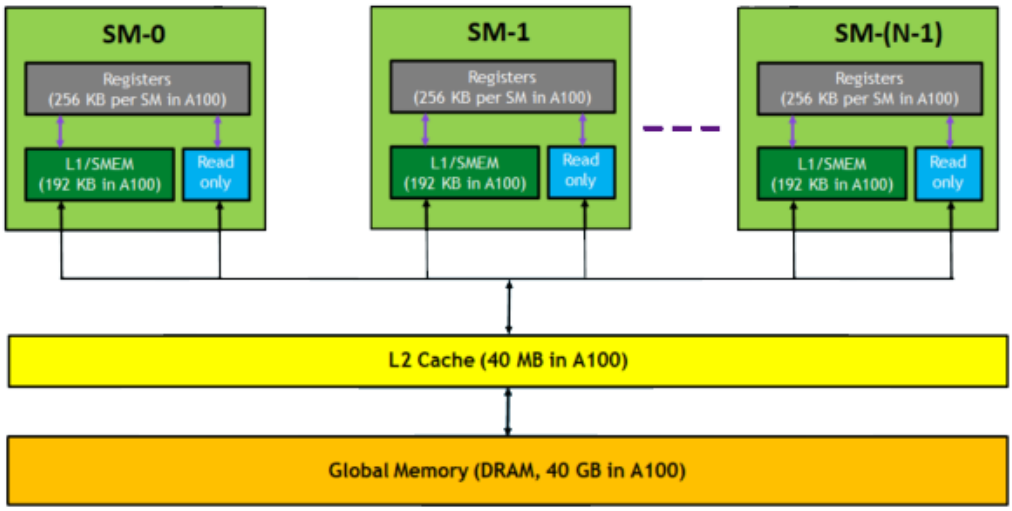


FIGURE 10 – The hierarchical memory architecture of a modern GPU [ArcCompute, 2023].

operate on it collaboratively, and share results before writing the final output back. This avoids many redundant, high-latency trips to the main memory and is a key technique for optimizing algorithms like matrix multiplication and convolutions.

A.4.2.2 Off-Chip Memory (Shared Across the GPU)

These memories are physically separate from the SMs and have higher latency and energy cost per access.

L2 Cache : A large, unified cache (e.g., 50 MB on an H100) that is shared by all SMs on the GPU. It acts as an intermediate buffer between the SMs' L1 caches and the main memory, capturing data reuse that occurs across different thread blocks or kernel launches. A high hit rate in the L2 cache is crucial for performance.

High-Bandwidth Memory (HBM) / Main Memory : This is the largest pool of memory available to the GPU (e.g., 80 GB on an H100), but also the slowest and most energy-intensive to access. This is where the model's parameters, input data, and large intermediate activation maps are stored. All data must ultimately originate from and be written back to this memory.

A.4.3 High-Bandwidth Memory (HBM) : A Technological Enabler

To meet the extreme bandwidth demands of its parallel architecture, high-end GPUs do not use the conventional DDR/GDDR DRAM found in CPU systems and traditional GPUs. Instead, they rely on **High-Bandwidth Memory (HBM)**. The key innovation of HBM is its approach to the physical interface, as illustrated in Figure 11. GDDR (left) uses a long, narrow, high-speed electrical bus to a separate DIMM. HBM (right) stacks multiple DRAM dies vertically

and connects them to the GPU through a very short, extremely wide, but lower-frequency bus via a "silicon interposer".

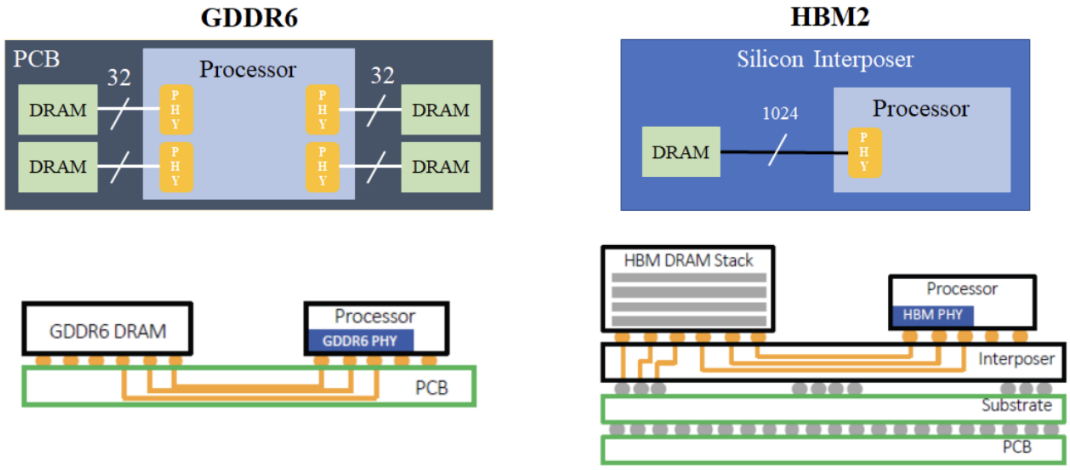


FIGURE 11 – A comparison of conventional GDDR memory and HBM [Jay, 2020].

Instead of connecting to separate memory modules (DIMMs) via a long, high-speed bus, HBM stacks multiple DRAM dies vertically. This 3D stack is then placed on the same package as the GPU, connected by a silicon **"interposer"**. This design has two profound advantages :

1. **Extremely Wide Bus :** The HBM interface can be thousands of bits wide (e.g., 4096-bit), compared to the 64-bit interface of a single DDR channel. Even running at a lower clock frequency, this immense width provides an order-of-magnitude higher total bandwidth (e.g., >3 TB/s for HBM3 vs. ∼460 GB/s for a dual-socket CPU with DDR5).
2. **Higher Energy Efficiency :** Driving signals across the short, wide interface of the interposer consumes significantly less energy per bit transferred compared to driving signals across the long traces of a motherboard to a separate DIMM.

HBM is a critical enabling technology for modern AI accelerators. Without its massive bandwidth and superior energy efficiency, it would be impossible to keep the thousands of cores in a modern GPU supplied with data.

A.4.4 Optimizing Data Movement : The Principle of Locality on a GPU

The entire memory hierarchy is designed to exploit the principle of locality. For a GPU programmer, this translates into a key goal : *minimize data movement from HBM*. This is achieved primarily through two techniques :

1. Memory Coalescing. When the threads in a warp access HBM, the hardware attempts to group, or *"coalesce"*, these individual accesses into as few memory transactions as possible. If all 32 threads in a warp access consecutive locations in memory, the GPU can satisfy all 32 requests with a single 128-byte memory transaction. Conversely, if the threads access memory in a random, scattered pattern, it may require up to 32 separate transactions, decimating the

effective memory bandwidth and dramatically increasing energy consumption. Writing memory-coalesced code is arguably the single most important optimization for GPU performance.

2. Maximizing Data Reuse. The second goal is to load data from HBM into the fast on-chip memory once and operate on it as many times as possible before discarding it. This is the essence of data reuse. An algorithm with a high **arithmetic intensity** (ratio of arithmetic operations to bytes of data moved) will perform well on a GPU. A well-known example is tiled matrix multiplication, where small blocks (tiles) of the input matrices are explicitly loaded into shared memory, and all necessary partial products are computed before the next tiles are loaded. This strategy is precisely what makes the GPU so efficient for the dense matrix algebra at the heart of Deep Learning.

The models developed in this thesis (Chapter 4) implicitly capture these effects. The model for memory time (T_{mem}), for instance, depends on the total volume of data moved and the L2 cache efficiency, which are directly influenced by the locality and data reuse patterns of the workload. Understanding this memory hierarchy provides the physical intuition behind why these modeling parameters are so crucial.

A.5 Observing the Machine : Performance and Energy Monitoring on Heterogeneous Systems

The preceding sections have detailed the immense complexity of modern processor architectures. The intricate dance of pipelined, out-of-order execution on CPUs and the massive concurrency of thousands of threads on GPUs create a system whose behavior is impossible to fully predict from software source code alone. To bridge this gap between theory and reality, we need a way to directly observe the machine in action. This is the role of the **Performance Monitoring Unit (PMU)**, a specialized subsystem embedded within modern processors that provides the empirical ground truth for all rigorous performance and energy analysis.

This appendix provides a comprehensive overview of performance monitoring, a practice central to this thesis. We will explore the architecture of PMUs, the key events they can track, and the different methods for collecting this data. Crucially, we will detail the distinct tools and challenges associated with monitoring both the CPU (host) and the GPU (device), highlighting the complexities of building a unified measurement framework like EA2P.

A.5.1 The Role and Architecture of the Performance Monitoring Unit

A Performance Monitoring Unit is not part of the primary data path of a processor ; it is a dedicated, non-intrusive hardware subsystem designed specifically for observability. Its core components are a set of special-purpose registers known as **Hardware Performance Counters (HPCs)**. These are counters that can be programmed to increment each time a specific micro-architectural event occurs. Modern architectures contain multiple PMUs distributed throughout the chip to provide a granular view of the system's operation, as illustrated in Figure 12. PMUs are distributed, with dedicated units per core (**core PMU**) to track core-specific events, and a

single, shared unit for the rest of the chip (**uncore PMU**) to monitor shared resources like the Last-Level Cache and memory controllers.

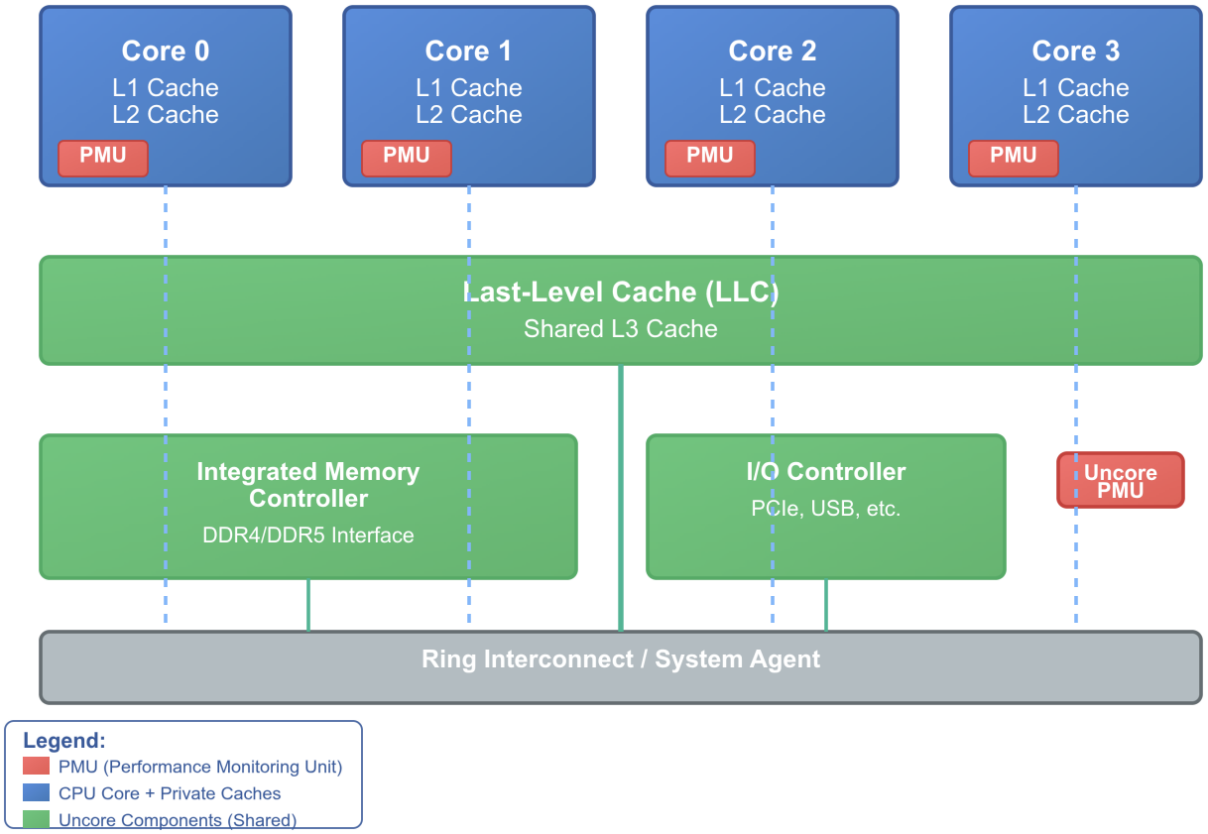


FIGURE 12 – Conceptual placement of PMUs in a modern multi-core processor.

Core PMUs : Each individual CPU core has its own private PMU. This allows for the precise measurement of events directly related to the instruction stream being executed on that core, such as instructions retired, branch mispredictions, or L1/L2 cache misses.

Uncore PMUs : The parts of the processor outside the cores (such as the Last-Level Cache (L3), the on-chip interconnect (ring bus or mesh), and the main memory controllers) are collectively referred to as the "uncore". A dedicated uncore PMU monitors the activity of these shared resources, providing critical system-wide data like memory bandwidth and L3 cache utilization.

A.5.2 Hardware Performance Counters and Key Events

HPCs fall into two categories : *Fixed-function counters* and *Programmable counters*. **Fixed-function counters** are hardwired to measure the most fundamental events, such as instructions completed and reference clock cycles. **Programmable counters** are more flexible and can be configured to count one of hundreds of possible architecture-specific events. The choice of which events to monitor is the primary task of performance analysis. Table 5 lists a selection of the most vital events and what they reveal about an application's behavior. A high IPC indicates

a compute-bound workload, while a high L3 miss rate and high memory bandwidth indicate a memory-bound workload.

TABLE 5 – A selection of key hardware performance events and their interpretation.

Key Performance Event	**Interpretation and Significance**
`Instructions Retired / CPU Cycles`	The ratio of these two events gives the *Instructions Per Cycle (IPC)*, the single most important metric of core efficiency. An IPC > 1 suggests the core is well-utilized ; an IPC < 1 suggests it is frequently stalled.
`L3_CACHE_MISSES`	Indicates that data was not found in the last level of cache and a costly request to main memory (DRAM) was required. A high miss rate is the primary symptom of a *"memory-bound"* application.
`BRANCH_MISSES`	The number of times the CPU's branch predictor guessed the outcome of a conditional statement (like an 'if' block) incorrectly. Each misprediction causes the entire instruction pipeline to be flushed and refilled, wasting dozens of cycles and significant energy.
`FP_ARITH_INST_RETIRED`	Counts the number of floating-point operations executed. This is the basis for calculating the achieved FLOPS (Floating-Point Operations Per Second) of a scientific or AI workload.
`UNC_M_CAS_COUNT.RD/WR`	An *uncore* event that counts the number of read/write commands sent by the memory controller to DRAM. This provides a direct measure of *"memory bandwidth"*, a critical bottleneck for many AI applications.

A.5.3 Monitoring the CPU (Host) : Tools and Challenges

Accessing the PMU on a CPU requires privileged operations, as it involves writing to Model-Specific Registers (MSRs). Several layers of software abstraction have been built to make this accessible to developers.

Low-Level Interfaces. At the lowest level, the Linux kernel provides the `perf_events` subsystem, a powerful system call that is the foundation for most modern profiling. It handles the complex tasks of programming the PMU, managing counter multiplexing (when more events are requested than available hardware counters), and collecting the data.

User-Space Tools. Developers typically interact with these interfaces through high-level tools :

— **Linux 'perf'** [Firefox, 2023] : A command-line tool that provides a comprehensive interface to the 'perf_events' subsystem. A command like '`perf stat -e instructions,cycles,`

`cache-misses ./my_app'` provides an aggregate count of these events for an entire application run.

— **PAPI (Performance Application Programming Interface)** [McCraw, 2014] : A standardized C/Fortran library that provides a portable API for accessing HPCs across different architectures (Intel, AMD, ARM). It simplifies the process of instrumenting specific sections of code.

— **LIKWID (Like I Knew What I'm Doing)** [Treibig, 2010a] : A powerful suite of command-line tools that simplifies pinning threads to specific cores and measuring performance groups, making it particularly useful for HPC applications.

Challenges of Uncore Monitoring. While core events are relatively straightforward to attribute to a specific process, monitoring uncore events like memory bandwidth is more complex. Because the uncore resources are shared by all cores, the data is system-wide. Disentangling which process is responsible for which portion of the memory traffic is a significant attribution challenge.

A.5.4 Monitoring the GPU (Device) : A Distinct Ecosystem

Monitoring GPUs presents a different set of challenges and relies on a separate ecosystem of vendor-specific tools. Unlike CPUs, there is currently no standardized, kernel-level interface like 'perf_events' for GPUs.

Vendor APIs and Tools. Access to GPU performance data is primarily provided through libraries supplied by the hardware vendor.

For NVIDIA GPUs, the primary tool is the NVIDIA Management Library (NVML), which is the programmatic interface underlying the popular 'nvidia-smi' command-line utility. NVML is excellent for querying high-level, slowly-changing state information about the GPU, such as : *Overall GPU and Memory Utilization (%), Power Consumption (Watts) and Temperature (°C),* and *Clock Frequencies (MHz).* This is the interface used by EA2P for its GPU power measurement, as it is simple, robust, and universally available. For more detailed, micro-architectural insights, NVIDIA provides the **Nsight suite** [Nvidia, 2025a] (Nsight Systems for system-level traces and Nsight Compute for in-depth kernel profiling). These tools can collect extremely detailed information directly from the SM's internal PMUs, such as SM occupancy, Tensor Core activity, L1/L2 hit rates, and memory transaction efficiency. This level of detail is invaluable for deep optimization but is too high-overhead for continuous, live monitoring.

A.5.5 Fundamental Challenges and the Justification for EA2P

This overview highlights the complex, fragmented landscape of performance and energy monitoring, which directly motivates the contributions of this thesis.

1. **Heterogeneity and Lack of Standardization :** Monitoring a CPU and a GPU requires two completely different sets of tools, APIs, and event names. There is no single, unified

interface. This was a primary driver for developing **EA2P**, which aims to provide a single, consistent Python interface that abstracts away these vendor-specific differences for the user.

2. **The Measurement "Blind Spot" (DRAM) :** While CPU uncore counters can provide memory bandwidth data, they do not directly measure the energy consumed by the DRAM modules themselves. As we have established, this is a significant and often unmeasured component of system power, justifying our development of an *analytical DRAM energy model* to fill this crucial gap.
3. **The Observer Effect :** The act of measurement always perturbs the system. High-frequency sampling, especially with detailed tools like Nsight Compute, can introduce significant overhead, altering the very behavior one is trying to measure. Tools like EA2P, which rely on low-overhead polling of high-level interfaces like RAPL and NVML, are designed to minimize this effect for application-level profiling.
4. **The Challenge of Energy Attribution :** The final and most profound challenge is attribution. The power reported by `'nvidia-smi'` is for the entire GPU board. The power reported by Intel's RAPL is for the entire CPU package. In a system running multiple processes, the operating system, and drivers, definitively attributing a specific portion of this energy to a single line of Python code is a fundamentally difficult, if not impossible, task. Our approach in this thesis is pragmatic : we focus on measuring the total system delta—the change in consumption of the entire system when the target application is running versus when it is idle. This differential measurement is a robust and actionable metric for quantifying the application's overall energy impact and for guiding system-level optimizations like scheduling.

In conclusion, the ability to observe the machine via its performance monitoring hardware is the bedrock of empirical computer science. However, the complexity, heterogeneity, and inherent limitations of these tools necessitate the development of higher-level, integrated frameworks to make their power accessible and actionable for the developers and system administrators working to build a more efficient and sustainable AI ecosystem.

Bibliography

[Aarti, 2023] Dhapte Aarti. *High-Performance Computing Market Report*. Accessed : 2025-09-21. 2023. URL : https://www.marketresearchfuture.com/reports/high-performance-computing-market-2698 (cf. p. 8).

[Adhinarayanan, 2025] Vignesh Adhinarayanan et Wu-chun Feng. « Looking Back to Look Forward : 15 Years of the Green500 ». *Computer* 58.01 (2025), p. 76-86. URL : https://doi.ieeecomputersociety.org/10.1109/MC.2023.3333316 (cf. p. 20).

[AKCP, 2021] AKCP. *Rear-Door Heat Exchanger for High-density Data Center*. Accessed : 2025-09-21. 2021. URL : https://www.akcp.com/articles/rear-door-heat-exchanger-for-high-density-data-center/ (cf. p. 35).

[Alan, 2014] Ismail Alan, Engin Arslan et Tevfik Kosar. « Energy-Aware Data Transfer Tuning ». *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2014, p. 626-634 (cf. p. 48).

[Alberto, 2021] Romero Alberto. *Meet M6 - 10 Trillion Parameters at 1% GPT-3's Energy Cost*. Accessed : 2025-09-01. 2021. URL : https://towardsdatascience.com/meet-m6-10-trillion-parameters-at-1-gpt-3s-energy-cost-997092cbe5e8 (cf. p. 7, 39).

[Amaral, 2023] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee et al. « Kepler : A Framework to Calculate the Energy Consumption of Containerized Applications ». *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 2023, p. 69-71 (cf. p. 53).

[AMD, 2014] AMD. *ROCm System Management*. Accessed : 2023-04-21. 2014. URL : https://sep5.readthedocs.io/en/latest/ROCm_System_Managment/ROCm-System-Managment.html (cf. p. 25).

[AMD, 2021] AMD. *AMD INSTINCT MI200 SERIES ACCELERATOR*. Accessed : 2025-07-25. 2021. URL : https://www.amd.com/system/files/documents/amd-instinct-mi200-datasheet.pdf (cf. p. 31).

[AMD, 2023] AMD. *AMD EPYC Energy Efficiency*. https://www.amd.com/en/campaigns/epyc-energy-efficiency. Accessed : 2025-09-01. 2023 (cf. p. 32).

[Anthony, 2020] Lasse F. Wolff Anthony, Benjamin Kanding et Raghavendra Selvan. *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. 2020 (cf. p. 28, 64).

[Appleyard, 2017] Jeremy Appleyard et Scott Yokim. *Programming Tensor Cores in CUDA 9*. Accessed : 2025-09-24. 2017. URL : https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/ (cf. p. 200).

[Appuswamy, 2015] Raja Appuswamy, Matthaios Olma et Anastasia Ailamaki. « Scaling the Memory Power Wall With DRAM-Aware Data Management ». *Proceedings of the 11th International Workshop on Data Management on New Hardware* (2015) (cf. p. 34, 51, 63, 103-105).

[ArcCompute, 2023] ArcCompute. *GPU 101 : Memory Hierarchy*. Accessed : 2025-09-24. 2023. URL : https://www.arccompute.io/arc-blog/gpu-101-memory-hierarchy (cf. p. 204).

[Archet, 2023] Agathe Archet, Nicolas Ventroux, Nicolas Gac et François Orieux. « Energy-Efficient Use of an Embedded Heterogeneous SoC for the Inference of CNNs ». *2023 26th Euromicro Conference on Digital System Design (DSD)*. 2023, p. 30-38 (cf. p. 53).

[Arduino, 2023] Arduino. *Arduino Portenta H7 product Overview*. Accessed : 2025-09-01. 2023. URL : https://store.arduino.cc/products/portenta-h7 (cf. p. 33).

[Arne, 2022] Tarara Arne. *TDP AND ACP FOR ENERGY ESTIMATION IN PROCESSORS*. Accessed : 2023-04-21. 2022. URL : https://www.green-coding.berlin/blog/tdp-and-acp/ (cf. p. 22).

[Barroso, 2013] Luiz André Barroso, Jimmy Clidaras et Urs Hölzle. *The Datacenter as a Computer : An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013. URL : http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024 (cf. p. 5).

[Bartezzaghi, 2015] Andrea Bartezzaghi, Massimiliano Cremonesi, Nicola Parolini et Umberto Perego. « An explicit dynamics GPU structural solver for thin shell finite elements ». *Computers & Structures* 154 (2015) (cf. p. 197).

[BCG, 2020] GAMMA BCG. *CodeCarbon*. Accessed : 2025-09-21. 2020. URL : https://mlco2.github.io/codecarbon/index.html (cf. p. 28, 45, 64).

[Belady, 2008] Christian Belady et Andrew Rawson. « GREEN GRID DATA CENTER POWER EFFICIENCY METRICS : PUE AND DCIE ». 2008 (cf. p. 20).

[Beloglazov, 2012] Anton Beloglazov, Jemal Abawajy et Rajkumar Buyya. « Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing ». *Future Generation Computer Systems* 28.5 (2012). Special Section : Energy efficiency in large-scale distributed systems, p. 755-768. URL : https://www.sciencedirect.com/science/article/pii/S0167739X11000689 (cf. p. 47).

[Beloglazov, 2010] Anton Beloglazov, Rajkumar Buyya, Young Lee et Albert Zomaya. « A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems ». *Advances in Computers* 82 (2010) (cf. p. 54, 55).

[Benoit, 2018] Anne Benoit, Laurent Lefèvre, Anne-Cécile Orgerie et Issam Raïs. « Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints ». *Int. Journal of High Performance Computing Applications* 32.1 (2018), p. 176-188. URL : https://inria.hal.science/hal-01557025 (cf. p. 56).

[Berral, 2011] Josep Ll. Berral, Ricard Gavalda et Jordi Torres. « Adaptive Scheduling on Power-Aware Managed Data-Centers Using Machine Learning ». *2011 IEEE/ACM 12th International Conference on Grid Computing*. 2011, p. 66-73 (cf. p. 48).

[Berral, 2010] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà et al. « Towards energy-aware scheduling in data centers using machine learning ». *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. e-Energy '10. Passau, Germany : Association for Computing Machinery, 2010, p. 215-224. URL : https://doi.org/10.1145/1791314.1791349 (cf. p. 48).

[Bertran, 2010] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro et Eduard Ayguade. « Decomposable and responsive power models for multicore processors using performance counters ». *24th ACM International Conference on Supercomputing*. ICS '10. Tsukuba, Ibaraki, Japan, 2010, p. 147-158 (cf. p. 50, 52).

[Bohrer, 2002] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell et al. « The case for power management in web servers ». *Power Aware Computing*. USA : Kluwer Academic Publishers, 2002, p. 261-289 (cf. p. 47).

[Boku, 2022] Taisuke Boku. « How FPGA can contribute to HPC ? » *2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 2022 (cf. p. 32).

[Bourdon, 2013] Aurelien Bourdon, Adel Noureddine, Romain Rouvoy et Lionel Seinturier. « PowerAPI : A Software Library to Monitor the Energy Consumed at the Process-Level ». *ERCIM News* (2013) (cf. p. 28).

[Brace, 2022] Alexander Brace, Igor Yakushin, Heng Ma, Anda Trifan, Todd Munson, Ian Foster et al. *Coupling streaming AI and HPC ensembles to achieve 100-1000x faster biomolecular simulations*. 2022. arXiv : 2104.04797 [cs.DC]. URL : https://arxiv.org/abs/2104.04797 (cf. p. 6).

[Browne, 1999] Shirley Browne, Christine Deane, George Ho et Philip Mucci. « PAPI : A Portable Interface to Hardware Performance Counters ». 1999 (cf. p. 27).

[Budennyy, 2023] SA Budennyy, VD Lazarev, NN Zakharenko, AN Korovin, OA Plosskaya, DV Dimitrov et al. « Eco2ai : carbon emissions tracking of machine learning models as the first step towards sustainable ai ». *Doklady Mathematics*. 2023 (cf. p. 26, 28, 64).

[Cabrera, 2019] Alberto Cabrera, Francisco Almeida, Vicente Blanco et Dagoberto Nieves. « Finding energy efficient hardware configurations under a power cap ». 2019 (cf. p. 55).

[Calheiros, 2011] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose et Rajkumar Buyya. « CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms ». *Software : Practice and Experience* 41.1 (2011), p. 23-50. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.995. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.995 (cf. p. 47).

[Canuto, 2016] Mauro Canuto, Raimon Bosch, Mario Macias et Jordi Guitart. « A methodology for full-system power modeling in heterogeneous data centers ». *Proceedings of the 9th International Conference on Utility and Cloud Computing*. UCC '16. Shanghai, China : Association for Computing Machinery, 2016, p. 20-29. URL : https://doi.org/10.1145/2996890.2996899 (cf. p. 48).

[Castro, 2018] Francisco Manuel Castro, Nicolás Guil Mata, Manuel J. Marín-Jiménez, Jesús Pérez Serrano et Manuel Ujaldón. « Energy-based tuning of convolutional neural networks on multi-GPUs ». *Concurrency and Computation : Practice and Experience* 31 (2018) (cf. p. 42).

[Chen, 2008] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao et al. « Energy-aware server provisioning and load dispatching for connection-intensive internet services ». *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. San Francisco, California : USENIX Association, 2008, p. 337-350 (cf. p. 48).

[Chen, 2017] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer et Vivienne Sze. « Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks ». *IEEE Journal of Solid-State Circuits* 52.1 (2017), p. 127-138 (cf. p. 42).

[Cheung, 2018] Howard Cheung, Shengwei Wang, Chaoqun Zhuang et Jiefan Gu. « A simplified power consumption model of information technology (IT) equipment in data centers for energy system real-time dynamic simulation ». *Applied Energy* 222 (2018), p. 329-342. URL : https://www.sciencedirect.com/science/article/pii/S0306261918304768 (cf. p. 48).

[Chowdhery, 2022] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts et al. « PaLM : Scaling Language Modeling with Pathways ». *J. Mach. Learn. Res.* 24 (2022), 240 :1-240 :113 (cf. p. 43).

[CNDC, 2021] CNDC. *Climate Neutral Data Centre Webpage*. Accessed : 2025-09-21. 2021. URL : https://www.climateneutraldatacentre.net/ (cf. p. 10).

[Colin, 2021] King Colin Ian. *powerstat - a tool to measure power consumption*. Accessed : 2023-05-26. 2021. URL : https://manpages.ubuntu.com/manpages/bionic/man8/powerstat.8.html (cf. p. 28).

[CPU-world, 2023] CPU-world. *Intel Xeon 8358 specifications*. Accessed : 2024-05-07. 2023. URL : https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%5C%208358.html (cf. p. 81).

[Cupertino, 2015] Leandro Fontoura Cupertino, Georges Costa et Jean-Marc Pierson. « Towards a generic power estimator ». *Comput. Sci.* 30.2 (2015), p. 145-153. URL : https://doi.org/10.1007/s00450-014-0264-x (cf. p. 49, 50, 52).

[Da Costa, 2010] Georges Da Costa et Helmut Hlavacs. « Methodology of measurement for energy consumption of applications ». *2010 11th IEEE/ACM International Conference on Grid Computing*. 2010, p. 290-297 (cf. p. 50).

[Dai, 2016] Xiangming Dai, Jason Min Wang et Brahim Bensaou. « Energy-Efficient Virtual Machines Scheduling in Multi-Tenant Data Centers ». *IEEE Transactions on Cloud Computing* 4.2 (2016), p. 210-221 (cf. p. 47).

[Dalcin, 2021] Lisandro Dalcin et Yao-Lung L. Fang. « mpi4py : Status Update After 12 Years of Development ». *Computing in Science & Engineering* 23.4 (2021), p. 47-54 (cf. p. 70).

[Daniel, 2021] Bizo Daniel, Ascierto Rhonda, Lawrence Andy et Davis Jacqueline. *2021 Data Center Industry Survey Results*. Accessed : 2023-04-21. 2021. URL : https://uptimeinstitute.com/2021-data-center-industry-survey-results (cf. p. 20).

[David, 2013] Greenhill David. *SWaP Space Watts and Power*. Accessed : 2023-04-21. 2013. URL : https://www.energystar.gov/ia/products/downloads/Greenhill%5C_Pres.pdf (cf. p. 22).

[Dayarathna, 2016] Miyuru Dayarathna, Yonggang Wen et Rui Fan. « Data Center Energy Consumption Modeling : A Survey ». *IEEE Communications Surveys & Tutorials* 18.1 (2016), p. 732-794 (cf. p. 46).

[Deng, 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li et Li Fei-Fei. « ImageNet : A large-scale hierarchical image database ». *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, p. 248-255 (cf. p. 72).

[Dennard, 1974] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous et A.R. LeBlanc. « Design of ion-implanted MOSFET's with very small physical dimensions ». *IEEE Journal of Solid-State Circuits* 9.5 (1974), p. 256-268 (cf. p. 191).

[Desislavov, 2023] Radosvet Desislavov, Fernando Martínez-Plumed et José Hernández-Orallo. « Trends in AI inference energy consumption : Beyond the performance-vs-parameter laws of deep learning ». *Sustainable Computing : Informatics and Systems* 38 (2023), p. 100857 (cf. p. 39, 42).

[Devlin, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee et Kristina Toutanova. « BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding ». *North American Chapter of the Association for Computational Linguistics*. 2019 (cf. p. 42).

[Dhiman, 2010] Gaurav Dhiman, Kresimir Mihic et Tajana Rosing. « A system for online power prediction in virtualized environments using Gaussian mixture models ». *Proceedings of the 47th Design Automation Conference*. DAC '10. Anaheim, California : Association for Computing Machinery, 2010, p. 807-812. URL : https://doi.org/10.1145/1837274.1837478 (cf. p. 49).

[Diel, 2025] Guilherme Diel, Ana Kraus et Guilherme Koslovski. « Knowledge-based job scheduling for HPC ». *Cluster Computing* 28 (2025) (cf. p. 56).

[Ding, 2019] Nan Ding et Samuel Williams. « An Instruction Roofline Model for GPUs ». *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2019, p. 7-18 (cf. p. 52, 107).

[Dolz, 2016] Manuel F. Dolz, Julian Kunkel, Konstantinos Chasapis et Sandra Catalán. « An analytical methodology to derive power models based on hardware and software metrics ». *Comput. Sci.* 31.4 (2016), p. 165-174 (cf. p. 50, 52).

[Dongarra, 2003] Jack Dongarra, Piotr Luszczek et Antoine Petitet. « The LINPACK Benchmark : past, present and future ». *Concurrency and Computation : Practice and Experience* 15 (2003), p. 803-820 (cf. p. 19).

[Dosovitskiy, 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner et al. *An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale*. 2021. arXiv : 2010.11929 [cs.CV]. URL : https://arxiv.org/abs/2010.11929 (cf. p. 124, 149).

[Eastep, 2017] Jonathan M. Eastep, Steve Sylvester, Christopher M. Cantalupo, Brad Geltz, Federico Ardanaz, Asma H. Al-rawi et al. « Global Extensible Open Power Manager : A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions ». *Information Security Conference*. 2017. URL : https://api.semanticscholar.org/CorpusID:791727 (cf. p. 27, 28).

[Eberius, 2022] David Eberius, Philip Roth et David M. Rogers. « Understanding Strong Scaling on GPUs Using Empirical Performance Saturation Size ». *2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. 2022, p. 26-35 (cf. p. 52).

[Economou, 2006] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis et Partha Ranganathan. *Full-system Power Analysis and Modeling for Server Environments*. 2006 (cf. p. 48, 49).

[EEMBC, 2014] EEMBC. *An EEMBC Benchmark*. Accessed : 2025-04-21. 2014. URL : https://www.eembc.org/ulpmark/ulp-cp/ (cf. p. 33).

[ElectronicsLessons, 2025] ElectronicsLessons. *Types of Transistors*. Accessed : 2025-09-24. 2025. URL : https://electronicslesson.com/types-of-transistors/ (cf. p. 190).

[Esser, 2024] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini et al. « Scaling rectified flow transformers for high-resolution image synthesis ». *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria : JMLR.org, 2024 (cf. p. 44).

[Fan, 2007a] Xiaobo Fan, Wolf-Dietrich Weber et Luiz Andre Barroso. « Power provisioning for a warehouse-sized computer ». *Proceedings of the 34th Annual International Symposium on Computer Architecture*. ISCA '07. San Diego, California, USA : Association for Computing Machinery, 2007, p. 13-23. URL : https://doi.org/10.1145/1250662.1250665 (cf. p. 47).

[Fan, 2007b] Xiaobo Fan, Wolf-Dietrich Weber et Luiz Andre Barroso. « Power provisioning for a warehouse-sized computer ». *SIGARCH Comput. Archit. News* 35.2 (2007), p. 13-23. URL : https://doi.org/10.1145/1273440.1250665 (cf. p. 47, 48).

[Fieni, 2020] Guillaume Fieni, Romain Rouvoy et Lionel Seinturier. « SmartWatts : Self-Calibrating Software-Defined Power Meter for Containers ». *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020. URL : http://dx.doi.org/10.1109/CCGrid49817.2020.00-45 (cf. p. 53).

[Filippini, 2024] Federica Filippini, Jonatha Anselmi, Danilo Ardagna et Bruno Gaujal. « A Stochastic Approach for Scheduling AI Training Jobs in GPU-Based Systems ». *IEEE Transactions on Cloud Computing* 12.1 (2024), p. 53-69 (cf. p. 56).

[Firefox, 2023] Firefox. *Firefox Documentation - Energy estimates*. Accessed : 2025-09-21. 2023. URL : https://firefox-source-docs.mozilla.org/performance/perf.html (cf. p. 28, 208).

[Freitag, 2021] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S Blair et Adrian Friday. « The real climate and transformative impact of ICT : A critique of estimates, trends, and regulations ». *Patterns* 2.9 (2021) (cf. p. 8).

[Frey, 2022] Nathan C. Frey, Dan Zhao, Simon Axelrod, Michael Jones, David Bestor, Vijay Gadepally et al. « Energy-aware neural architecture selection and hyperparameter optimization ». *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2022, p. 732-741 (cf. p. 52, 57).

[Gao, 2021] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen et Tianwei Zhang. « Chronus : A Novel Deadline-aware Scheduler for Deep Learning Training Jobs ». *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '21. Association for Computing Machinery, 2021 (cf. p. 136, 137, 143).

[Ghafouri, 2023] Saeid Ghafouri, Sina Abdipoor et Joseph Doyle. « Smart-Kube : Energy-Aware and Fair Kubernetes Job Scheduler Using Deep Reinforcement Learning ». *2023 IEEE 8th International Conference on Smart Cloud (SmartCloud)*. 2023, p. 154-163 (cf. p. 135).

[Gholami, 2021a] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney et Kurt Keutzer. « A Survey of Quantization Methods for Efficient Neural Network Inference ». *ArXiv* abs/2103.13630 (2021) (cf. p. 40, 42).

[Gholami, 2021b] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney et Kurt Keutzer. « A Survey of Quantization Methods for Efficient Neural Network Inference ». *ArXiv* abs/2103.13630 (2021) (cf. p. 57).

[GlobalPetrolPricecom, 2025] GlobalPetrolPrice.com. *Global Electricity prices September 2022*. Accessed : 2025-09-01. 2025. URL : https://www.globalpetrolprices.com/electricity_prices/ (cf. p. 21).

[Gmach, 2009] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova et Alfons Kemper. « Resource pool management : Reactive versus proactive or let's be friends ». *Computer Networks* 53.17 (2009). Virtualized Data Centers, p. 2905-2922. URL : https://www.sciencedirect.com/science/article/pii/S1389128609002655 (cf. p. 47).

[Gonzalez, 1996] R. Gonzalez et M. Horowitz. « Energy dissipation in general purpose microprocessors ». *IEEE Journal of Solid-State Circuits* 31.9 (1996), p. 1277-1284 (cf. p. 22).

[Goodfellow, 2016] Ian Goodfellow, Yoshua Bengio et Aaron Courville. *Deep Learning*. The MIT Press, 2016 (cf. p. 2).

[Google, 2023] Google. *system architecture tpu v4*. Accessed : 2025-04-21. 2023. URL : https://cloud.google.com/tpu/docs/system-architecture-tpu-vm (cf. p. 31).

[GoogleLLC, 2020a] GoogleLLC. *Dev Board datasheet*. Accessed : 2025-09-01. 2020. URL : https://coral.ai/docs/dev-board/datasheet/ (cf. p. 31, 33).

[GoogleLLC, 2020b] GoogleLLC. *Edge TPU performance benchmarks*. Accessed : 2025-09-01. 2020. URL : https://coral.ai/docs/edgetpu/benchmarks/ (cf. p. 31).

[Gou, 2021] Jianping Gou, Baosheng Yu, Stephen J. Maybank et Dacheng Tao. « Knowledge Distillation : A Survey ». *Int. J. Comput. Vision* 129.6 (2021), p. 1789-1819. URL : https://doi.org/10.1007/s11263-021-01453-z (cf. p. 42, 57).

[Grant, 2017] Ryan E. Grant, James H. Laros, Michael Levenhagen, Stephen L. Olivier, Kevin Pedretti, Lee Ward et al. « Evaluating energy and power profiling techniques for HPC workloads ». *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. 2017, p. 1-8 (cf. p. 57).

[Green500, 2025] Green500. *GREEN500*. Accessed : 2025-07-15. 2025. URL : https://www.top500.org/lists/green500/ (cf. p. 19).

[Grid5000, 2025] Grid5000. *Grid5000 :Home*. Accessed : 2025-09-01. 2025. URL : https://www.grid5000.fr/w/Grid5000:Home (cf. p. 80).

[Gu, 2023] Diandian Gu, Xintong Xie, Gang Huang, Xin Jin et Xuanzhe Liu. *Energy-Efficient GPU Clusters Scheduling for Deep Learning*. 2023. arXiv : 2304.06381 [cs.DC]. URL : https://arxiv.org/abs/2304.06381 (cf. p. 134, 135).

[Gu, 2025] Diandian Gu, Yihao Zhao, Peng Sun, Xin Jin et Xuanzhe Liu. « GreenFlow : A Carbon-Efficient Scheduler for Deep Learning Workloads ». *IEEE Transactions on Parallel & Distributed Systems* 36.02 (2025), p. 168-184. URL : https://doi.ieeecomputersociety.org/10.1109/TPDS.2024.3470074 (cf. p. 135).

[Guan, 2022] Wenkai Guan et Cristinel Ababei. « Unified Cross-Layer Cluster-Node Scheduling for Heterogeneous Datacenters ». *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*. 2022, p. 1-8 (cf. p. 56).

[Gutiérrez, 2023] Hermosillo Muriedas Gutiérrez, Pedro Juan, Katharina Flügel, Charlotte Debus, Holger Obermaier, Achim Streit et al. « perun : Benchmarking Energy Consumption of High-Performance Computing Applications ». *Euro-Par 2023 : Parallel Processing : 29th International Conference on Parallel and Distributed Computing, Limassol, Cyprus, August 28 – September 1, 2023, Proceedings*. Limassol, Cyprus : Springer-Verlag, 2023, p. 17-31 (cf. p. 28, 64).

[Hackenberg, 2014] Daniel Hackenberg, Thomas Ilsche, Joseph Schuchart, Robert Schöne, Wolfgang E. Nagel, Marc Simon et al. « HDEEM : High Definition Energy Efficiency Monitoring ». *Energy Efficient Supercomp. Workshop* (2014), p. 1-10 (cf. p. 24).

[He, 2015a] Kaiming He, X. Zhang, Shaoqing Ren et Jian Sun. « Deep Residual Learning for Image Recognition ». *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), p. 770-778 (cf. p. 42).

[He, 2015b] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv : 1512.03385 [cs.CV]. URL : https://arxiv.org/abs/1512.03385 (cf. p. 124, 149).

[He, 2015c] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun. « Deep Residual Learning for Image Recognition ». *CoRR* abs/1512.03385 (2015). arXiv : 1512.03385. URL : http://arxiv.org/abs/1512.03385 (cf. p. 57).

[Helmenstine, 2016] Anne Helmenstine. *Sources of Error in Science Experiments*. Accessed : 2025-08-13. 2016. URL : https://sciencenotes.org/error-in-science/ (cf. p. 141).

[Henderson, 2020] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky et Joelle Pineau. *Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning*. 2020. arXiv : 2002.05651 [cs.CY] (cf. p. 28, 64).

[Hennessy, 2012] John L. Hennessy et David A. Patterson. *Computer Architecture : A Quantitative Approach (5th ed.)* 2012, p. 22 (cf. p. 22).

[Hoffmann, 2022] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv : 2203.15556 [cs.CL]. URL : https://arxiv.org/abs/2203.15556 (cf. p. 52).

[Horowitz, 1994] M. Horowitz, T. Indermaur et R. Gonzalez. « Low-power digital design ». *Proc. IEEE Symp. on Low Power Electronics*. 1994 (cf. p. 144).

[Hosseinabady, 2018] Mohammad Hosseinabady et José Luis Núñez-Yáñez. « Dynamic Energy Management of FPGA Accelerators in Embedded Systems ». *ACM Transactions on Embedded Computing Systems (TECS)* 17 (2018), p. 1-26 (cf. p. 32).

[Husain Bohra, 2010] Ata E Husain Bohra et Vipin Chaudhary. « VMeter : Power modelling for virtualized clouds ». *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 2010, p. 1-8 (cf. p. 49).

[Hussain, 2021] Asim Hussain. *What is Green software ?* Accessed : 2025-09-21. 2021. URL : https://greensoftware.foundation/articles/what-is-green-software (cf. p. 11).

[IEA, 2024] IEA. *Electricity 2024 : Analysis and Forecast to 2026*. Accessed : 2025-09-21. 2024. URL : https://www.iea.org/reports/electricity-2024 (cf. p. 7).

[Ikram, 2018] Muhammad Ikram. « Energy-Efficient GPU-Based High-Performance Computing ». Thèse de doct. 2018 (cf. p. 30).

[Ilsche, 2017] Thomas Ilsche, Robert Schöne, Mario Bielert, Andreas Gocht et Daniel Hackenberg. « lo2s — Multi-core System and Application Performance Analysis for Linux ». *IEEE Int. Conf. on Cluster Comp. (CLUSTER)*. 2017, p. 801-804 (cf. p. 28).

[INRIA-Lille, 2019] INRIA-Lille. *Welcome to pyJoules's documentation !* https://pyjoules.readthedocs.io/en/latest/. Accessed : 2023-05-26. 2019 (cf. p. 28, 64).

[Intel, 2019] Intel. *HeadlineLakefield : Hybrid CPU with Foveros Technology*. Accessed : 2023-05-16. 2019. URL : https://www.intel.com/content/www/us/en/newsroom/resources/lakefield.html (cf. p. 54).

[Intel, 2020] Intel. *PowerTOP*. Accessed : 2023-05-26. 2020. URL : https://github.com/fenrus75/powertop (cf. p. 28).

[Intel, 2022a] Intel. *Intel Data Center GPU Max 1550*. Accessed : 2025-05-10. 2022. URL : https://ark.intel.com/content/www/us/en/ark/products/232873/intel-data-center-gpu-max-1550.html (cf. p. 31).

[Intel, 2022b] Intel. *Intel Neural Compute Stick 2 (Intel NCS2)*. Accessed : 2025-09-01. 2022. URL : https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html (cf. p. 33).

[Intel, 2022c] Corporation Intel. *Running Average Power Limit Energy Reporting / CVE-2020-8694 , CVE-2020-8695 / INTEL-SA-00389*. Accessed : 2023-04-21. 2022. URL : https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html (cf. p. 25, 27).

[IPCC, 2014] IPCC. « Summary for Policymakers ». *Climate Change 2013 – The Physical Science Basis : Working Group I Contribution to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 2014, p. 1-30 (cf. p. 50).

[Jacqueline, 2022] Davis Jacqueline, Bizo Daniel, Lawrence Andy, Rogers Owen et Smolaks Max. *2022 Data Center Industry Survey Results*. Accessed : 2023-04-21. 2022. URL : https://uptimeinstitute.com/resources/research-and-reports/uptime-institute-global-data-center-survey-results-2022 (cf. p. 20).

[Jafari-Nodoushan, 2020] Mostafa Jafari-Nodoushan, Bardia Safaei, Alireza Ejlali et Jian-Jia Chen. « Leakage-Aware Battery Lifetime Analysis Using the Calculus of Variations ». *IEEE Trans. on Circuits and Systems I* 67.12 (2020), p. 4829-4841 (cf. p. 57).

[Janacek, 2012] Stefan Janacek, Kiril Schröder, Gunnar Schomaker, Wolfgang Nebel, Marco Rüschen et Günter Pistoor. « Modeling and approaching a cost transparent, specific data center power consumption ». *2012 International Conference on Energy Aware Computing*. 2012, p. 1-6 (cf. p. 48).

[Janković, 2015] Strahinja Janković et Vujo Drndarević. « Microcontroller power consumption measurement based on PSoC ». *2015 23rd Telecommunications Forum Telfor (TELFOR)* (2015), p. 673-676 (cf. p. 24).

[Jarus, 2014] M. Jarus, A. Oleksiak, T. Piontek et J. Węglarz. « Runtime power usage estimation of HPC servers for various classes of real-life applications ». *Future Generation Computer Systems* 36 (2014). Special Section : Intelligent Big Data Processing Special Section : Behavior Data Security Issues in Network Information Propagation Special Section : Energy-efficiency in Large Distributed Computing Architectures Special Section : eScience Infrastructure and Applications, p. 299-310. URL : https://www.sciencedirect.com/science/article/pii/S0167739X1300157X (cf. p. 50).

[Jay, 2020] Jay. *Memory Systems for the Data-Intensive Applications (GDDR6, HBM2, HBM2E, and More...)* Accessed : 2025-09-24. 2020. URL : https://mediaupdate2019.wordpress.com/2020/03/28/memory-systems-for-the-data-intensive-applications/ (cf. p. 205).

[Jegham, 2025] Nidhal Jegham, Marwan Abdelatti, Lassad Elmoubarki et Abdeltawab Hendawi. *How Hungry is AI ? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference*. 2025. arXiv : 2505.09598 [cs.CY]. URL : https://arxiv.org/abs/2505.09598 (cf. p. 43).

[Jin, 2016] Xibo Jin, Fa Zhang, Athanasios V. Vasilakos et Zhiyong Liu. « Green Data Centers : A Survey, Perspectives and Future Directions ». *arXiv :1608.00687* (2016) (cf. p. 20).

[Jin, 2013] Yichao Jin, Yonggang Wen, Qinghua Chen et Zuqing Zhu. « An Empirical Investigation of the Impact of Server Virtualization on Energy Efficiency for Green Data Center ». *The Computer Journal* 56.8 (2013), p. 977-990. eprint : https://academic.oup.com/comjnl/article-pdf/56/8/977/1018769/bxt017.pdf. URL : https://doi.org/10.1093/comjnl/bxt017 (cf. p. 47).

[Kakolyris, 2024] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis et Dimitrios Soudris. « SLO-Aware GPU DVFS for Energy-Efficient LLM Inference Serving ». *IEEE Computer Architecture Letters* 23.2 (2024), p. 150-153 (cf. p. 53).

[Kansal, 2010] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari et Arka A. Bhattacharya. « Virtual machine power metering and provisioning ». *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA : Association for Computing Machinery, 2010, p. 39-50. URL : https://doi.org/10.1145/1807128.1807136 (cf. p. 48).

[Khosla, 2011] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao et Li Fei-Fei. « Novel Dataset for Fine-Grained Image Categorization ». *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO, 2011 (cf. p. 72).

[Knüpfer, 2008] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler et al. « The Vampir Performance Analysis Tool-Set ». *Parallel Tools Workshop*. 2008 (cf. p. 29).

[Koller, 2010] Ricardo Koller, Akshat Verma et Anindya Neogi. « WattApp : an application aware power meter for shared data centers ». ICAC '10. Washington, DC, USA : Association for Computing Machinery, 2010, p. 31-40. URL : https://doi.org/10.1145/1809049.1809055 (cf. p. 48).

[Koo, 1986] Richard Koo et Sam Toueg. « Checkpointing and rollback-recovery for distributed systems ». *Proceedings of 1986 ACM Fall Joint Computer Conference*. ACM '86. Dallas, Texas, USA : IEEE Computer Society Press, 1986, p. 1150-1158 (cf. p. 158).

[Krizhevsky, 2009a] A. Krizhevsky et G. Hinton. « Learning multiple layers of features from tiny images ». *Master's thesis, Department of Computer Science, University of Toronto* (2009) (cf. p. 124, 149).

[Krizhevsky, 2009b] Alex Krizhevsky et Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Rapp. tech. 0. Toronto, Ontario : University of Toronto, 2009 (cf. p. 72).

[Kumar, 2025] Iyngkarran Kumar et Sam Manning. *Trends in Frontier AI Model Count : A Forecast to 2028*. 2025. arXiv : 2504.16138 [cs.CY]. URL : https://arxiv.org/abs/2504.16138 (cf. p. 52).

[Kwon, 2022] Seokmin Kwon et Hyokyung Bahn. « Classification and Characterization of Memory Reference Behavior in Machine Learning Workloads ». *2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. 2022, p. 103-108 (cf. p. 40).

[Lacoste, 2019] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt et Thomas Dandres. « Quantifying the Carbon Emissions of Machine Learning ». *arXiv :1910.09700* (2019) (cf. p. 45).

[Lahmer, 2022] Seyyidahmed Lahmer, Aria Khoshsirat, Michele Rossi et Andrea Zanella. « Energy Consumption of Neural Networks on NVIDIA Edge Boards : an Empirical Model ». *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*. 2022, p. 365-371 (cf. p. 53).

[Lan, 2019] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma et Radu Soricut. « ALBERT : A Lite BERT for Self-supervised Learning of Language Representations ». *CoRR* abs/1909.11942 (2019). arXiv : 1909.11942. URL : http://arxiv.org/abs/1909.11942 (cf. p. 107, 124, 149).

[Lang, 2023] Matthias Lang et Ronald Hendrikx. *Energy Outlook 2023 : Energy Digitalisation*. Accessed : 2025-07-24. 2023. URL : https://www.twobirds.com/en/insights/2023/global/energy-outlook-2023-energy-digitalisation (cf. p. 7).

[Lannelongue, 2023] Loïc Lannelongue, Hans-Erik G Aronson, Alex Bateman, Ewan Birney, Talia Caplan, Martin Juckes et al. « GREENER principles for environmentally sustainable computational science ». *Nature Computational Science* 3.6 (2023), p. 514-521 (cf. p. 11).

[Lannelongue, 2021] Loïc Lannelongue, Jason Grealey et Michael Inouye. « Green Algorithms : Quantifying the Carbon Footprint of Computation ». *Advanced Science* 8.12 (2021) (cf. p. 28).

[Lazuka, 2024] Malgorzata Lazuka, Andreea Anghel et Thomas Parnell. « LLM-Pilot : Characterize and Optimize Performance of your LLM Inference Services ». *SC24 : International Conference for High Performance Computing, Networking, Storage and Analysis*. 2024, p. 1-18 (cf. p. 52).

[Leite, 2014a] Alessandro Leite, Claude Tadonki, Christine Eisenbeis et Alba de Melo. « A Fine-grained Approach for Power Consumption Analysis and Prediction ». *Procedia Computer Science (ICCS2014)* 29 (2014), p. 2260-2271 (cf. p. 55).

[Leite, 2014b] Alessandro Ferreira Leite, Tainá Raiol, Claude Tadonki, Maria Emilia MT Walter, Christine Eisenbeis et Alba Cristina Magalhaes Alves de Melo. « Excalibur : An autonomic cloud architecture for executing parallel applications ». *Proceedings of the 4th International Workshop on Cloud Data and Platforms*. 2014, p. 1-6 (cf. p. 56).

[Lenovo, 2022] Lenovo. *Optimizing Power and Energy in HPC Data Centers with Energy Aware Runtime*. Accessed : 2025-09-01. 2022. URL : https://lenovopress.lenovo.com/lp1646.pdf (cf. p. 28).

[Li, 2022] Baolin Li, Tirthak Patel, Vijay Gadepally, Karen Gettings, Siddharth Samsi et Devesh Tiwari. « DASH : Scheduling Deep Learning Workloads on Multi-Generational GPU-Accelerated Clusters ». *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 2022, p. 1-7 (cf. p. 56).

[Li, 2021] Bo Li, Xinyang Jiang, Donglin Bai, Yuge Zhang, Ningxin Zheng, Xuanyi Dong et al. « Full-Cycle Energy Consumption Benchmark for Low-Carbon Computer Vision ». *ArXiv* abs/2108.13465 (2021) (cf. p. 39).

[Li, 2025a] Haoyang Li, Fangcheng Fu, Hao Ge, Sheng Lin, Xuanyu Wang, Jiawen Niu et al. « Malleus : Straggler-Resilient Hybrid Parallel Training of Large-scale Models via Malleable Data and Model Parallelization ». *Proc. ACM Manag. Data* 3.3 (2025). URL : https://doi.org/10.1145/3725322 (cf. p. 137, 158).

[Li, 2025b] Ying Li, Yuhui Bao, Gongyu Wang, Xinxin Mei, Pranav Vaid, Anandaroop Ghosh et al. « TrioSim : A Lightweight Simulator for Large-Scale DNN Workloads on Multi-GPU Systems ». *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. ISCA '25. New York, NY, USA : Association for Computing Machinery, 2025, p. 1524-1538. URL : https://doi.org/10.1145/3695053.3731082 (cf. p. 138).

[Li, 2016] Yuanlong Li, Han Hu, Yonggang Wen et Jun Zhang. « Learning-based power prediction for data centre operations via deep neural networks ». *Proceedings of the 5th International Workshop on Energy Efficient Data Centres*. E2DC '16. Waterloo, Ontario, Canada : Association for Computing Machinery, 2016. URL : https://doi.org/10.1145/2940679.2940685 (cf. p. 49).

[Libri, 2018] Antonio Libri, Andrea Bartolini et Luca Benini. « Dwarf in a Giant : Enabling Scalable, High-Resolution HPC Energy Monitoring for Real-Time Profiling and Analytics ». *ArXiv* abs/1806.02698 (2018) (cf. p. 24).

[Lien, 2007] Chia-Hung Lien, Ying-Wen Bai et Ming-Bo Lin. « Estimation by Software for the Power Consumption of Streaming-Media Servers ». *IEEE Transactions on Instrumentation and Measurement* 56.5 (2007), p. 1859-1870 (cf. p. 48).

[Lin, 2013] Cheng-Yen Lin, Chi-Bang Kuan et Jenq Kuen Lee. « Compilers for Low Power with Design Patterns on Embedded Multicore Systems ». *2013 42nd International Conference on Parallel Processing*. 2013, p. 1052-1060 (cf. p. 55).

[Linaro, 2022] Linaro. *The Devicetree Specification*. Accessed : 2023-05-16. 2022. url : https://www.devicetree.org/specifications/ (cf. p. 27).

[Litzinger, 2023] Sebastian Litzinger. « Heuristic Scheduling of Streaming Applications for Energy Efficiency on Heterogeneous Multicores ». *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. 2023, p. 225-232 (cf. p. 56).

[Liu, 2022] Di Liu, Shi-Gui Yang, Zhenli He, Mingxiong Zhao et Weichen Liu. « CARTAD : Compiler-Assisted Reinforcement Learning for Thermal-Aware Task Scheduling and DVFS on Multicores ». *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.6 (2022), p. 1813-1826 (cf. p. 57).

[Liu, 2024] Ziyang Liu, Renyu Yang, Jin Ouyang, Weihan Jiang, Tianyu Ye, Menghao Zhang et al. « Kale : Elastic GPU Scheduling for Online DL Model Training ». *Proceedings of the 2024 ACM Symposium on Cloud Computing*. SoCC '24. Redmond, WA, USA : Association for Computing Machinery, 2024, p. 36-51. url : https://doi.org/10.1145/3698038.3698532 (cf. p. 141).

[Ljungdahl, 2022] V. Ljungdahl, M. Jradi et C. Veje. « A decision support model for waste heat recovery systems design in Data Center and High-Performance Computing clusters utilizing liquid cooling and Phase Change Materials ». *Applied Thermal Engineering* 201 (2022), p. 117671. url : https://www.sciencedirect.com/science/article/pii/S1359431121010966 (cf. p. 35).

[LLNS, 2023] LLNS. *Variorum*. Accessed : 2025-07-15. 2023. url : https://variorum.readthedocs.io/en/latest/ (cf. p. 27, 64).

[Luccioni, 2022] Alexandra Sasha Luccioni, Sylvain Viguier et Anne-Laure Ligozat. « Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model ». *ArXiv* abs/2211.02001 (2022) (cf. p. 43).

[Luccioni, 2024] Sasha Luccioni, Yacine Jernite et Emma Strubell. « Power Hungry Processing : Watts Driving the Cost of AI Deployment ? » *The 2024 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '24. ACM, 2024, p. 85-99. url : http://dx.doi.org/10.1145/3630106.3658542 (cf. p. 177).

[Ludvigsen, 2022] Kasper Groes Albin Ludvigsen. *How to estimate and reduce the carbon footprint of machine learning models*. Accessed : 2025-09-21. 2022. url : https://towardsdatascience.com/how-to-estimate-and-reduce-the-carbon-footprint-%5Cof-machine-learning-models-49f24510880 (cf. p. 45).

[Mahajan, 2020] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee et al. « Themis : Fair and Efficient GPU Cluster Scheduling ». *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA : USENIX Association, 2020, p. 289-304. url : https://www.usenix.org/conference/nsdi20/presentation/mahajan (cf. p. 135, 142).

[Mantovani, 2020] Filippo Mantovani, Marta Garcia-Gasulla, José Gracia, Esteban Stafford, Fabio Banchelli, Marc Josep-Fabregó et al. « Performance and energy consumption of HPC workloads on a cluster based on Arm ThunderX2 CPU ». *ArXiv* abs/2007.04868 (2020) (cf. p. 32).

[Marvell, 2019a] Marvell. *Manufacturing Applications on Marvell ThunderX2 : Withe paper*. Accessed : 2025-09-21. 2019. url : https://www.marvell.com/content/dam/marvell/en/products/assets/server-processors/thunderx2-arm-processors/pdf/ (cf. p. 25).

[Marvell, 2019b] Marvell. *tx2mon*. Accessed : 2023-05-16. 2019. url : https://github.com/jchandra-cavm/tx2mon (cf. p. 25).

[McCraw, 2014] Heike McCraw, James Ralph, Anthony Danalis et Jack Dongarra. « Power monitoring with PAPI for extreme scale architectures and dataflow-based programming models ». *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 2014, p. 385-391 (cf. p. 23, 27, 49, 209).

[McCullough, 2011] John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren et Rajesh K. Gupta. « Evaluating the effectiveness of model-based power characterization ». *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'11. Portland, OR : USENIX Association, 2011, p. 12 (cf. p. 49, 52).

[Merity, 2016] Stephen Merity, Caiming Xiong, James Bradbury et Richard Socher. *Pointer Sentinel Mixture Models*. 2016. arXiv : 1609.07843 [cs.CL] (cf. p. 124).

[Merkel, 2020] Cory Merkel. « Exploring Energy-Accuracy Tradeoffs in AI Hardware ». *2020 11th International Green and Sustainable Computing Workshops (IGSC)*. 2020, p. 1-7 (cf. p. 53).

[Meyer, 2013] Nils Meyer, Manfred Ries, Stefan Solbrig et Tilo Wettig. « iDataCool : HPC with Hot-Water Cooling and Energy Reuse ». *Inform. Security Conf.* 2013 (cf. p. 9, 35).

[Meyers, 2005] S. Meyers. *Effective C++ : 55 Specific Ways to Improve Your Programs and Designs*. Addison-Wesley Professional Computing Series. Pearson Education, 2005. URL : https://books.google.fr/books?id=Qx5oyB49poYC (cf. p. 55).

[Micikevicius, 2018] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia et al. *Mixed Precision Training*. 2018. arXiv : 1710.03740 [cs.AI] (cf. p. 107).

[Molka, 2010] Daniel Molka, Daniel Hackenberg, Robert Schöne et Matthias S. Müller. « Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors ». *Intl. Conference on Green Computing* (2010) (cf. p. 52).

[Morán, 2020] Marina Morán, Javier Balladini, Dolores Rexachs et Enzo Rucci. « Towards Management of Energy Consumption in HPC Systems with Fault Tolerance ». 2020, p. 1-8 (cf. p. 55).

[Nabavinejad, 2021] Seyed Morteza Nabavinejad, Sherief Reda et Masoumeh Ebrahimi. « BatchSizer : Power-Performance Trade-off for DNN Inference ». *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2021, p. 819-824 (cf. p. 52).

[Nana, 2024a] Roblex Nana, Claude Tadonki, Petr Dokládal et Youssef Mesri. « Power Consumption in HPC-AI Systems ». *Artificial Intelligence and High Performance Computing in the Cloud*. T. 1220. Lecture Notes in Networks and Systems. Springer Nature Switzerland, 2024, p. 89-116. URL : https://hal-ciheam.iamm.fr/hal-04861732 (cf. p. 2).

[Nana, 2024b] Tchakoute Roblex Nana et Claude Tadonki. « Experimental Study of Power Consumption of Basic Parallel Programs ». *2024 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*. 2024, p. 33-41 (cf. p. 84).

[Nana, 2024c] Tchakoute Roblex Nana, Claude Tadonki, Petr Dokladal et Youssef Mesri. « A Flexible Operational Framework for Energy Profiling of Programs ». *2024 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*. 2024, p. 12-22 (cf. p. 64, 81, 125).

[Nardin, 2022] Igor Fontana de Nardin, Patricia Stolf et Stephane Caux. « Evaluation of Heuristics to Manage a Data Center Under Power Constraints ». *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*. 2022, p. 1-8 (cf. p. 56).

[Nasrin, 2022] Shamma Nasrin, Ahish Shylendra, Nastaran Darabi, Theja Tulabandhula, Wilfred Gomes, Ankush Chakrabarty et al. « ENOS : Energy-Aware Network Operator Search in Deep Neural Networks ». *IEEE Access* 10 (2022), p. 81447-81457 (cf. p. 52, 57).

[Nonaka, 2020] Jorji Nonaka, Toshihiro Hanawa et Fumiyoshi Shoji. « Analysis of Cooling Water Temperature Impact on Computing Performance and Energy Consumption ». *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 2020, p. 169-175 (cf. p. 9, 35).

[NVIDIA, 2016] NVIDIA. *nvidia-smi - NVIDIA System Management Interface program*. Accessed : 2023-04-21. 2016. URL : https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf (cf. p. 25, 26).

[NVIDIA, 2020] NVIDIA. *NVIDIA A100 Tensor Core GPU Architecture - Whitepaper*. Accessed : 2025-07-25. 2020. URL : https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf (cf. p. 109, 110).

[NVIDIA, 2022] NVIDIA. *NVIDIA H100 Tensor Core GPU*. Accessed : 2025-07-25. 2022. URL : https://www.nvidia.com/en-us/data-center/h100/ (cf. p. 31).

[Nvidia, 2023] Nvidia. *NVIDIA Jetson AGX Orin for Robotics and Edge AI Applications*. Accessed : 2025-09-01. 2023. URL : https://www.nvidia.com/en-us/lp/embedded-computing/robotics-edge-ai-tech-brief/ (cf. p. 33).

[Nvidia, 2024] NVIDIA. *Introduction to DGX H100*. Accessed : 2025-09-21. 2024. URL : https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html (cf. p. 36, 40).

[Nvidia, 2025a] NVIDIA. *NVIDIA Nsight Systems*. Accessed : 2025-09-24. 2025. URL : https://developer.nvidia.com/nsight-systems (cf. p. 209).

[Nvidia, 2025b] NVIDIA. *The NVIDIA Blackwell Architecture : White Paper*. Accessed : 2025-09-24. 2025. URL : https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf (cf. p. 199).

[Oleynik, 2015] Yury OLEYNIK, Michael GERNDT, Joseph SCHUCHART, Per Gunnar KJELDSBERG et Wolfgang E. NAGEL. « Run-Time Exploitation of Application Dynamism for Energy-Efficient Exascale Computing (READEX) ». *IEEE 18th International Conference on Computational Science and Engineering* (2015), p. 347-350 (cf. p. 28).

[Orgerie, 2021] Anne-Cécile ORGERIE. *Measuring the Energy Consumption of HPC Systems*. Accessed : 2025-09-01. 2021. URL : https://ecoinfo.cnrs.fr/wp-content/uploads/2021/12/ORAP-2021-Orgerie.pdf (cf. p. 25).

[ourworldindataorg, 2025] OURWORLDINDATA.ORG. *Carbon intensity of electricity generation*. Accessed : 2025-09-01. 2025. URL : https://ourworldindata.org/electricity-mix (cf. p. 21).

[Pambudi, 2022] Nugroho Agung PAMBUDI, Alfan SARIFUDIN, Ridho Alfan FIRDAUS, Desita Kamila ULFA, Indra Mamad GANDIDI et Rahmat ROMADHON. « The immersion cooling technology : Current and future development in energy saving ». *Alexandria Engineering Journal* 61.12 (2022), p. 9509-9527. URL : https://www.sciencedirect.com/science/article/pii/S1110016822001557 (cf. p. 9, 35).

[Pandey, 2021] Pramesh PANDEY, Noel Daniel GUNDI, Koushik CHAKRABORTY et Sanghamitra ROY. « UPTPU : Improving Energy Efficiency of a Tensor Processing Unit through Underutilization Based Power-Gating ». *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, p. 325-330 (cf. p. 31).

[Park, 2022] S. PARK et H. BAHN. « Memory Access Characteristics of Neural Network Workloads and Their Implications ». *2022 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. Los Alamitos, CA, USA : IEEE Computer Society, 2022, p. 1-6 (cf. p. 40).

[Patterson, 2022] David PATTERSON, Joseph GONZALEZ, Urs HÖLZLE, Quoc LE, Chen LIANG, Lluis-Miquel MUNGUIA et al. « The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink ». *Computer* 55.7 (2022), p. 18-28 (cf. p. 21, 43, 45).

[Patterson, 2021] David A. PATTERSON, Joseph GONZALEZ, Quoc V. LE, Chen LIANG, Lluís-Miquel MUNGUÍA, Daniel ROTHCHILD et al. « Carbon Emissions and Large Neural Network Training ». *ArXiv* abs/2104.10350 (2021) (cf. p. 44, 45).

[Paul, 2017] Alcorn PAUL. *Intel Unveils Movidius Myriad X Vision Processing Unit*. Accessed : 2025-09-01. 2017. URL : https://www.tomshardware.com/news/intel-movidius-vpu-ai-inference,35327.html (cf. p. 33).

[Pedram, 2010] Massoud PEDRAM et Inkwon HWANG. « Power and Performance Modeling in a Virtualized Server System ». *2010 39th International Conference on Parallel Processing Workshops*. 2010, p. 520-526 (cf. p. 48).

[Peng, 2021] Yanghua PENG, Yixin BAO, Yangrui CHEN, Chuan WU, Chen MENG et Wei LIN. « DL2 : A Deep Learning-Driven Scheduler for Deep Learning Clusters ». *IEEE Transactions on Parallel and Distributed Systems* 32.8 (2021), p. 1947-1960 (cf. p. 135).

[Pereira, 2017] Rui PEREIRA, Marco COUTO, Francisco RIBEIRO, Rui RUA, Jácome CUNHA, João Paulo FERNANDES et al. « Energy Efficiency across Programming Languages : How Do Energy, Time, and Memory Relate ? » *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2017. Vancouver, BC, Canada : Association for Computing Machinery, 2017, p. 256-267. URL : https://doi.org/10.1145/3136014.3136031 (cf. p. 55).

[Poddar, 2025] Soham PODDAR, Paramita KOLEY, Janardan MISRA, Niloy GANGULY et Saptarshi GHOSH. *Towards Sustainable NLP : Insights from Benchmarking Inference Energy in Large Language Models*. 2025. arXiv : 2502.05610 [cs.CL]. URL : https://arxiv.org/abs/2502.05610 (cf. p. 52).

[Qi, 2017] Hang QI, Evan R. SPARKS et Ameet TALWALKAR. « Paleo : A Performance Model for Deep Neural Networks ». *Proceedings of the International Conference on Learning Representations*. 2017 (cf. p. 52).

[Qi, 2023] Xinxin QI, Juan CHEN, Yong DONG, Yuan YUAN, Tao XU, Rongyu DENG et al. « HighRPM : Combining Integrated Measurement and Sofware Power Modeling for High-Resolution Power Monitoring ». *Proceedings of the 52nd International Conference on Parallel Processing*. ICPP '23. Salt Lake City, UT, USA : Association for Computing Machinery, 2023, p. 369-379. URL : https://doi.org/10.1145/3605573.3605649 (cf. p. 53).

[Qiao, 2021] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang et al. « Pollux : Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning ». *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 2021, p. 1-18. URL : https://www.usenix.org/conference/osdi21/presentation/qiao (cf. p. 134, 136, 137, 142, 163).

[Qiu, 2022] Xinchi Qiu, Titouan Parcollet, Javier Fernandez-Marques, Pedro Porto Buarque de Gusmao, Yan Gao, Daniel J. Beutel et al. *A first look into the carbon footprint of federated learning*. 2022. arXiv : 2102.07627 [cs.LG] (cf. p. 44).

[Qureshi, 2009] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag et Bruce Maggs. « Cutting the electric bill for internet-scale systems ». *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. SIGCOMM '09. Barcelona, Spain : Association for Computing Machinery, 2009, p. 123-134. URL : https://doi.org/10.1145/1592568.1592584 (cf. p. 47, 48).

[Radford, 2018] Alec Radford et Karthik Narasimhan. « Improving Language Understanding by Generative Pre-Training ». *ArXiv*. 2018 (cf. p. 42).

[Raghavendra, 2008] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang et Xiaoyun Zhu. « No "power" struggles : coordinated multi-level power management for the data center ». *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XIII. Seattle, WA, USA : Association for Computing Machinery, 2008, p. 48-59. URL : https://doi.org/10.1145/1346281.1346289 (cf. p. 48).

[Ramesh, 2022] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu et Mark Chen. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv : 2204.06125 [cs.CV]. URL : https://arxiv.org/abs/2204.06125 (cf. p. 44).

[Rashti, 2015] Mohammad Rashti, Gerald Sabin, David Vansickle et Boyana Norris. « WattProf : A Flexible Platform for Fine-Grained HPC Power Profiling ». *2015 IEEE International Conference on Cluster Computing*. 2015, p. 698-705 (cf. p. 24).

[RaspberryPi, 2020] RaspberryPi. *Raspberry Pi 4 model B*. Accessed : 2025-09-01. 2020. URL : https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (cf. p. 33).

[Raycroft, 2014] Patrick Raycroft, Ryan Jansen, Mateusz Jarus et Paul R. Brenner. « Performance bounded energy efficient virtual machine allocation in the global cloud ». *Sustainable Computing : Informatics and Systems* 4.1 (2014), p. 1-9. URL : https://www.sciencedirect.com/science/article/pii/S2210537913000401 (cf. p. 47).

[Reghenzani, 2020] Federico Reghenzani, Giuseppe Massari et William Fornaciari. « Timing Predictability in High-Performance Computing With Probabilistic Real-Time ». *IEEE Access* 8 (2020), p. 208566-208582 (cf. p. 31).

[Ren, 2021] Jie Ren, Rajbhandari Samyam, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang et al. « ZeRO-Offload : Democratizing Billion-Scale Model Training ». *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. Sous la dir. d'Irina Calciu et Geoff Kuenning. USENIX Association, 2021, p. 551-564. URL : https://www.usenix.org/conference/atc21/presentation/ren-jie (cf. p. 196, 198).

[Reynolds, 2008] Douglas Reynolds. « Gaussian Mixture Models ». *Encyclopedia of Biometrics* (2008) (cf. p. 49).

[Rodrigues, 2017] Crefeda Faviola Rodrigues, Graham Riley et Mikel Luján. « Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1 ». *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 2017, p. 114-115 (cf. p. 53).

[Ronneberger, 2015] O. Ronneberger, P.Fischer et T. Brox. « U-Net : Convolutional Networks for Biomedical Image Segmentation ». *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. T. 9351. LNCS. Springer, 2015, p. 234-241 (cf. p. 42).

[Sai, 2013] Manoj P. D. Sai, Kanwen Wang et Hao Yu. « Peak power reduction and workload balancing by space-time multiplexing based demand-supply matching for 3D thousand-core microprocessor ». *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2013, p. 1-6 (cf. p. 56).

[Samsung, 2022] Samsung. *Here To Upgrade the World : Introducing Samsung's Game-Changing DDR5 Solution*. Accessed : 2024-08-29. 2022. URL : https://semiconductor.samsung.com/news-events/tech-blog/here-to-upgrade-the-world-introducing-samsungs-game-changing-ddr5-solution/ (cf. p. 69, 104).

[Sanh, 2019] Victor Sanh, Lysandre Debut, Julien Chaumond et Thomas Wolf. « DistilBERT, a distilled version of BERT : smaller, faster, cheaper and lighter ». *ArXiv :1910.01108* (2019) (cf. p. 44, 57, 124).

[Sanjeevi, 2017] P. Sanjeevi et Viswanathan Perumal. « Workload consolidation techniques to optimise energy in cloud : Review ». *International Journal of Internet Protocol Technology* 10 (2017), p. 115 (cf. p. 56).

[Schaller, 1997] Robert R. Schaller. « Moore's law : past, present, and future ». *IEEE Spectr.* 34.6 (1997), p. 52-59. URL : https://doi.org/10.1109/6.591665 (cf. p. 9, 191).

[Schirrmeister, 2023] Frank Schirrmeister. *Design Complexity In The Golden Age Of Semiconductors*. Accessed : 2025-09-24. 2023. URL : https://semiengineering.com/design-complexity-in-the-golden-age-of-semiconductors/ (cf. p. 192).

[Schmidt, 2009] Roger R. Schmidt et Madhusudan K. Iyengar. « Server Rack Rear Door Heat Exchanger and the New ASHRAE Recommended Environmental Guidelines ». 2009 (cf. p. 9, 35).

[Schölkopf, 2001] Bernhard Schölkopf et Alexander J. Smola. *Learning with Kernels : Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001. URL : https://doi.org/10.7551/mitpress/4175.001.0001 (cf. p. 49).

[Schöne, 2021] Robert Schöne, Thomas Ilsche, Mario Bielert, Markus Velten, Markus Schmidl et Daniel Hackenberg. « Energy Efficiency Aspects of the AMD Zen 2 Architecture ». *IEEE Int. Conference on Cluster Computing (CLUSTER)* (2021), p. 562-571 (cf. p. 25, 27).

[Schuchart, 2017] Joseph Schuchart, Michael Gerndt, Per Gunnar Kjeldsberg, Michael Lysaght, David Hořák, Lubomír Říha et al. « The READEX formalism for automatic tuning for energy efficiency ». *Computing* 99 (2017), p. 727-745 (cf. p. 28).

[scientific-computingcom, 2019] scientific-computing.com. *Cooling technology options for HPC*. Accessed : 2025-09-21. 2019. URL : https://www.scientific-computing.com/feature/cooling-technology-options-hpc (cf. p. 35).

[Scipy-Docs, 2008] Scipy-Docs. *scipy.optimize.differential_evolution*. Accessed : 2025-06-25. 2008. URL : https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html (cf. p. 122).

[Scott, 2011] Huck Scott. *Measuring Processor Power TDP vs ACP : withe paper*. Accessed : 2023-04-21. 2011. URL : https://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf (cf. p. 22).

[Seeedstudio, 2020] Seeedstudio. *NVIDIA Jetson Nano module Product details*. Accessed : 2025-09-01. 2020. URL : https://www.seeedstudio.com/NVIDIAr-Jetson-Nanotm-Developer-Kit-p-2916.html (cf. p. 33).

[Sharma, 2019] Neeraj Kumar Sharma et G. Ram Mohana Reddy. « Multi-Objective Energy Efficient Virtual Machines Allocation at the Cloud Data Center ». *IEEE Transactions on Services Computing* 12.1 (2019), p. 158-171 (cf. p. 47).

[Shuja, 2016] Junaid Shuja, Kashif Bilal, Sajjad A. Madani, Mazliza Othman, Rajiv Ranjan, Pavan Balaji et al. « Survey of Techniques and Architectures for Designing Energy-Efficient Data Centers ». *IEEE Systems Journal* 10.2 (2016), p. 507-519 (cf. p. 53).

[Simonyan, 2015a] Karen Simonyan et Andrew Zisserman. « Very Deep Convolutional Networks for Large-Scale Image Recognition ». *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Sous la dir. d'Yoshua Bengio et Yann LeCun. 2015. URL : http://arxiv.org/abs/1409.1556 (cf. p. 42, 72).

[Simonyan, 2015b] Karen Simonyan et Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv : 1409.1556 [cs.CV]. URL : https://arxiv.org/abs/1409.1556 (cf. p. 124).

[SimPy, 2002] SimPy. *SimPy : Discrete event simulation for Python*. Accessed : 2025-08-13. 2002. URL : https://simpy.readthedocs.io/en/latest/ (cf. p. 139).

[Singh, 2017] Amit Kumar Singh, Charles Leech, Basireddy Karunakar Reddy, Bashir M. Al-Hashimi et Geoff V. Merrett. « Learning-Based Run-Time Power and Energy Management of Multi/Many-Core Systems : Current and Future Trends ». *J. Low Power Electron.* 13 (2017), p. 310-325. URL : https://api.semanticscholar.org/CorpusID:7727571 (cf. p. 56).

[Skhynix, 2021] Skhynix. *256GB DDR4 RDIMM/LRDIMM Ultra-high capacities at industry's lowest power budget*. Accessed : 2024-08-29. 2021. URL : https://product.skhynix.com/products/dram/module/ddr4dm.go (cf. p. 34, 104).

[Socher, 2013] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng et al. « Recursive deep models for semantic compositionality over a sentiment treebank ». *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, p. 1631-1642 (cf. p. 124).

[Storn, 1997] Rainer Storn et Kenneth Price. « Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces ». *J. of Global Optimization* 11.4 (1997), p. 341-359. URL : https://doi.org/10.1023/A:1008202821328 (cf. p. 122).

[Strubell, 2019] Emma Strubell, Ananya Ganesh et Andrew Mccallum. « Energy and Policy Considerations for Deep Learning in NLP ». 2019, p. 3645-3650 (cf. p. 39, 44).

[Sueur, 2010] Etienne Le Sueur et Gernot Heiser. « Dynamic voltage and frequency scaling : the laws of diminishing returns ». 2010 (cf. p. 55).

[Tadonki, 2013] Claude Tadonki. « High performance computing as a combination of machines and methods and programming ». Thèse de doct. Université Paris Sud-Paris XI, 2013 (cf. p. 2).

[Tadonki, 2023] Claude Tadonki. « What Do HPC Applications Look Like ? » *High Performance Computing in Clouds : Moving HPC Applications to a Scalable and Cost-Effective Environment*. Cham : Springer International Publishing, 2023, p. 27-51. URL : https://doi.org/10.1007/978-3-031-29769-4%5C_3 (cf. p. 1, 2, 40).

[Tan, 2019] Mingxing Tan et Quoc V. Le. « EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks ». *ArXiv* abs/1905.11946 (2019) (cf. p. 42).

[Technology, 2024] Micron Technology. *How Much Power Does Memory Use ?* Accessed :2024-08-29. 2024. URL : https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use (cf. p. 104).

[Ting, 2021] Yu-Sheng Ting, Yu-Fan Teng et Tzi-Dar Chiueh. « Batch Normalization Processor Design for Convolution Neural Network Training and Inference ». *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2021, p. 1-4 (cf. p. 42).

[Toosi, 2022] Adel N. Toosi, Chayan Agarwal, Lena Mashayekhy, Sara K. Moghaddam, Redowan Mahmud et Zahir Tari. « GreenFog : A Framework for Sustainable Fog Computing ». *Service-Oriented Computing : 20th International Conference, ICSOC 2022, Seville, Spain, November 29 – December 2, 2022, Proceedings*. Seville, Spain : Springer-Verlag, 2022, p. 540-549. URL : https://doi.org/10.1007/978-3-031-20984-0_38 (cf. p. 56).

[Top500, 2025] Top500. *TOP500*. Accessed : 2025-07-15. 2025. URL : https://top500.org/ (cf. p. 19, 21, 32).

[Treibig, 2010a] Jan Treibig, Georg Hager et Gerhard Wellein. « LIKWID : A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments ». *39th Int. Conf. on Parallel Processing Workshops*. 2010, p. 207-216 (cf. p. 27, 64, 209).

[Treibig, 2010b] Jan Treibig, Georg Hager et Gerhard Wellein. « LIKWID : A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments ». *2010 39th International Conference on Parallel Processing Workshops*. 2010, p. 207-216 (cf. p. 80).

[UEFI, 2021] UEFI. *Advanced Configuration and Power Interface (ACPI) Specification, January 2021*. Accessed : 2023-05-16. 2021. URL : https://uefi.org/htmlspecs/ACPI_Spec_6_4_html/ (cf. p. 27).

[UpbeatLabs, 2023] UpbeatLabs. *Dr. Wattson Energy Monitoring Module for Arduino, Raspberry Pi and other Maker-Friendly Microcontrollers*. https://www.upbeatlabs.com/wattson/. Accessed : 2025-05-23. 2023 (cf. p. 24, 33).

[Vaddina, 2021] Kameswar Rao Vaddina, Laurent Lefèvre et Anne-Cécile Orgerie. « Experimental Workflow for Energy and Temperature Profiling on HPC Systems ». *IEEE Symposium on Computers and Communications, ISCC 2021, Athens, Greece, September 5-8, 2021*. IEEE, 2021, p. 1-7. URL : https://doi.org/10.1109/ISCC53001.2021.9631413 (cf. p. 56, 57).

[Valeye, 2022] Florian Valeye. *Tracarbon — Track your device's carbon footprint*. Accessed : 2025-09-21. 2022. URL : https://medium.com/@florian.valeye/tracarbon-track-your-devices-carbon-footprint-fb051fcc9009 (cf. p. 28, 64).

[Vallabhajosyula, 2023] Manikya Swathi Vallabhajosyula et Rajiv Ramnath. « Insights from the HARP Framework : Using an AI-Driven Approach for Efficient Resource Allocation in HPC Scientific Workflows ». *Practice and Experience in Advanced Research Computing 2023 : Computing for the Common Good*. PEARC '23. Portland, OR, USA : Association for Computing Machinery, 2023, p. 341-344. URL : https://doi.org/10.1145/3569951.3597595 (cf. p. 6).

[Vaswani, 2017a] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez et al. « Attention is All you Need ». *Advances in Neural Information Processing Systems*. Sous la dir. d'I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et al. T. 30. Curran Associates, Inc., 2017 (cf. p. 42).

[Vaswani, 2017b] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez et al. « Attention is all you need ». *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA : Curran Associates Inc., 2017, p. 6000-6010 (cf. p. 114).

[Verma, 2015] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune et John Wilkes. « Large-scale cluster management at Google with Borg ». *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France, 2015 (cf. p. 141).

[Vysocky, 2017] Ondrej Vysocky, Martin Beseda, Lubomír Říha, Jan Zapletal, Michael Lysaght et Venkatesh Kannan. « MERIC and RADAR Generator : Tools for Energy Evaluation and Runtime Tuning of HPC Applications ». *International Conference on High Performance Computing in Science and Engineering*. 2017 (cf. p. 28).

[Wallossek, 2014] Igor Wallossek et Chris Angelini. *Measuring DDR4 Power Consumption*. Accessed : 2024-08-29. 2014. URL : https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html (cf. p. 104, 105).

[Wang, 2019a] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy et Samuel R. Bowman. « GLUE : A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding ». In the Proceedings of ICLR. 2019 (cf. p. 124).

[Wang, 2019b] Dilin Wang, Meng Li, Lemeng Wu, Vikas Chandra et Qiang Liu. « Energy-Aware Neural Architecture Optimization with Fast Splitting Steepest Descent ». *arXiv preprint arXiv :1910.03103* (2019) (cf. p. 52, 57).

[Wang, 2020] Yunsong Wang, Charlene Yang, Steven Farrell, Yan Zhang, Thorsten Kurth et Samuel Williams. « Time-Based Roofline for Deep Learning Performance Analysis ». *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*. 2020, p. 10-19 (cf. p. 52).

[Wang, 2010] Zhikui Wang, Niraj Tolia et Cullen Bash. « Opportunities and challenges to unify workload, power, and cooling management in data centers ». *SIGOPS Oper. Syst. Rev.* 44.3 (2010), p. 41-46. URL : https://doi.org/10.1145/1842733.1842741 (cf. p. 48).

[Wikichip, 2021] Wikichip. *EPYC 7513 - AMD*. Accessed : 2024-05-07. 2021. URL : https://en.wikichip.org/wiki/amd/epyc/7513 (cf. p. 81).

[Wikichiporg, 2020] Wikichip.org. *Astra - Supercomputers*. Accessed : 2023-05-16. 2020. URL : https://en.wikichip.org/wiki/supercomputers/astra (cf. p. 33).

[Wikimedia, 2024] Wikimedia. *File :Cpu-gpu.svg*. Accessed : 2025-09-24. 2024. URL : https://commons.wikimedia.org/wiki/File:Cpu-gpu.svg (cf. p. 195).

[Wilde, 2013] Torsten Wilde, Axel Auweter et Hayk Shoukourian. « The 4 Pillar Framework for energy efficient HPC data centers ». *Computer Science - Research and Development* 29 (2013) (cf. p. 10).

[Williams, 2009] Samuel Williams, Andrew Waterman et David Patterson. « Roofline : an insightful visual performance model for multicore architectures ». *Commun. ACM* 52.4 (2009), p. 65-76. URL : https://doi.org/10.1145/1498765.1498785 (cf. p. 51, 107).

[Witkowski, 2013] M. Witkowski, A. Oleksiak, T. Piontek et J. Wundefinedglarz. « Practical power consumption estimation for real life HPC applications ». *Future Gener. Comput. Syst.* 29.1 (2013), p. 208-217. URL : https://doi.org/10.1016/j.future.2012.06.003 (cf. p. 50).

[Wolff, 1982] Ronald W. Wolff. « Poisson Arrivals See Time Averages ». *Operations Research* 30.2 (1982), p. 223-231. URL : https://ideas.repec.org/a/inm/oropre/v30y1982i2p223-231.html (cf. p. 150).

[Wu, 2021] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng et al. « Sustainable AI : Environmental Implications, Challenges and Opportunities ». *arXiv :2111.00364* (2021) (cf. p. 39, 44, 45).

[Wu, 2019] Lemeng Wu, Dilin Wang et Qiang Liu. « Splitting steepest descent for growing neural architectures ». *Advances in Neural Information Processing Systems*. 2019, p. 10655-10665 (cf. p. 52, 57).

[XENON, 2023] Systems XENON. *Liquid Cooling : Exceeding the Limits of Air Cooling to Unlock Greater Potential in HPC*. Accessed :2025-09-21. 2023. URL : https://xenon.com.au/white-papers/liquid-cooling-exceeding-the-limits-of-air-cooling-to-unlock-%5Cgreater-potential-in-hpc/ (cf. p. 9, 35).

[Xu, 2025] Kaiqiang Xu, Decang Sun, Han Tian, Junxue Zhang et Kai Chen. « GREEN : Carbon-efficient Resource Scheduling for Machine Learning Clusters ». *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. Philadelphia, PA : USENIX Association, 2025, p. 999-1014. URL : https://www.usenix.org/conference/nsdi25/presentation/xu-kaiqiang (cf. p. 135).

[Xu, 2023] Yi Xu, Silverio Martinez-Fernandez, Matias Martinez et Xavier Franch. « Energy Efficiency of Training Neural Network Architectures : An Empirical Study ». *ArXiv* abs/2302.00967 (2023) (cf. p. 42).

[Yang, 2021] Charlene Yang, Yunsong Wang, Thorsten Kurth, Steven Farrell et Samuel Williams. « Hierarchical Roofline Performance Analysis for Deep Learning Applications ». *Intelligent Computing*. Sous la dir. de Kohei Arai. Cham : Springer International Publishing, 2021, p. 473-491 (cf. p. 107).

[Yang, 2016] Tien-Ju Yang, Yu-hsin Chen et Vivienne Sze. « Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning ». *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), p. 6071-6079 (cf. p. 42, 57).

[You, 2023] Jie You, Jae-Won Chung et Mosharaf Chowdhury. « Zeus : Understanding and Optimizing GPU Energy Consumption of DNN Training ». *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA : USENIX Association, 2023, p. 119-139. URL : https://www.usenix.org/conference/nsdi23/presentation/you (cf. p. 141).

[Yu, 2020] Zheqi Yu, Pedro Machado, Adnan Zahid, Amir M. Abdulghani, Kia Dashtipour, Hadi Heidari et al. « Energy and Performance Trade-Off Optimization in Heterogeneous Computing via Reinforcement Learning ». *Electronics* 9 (2020), p. 1812. URL : https://api.semanticscholar.org/CorpusID:228816139 (cf. p. 56).

[Zhang, 2024] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang et Wei Lu. *TinyLlama : An Open-Source Small Language Model*. 2024. arXiv : 2401.02385 [cs.CL]. URL : https://arxiv.org/abs/2401.02385 (cf. p. 124).

[Zhang, 2019] Qingchen Zhang, Man Lin, Laurence T. Yang, Zhikui Chen, Samee U. Khan et Peng Li. « A Double Deep Q-Learning Model for Energy-Efficient Edge Scheduling ». *IEEE Transactions on Services Computing* 12.5 (2019), p. 739-749 (cf. p. 57).

[Zhang, 2013] Xiao Zhang, Jian-Jun Lu, Xiao Qin et Xiao-Nan Zhao. « A high-level energy consumption model for heterogeneous data centers ». *Simulation Modelling Practice and Theory* 39 (2013). S.I.Energy efficiency in grids and clouds, p. 41-55. URL : https://www.sciencedirect.com/science/article/pii/S1569190X13000853 (cf. p. 48).

[Zhao, 2022] Dan Zhao, Nathan C. Frey, Joseph McDonald, Matthew Hubbell, David Bestor, Michael Jones et al. « A Green(er) World for A.I. » *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2022, p. 742-750 (cf. p. 44, 45).

[Zheng, 2023] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman et Aditya Akella. « Shockwave : Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning ». *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA : USENIX Association, 2023, p. 703-723. URL : https://www.usenix.org/conference/nsdi23/presentation/zheng (cf. p. 158).

RÉSUMÉ

La convergence du calcul haute performance (HPC) et de l'intelligence artificielle (IA) est marquée par une demande en puissance de calcul sans précédent, portée par la croissance fulgurante des modèles d'apprentissage profond (Deep Learning, DL). Toutefois, le déclin des paradigmes classiques de passage à l'échelle, tels que la loi de Moore ou ou celle de Dennard, a déplacé le défi de la conception des systèmes de nouvelle génération : il ne s'agit plus seulement d'atteindre des performances maximales, mais également de garantir une efficacité énergétique accrue. En effet, la consommation électrique massive des infrastructures hétérogènes modernes constitue désormais une contrainte majeure, imposant une transition vers des approches de calcul plus durables.

Les approches actuelles présentent encore des limites, notamment le manque d'outils permettant une mesure fine et détaillée de l'énergie des systèmes. Pour faire face à celà, cette thèse propose un cadre méthodologique intégré, couvrant l'ensemble de la chaîne — analyse, modélisation et optimisation — afin d'améliorer l'efficacité énergétique des applications en IA. Notre démarche repose sur un cycle vertueux articulé autour de quatre étapes : « Mesurer, Comprendre, Modéliser et Optimiser », pour lesquelles nous avons conçu un ensemble cohérent d'outils et de stratégies innovantes.

Afin de permettre une analyse empirique robuste, nous introduisons d'abord un outil flexible et multiplateforme permettant un profilage énergétique fin. Ensuite, pour passer de la simple observation à la prédiction, nous développons un cadre analytique complet pour modéliser la performance et la consommation énergétique de l'entraînement de modèles de DL. Enfin, afin de transformer les prédictions en leviers concrets d'optimisation, nous proposons une méthodologie systématique de co-conception de planificateurs de tâches conscients des contraintes énergétiques.

En somme, cette thèse met à disposition un ensemble d'outils ainsi qu'une méthodologie structurée qui permettent de faire passer l'optimisation énergétique, d'une démarche ponctuelle et empirique à un processus systématique. Nous espérons ainsi contribuer à franchir une étape significative vers la conception et l'exploitation d'infrastructures d'IA plus intelligentes, plus efficaces et plus durables.

MOTS CLÉS

Efficacité énergétique, consommation électrique, planification des tâches, modélisation GPU, apprentissage profond, HPC-IA, profilage énergétique

ABSTRACT

The convergence of High-Performance Computing (HPC) and Artificial Intelligence (AI) has led to an era of unprecedented computational demand, driven by the explosive growth of Deep Learning (DL) models. However, the demise of traditional scaling paradigms like Moore's Law and Dennard scaling has extended the challenge in designing next-generation systems from raw performance to energy efficiency. The massive power consumption of modern heterogeneous infrastructures now represents a first-order constraint, thus necessitating a fundamental shift towards sustainable computing. State-of-the-art approaches often lack necessary tools for fine-grained energy measurement across heterogeneous components. To tackle this issue, this dissertation presents an integrated end-to-end methodological framework for the analysis, modeling, and optimization of energy efficiency of AI workloads. Our approach follows a virtuous cycle of "Measure, Understand, Model, and Optimize", for which we have developed an operational suite of novel tools and strategies.

Firstly, to enable robust empirical analysis, we introduce a flexible and multi-platform tool for fine-grained energy profiling. Secondly, to move from observation to prediction, we developed a comprehensive analytical framework to model the performance and energy of DL training on GPUs. Finally, to translate energy prediction into active optimization, we present a systematic methodology for the co-design of energy-aware job schedulers.

In summary, this thesis provides a set of tools and a structured engineering methodology that transform energy optimization from an ad-hoc practice into a systematic process. We hope that through our contributions, we have made a significant step towards the design and operation of more intelligent, efficient, and sustainable infrastructures for the future of AI.

KEYWORDS

Energy efficiency, power consumption, job scheduling, GPU modeling, Deep learning training, HPC-AI, energy profiling