Observing and Understanding Green Software Ecosystems

Adel Noureddine

Observing and Understanding Green Software Ecosystems

Habilitation à Diriger des Recherches

Université de Pau et des Pays de l'Adour

Spécialité : Informatique

Adel Noureddine

Soutenue le 13 février 2024

Composition du jury

Rapporteurs :
Mathieu ACHER, Professeur des Universités
INSA Rennes, Institut Universitaire de France (IUF), France
Jean-Marc MENAUD, Professeur des Universités
IMT Atlantique, Nantes, France
Chantal TACONET, Maître de Conférences HDR
Télécom SudParis, France

Examinateurs :
Rabih BASHROUSH, Professeur des Universités
University of East London, Royaume-Uni
Christophe CHASSOT, Professeur des Universités
Université de Toulouse, France
Khalil DRIRA, Directeur de Recherche
CNRS, Universté de Toulouse, France
Ernesto EXPOSITO, Professeur des Universités
Université de Pau et des Pays de l'Adour, France
Florence MARANINCHI, Professeur des Universités
Université Grenoble Alpes, France

To Dad and Mom

Contents

Chapter 1

Introduction

Contents

I've always wanted to be an academic researcher. Since high school, I questioned which path I wanted to go on, whether exploring history, computer science or being a writer. The young boy who likes playing video games on the Sega Master System II and on DOS/Windows computers, finally settled on pursuing a computer science degree. At university, my journey into research was ultimately due to a crucial week in 2009, where I had made the choice to continue in a research master thesis, even though the temptation to go a different path, at home and abroad, were high.

My master thesis defended in 2010 after an internship at Inria's ADAM research team (today called SPIRALS), was my first research encounter with *Green IT*, sustainable software engineering, and autonomic computing. After that, the story is known: a PhD defended in March 2014 titled *Towards a Better Understanding of the Energy Consumption of Software Systems*, two research fellow positions at the University of Edinburgh and the University of East London, an R&D position at Cat-Amania, and finally an associate professor position at the University of Pau and Pays de l'Adour in 2018.

This manuscript provides a summary of nearly a decade of research in green IT, software engineering and autonomic computing, after my PhD defense, from summer 2014 to summer 2023. In particular, my research focuses on green software and their ecosystems, including hardware and software layers, architectural concerns, and human actors.

1.1 Context of Green Software

Recent studies indicate an acceleration of earth's warmth and an alarming increase in temperatures with recent scenarios describing an increase of up to 7% by 2100 [cnr19]. The major reason for climate change is human activity, and in particular CO_2 emissions due to fossil fuel and related energy sources [ipc19]. Earth's resources depletion is also another problem that is also due to human activities, such as extracting mineral resources, water drainage or deforestation.
Human societies today live in the information age where economies and social activities are shifting towards information technology (IT). Energy demands and consumption in information systems, ranging from computers and servers to devices in Internet of Things (IoT) and data centers, have grown subsequently in the last decade [VVHC$^+$10, W$^+$08]. Information and Communication Technologies' (ICT) greenhouse gas emissions (GHGE) could exceed 14% of global GHGE by 2040 from around 1-1.6% in 2007 [BE18]. ICT account in 2020 to up to 7% of global electricity consumption and is expected to continue rising [SA20]. ICT also account in 2015 for 4% of European CO_2 emissions, and up to 10% of the total European's electricity consumption [ict15].
Today, people and businesses are more equipped with digital and technological devices and appliances. These devices are nowadays connected together and with distant servers (cloud computing), with complex software controling each devices and the collective of devices, thus transforming our equipment into smart devices and smart systems. This advent of devices and software in every aspect of our lives is a key element in major social changes, such as the fourth industrial revolution (Industry 4.0). Therefore, these smart environments open opportunities for energy management of those devices and appliances, as the energy consumption of appliances have doubled in 20 years, according to the French environment and energy agency, ADEME [ADE18].
Therefore, ICT has expanded to more than just servers in a data center, wired and wireless network equipment, and computer terminals. Internet of things devices are expected to continue rising, with specific numbers varying depending on sources, but one source expected up to 125 billion devices in 2030 (a 12% annual increase) [ihs17].
To achieve the goals in reducing our impact on the planet's resources and emissions, it is crucial to reduce the impact of ICT. Addressing the necessity of reducing the energy and CO_2 footprint of ICT can be achieved through many approaches.
The most prolific approaches revolve around optimizing the impact of the hardware that ICT need, such as optimizing its construction, recycling or usage. For the usage phase, optimizing ICT is essentially optimizing their energy consumption, thus, technical-centric approaches.
Hardware devices and equipment are, at their essence, just pieces of metal and stone that provide no real added value without software piloting them. For example, a smartphone without an operating system and applications, is just a collection of rare earth materials without any use. Software can be considered the brain of hardware, piloting them and giving commands and instructions. Therefore, optimizing the energy consumption of hardware, comes down to optimizing the instructions provided by software.
Another set of approaches question the needs of integrating ICT and intelligent software into our everyday lives, and to answer our social and economical problems. For instance, some approaches advocate technological degrowth, in part due to the increasing environnemental impact of ICT. Specifically, such approaches question if there is a business need today, or in

the future, that only an ICT-based approach can solve. In short, these approaches tackle the human and social sides of ICT.

End users, in particular, are often interacting with these smart devices and the software running and piloting them. Focusing on software systems optimizations without integrating users in the energy reduction strategies might lead to rebound effects. For example, optimizing energy consumption of an application might lead to an increase in using this application by end users, and therefore an increase in the total energy consumption rather than a reduction. Therefore, end users also have a key role in adopting sustainable and more energy-aware behavioral patterns, such as changing their usage of ICT devices and software in order to reduce their energy consumption.

This overview of the current situation in green ICT and green software sheds the light on three major changes:

- ICT and software are integrated more tightly in every aspect of our societies and economies, and energy management approaches need to address the new reality of **cyber-physical systems** (CPS) with software systems at its core. CPS are systems that integrates computing and software through sensing, computation, control and networking into physical objects, connecting them together and online. The days of compartmentalized and isolated systems are over, and ICT is not a support service anymore but the basis of our information societies.

- The spreading of smart devices, equipment and software systems is leading to scattered architectures, software, protocols and hardware designs. These **heterogeneous environments** are more complex to monitor and measure their energy consumption, and therefore optimize them.

- As more people are interacting with software, through new and diverse devices, they are leading demands in terms of features, requirements and needs. Optimizing energy of software environments cannot be sustained without **integrating humans** in the energy efficiency equation.

My approach aims at optimizing and reducing the energy consumption of such environments by following the fundamental research approach of understanding the physical world through observation and experimentation.

I argue we first need to observe software energy consumption, through modeling or measuring their energy consumption at various granularity levels, and everywhere software is present (*e.g.*, observing energy in heterogeneous ecosystems).

Then, we need to understand why energy is consumed, what is consuming it, how this is consumed, and what can we do to make things better. My approach aims to understand the factors impacting software energy with a goal to provide knowledge and best practices to help build green software.

However, observing and understanding green software, and ultimately optimizing it, leads us to a simple question: what is *green software*? The two notions of green and software are, individually, not intuitive to comprehend. In the next section, I define what the concept of *green software* is, with the definitions of the concepts of software and green.

1.1.1 Defining Software

Software is the brain of hardware, as computing hardware cannot function properly and usefully without the instructions and orders that software provide.
But what is software? Where does an application, a software architecture or a piece of code making the instructions of a software, exist?
According to the ISO/IEC 2382-1:1993 standard [ISO93], software is *all or part of the programs, procedures, rules, and associated documentation of an information processing system*. The same standard defines a program as a *syntactic unit that conforms to the rules of a particular programming language and that is composed of declarations and statements or instructions needed to solve a certain function, task, or problem*.
Defining software as just computer instructions and rules hides its most important nature: software is an intellectual object rather than a physical one. The ISO standard acknowledges this concept with a note associated to the definition of software stating that *software is an intellectual creation that is independent of the medium on which it is recorded*.
In the following, I strengthen ISO/IEC's definition of software with my own nuances and additions, and propose my refinement of the definition of software.
The code of a text-editing software, in source code or in binary format, resides in a memory space: either in the permanent storage memory (such as a hard disk or an SSD), or a temporary storage memory (typically, the RAM memory or the cache memory). But in either case, the only physical object, with real impact in the physical world and the environment, is the memory disk itself. The code is stored as a collection of physical objects characteristics, such as RAM capacitors or SSD transistors. Therefore, as software cannot be touched, smelled or seen, it does not physically exist. Software is therefore **intangible**.
In essence, software is an abstract concept, an **idea**. Like a poem or a novel, software is a cultural product, and an abstraction of reality. It is written in textual format, and using a specific language. Shakespeare's *Hamlet* is written in a specific language (16^{th} century's Early Modern English), with its specific set of conditions, regulations, and structure. No one can hold or touch *Hamlet*, as we'll only be able to touch the paper it is written on, or see the digital screen it is displayed on. Like *Hamlet*, the Linux kernel is written in a specific language (mainly the C programming language), also with its specific set of conditions, rules and structure.
But software is more than just an idea. It is a cohesive list of instructions, structured in variables, functions, methods, classes, or packages, like *Hamlet* is structured in sentences, paragraphs, or acts. Software is, therefore a **book**. More precisely, it is a book of instructions. Similar to a washing machine manual, indicating how to operate the machine, a software is a book of instructions, indicating how to operate a computing device.
But unlike a washing machine manual where the user, the human, is the one executing the instructions (*i.e.*, loading or unloading clothes, adding detergent, etc.), software is **automated**. The book of instructions is automatically executed by the computing machine. And the same instructions can run on different execution contexts: computers, operating systems, etc.
For me:

> Software is an automated book of instructions, an intangible idea, an abstract manual, a poem, and live in our cultural space.

It is important, through this definition, to state that only physical hardware and machine consume actual energy and physical resources. The earth metals are used to build the phys-

ical computing device. And the electric energy is consumed by the same physical device. However, these devices do not consume energy without the instructions provided by software. And they are not built unless there is a software need for them. The energy consumed by machines to execute the instructions of software, is defined as *software energy*.
The question that arises now, is what *green* means in our context.

1.1.2 Defining Green

The global ecological impact of ICT shows us the importance of reducing this impact in IT devices and software, in particular their energy consumption and costs. However, defining what a green IT, sustainable computing or a green software is, is still a challenging task, as definitions diverge, and their scope varies.
Looking at the state of the art, a few definitions of green IT have been proposed. Elliot in 2007 [Ell07] defined Green IT as: *The design, production, operation and disposal of ICT and ICT-enabled products and services in a manner that is not harmful and may be positively beneficial to the environment during the course of its whole-of-life*.
In this early definition, the *green* part of IT is on the entire life cycle of physical devices (design, production, operation and disposal).
Gartner's analyst, Mingay, proposed a similar definition of Green, more tailored to the context of an enterprise, the same year [Min07], with Green IT being popularized and defined as: *Optimal use of information and communication technology (ICT) for managing the environmental sustainability of enterprise operations and the supply chain, as well as that of its products, services and resources, throughout their life cycles*.
Finally, Molla in 2009 [Mol09] combined multiple previous definitions to define Green IT as: *Green IT is an organization's ability to systematically apply environmental sustainability criteria (such as pollution prevention, product stewardship, use of clean technologies) to the design, production, sourcing, use and disposal of the IT technical infrastructure as well as within the human and managerial components of the IT infrastructure*.
These early definitions cover the hardware side of IT and enterprise organizational aspects. In contrast, Murugesan in 2008 [Mur08] proposed a similar definition of Green IT also around the lifecycle of physical device: *Green IT refers to environmentally sound IT. The study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems - such as monitors, printers, storage devices, and networking and communications systems - efficiently and effectively with minimal or no effect on the environment. Green IT also strives to achieve economic viability and improved system performance and use, while abiding by our social and ethical responsibilities.*
As Murugesan defines Green IT in the scope of *system performance and use*, therefore he is proposing a first insight of what Green IT will be about in the operational phase of devices and, thus, software, which identifies a number of focus areas and activities such as *energy-efficient computing*, *power management*, and *green metrics, assessment tools, and methodology*.
These definitions of Green IT as application of sustainability in the fields of ICT. This raises the question about defining sustainability itself as it encompasses more than just energy efficiency.
Costanza et al. defines sustainability as the following [CP95]: *A sustainable system is one which survives or persists*.
The authors go into identifying three questions to qualify a sustainable system: 1) **What system** or subsystems or characteristics of systems persist?, 2) For **how long** they are to be

sustained?, and 3) **When** do we assess whether the system or subsystem or characteristic has persisted and has been sustained?
This definition and subsequent questions are in accordance to the definition of sustainable development that the UN's Report of the World Commission on Environment and Development (*Our Common Future*) [oED] has established, where sustainable development is defined as: *Humanity has the ability to make development sustainable to ensure that it meets the needs of the present without compromising the ability of future generations to meet their own needs*.
Sustainable computing is thus an application of the general concepts of sustainability, and sustainable development, to the computing, IT and ICT worlds. One of the earliest definitions of sustainable computing [Moc06] distinguishes six dimensions of this new concept: focusing on the product, on the production process, and on the consumption process, all these dimensions across hardware and software.
Common definitions of sustainable software [MA13, NDKJ11, Fou], often focus on energy consumption, carbon footprint, and eventually on software life cycle.
However, as I defined software as an abstract concept, or a cultural idea, the concept of sustainable software is harder to specify than what the state of the art proposes. Software, as an idea, is more than just energy, carbon emissions or a development process. Its sustainability is in the realm of culture, rather than technical factors. Nevertheless, the general concept of sustainably as defined by Costanza et al. [CP95] can also be applied on abstract concepts.
And like poems and books disappear from human memory, and therefore fade from existence and stop being sustained, software also can disappear and fade from our collective human or digital memories. The work of the Computer History Museum in California [1], or the Software Heritage project [2], is, in essence, the closest to preserve, and thus sustain, software.
For instance, is Microsoft Office more sustainable than LibreOffice? Regardless about their features or impact, is Microsoft Office of 2023 the same, but evolved version, of Office 2000? Or are they considered two different software, as the code base and supported file formats are quite different?
What this outline is that the concepts commonly implied by the research and practitioner communities about *sustainable, green, energy-efficient, or carbon-neutral* software and IT are not what the proper definition of each term is, and are often interchangeable. The community is referring to a general ill-defined but also hard-to-define concept, as the ultimate goal is to bring the physical reality (of energy, earth materials, environmental impact) to the world of culture, words and ideas.
In this manuscript, I prefer to use the term of *green software* rather than sustainable software, as my goal is studying the energy efficiency of software, and building software ecosystems that use less energy resources for the same workload, or questioning the need for the workload to answer the user's needs.

Epilogue: Software is an abstract concept that is difficult to understand and comprehend by end users. Software ecosystems go beyond software themselves and beyond old-school computing hardware (*i.e.*, computers and servers). For instance, in the data center environment, a large part of the electric energy is consumed by the cooling system rather than the computing hardware.

[1] https://computerhistory.org/
[2] https://www.softwareheritage.org/

In addition, in the last 10 years, the landscape of computing has much evolved with billions of new smart, connected or mobile devices. How we interact, as a society, with computing devices, software and IT as a whole, has much changed. This new reality encompasses the physical world (from the physical objects and humans, to socio-economic and political factors), and the cyber world (computing devices, sensors, systems, and software).
This new landscape of connected devices, smart sensors and actuators, and non-computing actors, is bound together with software. Such landscapes encompass smart homes, smart industries, but also computing systems such as data centers, computers or smartphones, or software systems (applications, source code, distributed or cloud-based software).
The challenges we face in this new reality are complex if we need our software to be energy-efficient, which I'll outline in the next section.

1.2 Scientific Challenges

Building green software ecosystems or transforming existing ones to be more energy-efficient, present a multitude of challenges. Throughout the last years, these challenges evolved with the changes in software, software architectures, cyber-physical systems and devices, and how users interact with them.

1.2.1 Monitoring the Energy of Heterogeneous Software Ecosystems

Software ecosystems have become complex and heterogeneous. In server systems and data centers, the dominance of CISC x86 architecture is more and more challenged by RISC's ARM, graphic cards, and other custom architectures and FPGA's. This is accentuated by the rapid rise of AI algorithms, mostly using graphic card and custom cores.
The social changes around IT and connected devices pushed heterogenous devices in the hands of users, such as smartphones, smart speakers and televisions, and other Internet of Things (IoT) devices, sensors and actuators. All of these new devices have different hardware architectures, components, usage patterns, and therefore energy consumption and energy profiles. The cement of these devices, making them smart and their ecosystem useful, is software.
We are also more often going towards rapid development and deployment cycles of software and systems, where new updates can improve or reduce the energy efficiency of software. Keeping track of all this evolution and heterogeneity requires being able to observe, monitor, measure and model the power and energy consumption of software.
The heterogeneity of such systems and its more complex architectures is making their energy monitoring much more difficult and inaccurate. Integrating hardware power meters or sensors is financially costly, and ecologically a disaster as these meters need to be manufactured and deployed, including onto small connected devices, such as Raspberry Pi mini-computers. In contrast, modeling the power consumption is a promising approach but is time costly to build models for the diversity of hardware. A crowd-sourced approach, such as involving a large number of device owners to participate in power modeling, or forcing manufacturers to deliver power models are better solutions, the former is one I used to build power models for connected devices.
Addressing the challenges of heterogeneity in software, hardware, and the architecture and design of software ecosystems was a difficult task, decades ago, and it is still a challenging

task, and will be so for the foreseeable future. My work and contributions aim at facilitating building solutions for monitoring the energy consumption of software in heterogeneous ecosystems.

1.2.2 Understanding the Impacts of Design Factors and Actors

Observing the energy consumption of software is a first step in energy efficiency. The next step is understanding this observation, and what factors impact energy efficiency, from hardware or software features and metrics, to procurement, requirements or design choices, to the role of the different actors (end users, developers, software systems or non-computing systems).
Improvements in hardware design, cooling and networking infrastructure, and in building architecture and construction (such as building data centers near clear water sources, or in cooler climate zones), have much improved the overall energy efficiency of data centers and reduced their PUE (Power Usage Effectiveness) to an average of 1.57 in 2021 [KDC22]. A similar trend is also happening for computers, connected devices and sensors, with processors and devices having more energy efficient hardware or build design. A typical Mac Pro in 2008 had an idle power at 155 watts and a maximum power at 318 watts, compared to 47 watts idle and 300 watts maximum for the more powerful 2023 Mac Pro [mac23]. Even though operating sytems and some applications have become more energy-aware, the rate of energy improvement in hardware wasn't matched by similar improvements in software energy efficiency.
With proper understanding of why and how energy is consumed by software, developers and practitioners would have the knowledge to build energy-efficient software, while managers and procurers would make proper choices in green design and procurement. End users awareness would be raised with a hope that this will trigger behavioral changes in using software.

1.2.3 Integrating Humans in the Loop

I finished my PhD [Nou14], a decade ago, with a bold statement that we need to *put the human outside the loop*, in software systems' energy management. When applying runtime adaptations and optimizations on software systems, I still argue that a fully automated approach is the best course of action. However, in our new cyber-physical world, software is only a part of a more complex system, and I argue that every actor in these cyber-physical systems needs to play its role in sustaining the system and making it energy-efficient.
Putting the human in the loop makes sense in this cyber-physical reality and systems. This integration targets all the humans involved: developers, architects and other software practitioners, managers, deciders and procurers, and finally end users.
For practitioners, there is a dual need for monitoring and understanding the energy consumption of software, including providing accurate monitoring tools that integrate into their workflow and their business needs. In a study we did in early 2016 [BWN16], surveyed software architects didn't have to deal with energy efficiency concerns in the last 5 years (83%), while a majority thinks it will be a major concern in the next 5 years (67%). The most important finding is that a majority of software architects consider they don't have the right tools to address energy efficiency at the design level, with only a quarter saying they do. Also,

another recent survey [PHAH16] showed that software developers rarely take energy efficiency into account (18%), and very few knew that software energy can be measured (12%). A larger survey of practitioners' perspectives on green software (464 surveyed developers and testers from ABB, Google, IBM and Microsoft) [MBZ+16a] indicated that: software engineers *care and think about energy when they build applications*, and they *are not as successful as they could be because they lack the necessary information and support infrastructure*. Finally, another survey by Pinto et al. [PC17a] identified energy efficiency as a new concern for software developers.
For managers and procurers, they need to understand, at a business logic, what actions they can apply to drive energy efficiency, and how they can monitor their environments with ease. Finally, end users need to know what their systems are consuming to be aware about their impact and how they can make things better.

Perspectives: The challenges of today and yesterday will stay with us for the coming years, as their complexities require many core building blocks towards green software. My vision for the next decade involves tackling additional challenges, which are detailed in the last chapter in Section 4.2.

1.2.4 Overview of Existing Approaches

Over the last decade, many approaches have been proposed to tackle the issues with green software. I present here a summary overview and limitations of the main categories of approaches: runtime optimizations, eco-design, and behavioral changes.

Runtime optimizations

Arguably the most popular approach in green software engineering is applying software optimizations and configuration changes to software and hardware during the execution phase, including approaches such as consolidation of services or virtual machines in server environments.
Research in energy efficiency in software systems start by monitoring and measuring energy consumption, with a wide range of approaches ranging from measurements in server systems and hardware components, to cloud, IoT and mobile devices [NRS13, CFRS17, FRS20, FRS21, LOR+18, LHHG13, CCC+14, HLHG13, WNLMM18].
Then, many approaches aim to optimize the energy consumption of software [APTV15, Jag17, NRS15, OBR+21, OBR+20, Bel22], to optimize the energy footprint of servers and equipment in data centers [Lev15], or to optimize the workload in servers, virtual machines, and in distributed environments [OAL14, CRK+18].
In data centers, approaches vary from operating system and application optimizations, to virtual machines migration and consolidation, to load balancing, to workload categorization, to batch execution, or to hardware or cooling approaches [KDC22, HGP16, TPS+15, SG13, CCM20].
Modern operating systems adopt techniques to manage the energy of the system and working with the underlaying hardware, such as DVFS (Dynamic Voltage and Frequency Scaling), or OSPM (Operatin System Power Management). Operating system governors and schedulers can also be tweaked to be more energy-aware, such as the Linux scheduler [SAL18].

Similar approaches in scheduling are also applied at the virtual machines level, such as in [KAK+18, XNLH21].
The advent of edge computing is also pushing new approaches, such as computation offloading strategies between the cloud and the edge (*i.e.*, offloading heavy tasks to the cloud, or offloading recurrent tasks to the end user devices) [LWP13, WW16].
Overall, existing approaches in runtime optimizations are quite diverse, but necessarily specific to a particular set of hardware and software configuration. Although they can be effective, existing approaches are bound to their limited scalability, as software or operating system updates, improvement in hardware design, or a change in business need and use cases, will render these optimizations less effective.

Eco-Design

In eco-design approaches, the goal is to build or update software systems or hardware while taking energy efficiency into consideration from the design phase. Developers and practitioners are interested in building energy-efficient software, as many studies have shown that energy is a rising concern for them [PC17b, MBZ+16b, BWN16]. However, they lack proper tools to measure, analyze, and apply good design green best practices.
Some approaches in eco-design use refactoring and reengineering, which try to refactor source code or reengineer the design of an application in order to reduce energy consumption. For example, in [PLCL16], the authors showed that using a different implementation of a HashTable in Java yielded 3.5x energy savings, and in [OH98], 20% energy was saved by uploading tasks from mobile devices to fixed servers.
Others use visualization techniques that help viewing and understanding energy consumption in software. For example, the authors in [LHHG13] colored lines of code according to their observed energy consumption. In [CCC+14], the authors detected anomalous energy consumption in Android applications and related that energy to methods and classes with a sunburst diagram.
And many approaches tried to understand the impact of certain design choices in software code or hardware configuration towards energy efficiency [PFL16]. For instance, mobile development is an area that had, recently, extensive energy research. Recent work studied the impact of Android APIs [LVBBC+14], developer knowledge [MZG15], or hardware components such as OLED displays or GPS [LHHG13, LTH14]. Recent tools, such as ecoCode [LGH23], help developers identify code structures that are known to have a negative impact on energy consumption. Such energy bad patterns or energy smells are pre-identified by experts in a curated list, and detected through automated tools, such as SonarQube.
In networking, data structures and parallel programming investigated the role of HTTP requests or network usage, and the impact of certain data structures (thread-safe, data sharing primitives, etc.) on energy consumption. In addition, recent work shows that parallel frameworks can yield better energy efficiency than thread-based ones [PCL14b].
Eco-design research is gaining traction in recent years with new approaches and studies helping practitioners build energy-efficient software. Most of such approaches either target a specific environment (such as Android development), or analyze software source code and low-level coding practices. A true green design approach is still lacking, such as studying or improving higher-level design choices in software construction and in software systems.

Behavioral Changes

Integrating end users in the energy efficiency equation is still quite rare in software engineering, but many existing research studied their role in smart systems, such as smart building or smart homes. Most of these approaches use behavioral techniques to push users to adopt greener behaviors in regards to non-computing devices, such as lighting, heating or using electric appliances.
There are three main approaches to drive user behavioral change: eco-feedback, social interactions, and gamification [PB18].
Eco-feedback studies focus on providing occupants with information regarding their energy consumption, current or historical data. Eco-feedback can indicate raw energy consumption, its equivalent in price or CO2 emissions, or an abstraction of scientific units, such as the number of trees saved [WN07]. A large-scale eco-feedback study of 2000 households reported saving of up to 15% [VOWD12]. However, the long-term effects of eco-feedback is still under scrutiny as energy savings could disappear in the long term with rebound effects and unintended consequences [BRA15]. Normative feedback has recently been investigated and suggested for a positive behavioral change in the long run [AL16]. Overall, eco-feedback is not a new scientific challenge, and, according to [FFLS10], research goes back decades ago to the origins of environmental psychology.
Social interactions take eco-feedback a step further to allow users to compare their energy usages with others. Thus, normative comparison interactions can yield energy reductions up to 55% [JGTC13]. Comparisons can be done locally in individual buildings or neighborhoods, or online through dedicated websites or social networks.
Finally, the authors in [PB18] portray gamification as a new approach for triggering behavioral change. Games offer interactive experiences, and the emotional engagement of users helps encourage new behaviors as aimed by the games. Game examples include suggesting specific or more appropriate actions for energy reductions, correcting bad energy usage, or motivations for tangible and social actions.
These behavioral approaches, quite popular in buildings and smart homes, are rarely applied or studied in software systems. The main difficulty is that software, as an abstract construct, is difficult to understand and comprehend by end users, and its energy impact in the physical world is still a complex subject even for tech-savvy users. Even simple objects, like a light bulb, which usually has one option (*on, off*), have become *smart* with software in it and having multiple options from brightness and intensity to colors, with each combination of options having a different energy consumption. Therefore, it is important to study how users understand and interact with software in their wide execution landscapes, and how we can push them to adopt greener software behaviors.

Epilogue: Most current approaches are technical or oriented towards developers and practitioners (whether runtime optimizations or eco-design approaches). Humans, the ones using software or making decisions around software, and not the ones coding them, are often forgotten in the energy efficiency equation and outside the sustainability loop. I argue that end users, as an integral part of software ecosystems, have an important role to play, and we also need user-oriented approaches in monitoring energy and understanding its impact.
The new reality of complex and heterogeneous software requires more than specialized technical solutions. Thinking about energy efficiency starts with better observation approaches, tailored towards developers, managers and users alike, and continues with better under-

standing of the factors in play, and ends in thinking about energy as a design choice in software development.
To this end, I envision addressing the challenges of green software ecosystems with both a technical approach and a human approach. Both starting from observing software energy ubiquitously, understanding the elements in play, and acting towards energy efficient software ecosystems. The next section details my vision.

1.3 Vision

My vision towards green software starts from my definition of green software. How I address the challenges and my contributions around them are driven by this abstract concept and idea that is software.
Optimizing software energy consumption is a necessity today, but doing so requires more than just optimizing the instructions and source code of software. Software, as an abstract concept, lives in a physical ecosystem of hardware components and devices, interacts with users and other systems, and its existence is driven by business needs and user requirements. Therefore, improving the energy efficiency of software requires more than optimizing software itself, and goes into optimizing its ecosystem. Software is everywhere, and our smart ecosystems are driven by software: from software sensing information, or complex algorithms analyzing data and deciding on actions, to interactions with other devices, software and users. These new landscapes involve more heterogeneous devices, actors, software and other contextual factors.
This vision boils down to two dimensions for optimizing software:

- Optimizing instructions: that is optimizing the technical aspect of software and its ecosystems. This requires measuring, understanding and optimizing software (from code to design factors and architectural concerns) in order to run more efficiently, individually, and as part of an ecosystem (*e.g.*, with a holistic approach).
- Optimizing the needs: which means addressing the business and human needs and requirements. Do we need a technological solution for our software problem? Can we act on the users or deciders in software? This is the human part in my vision and the necessity to integrate it in the green equation.

To achieve this vision across these dimensions, we first need to **observe** software energy, then **understand** what's happening before aiming to optimizing it. As software is nowadays everywhere (in the diversity of ecosystems they're executing on, from computers and servers, to smartphones and all other smart objects), we need to also **observe and understand software energy everywhere**, that is in its various objects, systems and layers.
A holistic approach means observing the entire ecosystem and capturing the relevant data to optimize energy efficiency. Observing such complex and diverse systems, with different layers (from low-level systems, to software architectures, to external actors), requires different approaches to measure or model the energy consumption. These measurements can be provided in real time, or aggregated and processed for a post-execution analysis.
In addition, capturing relevant data requires understanding what are the factors that have a real impact on energy consumption, whether the impact is direct or indirect. Then, collecting the data needs proper sensors and modeling of the environment, because many of the relevant data are contextual information, *i.e.*, generated knowledge from processing data from

multiple sources (such as detecting a human presence in a room through the usage patterns of a smart TV).
In all cases, we need to build and create the building blocks of this holistic approach. For instance, building measurements tools or energy models for each system or software, or each category of systems, or building knowledge and understanding patterns impacting energy consumption of software. I build tools, models and approaches, and provide analysis and empirical studies, including around human actors. These building blocks allow observing and understanding complex heterogeneous software ecosystems, and help achieve my vision in managing software energy in a holistic approach.

Where is software? Software is everywhere today, from computers and servers to smart devices. To optimize software energy in a holistic approach, we need to observe and understand software energy everywhere: on every layer of our complex heterogeneous environments and ecosystems.
There are many classifications for software and computing ecosystems and environments. The OSI model [iso94] is one of the most used ones dividing distributed computing systems into 7 layers, from the physical to the application layer. In this manuscript, I adopt the layers identified the by ACM Computing Curricula [acm05] in their *Problem Space of Computing* model. The model identifies 5 focus layers, going from top to bottom:

- Organizational Issues & Information Systems, where the focus is on people, information and the organizational workspace, with a goal to help organizations to operate, and satisfying human needs.
- Application Technologies, where the focus is on building applications tailored for specific needs or aimed for organizational purpose, such as web browsers, databases or search engines.
- Software Methods and Technologies, where the focus is on developing systematic models and reliable techniques for building high quality software. This is the core of software engineering.
- Systems Infrastructure, where the focus is on systems enabling the construction of more complex applications and technologies. This infrastructure is the basis of any modern computing ecosystem, with elements such as an operating system or communication programs.
- Computer Hardware and Architecture, where the focus is on the hardware itself and the construction of complex hardware architectures.

I adopt this classification as it covers the main hardware and software layers, but also the business and human factors with focus on people and workspaces. My contributions and future perspectives aim to cover observing and understanding green software in all these different layers.

Positioning: My research in the last decade, and my objectives for the next one, are a part of a global research community working on different aspects of green computing and green software engineering.

Related research teams tend to focus on specific environments (such as data centers, cloud services, or mobile phones), and on a specific building blocks, rather than the whole environment.

For instance, some research teams (such as France's Inria's STACK in Nantes and Myriads in Rennes, or the Enterprise Computing team in London, UK) focus, in relation to energy efficiency, on large scale distributed systems, data centers, cloud, networks and grids, and fog/edge computing.

France's Inria's SPIRALS in Lille focus on energy in software systems, in particular with measuring and analyzing power consumption and technical software factors. The Software and Sustainability team (S2) in the Netherlands, focus on software sustainability as a general concept (and not only energy efficiency), and also in green software patterns in robotics and mobile devices.

In France, the research community in green IT is grouped into multiple working groups, with two main ones: the Energy working group in the GDR RSD (networks and distributed systems), and the Sustainable Software working group in the GDR GPL (software engineering) which I co-founded and co-piloting. RSD's Energy working group has an important focus on networks, grids, distributed systems, cloud and data centers, while GPL's group focuses on the software engineering part, including with eco-design in writing software code or measuring software energy consumption. In my current university in Pau, France, the software engineering (GL) team focuses on proposing eco-design good practices and automated tools to detect bad practices in mobile environments, mainly Android's software development. T2I's team (information processing and interaction team) address the issues of green IT with a similar focus on eco-design in software code and units, such as middleware. My research team, CPSA (cyber-physical systems architecture), focuses on the CPS architectural aspects, in particular with a network-oriented approach.

In the last very few years, green IT and green software is generating hype again, mainly due to the social and political shift towards diversifying energy sources, and reducing the energy impact and GHGE impacts of developed societies. I think it is too early to judge the sustainability of recent research initiative, such as the Green Coding Berlin team in Germany, or industrial initiatives (such as Boavizta in France, or the global but USA-based Green Software Foundation). However, from their early results and manifests, these initiatives usually focus on a specific environment (mobile, data centers, web servers, or pure technical/software engineering) as current research teams do.

My vision, in contrast to existing research teams and initiatives, aims to address energy efficiency in software in both the technical and human dimensions, and manages software energy in a holistic approach taking into consideration different layers of complex ecosystems including human actors.

Holistics approaches, per se, are not new, but existing ones suffer from the same limitation as the holistic part have on a specific environment, *i.e.*, for example, we can manage energy in a data center in a holistic approach but only limited to the data center itself and not integrating use cases, edge devices, fine-grained software factors, and human actors.

The uniqueness of my vision is addressing the challenges of software everywhere, and in their whole diversity, heterogeneity, complexity and widespread applications, with a holistic approach consisting of building blocks in all layers, and integrating all actors.

This manuscript details my contributions in the last decade towards this vision, and the perspectives to sustain and fully enrich this vision for the next decade.

1.4 Contributions

My contributions in this manuscript aim to provide initial sets of answers towards achieving my vision for energy management, with approaches, solutions and knowledge in observing and understanding software energy along with steps towards improving and acting on software energy. I propose multiple contributions to address the challenges and the limitations observed in existing approaches.
Software, as an abstract concept, is not the one consuming electric energy. The hardware where it is running on is the actual physical object consuming energy, and hence it all starts with being able to properly monitor and measure the energy consumption of hardware. With the advent of connected computing devices and the explosion in IoT devices in our new cyber-physical world, software is *everywhere*. Therefore, to study software energy, we first need to study hardware energy, and do so *everywhere* too.

Observing green software ecosystems: The complex heterogeneity of such ecosystem makes observing and capture energy and power data of various devices and components challenging as I outlined in Section 1.2. Not all devices have energy sensors embedded or are plugged into power meters. Their heterogeneity makes building a one-off model complex and impossible. In addition, other contextual data about the device and its activities are essential for proper and efficient energy management. My approach towards these challenges boils down to modeling the energy dimension of these complex ecosystems.
For instance, in Section 2.2.1, we propose a modeling approach to contextualize complex environments and help in collecting contextual data useful for energy management. In particular, we extend the SAREF ontology with energy-related attributes to help understand such complex systems, and build an energy-aware knowledge base useful for autonomous energy management.
In Section 2.2.2, we propose a crowd-sourced approach to model the energy consumption of heterogeneous devices and hardware components in cyber-physical systems. With the wide variety of these devices, including for computing ones, and the lack of constructor-provided energy models or sensors, it is a necessity to generate power models with the help of the community of users.
In Section 2.2.3, we propose an implementation of our general approach aimed towards single-board computers, and in particular for Raspberry Pi devices. And in Section 2.2.4, we adapt our approach to model the energy consumption of non-computing devices and objects, such as a smart bulb or a computing monitor.
These contributions help creating and capturing the necessary energy information needed to observe energy, and thus being able to understand it and manage it.
After being able to model and measure the energy of hardware devices and components, we need to map the energy consumed to software instructions and structure. While hardware is consuming electric energy, software is giving the instructions for that consumption. Mapping this consumption to software is, therefore, crucial to understanding where software energy is being consumed and why.
To that end, I propose multiple contributions aimed to provide tools, approaches and architectures to observe energy consumption everywhere, *i.e.*, at various layers of software ecosystems.
On the hardware layer, PowerJoular (Section 2.3.1) and PowDroid (Section 2.3.2) are two ap-

proaches and tools aimed to monitor the energy consumption of various devices and hardware components. PowerJoular monitors the energy of computers, servers, and other computing devices, such as single-board computers, all across various CPU architectures (x86, ARM). PowerJoular can also monitor applications (through individual or multiple process IDs). PowDroid is capable of monitoring the energy consumption of Android devices with no modifications or installation on the device itself.
On the application layer, Jolinar (Section 2.3.3) and Demeter (Section 2.3.4) allow monitoring applications in Linux or Windows, in a user-friendly approach. Jolinar, on Linux, is tailored toward power users and can monitor individual applications with a granularity at the hardware component level (CPU, disk and memory). Demeter, on Windows, is a user-centric software architecture capable of monitoring all running applications on multiple computers (for CPU, disk and network), and can provide users with a summary of their energy consumption and potential recommendations.
On the software methods layer, JoularJX (Section 2.3.5) is a multi-platform approach capable of monitoring the energy consumption of methods and execution branches, across time. This fine-grained tool, aimed for developers, is an advanced energy *microscope* tailored to observe the energy dynamics in software source code.
My contributions in observing green software ecosystems provide building blocks for different layers in complex and heterogeneous software ecosystems, and are detailed in Chapter 2.

Understanding and acting towards green software ecosystems: Measuring and modeling complex software ecosystems is the first step of our journey towards green software. The second crucial step is to understand the energy dynamics of software ecosystems and answer the question: *why?* Therefore, it is important to understand the impact of design choices, software and hardware factors, and the role of actors, such as end users. Thus, we need to understand what's happening at every layer and phase in the software life, from design to usage and its interaction with users.
To understand the technical and human factors impacting software energy, I start again from the hardware layer and navigate up towards low-level features, software construct, design choices, testing, and the interaction with users.
For instance, in Section 3.2.1, we analyze the impact on energy consumption of around 100 hardware and software features and metrics, including dynamic and static ones.
In Section 3.2.2, we analyze the impact of design choices on energy consumption, and in particular of the impact of software design patterns. We also propose a transformation approach to optimize energy consumption of these patterns.
In Section 3.2.3, we explore measuring and analyzing software energy in a horizontal cross-cutting approach, to study software at the feature-level with program slicing.
In Section 3.2.4, we analyze the impact of code coverage and unit tests on software energy, and aim to answer this question: is well-tested code energy efficient?
In Section 3.2.5, we explore and analyze the impact of green feedback on user awareness about the energy consumption of their software and computing devices. My aim here is to implicate users in the energy efficiency equation, by raising their awareness and understanding about the impacts of their choices and behavior on energy consumption.
Observing and understanding the impact of technical or human factors on energy consumption provides us the knowledge needed to act towards green software. In my contributions, I propose initial approaches aimed towards energy management in heterogeneous ecosys-

tems and integrating human actors.
In Section 3.3.1, we propose a design tool aimed to help procurers assess the energy efficiency of their data centers and server rooms, and guide them in their procurement, purchase or optimization choices.
Implicating users also means that automating the management of complex heterogeneous ecosystems needs to take into account the human factor, such as users' preferences, and adapt according to the user's interactions and behaviors. Therefore, in Section 3.3.2, we propose an autonomous energy management architecture that uses knowledge collected from the environment (using our energy models from Section 2.2.2 and ontology extension from Section 2.2.1), and a reinforcement learning approach to apply green adaptations and changes to devices and software while respecting user preferences and learning from their behaviors.
My contributions help understanding and analyzing the factors impacting software energy, and providing steps and approaches in acting towards green software. My contributions are detailed in Chapter 3.

1.5 Structure of this Document

This document is read as a three-act structure with:

- Chapter 1 - First act, setup and opening narration: **Introduction: The Context of Green Software**. I present the definitions and context related to green computing and sustainability, motivating my work and defining the challenges and an overview of my contributions.
- Chapter 2 - Second act, confrontation and rising action: **Observing Green Software Ecosystems**. I address the challenges and work on observing power and energy-related information of computing hardware, devices and software, in multiple contexts and ecosystems, and aimed towards multiple actors.
- Chapter 3 - Third act, climax and resolution: **Understanding and Acting Towards Green Software Ecosystems**. In this climax, I discover and understand the energy impact of computing devices and software, and analyze and model the underlaying factors including the role of humans, along with elements on how we can make things better.

Finally, in Chapter 4, I conclude this unparalleled journey with a summary of my main achievements in this last decade, and presenting an epilogue for future adventures and research directions for my vision of the next decade.
In the remainder of this introductory chapter, I present my list of publications, research grants and projects, and PhD and research master supervisions.

1.6 Projects and Supervisions

I worked in the last decade to implement my vision towards green software ecosystems, and to address the challenge through my research and various projects and supervision. In both of the technical and research dimensions, and across the challenges of observing, understanding and acting towards green software ecosystems, I managed multiple projects, supervised many PhDs and Master students, and published relevant publications in journal and conferences.

Technical dimension: In the technical dimension, my work is driven by the two research directions I presented in my vision: observe and understand, in order to later act and optimize.

- To this end, I worked during my research position at the University of Edinburgh (2014, 2015) on providing analysis of the factors impacting energy consumption in software: mainly whether coding practices (such as using design pattens) have an impact, and whether software and hardware features have a relation with energy consumption.

- At the University of East London (UEL, 2015, 2016), I co-proposed a project and collaborated with two colleagues to further measuring and analyzing software at the feature level using software slicing techniques. The various understandings and analyses I performed during my postdoctoral stays provided key insights to the research community, and led me to think about the larger picture: how to observe and understand software ecosystems.

- I proposed in early 2019 a project to capture relevant contextual data to help in energy management of heterogeneous ecosystems. The project, funded by UPPA E2S, covered a PhD thesis (Houssam Kanso, successfully defended in late 2022), and allowed to work on a contextualization ontology, a crowed-sourced energy benchmarking approach, and a machine-learning autonomous system to manage energy in smart homes.

- At the same time, I started my Joular project aimed to provide models, tools and analysis on software energy. Through this project, multiple software and power models were created, such as PowerJoular or JoularJX. Therefore, I proposed approaches to observe the energy of various components of software ecosystems, from single-board computers (with initial proof-of-concept in a Master 2 internship I proposed and supervised in 2020), to mobile devices (with a cross-team collaboration at my laboratory and a co-supervision of a Master 2 internship in 2021, that led to the PowDroid software), and to various improvements to my research software through three Master 2 student internships and projects in 2023. Also, an industrial collaboration starting in early 2022 also led to an enterprise energy monitoring tool (Demeter, paper published 2023) capable of monitoring a park of computers in a corporate setup.

- As these software observation models and tools matured along with our understanding of software energy, the need to expand software energy analysis was becoming prevalent: observing energy and understanding the factors in play are crucial, but identifying whether there is an energy problem in software is harder. This challenge

is part of my perspectives, and I proposed a project to identify software energy bugs and anomalies, and to detect energy variations through software traces analysis. The project is accepted and funded by the local community of Lacq-Orthez (CCLO) in late 2022 with a PhD thesis that started in early 2023.

- Finally, in this technical dimension, I co-proposed and co-supervised a Master internship in 2022 in a cross-team collaboration at my laboratory, to visualize and automate energy management in servers and systems. This collaboration led to a cross-border project with the University of Zaragoza aimed to work on energy efficiency through load shifting for multi-platform services. The project will be within my mid-term perspectives, and funded a PhD student I co-supervise and who started in late 2023.

Human dimension: In the human dimension of my vision, the quest to observe and analyze users' perceptions and behaviors regarding computing and software started back at my research position at UEL in 2015 and 2016.

- At UEL, I proposed and co-supervised an undergraduate student project on computer behavioral impacts on energy consumption. In this project, we monitored software usage of multiple users from different age groups for a week, and analyzed their usage patterns. This first experiment provided initial encouraging results that end users might have a role to play in energy efficiency of devices and of software.
- In 2019, I started a research direction on analyzing user behaviors in regards to software energy consumption. I led a large-scale scientific experiment, with a control group, where the main field experiment was conducted in 2020 (culminating in a major journal paper published in 2023). The study analyzed the role of energy feedback on user behaviors, and is detailed in my contributions chapters.
- This led me to propose a major research project to further study the impact of various feedback methods on diverse user profiles, and also to automate the feedback in order to sustain behavioral changes through time, and even with user social changes. The project, Behave, was accepted and funded by the French national research agency (ANR) in the JCJC instrument, and started in late 2022 with the recruitment of a PhD thesis. The results of this project, which also includes two Master 2 students and a postdoctoral grant, will be within my mid-term perspective on user behaviors that I detail in the concluding chapter of this manuscript.

In the following sections, I list my PhD supervisions, my Master supervisions, including full research internships and projects, and my research projects and grants.

Supervision

PhD Supervision

- **Nicolas TIREL**, 11/2023-2026 (25%, co-supervision with Prof. Philippe Roose, Dr. Olivier Le Goaër and Prof. Sergio Ilarri (UNIZAR))
  Load shifting and multi-platform services for energy efficiency.
  Funded by UPPA and UNIZAR (University of Zaragoza, Spain).

- **Thi Mai Phuong NGUYEN**, 01/2023-2026 (50%, co-supervision with Prof. Congduc Pham)
  Towards sustainable computing : analyzing and detecting software energy anomalies.
  Funded by CCLO and UPPA.

- **Thomas ZARAGOZA**, 11/2022-2025 (50%, co-supervision with Prof. Ernesto Exposito)
  Reducing the energy footprint of software through users' behavioral changes.
  Funded by ANR JCJC Behave project.

- **Houssam KANSO**, 10/2019-12/2022 (50%, co-supervision with Prof. Ernesto Exposito)
  Contributing to the Energy Efficiency of Smart Homes : An Automated Management Frameworks.
  Funded by UPPA E2S. Defended on December 2022.

Master Research Thesis Supervision

- **Thibaut Soulance**, Master 2, 2024 (100%, sole supervisor)
  Users' perception and awareness of software energy consumption.
  Funded by ANR Behave project.

- **Halima Amekran**, Master 2, 2024 (100%, sole supervisor)
  Eco-design and purchasing or renewal practices of software and hardware resources in national education.
  Funded by GTnum AESON project.

- **Slim Khiari**, Master 1, 2022 (33%, co-supervision with Prof. Philippe Roose and Dr. Olivier Le Goaër)
  Visualize, automate and monitor the energy management of enterprise systems.
  Funded by LIUPPA under the The Green Campus Bot project-team.

- **Fares Bouaffar**, Master 2, 2021 (50%, co-supervision with Dr. Olivier Le Goaër)
  Measure the energy consumption of Android applications: what solutions for software developers?
  Funded by CPSA and GL research teams.

- **Kamar Kesrouani**, Master 2, 2020 (100%, sole supervisor)
  Model the energy consumption of connected devices.
  Funded by UPPA E2S.

Master Project Supervision

- **Geoffrey Herman**, 2 months, 2024 (50%, co-supervision with Dr. Clément Quinton (University of Lille)).
  REST API for Power Models.

- **Oumar Kaba**, 1,25 months (5 weeks), 2023 (100%, sole supervisor).
  Development of a graphical interface for JoularJX.

- **Maxime Coschiera, Thomas Ergaud, Yanis Monnet, Hugo Pierre, and Seraphin Wattenne**, 1,25 months (5 weeks), 2023 (50%, co-supervision with Dr. Olivier Le Goaër).
  Ranking of Android applications by their energy consumption.

- **Jérémy Nison**, 2 months, 2023 (50%, co-supervision with Dr. Clément Quinton (University of Lille)).
  Development of version 2.0 of JoularJX.

- **Thomas Liard**, 2 months, 2023 (33%, co-supervision with Dr. Clément Quinton (University of Lille) and Dr. Olivier Le Goaër).
  Analyzing the energy consumption of design pattern.

Research Grants

- **ANR JCJC, 2022. BEHAVE! Reducing the energy footprint of software through users' behavioral changes**.
  The Behave project is a fundamental and applied research project that aims to study the role of end users in software energy reductions, and drive users' behavioral changes to achieve that goal.
  The project started in November 2022 with the recruitment of Thomas ZARAGOZA (PhD student), and includes funding for a postdoctoral researcher, and two master 2 research internships. Total funding: 219k€.

- Cross-border UPPA-UNIZAR grant, 2023.**Load shifting and multi-platform services for energy efficiency**.
  This project aims at evaluating the energy consumption of applications and hardware and proposing green redeployment scenarios, while also informing users of their consumption in order to influence it for better habits. This grant financed Nicolas Tirel's PhD contract.

- CCLO (Communauté de communes Lacq-Orhtez) grant, 2022. **Towards sustainable computing : analyzing and detecting software energy anomalies**.
  This project aims to detect, through software traces, software energy anomalies, with an aim to understand and reduce the energy consumption of software.
  This grant financed Thi Mai Phuong NGUYEN's PhD contract.

- E2S UPPA grant, 2019. **Capture and propagate contextual green software data in cyber-physical environments**.
  This project aims to understand and capture contextual data from software and other devices and actors in cyber-physical environments, and apply autonomous actions in order to improve the energy efficiency of the environment.
  This grant financed Houssam KANSO's PhD contract.

- Microsoft Azure grant, 2015. **Feature-based Energy Measurement in the Cloud**. With Syed Islam and Rabih Bashroush. University of East London.

1.7 Publications

The following is the list of my publications on research activities explored after my PhD defense. It includes my publications during my two postdoctoral contracts in the University of Edinburgh and the University of East London, and my publications as an associate professor at the University of Pau and Pays de l'Adour.

Journal Publications

- **The Impact of Green Feedback on Users' Software Usage**. Adel Noureddine, Martín Diéguez Lodeiro, Noelle Bru, and Richard Chbeir. In IEEE Transactions on Sustainable Computing journal (T-SUSC). Volume 8, number 2, pages 280-292. April-June 2023. SJR-Q1.

  Paper also accepted for the journal-first track at the 9th International Conference on Information and Communications Technology for Sustainability (ICT4S 2023). Rennes, France, June 2023.

- **Automated Power Modeling of Computing Devices: Implementation and Use Case for Raspberry Pis**. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. In Sustainable Computing: Informatics and Systems journal (SUSCOM). Volume 37. January 2023. SJR-Q1.

  Paper also accepted for the journal-first track at the 9th International Conference on Information and Communications Technology for Sustainability (ICT4S 2023). Rennes, France, June 2023.

- **An Automated Energy Management Framework for Smart Homes**. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. In Journal of Ambient Intelligence and Smart Environments. 2023. SJR-Q2

- **A Review of Energy Aware Cyber-Physical Systems**. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. In Cyber-Physical Systems journal (TCYB). 2023. SJR-Q2

- **Software Energy Efficiency: New Tools for Developers**. Adel Noureddine, and Olivier Le Goaër. In ERCIM News, No. 131. October 2022.

- **Data Center Energy Demand: What Got Us Here Won't Get Us There**. Rabih Bashroush, Eoin Woods, and Adel Noureddine. In IEEE Software. Volume 33, Issue 2, pages 18-21, March-April 2016. SJR-Q1

- **Monitoring Energy Hotspots in Software**. Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. In Automated Software Engineering Journal (ASEJ). Volume 22, Issue 3, pages 291-332, September 2015. SJR-Q1.

Peer-Reviewed Conference Publications

- **Analyzing Software Energy Consumption**. Adel Noureddine. In the 46th International Conference on Software Engineering (ICSE 2024), Technical Briefing track. Lisbon, Portugal, April 2024. Core A*.

- **Studying the Impact of User Feedback on Software Energy Consumption**. Thomas Zaragoza, Adel Noureddine, and Ernesto Exposito. In the 22nd International Conference on Pervasive Computing and Communications (PerCom 2024), WIP track. Biarritz, France, March 2024. Core A*.
- **Demeter: An Architecture for Long-Term Monitoring of Software Power Consumption**. Lylian Siffre, Gabriel Breuil, Adel Noureddine, and Renaud Pawlak. In the 17th European Conference on Software Architecture (ECSA 2023). Istanbul, Turkey, September 2023. Core A.
- **PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools**. Adel Noureddine. In the 18th International Conference on Intelligent Environments (IE2022). Biarritz, France, June 2022. Core B.
- **A Preliminary Study of the Energy Impact of Software in Raspberry Pi devices**. Kamar Kesrouani, Houssam Kanso, and Adel Noureddine. In the 29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2020). Bayonne, France, 2020. Core B.
- **gUML: Reasoning about Energy at Design Time by Extending UML Deployment Diagrams with Data Centre Contextual Information**. Nigar Jebraeil, Adel Noureddine, Joseph Doyle, Syed Islam, Rabih Bashroush. In IEEE World Congress on Services (SERVICES 2017). Honolulu, United States, June 2017. Core B.
- **A Study on the Influence of Software and Hardware Features on Program Energy**. Ajitha Rajan, Adel Noureddine, and Panagiotis Stratis. In International Symposium on Empirical Software Engineering and Measurement (ESEM'16). Ciudad Real, Spain, September 2016. Core A.
- **Jolinar: Analysing the Energy Footprint of Software Applications**. Adel Noureddine, Syed Islam, and Rabih Bashroush. In International Symposium on Software Testing and Analysis (ISSTA'16), demonstration track. Saarbrücken, Germany, July 2016. Core A.
- **Measuring Energy Footprint of Software Features**. Syed Islam, Adel Noureddine, and Rabih Bashroush. In International Conference on Program Comprehension (ICPC'16). Austin, USA, May 2016. Core A.
- **MUSA: A Scalable Multi-Touch and Multi-Perspective Variability Management Tool**. Muhammad Garba, Adel Noureddine, and Rabih Bashroush. In Tool Demos track at WICSA/CompArch 2016. Venice, Italy, April 2016. Core A.
- **Optimising Energy Consumption of Design Patterns**. Adel Noureddine, and Ajitha Rajan. In New Ideas and Emerging Resuts (NIER) of the 37th International Conference on Software Engineering (ICSE'15). Florence, Italy, May 2015. Core A*.

Peer-Reviewed Workshop Publications

- **Software Energy Efficiency: Between Technical and Human Approaches** (Talk). Adel Noureddine. In Green Software and Human Actors: design, code, and behavior workshop, co-located with ICT4S 2023 conference. Rennes, France. June 2023.

- **Is well-tested code more energy efficient?** Adel Noureddine, Matias Martinez, Houssam Kanso and Noelle Bru. In the 11th Workshop on the Reliability of Intelligent Environments (WoRIE'22)/(workshop at Intelligent Environments conference 2022. Core B). Biarritz, France, June 2022.

- **A Preliminary Study of the Impact of Code Coverage on Software Energy Consumption**. Adel Noureddine, Matias Martinez, and Houssam Kanso. In the Second International Workshop on Sustainable Software Engineering (SUSTAINSE)/(workshop at Automated Software Engineering conference 2021. Core A*), Melbourne, Australia, November 2021.

- **PowDroid: Energy Profiling of Android Applications**. Fares Bouaffar, Olivier Le Goaer, and Adel Noureddine. In the Second International Workshop on Sustainable Software Engineering (SUSTAINSE)/(workshop at Automated Software Engineering conference 2021. Core A*), Melbourne, Australia, November 2021.

Chapter 2

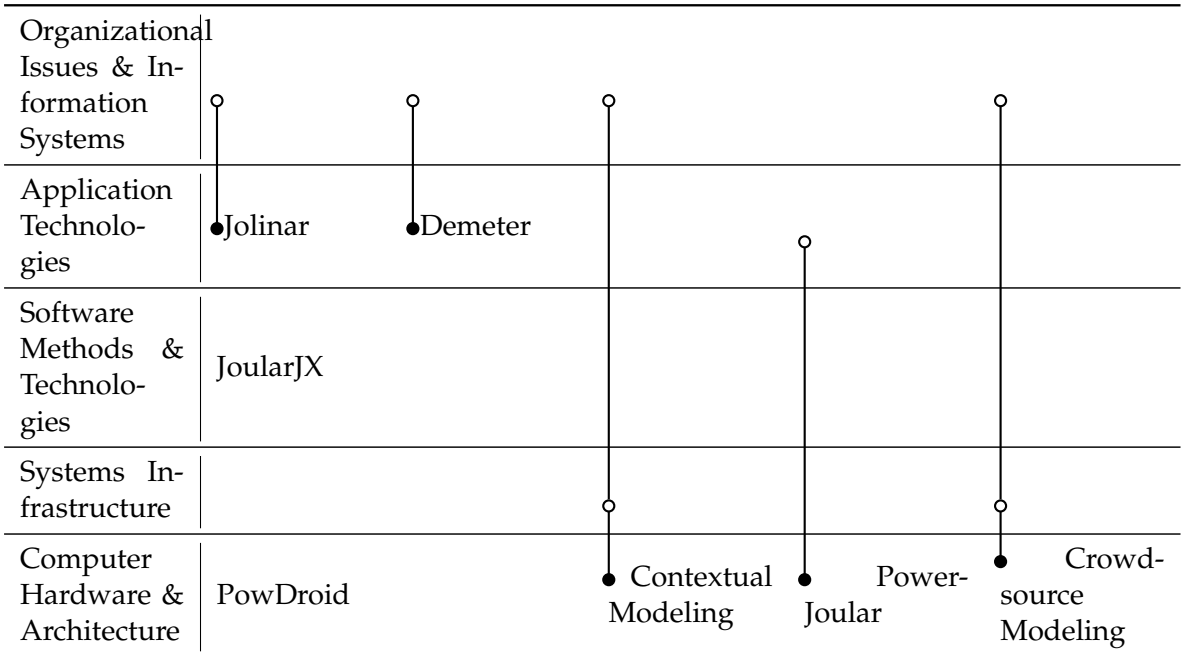
Observing Green Software Ecosystems

Contents

2.1 Introduction

Software ecosystems are complex and heterogeneous systems, with many components and actors interacting with each other. This *music of creation* between all the players in the environment is what makes such environments smart, and computationally useful. It is therefore important to observe this environment, with all its complexity, in order to properly and accurately get its power and energy consumption.
In this new reality where software is everywhere, my contributions in this chapter covers observing, through modeling and measuring, software everywhere, *i.e.*, ubiquitously. As mentioned in my vision in the introduction of this manuscript, I adopt the layers identified the by ACM Computing Curricula [acm05] in their *Problem Space of Computing* model, in order to categorize the layers and refine what *everywhere* means.

Table 2.1: An overview of my contributions in observing green software ecosystems



My contributions are detailed in the following sections, as adaptations of my different scientific publications. Table 2.1 summarize my contributions using ACM's aforementioned model. Many of my contributions target multiple layers, with a main layer (symbolized with a dark circle), and secondary layers (symbolized with an open circle).
In particular, I present two main research axes: modeling green software systems, and measuring those systems.
In modeling, I aim to create a clear picture of software and their ecosystems in regards to energy consumption. This means observing software, hardware, and the computing and non-computing systems and environment in order to collect relevant data to generate this *energy picture*. Specifically:

- Software ecosystems are complex due to the heterogeneity of devices, actors and a plethora of generated data, including contextual information crucial for any viable en-

ergy management system. To make sense of all this complexity, we propose, in Section 2.2.1, an approach to model the complexity of devices, mainly in cyber-physical systems, through an energy-aware extension of the SAREF ontology.

- In Section 2.2.2, we propose an approach to automate the generation of power models for any computing device or electric appliance. The approach leverages the power of the crowd, and provides a distributed architecture aimed to collect benchmark data from multiple users and devices, generate empirical and validated power models, and provide an interface for monitoring clients to get accurate and up-to-date models at any moment.

- Our approach is validated in Section 2.2.3 through an implementation for modeling the power consumption of single-board computers, and in particular, of the CPU of Raspberry Pi devices with a very low margin of error.

- Finally, in Section 2.2.4, we explore modeling the power consumption beyond computing devices, towards modeling the energy of devices and objects such as a smart bulb or a computer monitor.

In measuring, my goal is to provide tools, architectures and approaches for developers, practitioners, managers, end users and also automated systems, to measure the energy consumption of software and its ecosystem at every layer possible.
To achieve software energy efficiency, the role of developers is paramount, and having proper tools to diagnose energy consumption at different granularities and layers is an important requested need as shown by multiple studies [BWN16, PHAH16, MBZ+16a, PC17a].
On the other spectrum, users are interacting with software environments without having direct technical control over them. This includes end users, but also procurers, managers and other non-technical practitioners. For them, it is important to observe energy consumption in a user-friendly way.
My contributions cover multiple layers, from computing hardware and architecture (computers, servers, smartphones, IoT devices), to software methods (source code, execution branches), and applications on various operating systems. These contributions are aimed for developers and practitioners, but also for managers and end users. In particular:

- PowerJoular, in Section 2.3.1, is a cross-platform tool aimed to monitor the power consumption of hardware components, processes and applications. It leverages multiple measurement techniques, including the power models we generated in the automated power modeling approach, and delivers a command-line system utility for runtime measurements. PowerJoular is aimed for computers and servers (including in cloud environments) but also for cyber-physical and IoT systems.

- PowDroid, in Section 2.3.2, is a tool aimed for measuring the energy consumption of Android devices and applications. Smartphones are widespread today, and their energy consumption is often abstracted with battery percentages and the heterogeneity of devices. PowDroid leverages the Android platform, and provides an easy-to-use command line tool to collect, process and analyze data from a smartphone, and generates a timestamp-based energy report.

- Jolinar, in Section 2.3.3, is an easy-to-use tool with a graphical user interface allowing users to quickly start monitoring a specific application with ease. The tool provides detailed energy analysis on the energy consumption of the application, and also per-hardware component (CPU, disk and memory).

- Demeter, in Section 2.3.4, is an advanced software architecture providing energy monitoring capabilities aimed for long-term observations (in weeks or months). The distributed architecture encompasses a client tool aimed to collect data about every application running and provide energy consumption information for applications, and, optionally, for each hardware component (CPU, network and disk). Energy data can then be shared and stored in a centralized server that provides guidances and summary analysis for users about their energy patterns.

- JoularJX, in Section 2.3.5, is a cross-platform and cross-OS software aimed to measure the energy consumption of applications' source code. In addition to measuring individual methods and functions in a Java application (and, subsequently, classes and packages), it can measure energy at runtime allowing following energy of every method through time. JoularJX can also measure the energy consumption of individual execution branches, thus allowing to map the energy consumption of the call tree of an application. A graphical user interface is also provided to analyze and display the energy data, as JoularJX generates lots of various data providing a very detailed diagnosis of the energy consumption of applications.

My contributions cover observing, through modeling and measuring, a large spectrum of the heterogeneity of software ecosystems, from fine-grained source code and execution branches measurements, to applications in multiple platforms, hardware components, end users' devices, and servers environments. In the following sections, I present in detail my different contributions in observing green software ecosystems.

2.2 Modeling Green Software Systems

My contributions in modeling the energy consumption of software systems aim to map and make sense of the power consumption of devices and software, along with other relevant data useful for measuring, analyzing or managing energy. In this section, I present my contributions aimed to map *green* data and generate relevant knowledge of software ecosystem, including generating power models for devices and hardware components.
First, I present our approach to contextualize green data and generate green knowledge of cyber-physical systems through a green ontology extension. With this ontology, we can get a view of the energy mapping of objects and devices, using their energy sensors or an estimation power model. The latter is often absent and need to be built for the specific device or hardware component. This is where I present our approach to create power models for devices leveraging the power of the crowd with an automated architecture. An implementation for single-board computers, such as Raspberry Pi devices, is then presented with an empirical validation and analysis of the impact of many factors. Finally, I present our vision beyond computing devices and how our approach can model and map energy for all sorts of devices and configurations.

2.2.1 Contextual Modeling

The content of this section is adapted from the following publication:
An Automated Energy Management Framework for Smart Homes. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. Journal of Ambient Intelligence and Smart Environments (JAISE). 2023.

Industrial and residential sectors consume the majority of final electrical energy [IEA19]. In residential sectors, conventional electrical systems and devices are not very eco-friendly. Some are considered high energy demanding (*e.g.*, water heaters, fridge, and laundry machines), and others are smaller devices and do not consume as much (e.g., internet modem, led lamp, and mobile phones). However, their large number leads to high energy consumption of the entire environment. Therefore, optimizing a large number of these small devices can have a significant impact. These devices consume energy directly from the electrical grid (e.g., television) or by storing energy in an internal battery (e.g., mobile phones or laptops).
The introduction of IoT and smart devices led to the concept of a smart home. A smart home is a residence equipped with advanced technology devices that allow homeowners to remotely control lighting, appliances, thermostats, security systems, and Heating, Ventilation, and Air Conditioning (HVAC) systems.
Many IoT devices now offer built-in intelligence capabilities (*e.g.*, software) that enables them to communicate with each other and connect to the internet. These intelligent devices help automate tasks, reduce cost, and save energy. A large number of devices integrate sensors allowing them to produce data describing the state of a device or the behavior of a person. In addition, these devices are equipped with actuators accomplishing tasks impacting the environment. Therefore, they have the potential of making homes greener and more sustainable.
In smart homes, several heterogeneous devices are connected using different protocols and sharing all kinds of data. They interact with each other to accomplish specific tasks, in addition to their interactions with humans which adds an extra level of complexity to the system. Smart homes also host a growing number of devices with heterogeneous architectures, types or interfaces. However, these environments are dynamic as changes can occur to existing devices such as software updates, hardware updates, location changes, etc.
Each device has its power consumption that needs to be estimated or measured in order to be able to estimate global energy consumption and implement the appropriate power management strategies. However, getting energy consumption is not sufficient easily done as most devices don't have an integrated energy sensors or connected to a power meter, and the remaining ones don't have a model estimating their energy consumption. Also, to properly observe and then understand and manage the energy of devices and the system as a whole, contextual data need to be collected about these devices in order to create an energy-aware knowledge base of the system. We use an ontology to model the contextual energy data of these devices.

Ontology Knowledge Base: An ontology is a defined set of concepts and relationships used to represent the meanings and a shared understanding in a certain domain [NM01]. Ontologies are considered highly effective for knowledge modeling and information retrieval. They provide a common representation and definition of the concepts and the rela-

tionship between them. In addition, ontologies support inference to discover new relationships and knowledge. They are scalable and their reuse is possible. Therefore, an ontology is used to formally represent the environment's knowledge.
Smart Applications REFerence Ontology (SAREF) [CBB+12] is a well known and adopted ontology. It focuses on the concept of a device as a central concern and defines its properties. These properties can be measurable, controllable, or both. In addition, it defines the type of the device (i.e., appliance, sensor, actuator, meter, or HVAC).
Ontologies do not usually consider power estimation and measurement. Therefore, it is necessary to extend one of these ontologies to represent this energy concept. The way these concepts are defined in SAREF provides a compatible and easy way to represent the knowledge base for reinforcement learning model generation.

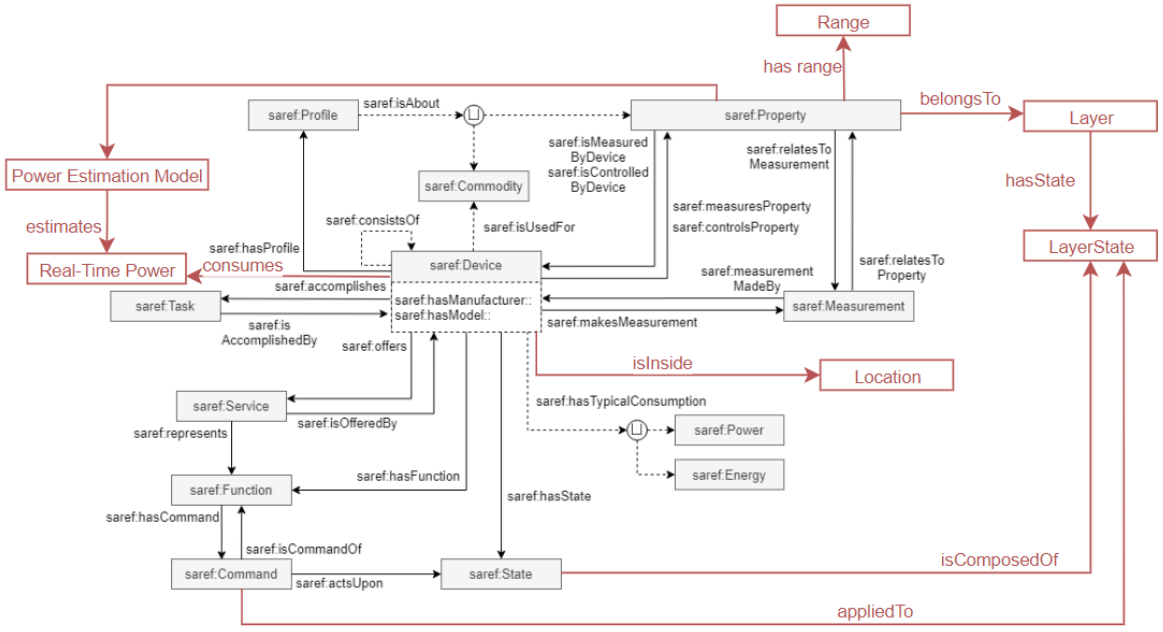


Figure 2.1: SAREF Extention Overview

In order to include the real-time power and different layers of a device into the knowledge base, we propose an extension of the SAREF ontology with energy-aware contextual information as seen in Figure 2.1. The extension includes the following classes, attributes, and relations:

- *ecps:Location* is the geographic space in which a device is implemented. It helps build a contextual understanding of the environment and identify devices that share the same location.

- *ecps:Layer* is the structural level of the system to which a property belongs. It can be application, computational, network, storage, and contextual.

- *ecps:LayerState* is the state in which a specific layer is running. It composes the state of the device.

- *ecps:Power Estimation Model* is the mathematical model that takes as input the properties of a device and returns the power consumption.
- *ecps:Real Power* is the real-time power consumption of a device. It can be measured directly by a hardware wattmeter or estimated using an estimation model.
- *ecps:hasRange* represents the accepted values for each property. It takes the form of a list of accepted values or minimum and maximum accepted values. Ranges specified with a minimum and maximum can include a step that defines a fixed value increased or decreased from a property value.
- *ecps:deviceId, ecps:deviceName, ecps:propertyId, ecps:propertyName* represent respectively the id and name of a device used to identify a device from another. In addition to the id and name of a property.

(a) Class Hierarchy

(b) Data Property Hierarchy

(c) Object Property Hierarchy

Figure 2.2: SAREF Ontology Extension Implementation in Protégé

Ontology Implementation Our ontology is defined using the Web Ontology Language (OWL) [Bec09], a language of knowledge representation for ontologies in the Protégé ontology editor and knowledge management system [Mus15]. As seen in Figure 2.2, a series of

classes, data properties (attributes), and object properties (relations) are defined in order to make the ontology suitable with the agent generator. Newly introduced concepts' names start with the prefix *ecps*.
Our contextual modeling, through our energy-aware extension of the SAREF ontology allows mapping and observing the energy dimension of complex cyber-physical systems and of heterogeneous devices and objects.
However, not all devices have a power sensor or a power model. For these devices, which are the vast majority, we need to build and generate models estimating their energy consumption. Such model needs to be accurate, automated and crowd-sourced as the variety and heterogeneity of devices, software and configurations is too large for manual model generation. In the next section, we present our automated and crowd-sourced power modeling approach.

2.2.2 Crowd-sourced Power Modeling

The content of this section is adapted from the following publications:
Automated Power Modeling of Computing Devices: Implementation and Use Case for Raspberry Pis. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. In Sustainable Computing: Informatics and Systems journal (SUSCOM). Volume 37. January 2023. [KNE23]
CPU Power Bench: An Automated Benchmark Tool for Power Estimation in Single-Board Computers. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. In the 9th International Conference on Information and Communications Technology for Sustainability (ICT4S 2023). Rennes, France, June 2023.

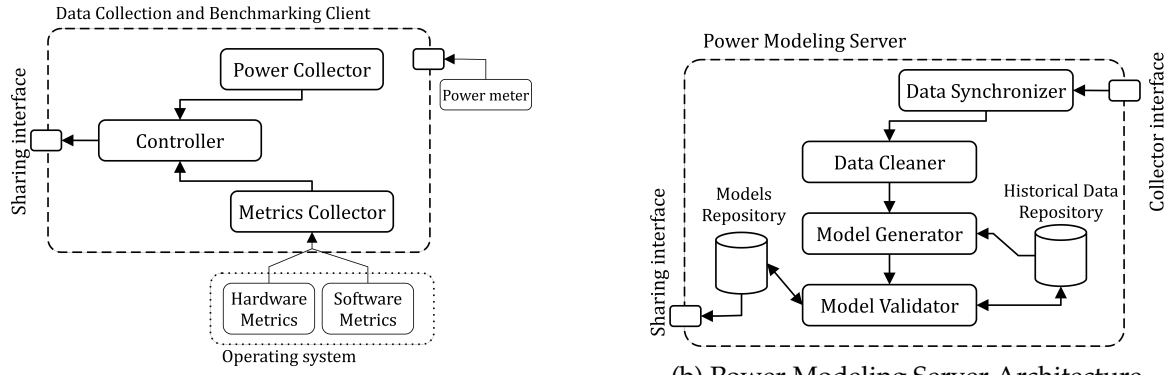
Monitoring the power consumption of smart and connected devices is a challenging task with heterogeneous devices and a variety of hardware and software configurations. In order to accurately monitor these devices and follow the speedy changes in their configuration, a new approach is needed. We present an automated architecture and approach to empirically generate power models for a large set of devices. Our approach allows conducting automated benchmarks to collect power data and metrics, generating or updating accurate power models, and allowing software tools to query and retrieve the most accurate and up-to-date power model of a specific device configuration. We also present a proof-of-concept implementation for modeling the power consumption of single-board computers, in particular, Raspberry Pi devices. Finally, we conduct a comprehensive experiment, modeling the entire current lineup of Raspberry Pi devices with error margins as low as 0.3%, and then we discuss the impact of multiple device configurations on power consumption.
Connected devices have different CPU architectures than current and legacy computing devices (*i.e.*, ARM/RISC compared to x86/CISC architecture), and have a huge variety of hardware and software configuration and components. External power meters can provide accurate power measurements for specific workloads and environments (such as in [ASBSVRSQ16]). However, these meters are costly financially, scale badly for a large park of devices, have a time-consuming setup, and require physical access to each device. Therefore, it is important to provide software-based power models for these devices. However, without embedded power sensors or constructors' power models and API, it is challenging to provide accurate power models for this variety of device configurations. In addition, monitoring the power

consumption at run-time (in addition to other performance metrics) helps software developers to detect misbehaving software, or specific power drains due to a particular hardware configuration.
Current power estimation techniques are either based on mathematical formulas (such as in [NRS15]), or on a static data set used to generate an empirical model (such as in [RWB+17]). In addition, such models target a single device release with no way to estimate other revisions or variations of the device without manually conducting the experiments again.
Our main motivation is to provide an automated approach to model the power consumption of various devices, while allowing the models to be updated, extended, improved, and shared. We argue that such an approach leads to democratizing power comprehension of hardware and software in different devices and environments.
Providing an accurate and automated approach to solve these questions is challenging, and in particular:

- **Heterogeneous environment**: monitoring the power consumption of heterogeneous devices is challenging: different hardware configuration, architectures, revisions, or cooling. Also, software heterogeneity has an impact on power, such as the operating system, software workloads, or libraries.
- **Empirical validity**: generating empirical power models requires a large set of valid data and metrics, which in many cases is difficult to collect in statistically sufficient numbers.
- **Automated power modeling**: automating such an approach with large heterogeneity and an empirical backbone requires a crowd-sourcing architecture that facilitates benchmarking devices and data collection. Automated safety and security checkups should also prevent erroneous or malicious data to impact the accuracy of the generated power models.

We present an automated approach and architecture to empirically generate power models for, potentially, unlimited devices and configurations. Our approach provides always up-to-date and accurate power models with low error rates. The approach follows a crowd-sourcing architecture where benchmarking components can run on any device, generate empirical data, and lastly, our power model generator component will generate an accurate power model for the specific device, or improve the model if a previous one already exists. Monitoring software can connect to our architecture to query and retrieve the most accurate and up-to-date power model for their devices. The data collection and model generation phases are relatively fast as they only take a few minutes. Once the model is generated, power consumption can be estimated with negligible overhead in real time. With time, the accuracy of the model can be improved as more data are collected and fed to generate more accurate regression models.
The main novelties of this work can be summarized as follows:

- Energy estimation models are always up-to-date due to the continuous data benchmarking as new models are automatically generated when necessary.
- Model generation is done in an automated manner from the data collection to the testing (with human intervention needed only to start the benchmarking process.
- Our proposed approach is a collaborative one where benchmarks can also be crowd-sourced, and energy estimation models are shared across users and devices.



(a) Benchmarking Client Architecture

(b) Power Modeling Server Architecture

Figure 2.3: The architecture of our crowd-sourced approach

General Architecture: Our architecture aims to generate an always up-to-date and accurate power model for various computing devices, such as servers, PCs, single-board computers, or embedded and IoT devices. We achieve this automatic generation with a multi-component architecture aimed to collect and process power data and metrics, apply machine learning algorithms, and generate an updated, more accurate, power model.

Concretely, our architecture, is composed of three distinct but complementary components: a data collection and benchmarking client, a machine learning and power modeling server, and a power estimator client. Simply put, our benchmarking client will collect run-time software, hardware, and power metrics, which are then sent to the power modeling server. The latter will then generate empirical power models based on the current and previously collected metrics, using machine learning techniques. Finally, the power estimator client will query the server for the most up-to-date and accurate model for the device it's running on, and use that model to estimate the power consumption of the device.

Each component of our architecture can be independently implemented as we aim to provide a decentralized and decoupled architecture. For instance, our generated power models can be used by third-party power estimator clients to provide run-time and live power estimations, or used by hardware management software to supervise the power consumption of a set of devices.

In addition, the architecture is designed to maintain a high level of flexibility to manage: new devices introduced to the environment, changes occurring to the existing ones (such as OS or kernel updates), new metrics to consider and collect, or changing the model generation algorithm.

Data Collection and Benchmarking Client Architecture: Figure 2.3a presents the architecture of our data collection and benchmarking client. Its main role is to collect software, hardware, and power metrics of run-time and real-world workloads. The collected data will then be used as a training and validation data set for the machine-learning algorithms in our server.

Our benchmarking client first needs to collect the power consumption of the workload. This first step requires an accurate power measurement component, as this metric will be used as the *truth* for this power metric. Therefore, we recommend using a physical power device,

such as an external power meter, or an integrated physical power sensor.
The main components of our architecture are the following:

Power Collector : this component connects to the power measurement sensor and collects run-time power metrics (such as the power consumption in watts, the current, the voltage, or any other power-related metrics), and associates each measure to a timestamp.

Metrics Collector : this component collects various metrics from the operating system, hardware (through the OS), and software. For example, it can collect the number of CPU cycles, the transmitted data packets in a network, the number of storage access requests, or more complex metrics and data (such as software running, network throughput, quality of service, or metadata about the operating system or the software workload).

Controller : this component orchestrates the data collection from the energy and metrics collectors. It controls the frequency of data collection and sharing with the server. It also makes sure that the metric collector is running simultaneously with the power collector.

Sharing Interface : this component's role is to share the collected data to the power modeling server. It can be implemented as a web service API, or through a file-sharing mechanism (on a local network, over FTP, etc.), or any other sharing method understood by the server.

Power Modeling Server Architecture: Figure 2.3b presents our power modeling server architecture. Its main role is to receive generate accurate and always up-to-date power models using the data collected from the benchmarking client. The server acts as a centralized entity towards multiple benchmarking clients, receiving data from multiple similar or different devices.
For example, benchmarking clients can collect data from workloads running on multiple instances of a same-type device. This data collection can also spread across time, and the server will regenerate a new updated power model each time a new data set of metrics is received from a benchmarking client.
The main components of our architecture are the following:

Collector Interface : this component receives the data collected by the benchmarking clients. At this point, the data is received as is, and further processing is handled in the next components.

Data Synchronizer : this component processes the received data (which might be in multiple formats or files), synchronize timestamps between the metric and power data, verifies and synchronizes clock diversion between the timestamp of the computing device and the one from the physical power sensors or meters.

Data Cleaner : this component cleans the synchronized data by identifying and eliminating redundant, erroneous, or out-of-context data. For example, the cleaner tries to identify when the useful workload started and ended, and discards data points outside this range. The cleaner also verifies that the received data is valid, such as if the provided power values are within the power range of the device it was run on, or whether bogus or spamming data are present.

Model Generator : this component generates a power estimation model based on empirical machine learning techniques, using the newly received data along with the data already stored in the server about the particular device. For example, the estimation models can be based on any prediction algorithm (such as regressions, decision trees, neural networks, etc.).

Model Validator : this component validates that the new model is more accurate than the current model stored in the server for the device. For example, using all available data sets (including the newly received ones), it can compare the average error of the new model to the currently stored one, and then keep the more accurate one.

Historical Data Repository : this database contains all the collected data of all devices, benchmarks, metrics, and workloads. The newly received, cleaned and validated data are added to this repository, and therefore contributing to building big data set associating various metrics and data to power consumption, and gradually improving the accuracy of our model generator.

Power Models Repository : this component contains all the generated power models for all devices. Only the most accurate power model is saved to this repository, along with the model and estimation parameters, and the average error of the model. The repository distinguishes between devices, but also between revisions of devices (for example, Raspberry Pi 4B revision 1 and revision 2), between operating systems architectures (32 or 64 bits), etc.

Sharing Interface : this component provides a sharing mechanism for power estimator clients to retrieve the latest up-to-date power model of their device. As with our other sharing interfaces, it might be implemented as a web service API, a format-specific file, or any other sharing mechanism and format.

In the next section, I present an implementation of our approach and architecture aimed towards single-board computers, and in particular with a validation on Raspberry Pi devices.

2.2.3 Implementation for Single-Board Computers

In this section, we present a specific implementation of our architecture aimed to generate accurate power models for single-board computers in general and Raspberry Pi devices in particular. To the best of our knowledge, our implementation is the first to provide a comprehensive set of benchmarks and power models for the entire current set of Raspberry Pi devices.

2.2.3.1 Architectural Implementation

Data Collection and Benchmarking Client Our implementation of the benchmarking client consists of two main components: a workload-generating benchmark and a data collector. In our proof-of-concept implementation, we focus on generating an accurate power model for the ARM processor of Raspberry Pi devices by collecting CPU metrics.
The workload generated by the benchmark consists of applying variable loads on the device's processor. We decide to apply a stress load on the CPU covering the entirety of the load range, i.e., we stress the CPU from 0% all up to 100%, with an incremental step of 5%.

The load is applied for 60 seconds for each percentage step. We also saved the workload timestamp and store everything in a CSV file.
The data collector component is a program collecting CPU metrics. In particular, we collect CPU cycles from the Linux *proc* interface (`/proc/stat`). Then we calculate the CPU utilization (ranging from 0 to 1) and save this data to another CSV file.
We calculate the CPU utilization by calculating the ratio of the busy cycles (CPU cycles in user and kernel mode) with the total cycles which includes idle ones:

$$u[t] = \frac{c_{busy}[t] - c_{busy}[t-1]}{c_{total}[t] - c_{total}[t-1]} \tag{2.1}$$

where:

- $c_{busy}[t]$ is the total number of busy cycles up to time t (busy is here equal to: user + nice + system from */proc/stat*).
- and $c_{total}[t]$ is the sum of $c_{busy}[t]$ and the number of idle cycles $c_{idle}[t]$.

In addition, we collect the actual power usage using a power meter and store the power data in a third separate CSV file. The power data is collected from another device to reduce the impact on the workload and accuracy of the benchmark. The controller makes sure the collection of metrics and power is done with the same frequency and at the same time. It gathers the three CSV files and prepares them to be shared with the power modeling server.

Power Modeling Server We build our power modeling server following a decision algorithm presented in Figure 2.4.
We first collect the three generated CSV files from the client (in our implementation, sharing the CSV file from a common storage location). We then normalize and clean the data (remove irrelevant data points, synchronize timestamps, synchronize the clock if needed, aggregate files, remove inconsistency in the data).

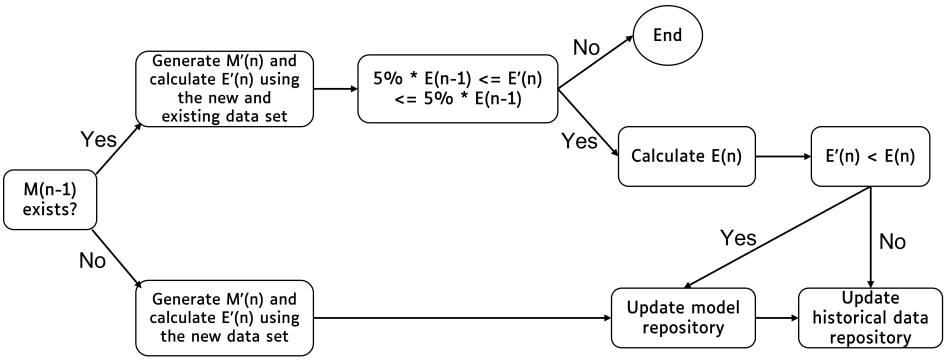


Figure 2.4: Our Implementation Approach for Power Modeling

We then proceed with the generation and validation of our power model. First, we check if an estimation model already exists on the server for the specific Raspberry Pi device and version. If no model exists, then we proceed to generate a new power model ($M'(n)$) using

the collected data, calculate its average error $E'(n)$. Furthermore, we save the new power model to the model repository and the collected data to the historical data repository.
However, if a current power model exists ($M(n-1)$ along with its average error $E(n-1)$), then we proceed with the following process:

- We first read all the data saved in the historical data repository of the specific device, and temporally add the newly collected data to form a new data set. This data set is then used to generate a new power model $M'(n)$, and the average error for this new model is also calculated $E'(n)$.
- If the error rate of this new model $E'(n)$ is outside of an accepted predefined range compared to the previous model $E(n-1)$, then we discard the collected data (as we consider it is not valid), and the server keeps its data and power model. In our implementation, we consider a 5% range around the error rate as a good indicator of whether the data is valid or has been trafficked. The latter can happen if the power data of a device has been mixed with the collected metrics of another, or fake data has been sent, or an error in converting numbers happens in the client.
- If the error rate is within the predefined range, then we calculate the error rate $E(n)$ of the existing model $M(n-1)$ using all the data (historical and new ones). We do this additional calculation because the current error rate $E(n-1)$ has been calculated using the historical data only.
- We then compare the new error rate $E(n)$ of the current model $M(n-1)$, with the error rate $E'(n)$ of the new model $M'(n)$. The model with the lowest error rate will then be stored in the models repository as the new up-to-date and accurate power model of the specific device. And lastly, we store the newly collected data in the historical data repository.

At the end of this process, the server will contain additional data points which will, over time, improve the accuracy of our empirical power model generation.
In our implementation, we use linear and polynomial regression algorithms to generate power models, as a correlation was observed between CPU utilization and power consumption in Raspberry Pi devices. In the next section, we detail the empirical experimentation to validate our implementation and power models.

2.2.3.2 Empirical Validation and Discussions

In this section, we detail the empirical experimentations that validate our approach, implementation, and generated power models.

Experimental Setup Our experimental setup consists of 8 Raspberry Pi devices from different generations and revisions, dating back from 2012 until the latest current model: Zero W, 1B, 1B+, 2B, 3B, 3B+, and 4B. We run our workload on both 32 bits (armv6l and armv7l) and 64 bits (aarch64) operating systems for model 4B (for each of the 2 revisions we used).
We used the same SD card to boot Raspberry OS on all devices, switching operating systems and ARM architecture accordingly. We also automated the benchmark experimentation by adding a boot script to `/etc/rc.local`.

To collect power consumption, we use the PowerSpy2 power meter [3]. PowerSpy2 is a Bluetooth power meter used for advanced and accurate analysis. To reduce interference in the experiments, we use a separate computer to connect to the meter and collect power metrics.
We run all our experiments on Raspberry Pi OS (version based on Debian 9 stretch), with Linux kernel 4.14. Our components and tools are written in Python and run with version 2.8, and in C compiled with GCC 6.3.0. For Raspberry Pi 4B, we the supported version of the OS based on Debian 10 buster with Linux Kernel 5.4, and GCC 8.3.0.
To further reduce interference on the accuracy of our experiments, and as we aim to generate a CPU power model, we disconnected all external peripherals during the workload (including the monitor through HDMI, keyboard, and mouse through USB ports and the network through the Ethernet interface). We also disabled from the operating system all network interface cards (i.e., WiFi and Bluetooth). Furthermore, we made sure that every device was cooled down before running the experiments, as overheating can have an impact on power consumption. Specifically, each Raspberry Pi was disconnected from its power supply until the device was cooled down.

Benchmark Data Collection To collect our CPU metrics, we wrote a minimal C program that read the `/proc/stat` file every second and calculated the CPU utilization. The latter was then saved into a CSV file. As described in our implementation in Section 2.2.3.1, we stressed the CPU from 0% to 100% with a 5% increment.
We initially used the same stress command (or stress-ng) used in the literature [KKN20] to specify a percentage CPU load. However, we noticed that the CPU load was inconsistent, with the actual CPU load altering between 0% or 100% in various time duration, rather than consistently stabilizing at the asked percentage load. Instead, we used a Python script, CPULoadGenerator [4], which consistently stressed the CPU at the asked percentage with a small degree of variation.
For each experiment, we stressed the CPU for 60 seconds for each CPU load step and is followed by a 10-second pause. A 60-second pause precedes each experiment in order to reduce the impact of our script on the results. In total, each experimental benchmark runs on average for about 25 minutes (24 min and 20 sec).
Each benchmark generates a total of 1460 data points. After the cleaning phase, we end up with more than a thousand data points. These are then used to generate our empirical power models. For the purpose of our experiments, we run our benchmark a few times for the Raspberry Pi 3B+ and Raspberry Pi 4B rev 1.2 (64 bits) and ended up with around 5400 data points for the former, and around 2000 data points for the latter. In total, RPi Zero has 666 data points, 1B has 1034, 1B+ has 954, 2B has 1137, 3B has 1105, 3B+ has 5383, 4B 1.1 (32 bits) has 1089 and its 64-bit version has 998, 4B 1.2 (32 bits) has 1017 and its 64 bits has 2014 data points. The difference in the number of data points for each device type is due to our additional experiments (in particular for the 3B+ and 4B, for instance, to compare 32 bits vs. 64 bits, or the validation of the power model generator). Additionally, our cleaning script strips the waiting time between each experiment run of the stress benchmark. As the stress command is not precisely perfect in its timing, the duration of each test might vary by a few seconds, hence the additional data points.

[3] https://www.alciom.com/en/our-trades/products/powerspy2/
[4] https://github.com/GaetanoCarlucci/CPULoadGenerator

Regression Power Models For our experiments, we used two regression algorithms in our server to generate the power models: linear and polynomial regression. We choose these two regression algorithms because of the visible correlation between power and CPU utilization. However, other machine learning techniques and algorithms can be applied to generate different regression models.
Our power modeling server is implemented in Python scripts, automating the data cleaning, synchronization, power modeling, and storing data into the repositories. We, then, generated power models for all different Raspberry Pi models. All generated polynomial models were of a degree between 7 and 9.

Validation of Power Models To validate the accuracy of our power models, we compare the power consumption calculated by our models to the power consumption measured from the power meter. This allows us to calculate the absolute difference between these two values for every data point in the collected metrics. We then calculate an average error using all measurements from all devices with different regression models.
Figure 2.6 presents the measured correlation between CPU load (in percentage), and the power consumption (in watts) for our two regression models (linear and polynomial) and the actual measurements from the power meter. Except for Raspberry Pi 1B and 1B+, our empirical benchmarks show a better fit for the polynomial regression.
This translates into a lower average error for the polynomial model as compared to the linear model for all experiments and Raspberry Pi devices, as seen in Figure 2.5. The average error for the linear models varies from as low as 0.34% for RPi 1B+, to 7.81% for RPi 3B. In contrast, the highest average error for the polynomial model is 3.83% to the RPi 3B+.

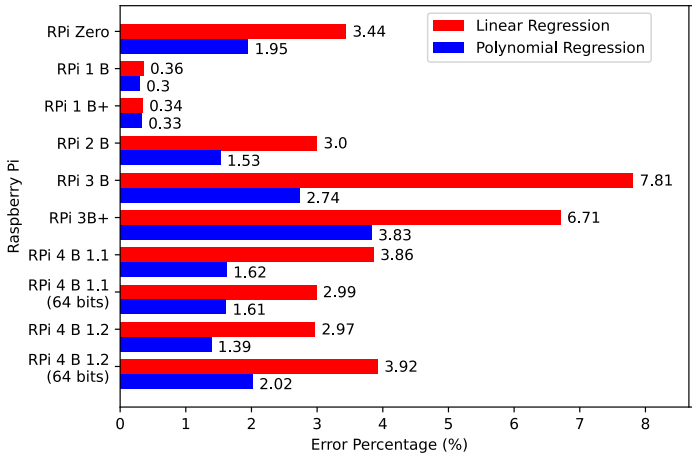


Figure 2.5: Average Error percentage for linear and polynomial regression algorithms per Raspberry Pi

Impact of Raspberry Pi Revisions The Raspberry Pi Foundation often revises its current offering of devices, with modifications to various hardware components. As our implementation focuses on generating power models for the CPU, we suspect that revisions on the

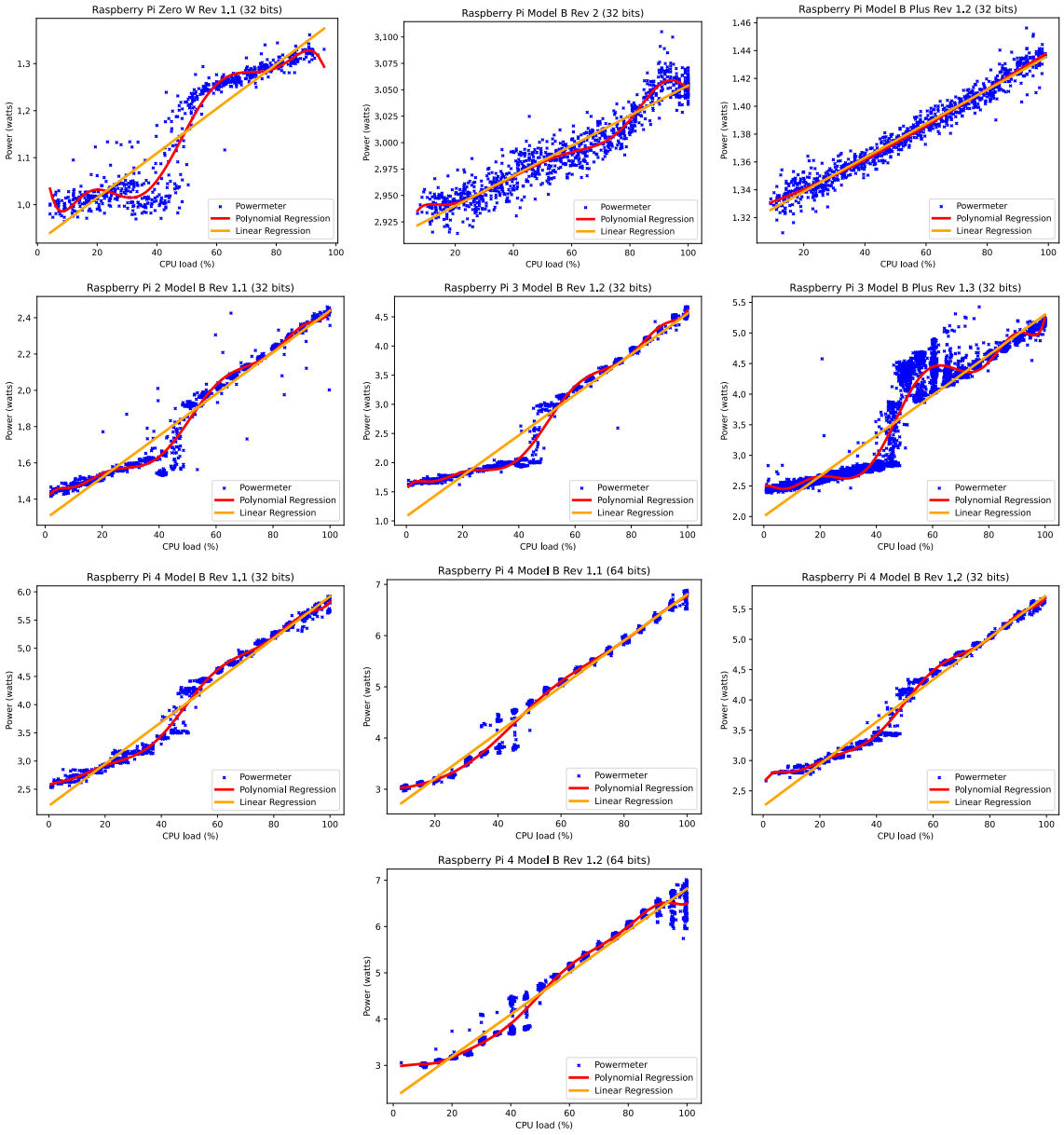


Figure 2.6: Linear vs. polynomial regression power estimation models of the (a) RPi Zero W (b) RPi 1B (c) RPi 1B+ (d) RPi 2B (e) RPi 3B (f) RPi 3B+ (g) RPi 4B 1.1 (h) RPi 4B 1.1 (64 bits) (i) RPi 4B 1.2 (j) RPi 4B 1.2 (64 bits)

USB port or other minor modifications will only have a small impact on the average error of our models.
We proceed to run our benchmarking client and generate power models for the Raspberry Pi 4B revisions 1.1 and 1.2.For this particular device, the differences between revisions 1.1 and 1.2 are minimal and related to the USB-C connector as some electronic components were added and reallocated to fix a fault regarding the connector.
Our experiments show minor differences in the average error between the revisions 1.1 and 1.2 for the same OS architecture (32 or 64 bits). This difference is not negligible for accurate

power measurements as an increase of up to 56% was observed for using the power model of another revision. However, the average error compared to the power meter is still low in both power models when switching revisions (*i.e.*, from around 3% to 4% for rev 1.1 to rev 1.2). Therefore, we recommend generating and using power models for specific revisions, while still allowing estimator clients to use another revision power model if one isn't provided for the specific revision.

Impact of 32- and 64-bit Raspberry Pi Versions Newer Raspberry Pi devices have a 64 bit supported ARM architecture, where users can run either a 32 or a 64-bit operating system. As a stable 64-bit version of Raspberry OS hadn't been released during our experiments, we use its latest beta version (arm64-2020-08-24). Recent experiments had shown that a 64-bit OS on a Raspberry Pi 4 provides a much higher performance compared to a 32-bit OS, up to doubling the performances in benchmarks [Cro20]. Therefore, we suspect that our power models generated in a 32-bit OS would not provide a similar accuracy on a 64-bit OS.
For both Raspberry Pi 4B revisions, we generate power models running our benchmarks on a 64-bit OS.As we suspected, our benchmarks running on a 64-bit OS have, on average, higher-power consumption than the same device running the same benchmarks on a 32-bit OS.
We observe, consistently, that a different architecture highly impacts the accuracy of the generated power models, with a margin of error higher up to more than 5 times. For instance, RPi 4B rev 1.2 32 bits power models are nearly 5 times less accurate when used on the same revision but with a 64-bit OS. These results confirm our hypothesis and the higher performance of a 64-bit OS on supported devices as seen in the literature [Cro20].
Consequently, we recommend using power models generated specifically for the device's architecture.

Impact of Connected Peripherals Raspberry Pi devices are designed to be easily connected to external peripherals via a variety of physical interfaces. To assess the impact of connected peripherals on the validity of our energy models, we conduct the same benchmark experience in two different scenarios on a Raspberry Pi 4B, revision 1.2. In the first scenario, we disconnect all peripherals and follow the procedure mentioned in Section 2.2.3.2. In the second scenario, we launch the benchmark after connecting the Raspberry Pi to a screen using the mini HDMI interface, a USB wired keyboard, and a wireless mouse, and we activate WiFi.
We generate two power models, one for each scenario, and calculate its average error. We observe that both CPU linear and polynomial models have a lower error when disconnecting the peripherals. However, the models generated with the peripherals are still within an acceptable margin below 8%. This proves that our approach can generate CPU power models with an acceptable accuracy even with interference from connected peripherals.
Our experiments also show that the average error when using the power model of one scenario onto the data of the other scenario, *i.e.*, using the power model generated with the peripherals on benchmarking data generated without the peripherals, and vice versa, is higher, but still within a range below 8%. This means our CPU power models generated in an ideal benchmark setup (with peripherals disconnected), are still accurate enough to, not only estimate the power of the CPU, but to estimate the power of the Raspberry Pi device (as the CPU is shown to be the most power-consuming component).

Raspberry Pi devices are often used in a headless server setup (such as to control industrial machines, control heating or lightning in a smart home or city, a web or NAS server, etc.). In these situations, our approach generates accurate CPU power models without interference from peripherals.

Comparison to the State of the Art Models To assess the validity of our models and our automated approach in generating up-to-date models, we compare our generated power models to the ones provided by the state of the art. In particular, we tested and compared our models to two models: PowerPi [KGH14] and EMM [KKN20]. Both models announced an average error rate of 1.2% and 1.25%, respectively. However, in our experiments, we found that both models have a much higher error rate: 14.56% for PowerPi and 40.76% for EMM. In contrast, our linear models provide an error rate of 3% for RPi 2B, and 6.71% for RPi 3B+, and even much lower error rates for our polynomial models (1.53% and 3.83%, respectively).

Overhead of Regression Models Most of our generated polynomial models have a degree of 9, requiring calculations up to the power of 9. As these models have a much higher accuracy than the linear ones, we compared the overhead of running both models in our implementation of the power estimator client.
Our client is a minimal C program reading CPU cycles, calculating CPU utilization, and applying the power models. The client monitors power consumption at run-time and provides a power value every second. We compare the power overhead of running the client with both power models, and also in comparison to the base power consumption without our client. We conduct our experiment on two Raspberry Pi models: RPi Zero W for a low-power device, and the 3B+ for a more recent higher-power one.
For both devices, although we observe some rare data points of higher diversion, the overall difference is quite low with an absolute average difference of 0.084 watts (corresponds to a relative difference of 1.32%) for the RPi Zero W, and only 0.109 watts (corresponds to a relative difference of 0.72%) for the RPi 3B+. These numbers show a negligible overhead for using our polynomial models over the linear ones, even on low-power devices. We, therefore, recommend using the polynomial power models even for run-time power monitoring.

Validation of the Power Model Generator The core idea of our power modeling generator, described in Section 2.2.3.1, is to allow third party benchmarking clients to send new benchmark metrics to further improve the accuracy of the generated power models. In this section, we validate our approach with a breakdown of a step-by-step experiment of the linear power model in a Raspberry Pi 3B+ device.
We run our experiment 10 times, emulating 7 benchmarks sending data incrementally one after the other for one particular device (Raspberry Pi 3B+). Our server implementation then runs our model generator algorithms and keeps the best accurate power model in every step. The result of this breakdown is outlined in Table 2.2a for the linear model, and Table 2.2b for the polynomial model. Each row of the table represents a new server iteration (receiving new data, data normalization and cleaning, validation, power model generation, and comparison, etc.). $M'(n)$ indicates the newly generated power model in the server, along with its error rate $E'(n)$. $E(n)$ is the error rate of the currently saved power model using all the data. And $M(Server)$ is the power model that is saved after the current iteration.

(a) Linear model

Step	E′(n)	E(n)	M(Server)
S1	**7.64**%	-	$S1$
S2	**6.61**%	7.07%	$S2$
S3	**6.13**%	6.23%	$S3$
S4	6.58%	**6.51**%	$S3$
S5	6.72%	**6.66**%	$S3$
S6	6.45%	**6.43**%	$S3$
S7	**6.25**%	6.27%	$S7$

(b) Polynomial model

Step	E′(n)	E(n)	M(Server)
S1	**3.68**	-	$S1$
S2	**3.73**	4.15	$S2$
S3	**3.63**	3.79	$S3$
S4	3.8	**3.71**	$S3$
S5	3.82	**3.73**	$S3$
S6	3.78	**3.71**	$S3$
S7	3.74	**3.69**	$S3$

Table 2.2: Breakdown of our Power Model Approach (RPi 3B+, bold for the selected model)

As we can observe in this breakdown, the newly generated model is not always the most accurate. For instance, for the linear model, at step 4, the new model has a worst average error (6.58%) compared to the current model (6.51% calculated with all data including the new ones). This also happens in steps 5 and 6. However, across multiple iterations, we observe a decrease in the average error, which started at 7.64%, then gradually went down up to 6.25% after only 7 benchmarks and model iterations. We observe a similar breakdown for the polynomial model with an improvement of the error rate and our approach uses the most accurate power model on every step.
We argue that the more benchmark data we have, the more our architecture and approach will provide empirical power models with better accuracy.

Use Case of Remote Power Monitoring A use case illustrating the advantages of our approach is remote power monitoring of a park of deployed Raspberry Pi devices. In particular, smart devices, such as Raspberry Pis, send collected metrics and their status (including CPU statistics) to a central monitoring service.
An example of the latter is Zabbix [5], an open-source server used for real-time monitoring of a large number of clients. In each Raspberry Pi client, a Zabbix agent is installed to allow remote monitoring and management. It can send CPU utilization and many other metrics for the device in real time. We developed a prototype plugin for Zabbix to integrate our power models and architecture into its web interface. Our plugin updates the power models of the monitored devices by connecting to our power modeling server. It also tweaks the Zabbix web interface to calculate the power consumption of monitored devices, in real time, and displays them along with the CPU utilization.
With our approach and power models, remote management tools can efficiently, accurately and with no overhead on the monitored devices, monitor the power consumption in real time. It also allows these tools to always have updated and accurate power models, and to support new device power models easily by just calling our sharing interface.

Epilogue: We presented our crowd-sourced architecture to automate the generation of always up-to-date and accurate power models for a variety of devices. We implemented a

[5] https://www.zabbix.com/

proof-of-concept client and server to automate the generation of power models for Raspberry Pi devices, where the generated power models have high accuracy with error rates as low as 0.33%.
Our approach is cable of generating power models for various devices, and hence provide a key building block in the observation of the energy consumption of complex software ecosystems by starting to measure the devices they run on with all their heterogeneity.
But these ecosystems and cyber-physical systems include more than just computing devices. Other devices, either *dumb* ones or smart and connected ones, consume energy, are piloted by software or used in a software-based ecosystem. Therefore, we need to capture their energy consumption too. Our approach is capable of generating power models for these devices as we present in the following section.

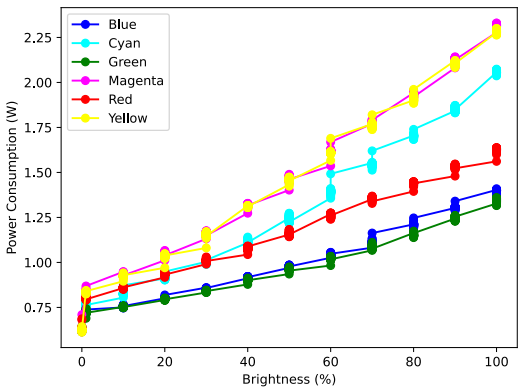
2.2.4 Beyond Computing Devices

The architectural approach we presented in Section 2.2.2 is not limited to generating accurate power models for computing devices and software systems. In computing systems, such as for desktop users, or in smart systems (such as a smart home or a smart office), other devices have a non-negligible energy consumption. We present two use cases where our approach generated accurate power models: a smart light bulb, and a *dumb* desktop computer monitor.

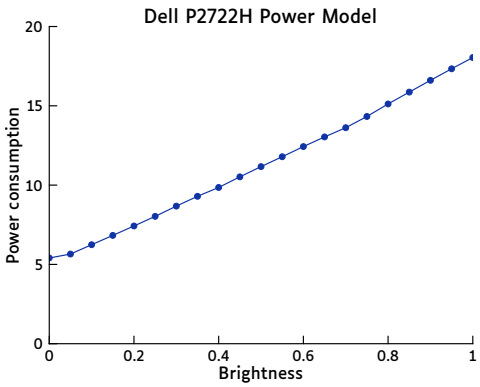
Smart light bulb: We model the power consumption of smart connected RGB light bulb (manufactured by LSC Smart Connect with reference 2578539, 9 watts, 806 lumens) [6]. The bulb communicates using 2,4 GHz WiFi.
We use PowerSpy2 powermeter to collect power consumption, and for each experiment, a total of approximately 800 data points is collected. We exclude the data when the brightness is equal to 0 because at that level of brightness none of the single led lights that consists the LED bulb is turned on so the only component consuming energy is the Wi-Fi circuit. The power consumption in this particular condition is 0.62919 W. We vary brightness from 0 to 100% with a 10% step, for each for the six basic colors: white, blue, cyan, green, magenta, red, and yellow. Figure 2.7a presents the measured correlation between brightness (in percentage), and the power consumption (in watts) for our linear regression model and the actual measurements from the powermeter. Our generated power models, for every color, have an error rate varying between 0.5% and 4%.

Computer monitor: We also modeled the power consumption of a computer monitor: Dell P2722H, according to its brightness level. We manually vary the brightness level from 0% to 100%, with a 5% step, and monitor its power consumption for one minute using a PowerSpy2 powermeter.
We apply our regression algorithms (linear and polynomial) and calculate the power model for the monitor and the margin of error of the models. The linear model generated is: P = $12.799585453870126 \times Brightness + 4.898205413922081$, and the polynomial model is: P = $10.63799768 \times Brightness + 2.16158777 \times B^2 + 5.240456810726142$. The linear model has 1.51% average error, and the polynomial model (degree 2) has a 0.65% average error.

[6] https://eprel.ec.europa.eu/screen/product/lightsources/1286006



(a) Power consumption of a smart light bulb



(b) Power consumption of the Dell P2722H monitor

Figure 2.7: Power consumption of non-computing devices

Epilogue: The models generated by CPU Power Benchmark, and those we generated with custom benchmarking suites or manually, are published in a Power Model Database available online [7]. The database contains power models for various computing components and hardware devices. And we aim to centralize power models with an open data format, and to provide an API to download or update power models.
Generating accurate power models for computing and other smart devices allows us to better observe the hardware devices and components that software run on, use or interact with. However, in such ecosystems, observing, understanding and managing energy consumption requires more than just the raw energy values of devices. Observing what's happening in these ecosystems, including from human actors, and collecting relevant contextual data is key to better understanding the dynamics of energy consumption.
Therefore, making sense of these ecosystems in regards to energy requires more than just power models. In the next section, I present our approach to model, through an ontology, contextual data and generate green knowledge about the environment and systems.

2.3 Measuring Green Software Systems

My contributions in measuring the energy consumption of software systems aim to provide relevant tools, approaches and architectures for practitioners, managers and users. These approaches allow measuring power and energy across a wide variety of layers and granularities: from devices and hardware components, to applications, and to software source code and execution branches.
As expressed in multiple studies, developers lack the proper tools to measure the energy consumption of software and devices [BWN16, PHAH16, MBZ+16a, PC17a]. And as software is everywhere, it is important to provide measurement approaches everywhere.
In this section, I present 5 of my contributions, covering all the layers of ACM's computing curricula, with the exception of systems infrastructure.

[7] https://github.com/joular/powermodels

PowerJoular and PowDroid are device- and hardware-oriented, Jolinar and Demeter are application-oriented and end users oriented, and finally JoularJX is software source code-oriented.

2.3.1 PowerJoular: Multi-platform Hardware and Software Monitoring

The content of this section is adapted from the following publication:
PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. Adel Noureddine. In the 18th International Conference on Intelligent Environments (IE2022). Biarritz, France, June 2022. [Nou22]

The landscape of computing architecture is moving towards a heterogeneity of CPU and GPU architectures (*i.e.*, x86, ARM), and device types (*i.e.*, PCs, servers, single-board computers, mobile). Therefore, software developers are more often building multi-platform software.
To help bridge the gap in measuring energy across devices and platforms, we developed PowerJoular, a software that monitors the power consumption of CPU and GPU for PCs, servers and single-board computers (such as Raspberry Pi).
In the last decade, multiple power monitoring tools were released with varying approaches and accuracy. Approaches range from hardware-based tools (using power meters) to software-based ones (using power estimation models).
The former [NRS13] uses a physical power meter or multimeter to measures the energy consumption of the device, and correlates the data with software-based monitoring. The main limitation of these approaches is the high installation cost, and the limited scalability as they require an additional hardware device.
For the latter, earlier software tools used their own power estimation models, such as the first version of PowerAPI [BNRS13] or pTop [DRS09]. These models are either based on CMOS power formulas, or on empirical experiments.
However, newer approaches and tools are based on hardware manufacturers' APIs. In particular, most server tools use the Intel RAPL interface, either directly from the registers or through the Linux kernel's implementation (for example, using powercap interface). For instance, newer PowerAPI versions or Scaphandre [8] use Intel RAPL for power monitoring. However, these tools target server and cloud environments and provide features mostly used by their uses cases, such as monitoring virtual machines or exposing metrics to hypervisors and cloud dashboards.
In addition, most tools focus on one particular platform (mainly Intel servers) and are aimed towards system administrators or automated monitoring platforms. In contrast, we aim with our approach, to provide a multi-platform power monitoring tool (starting with x86_64 servers and ARM single-board devices), and help software developers with up-to-date and easy-to-use tools to analyze the power consumption of hardware and software.
PowerJoular allows runtime power monitoring of multiple hardware components of different devices and architectures. In particular, our initial version monitors the CPU and GPU power consumption in computers and servers, and the CPU in single-board computers such as Raspberry Pi devices. PowerJoular is written in Ada in order to provide a low-

[8] https://github.com/hubblo-org/scaphandre

impact tool as Ada is constantly ranked among the most energy efficient programming languages [PCR+21], while also improving code maintainability and safety in particular as we also target monitoring single-board computers and embedded devices.
PowerJoular is aimed to software developers, system administrators and to automated tools, with a goal to help these users understand the power consumption of their devices and software, and to build more in-depth tools using our proposed platform.

Power Monitoring Approach: PowerJoular power monitoring is based on two modules:

- for PC/servers: the Intel RAPL through the Linux Power Capping Framework [9], and, optionally, NVIDIA's System Management Interface [10],
- for ARM devices (Raspberry Pi, Asus Tinker Board): our own empirical regression power models, detailed in Section 2.2.2.

PowerJoular automatically detects the computer configuration and supported modules, and provides power data accordingly. For the GPU, PowerJoular checks if NVIDIA SMI is installed, then uses it to verify if GPU power monitoring is supported to the specific graphic card, and to read GPU power consumption every second.
For the CPU, PowerJoular uses the Intel RAPL power data through the Linux powercap interface by reading the appropriate system files. It first detects which power domains are supported by the CPU:

- Pkg: which is supported since Intel Sandy Bridge CPUs, and provides energy consumption for the CPU cores, integrated graphics, memory controller and last level caches. PowerJoular also checks if DRAM power domain is supported (RAM attached to the memory controller) and adds its power readings to the total.
- Psys: which is supported since Intel Skylake CPUs, and provides energy consumption for the entire SOC (including Pkg along with other components, such as eDRAM, PCH, System Agent [KHN+18]). If Psys is supported, it will be exclusively used by PowerJoular for CPU power consumption instead of Pkg and DRAM, as it provides a more comprehensive power reading of the CPU SOC.

Finally, PowerJoular aggregates power readings from all supported components to provide an overall power consumption. For instance, if both Intel RAPL and NVIDIA SMI are supported, the tool will provide an aggregated power value for both CPU and GPU.
On ARM devices (Raspberry Pi, Asus Tinker Board), PowerJoular uses our own power polynomial regression models 2.2.2 that map the CPU utilization to power consumption. These models are accurate and have very low error rates, between 0.3% and 3.83%, far more accurate than the state-of-the-art models. The tool reads CPU cycles from `/proc/stat` system file, and calculates CPU utilization. The latter is then used in the polynomial models to provide an accurate estimation of the CPU power consumption.

[9] https://www.kernel.org/doc/html/latest/power/powercap/powercap.html
[10] https://developer.nvidia.com/nvidia-system-management-interface

Features: We designed PowerJoular to be efficient, low on resources, flexible and intuitive to use. The interaction with the tool is achieved through a command-line interface. Such interface offers flexibility for scientific experimentations, headless monitoring in server environments, and can be easily incorporated into external frameworks or dashboards.

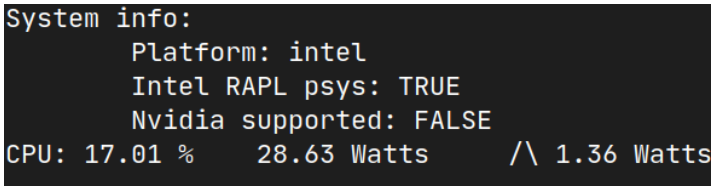


Figure 2.8: Default output of the PowerJoular command-line interface

Runtime power monitoring can be displayed on the terminal and/or written to CSV files. Figure 2.8 shows the command-line interface of PowerJoular. The latter CSV option stores power data every second and can then be read to retrace the historical power consumption of a device or a specific process.
PowerJoular can monitor the CPU power consumption of an individual process by providing its PID on runtime. For monitoring a specific process through its ID (PID), PowerJoular uses the Linux statistics in `/proc/stat` and `/proc/pid/stat`, and calculates the proportion of CPU cycles used by the process, and thus calculates the power consumed by the process accordingly.
If a PID is monitored, its power data will be displayed on the terminal, and will be stored to a distinct CSV file, while the device's power is saved independently. At the end of each monitoring session, PowerJoular displays the total energy consumption (in Joules) of the session (and for the monitored PID).
PowerJoular can also monitor an application by its name. It monitors an application by searching for all its PIDs, on every monitoring cycle (by default, every second). We verify the PIDs of the application on every cycle as applications can create or destroy processes during their runtime. Then, PowerJoular applies the same approach to monitor a PID, but by monitoring all the applications PIDs and calculating the sum of the power of these PIDs.
In addition, writing to a file can also be done in *overwrite* mode, *i.e.*, only the last power data is saved to the file. Therefore, the tool can run for long periods of time without generating a large CSV file. This mode allows external tools to connect to PowerJoular's power data in runtime to build dashboard or monitoring interfaces. For instance, a centralized dashboard can read and visualize power data of multiple servers or Raspberry Pi devices running PowerJoular.
Finally, PowerJoular provides a systemd service [11] that can be enabled and run automatically on Linux boot. The service monitors the computer's power consumption and stores data in a CSV file (with overwrite mode) in `/tmp` folder. This allows continuous and automated monitoring of servers and devices, and provides accurate runtime power data. Writing to `/tmp` while running automatically as a service, allows to bypass the added restrictions on reading powercap energy meters to non-privileged users in Intel CPUs [12]. The restriction

[11] https://systemd.io/

[12] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=949dd0104c496fa7c14991a23c03c62e44637e71

was added due to the recently discovered PLATYPUS vulnerability [LKO+21]. However, the systemd service still requires privileged access (root/sudo) to be enabled, and the data provided is the runtime power consumption (every second) from aggregate sources (CPU and GPU when available).
Currently, PowerJoular supports traditional x86 computers and servers (32 and 64 bits), a variety of Raspberry Pi devices (from Model Zero to Model 400, in 32 and/or 64 bits), and Asus Tinker Board S.

Use Case Scenario: To showcase the multi-platform capabilities of PowerJoular, we measure the energy consumed by three implementations of the Ray-casting algorithm taken from Rosetta Code [13]. We run the Python implementation in a loop for 10 000 iterations, the C version for 100 000 iterations and the Java version for 5 000 iterations. We conduct our experiments on three devices: a Dell 5530 laptop (Intel Core i7-8850H) running Fedora Linux 34 with kernel 5.11.19, GCC 11.1, Java 11, and Python 3.9.5. A Raspberry Pi 3b+ revision 1.3, running Raspberry Pi OS 32 bits (based on Debian 10), and a Raspberry Pi 4B revision 1.2, running Raspberry Pi OS 64 bits, both running with kernel 5.10.17, GCC 8.3, Java 11, and Python 2.7.16.

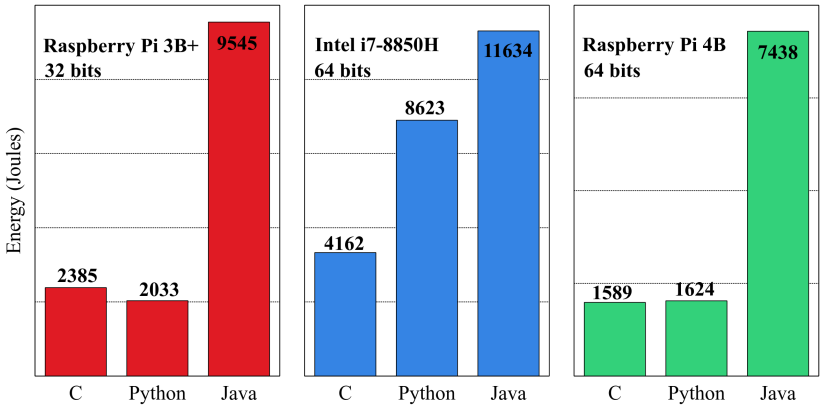


Figure 2.9: Energy consumption of Ray casting algorithm on different programming languages and platforms

Figure 2.9 shows the total energy consumption of all experiments for both platforms. However, each implementation prints different output to the Linux terminal. In particular, the Java and Python version prints multiple lines of text, while the C version prints 3 numbers on each loop. Therefore, the results should not be used to compare programming languages, but rather to compare energy distribution across different platforms. In particular, we observe that the Python program consumed much more energy, compared to the C program, on the Intel-based computer, while it consumed less on the Raspberry Pi devices. This can be partially explained by the different software stack (Python and GCC versions), and the execution time of both programs: Python code on Intel took 3 minutes and 28 seconds (for C: 3 min 14 sec), and on Raspberry Pi it took 12 min and 32 sec (RPi 3B+) and 7 min and 38 sec (RPi 4B) (for C: 14 min and 8 sec for RPi 3B+ and 7 min and 23 sec for RPi 4B).

[13] https://rosettacode.org/wiki/Ray-casting_algorithm

PowerJoular allows such comparisons and studies across multiple platforms. In this particular example, a software programmer might decide to use the C version on a server, and the Python version on a Raspberry Pi, instead of using the C version on both platforms if the programmer solely relied on energy consumption of only the Intel computer.
PowerJoular offers a light software capable of monitoring power consumption in server environments and in cyber-physical systems alike. Its aim is to be a simple, yet capable, tool to help practitioners monitor the power consumption of hardware components and devices, and of applications.
However, as many other monitoring tools, it is aimed for a tech-savvy public and runs on Linux systems. End users, whether on home computers or corporate devices, often run Windows, and lack technical skills to use such tools. For this public, we developed Jolinar and Demeter, which are detailed in future sections in this chapter.
PowerJoular targets computers, servers and cyber-physical systems. However, one of the most popular devices today are smartphones and tablets. These devices are more powerful than many CPS devices, and their energy impact is non-negligible. Therefore, it is crucial to be able to measure their energy consumption too, and for this need, we designed and built PowDroid which is detailed in the next section.

2.3.2 PowDroid: Monitoring Android Devices

> The content of this section is adapted from the following publication:
> PowDroid: Energy Profiling of Android Applications. Fares Bouaffar, Olivier Le Goaer, and Adel Noureddine. In IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), SUSTAINSE/ASE'21, Melbourne, Australia, November 2021. [BGN21].

While the energy efficiency of mobile apps is receiving considerable attention in recent years, Android developers have little tools to assess the energy footprint of their devices and applications. We introduce, PowDroid, our tool to estimate the energy consumption of Android devices, and can be used in a benchmark to estimate the energy of Android applications. It uses system-wide metrics and does not require access to applications' source code, or modifications to the smartphone or the Android operating system.
Android is the most popular operating system in the world, with over 2.5 billion active users spanning over 190 countries. Battery-limited devices powered by Android like smartphones and tablets are particularly concerned by energy-heavy hardware components like the screen, GPS, Wi-Fi and so on. But as noticed in [Hin16], developers have little or no tools to address this concern.
In the Android development field, developers are still not able to measure their devices, compare their applications with competing ones, nor conduct studies of energy consumption at large. On one side, hardware-based solutions require electrical tinkering to avoid damaging the device, are hardly scalable and come at an additional cost. On the other side, software-based solutions are often untrustworthy and quickly deprecated as the Android platform evolves. The only tool officially supported is the Energy Profiler[14], which is integrated within Android Studio development environment. However, it requires access to

[14] https://developer.android.com/studio/profile/energy-profiler

the source code and does not provide runtime energy consumption but rather displays an overview of the energy levels (*e.g.*, light, medium, etc.) of hardware components over the runtime of the application. In addition, no export functionality is available, which would have allowed for offline analysis and decision-making.

PowDroid monitors the entire Android device, rather than specific applications, but it can be used to evaluate the energy consumed by an application run in isolation in a given time period.

The general architecture of PowDroid is described in Figure 2.10. The key idea is to pull system-wide battery data from any Android device using the Android Debug Bridge (ADB). The initial version of PowDroid connects to the device using WiFi instead of a USB connection, in order to avoid having the device falls into a charging state while connected to USB (and thus preventing the proper observation of the battery drain in real time). However, it is still possible to use a USB cable by disabling its charging feature, as in PETrA [DNPP+17]. Version 1.5 and later, released in 2022, switches to using USB connection to start and stop monitoring and to collect data, but it does not require continuous USB connection, therefore removing the interference of USB charging on the measurements.

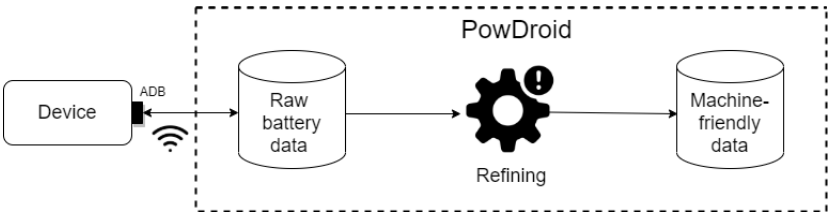


Figure 2.10: PowDroid Architecture

The collected data are generated by the Android system itself and serves as a ground proof in our approach. Hence, we do not introduce over-estimations than those already calculated natively by the Android platform. Instead, we refine the raw data through a specific workflow to spit out a machine-friendly output. The output contains energy consumption in joules, and also hardware components status along a fine-grained time sampling. We argue that the resulting set of metrics is valuable to perform a large panel of analysis and statistics.

Our architecture uses multiple components to collect and process metrics and calculates energy consumption.

Batterystats: is a tool included in the Android framework that collects raw battery data on a device. The corresponding ADB command is `adb shell batterystats`. To avoid unnecessary accumulation of data between test sessions, we clean the history with the command `adb shell batterystats reset`. At this stage, the data are located on the device in a file named `data/local/tmp/battery.txt`.

Bugreport: is also a tool included in the Android framework which generates a report file in a ZIP format on the device from the aforementioned `battery.txt`. The corresponding command is `adb bugreport [filename].zip`. A bugreport file is a rich format compatible with Battery Historian.

Battery Historian: Google's Battery Historian[15] converts the report from Bugreport into a pretty web-based visualization that can be viewed in a web browser. We reuse the Go script `local_history_parse.go` to convert the zip file to a human-readable CSV file. We use the `-summary=totalTime` argument to produce results based on time, rather than the evolution of the battery charge.

Processing energy scripts: Finally, we implement the workflow in a Python program: *PowDroid.py*, which handles all the communication process with the device, resetting Batterystats, processing and analyzing the data, and producing the final output file with energy consumption across the time windows.

Energy Metrics: Each time the battery evolves or the status of the hardware components change, an entry is written in the bugreport file. This is called an energy-related event, delimited by a start time and an end time (of type Timestamp - TS). The entry provides a lot of information, including the voltage, Coulomb charge, or the top application packages.
PowDroid processing starts with the calculation of the duration of a state (T_{Δ}), using start time and end time:

$$T_{\Delta(event)}(ms) = endtime_{event}(TS) - starttime_{event}(TS)$$

These factors influence the energy consumption, and can be sorted chronologically.
In addition to the time intervals, we calculate the intensity of the current from the difference between two remaining charges (C_{Δ}) and the duration of the smartphone charge drain (T_w), by using this formula:

$$Intensity(mA) = \frac{C_{\Delta}(mAh)}{T_{w(phonecharge)}(hr)}.$$

With intensity, voltage and duration of events, we can calculate the power and energy consumption of our device for the monitored event. First, we calculate the power with:

$$Power(Watt) = Voltage(Volt) * Intensity(Amp).$$

Then, we calculate the energy consumption for each event in Joules:

$$Energy_{(event)}(Joule) = Power(Watt) * T_{\Delta(event)}(Second)$$

We are then able to calculate the energy consumption of the application by adding every energy consumption of all events:

$$Energy_{(app)}(Joule) = \sum_{i=0}^{n} Energy_{(event_i)}(Joule)$$

In summary, our tool will generate one CSV file, crossing all events, their time intervals and energy consumption. The final structure of our file is presented in Table 2.3.

[15] https://github.com/google/battery-historian

Metric	Description	Unit
Start time	When the event start	Timestamp
End time	When the event ends	Timestamp
Duration	Duration of the event	Millisecond
Voltage	Electric voltage emitted by the battery.	Millivolt
Remaining charge	Electric charge remaining in the battery	Milliampere-hour
Intensity	Electric intensity calculated from remaining charge	Milliampere
Power	Amount of energy during a given time, usually 1 second.	Watt
Consumed charge	Amount of charge passing through the cross-section of smartphone	Milliampere-hour
Energy	Total energy consumed when the application was run.	Joule
Top app	Name of the package running in the foreground.	String
Wakelock	Wakelocks acquired by the application.	String
Screen	Indicates if the screen is active or not.	Boolean
GPS	Indicates if GPS is on or not	Boolean
Mobile radio	Indicates if mobile GSM is active for scanning or transmitting data	Boolean
Wifi on	Indicates the status of wifi (On/Off)	Boolean
Wifi radio	Indicates if the wifi is transferring data	Boolean
Camera	Indicates if the camera is active or not	Boolean
Video	Indicates if there is a video like video reading or a video call	Boolean
Audio	Indicates if the audio component (like speakers) is active	Boolean

Table 2.3: Metrics of the Output CSV file

Use Case Example: To show the capabilities of PowDroid, we compare the energy consumption of several Android applications from three categories: web browsers, camera, and weather applications. We chose these applications in order to get a wide spectrum of use cases using various hardware components (Wi-Fi, GPS, Camera, CPU, GPU, etc.).

To conduct our experiments, we use a *Google Pixel 3a* device in debug mode. We run the applications isolation: 1) we remove the SIM card to prevent accidental phone calls and messages, and to avoid the continuous impact of cellular network connectivity, 2) we disable running background services such as Google Play Store and close all forefront applications, and 3) finally, we disable the Battery Saver Mode.

For each application category, we define an experimental but realistic scenario that is conducted for all applications. We have installed the latest versions of the applications available at the time of the experiment. We run each application for 4 minutes, with the battery level at 50% at the start of the measurement (we recharge back to 50% before running the next experiment), screen brightness at 50%, and audio at 50%.

We run each scenario three times for each application and average the results in Joules. For

web browsers, the scenario includes typing a URL, searching with a keyword, and selecting and watching a link from the search results, and we repeat all of this for 3 different websites. For camera applications, we take pictures from both the front and back cameras in normal, panoramic and HDR modes, and record a 10-second video clip. And for weather applications, we locate our position, get weather information for today, check monthly view and another day, and view the radar map.

Web Browsers

Application	Energy (Joule)
Mozilla Firefox	427,78
Google Chrome	404,27
Microsoft Edge	386,18
Opera	368,33
Samsung Browser	345,93
Brave	319,66

Camera Applications

Application	Energy (Joule)
Open Camera	711,35
Google camera	683,45
Samsung S10 camera	661,92
ProCam X	660,54
Manual camera DSLR Pro	610,54

Weather Applications

Application	Energy (Joule)
AccuWeather	441,12
The weather channel	398,92
Weather Forecast	318,82
Advanced Weather	236,92

Table 2.4: Energy consumption of our different application domains

Table 2.4 outlines the energy consumption of all our tested applications and domains: web browsers, camera and weather applications. Overall, our results are consistent with recent comparative studies (such as for web browsers [16]).

For web browsers, our experiment found that Brave is the most energy efficient, while Firefox was the worst with a 33.8% increase in energy consumption (a increase of 108.12 Joules for the same workload). However, this can be partially attributed to the default built-in adblock feature in Brave, which also blocks YouTube and video ads. However, Samsung Internet Browser achieved the second place in energy efficiency without blocking ads. Google Chrome and Mozilla Firefox were within a slim margin of 5.8% in energy consumption.

For camera applications, we also observe an energy increase of 16.5% (100.81 Joules) between the most efficient of our experiment (Manual camera DSLR Pro) and the least (Open Camera).

Finally, we tested 4 weather applications with even bigger variation: an increase of 86.19% (204.2 Joules) between Advanced Weather (the most efficient), and AccuWeather (the least efficient). This can be explained by the presence of ads in all applications, with the exception of Advanced Weather which does not contain any ads.

These results for ad-supported applications, for the different categories, are consistent with state-of-the art studies which show that ads are known to have a huge impact on energy consumption in mobile applications [GMNH15].

[16] https://greenspector.com/en/what-are-the-best-web-browsers-to-use-in-2020/

PowDroid fills a gap in software energy measurement as it allows monitoring the most used devices today, which are still, surprisingly, lacking accessible energy measurement approaches and tools. Smartphones and tablets are part of our complex software ecosystem and modern cyber-physical systems, and their measurement, and later optimization, is crucial for a holistic energy management.
PowDroid measure Android devices, and can be used through specific benchmarks and isolation to estimate the energy of applications. This manual approach for application monitoring is limiting the accessibility and widespread usage of software monitoring. To address this issue, we built two tools and architectures capable of directly measuring the energy consumption of applications, along with an easy interface for more widespread utilization. These approaches, Jolinar and Demeter, are detailed in the following two sections.

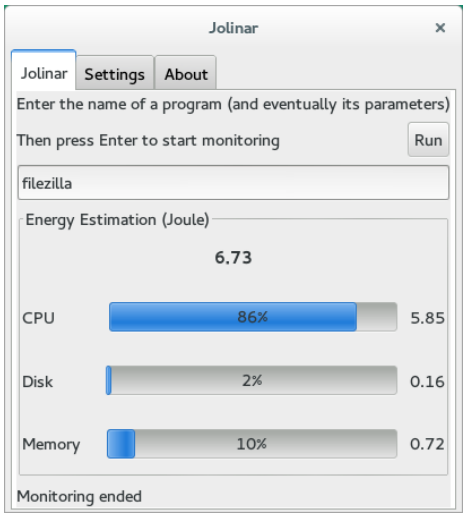
2.3.3 Jolinar: GUI-based Software Monitoring

> The content of this section is adapted from the following publication:
> Jolinar: Analysing the Energy Footprint of Software Applications. Adel Noureddine, Syed Islam, and Rabih Bashroush. In International Symposium on Software Testing and Analysis (ISSTA'16). Saarbrücken, Germany, July 2016. [NIB16]

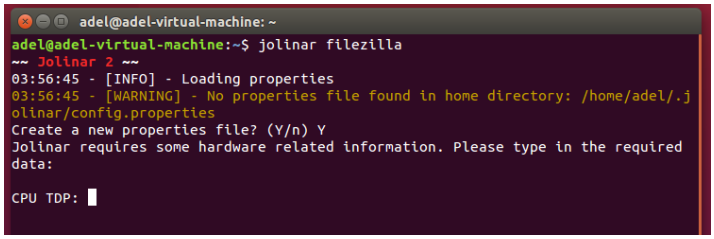
Monitoring energy consumption of applications is crucial for energy optimization and improvements in software systems. Jolinar is a tool that bridges the gap between energy measurements and accessibility to end users. The tool builds on top of recent energy models to provide an accurate, light and easy-to-use interface for energy measurements. The target audience of Jolinar is both software developers and non-technical end users who want to monitor their applications' energy footprint.
Limited research has been conducted on measuring energy consumption of software applications to raise the awareness of end users and developers of the impact of their applications [PCL14a]. The importance of energy feedback for end users is an ongoing research challenge, though, most of the focus so far has been on home appliances and providing feedback in physical environments [CSJS14]. As such, there is a missed opportunity to try and influence end-user behavior through increased awareness.
A more accessible tool to end users, with an easy-to-use and intuitive interface, can contribute substantially to raising awareness of software energy consumption. Indeed, providing energy consumption feedback to end users has shown its relevance in energy conservation as early as the 1970s [Bec78, JHvH89]. We argue that for a tool to be easily accessible to end users, it needs to address the following two design aspects:

- **Visual feedback**. Providing a clear visual feedback of software energy consumption is key for raising energy awareness. HCI researchers distinguishes between two types of data visualization: pragmatic and artistic visualization [POB08]. The former provides more concrete quantitative information while the latter is richer visually but at a cost of explicitness. Studies found that users, although appreciate artistic metaphors, want more precise information [Fro09].

- **Ease of installation and usage**. Being appealing and concise only helps if users can install and use the software easily. Complex installations requiring hardware tools,



(a) A successful energy monitoring by Jolinar.



(b) Command line version of Jolinar.

Figure 2.11: Jolinar's GUI and CLI

compilation, text-based configuration or command line instructions will severely restrict the potential user base to those who can afford the investment, or possess the required technical knowhow.

Jolinar addresses these challenges by providing a lightweight and easy to install and use GUI-based energy monitoring tool. Jolinar requires no power meter or source code modifications.
The visual information provided by Jolinar is pragmatic for precise energy data, while deploying familiar indicators and gauges that the average end user can easily understand. Jolinar is an easy-to-use software tool that allows end users to visualize the energy consumed by their applications. It is designed to be accessible by non-technical users and software architects alike, providing an intuitive graphical user interface and a command-line tool for advanced users.
Jolinar is based on our previously defined energy models [NBRS12a] and bridges the gap between accurate energy estimations and end user usability. It monitors one application at a time and provides energy feedback in a visual and easy-to-understand format.
Figure 2.11a, display an example of using Jolinar GUI to monitor the energy consumed by FileZilla, an FTP client on GNU/Linux. A typical use case where an end–user wants to understand the amount of energy consumed by usage of software.
Jolinar was designed to be an easy to install and use tool. The tool is distributed as *executable Jar*, *.deb* and *.rpm* packages where dependencies are automatically managed. As such, no compilation or command line instructions are required as the tool is packaged in *.deb* or *.rpm* packages.
The initial view of Jolinar is divided into two areas: the input box for choosing the application to monitor, and the visual output display, which shows the energy consumed by the application (Figure 2.11a). Users simply need to indicate the name of the application to start monitoring. Jolinar launches the application and starts monitoring its energy consumption

instantaneously. This presents the best trade-off between swiftness, monitoring and accessibility avoiding the need to deal with process ids.
We chose to use a text box to input the application name instead of a list to choose from, or permanently monitoring all applications, because this option presents the best trade-off between swiftness of monitoring and accessibility. Therefore, there is no need for software modification, memorizing the application's process ID, or fiddling with a long list of monitored applications to find the one the user needs.
The second tab in Jolinar's GUI window is where users can specify the settings and configuration information needed for the application and the energy models.Our models use hardware specification data that are usually provided by hardware OEMs, such as the TDP, frequency and voltage, or read/write power rates. An option to generate logs is also available for developers and will store runtime software power consumption (in Watts) each 500 milliseconds.
When Jolinar launches the application and starts monitoring, it will do so in the background without impacting the performance of the application. Users can then use their software normally. On exit, Jolinar displays the energy estimations in both numerical values (in Joule and percentage) and using visual feedback.
Energy estimations are broken into individual hardware components along with the total energy consumed by the application. This provides better understanding and awareness of which hardware component is responsible for the highest/lowest energy impact in the monitored application (which could help in identifying energy hotspots).
In the example of Figure 2.11a, FileZilla consumed 6.73 Joules in a benchmark run consisting of sending and receiving 5 files to an FTP server, with most of that energy (5.85 Joules) being consumed by the processor (86% of the total energy of the application). Disk energy was around 2% due to reading files for FTP transfer, while the RAM memory consumed 10% of FileZilla's energy (0.72 Joules).
This breakdown, both in Joules and in percentage, helps both software developers and end users understand which hardware components are consuming the most energy. For developers, this will provide them with a first indication of the energy consumption of applications and help them focus their optimization and improvement work to increase the energy efficiency of their software. Therefore, Jolinar is complementary, rather than a replacement, to tools such as `Jalen` [NRS15] which monitors energy consumption of software source code (at the methods level). For end users, the breakdown would allow them to better understand the energy footprint of each hardware component in their computer.

Analysis use case using CLI: The command line interface of Jolinar is used in a use case where Jolinar is used to analyze libraries and executables, helping software engineers make energy-conscious decision.
In addition to the graphical user interface, Jolinar provides a command-line tool (JCL) that is suited for large runs, scientific experiments, for headless applications in server configurations and incorporation into external frameworks.
We designed the command-line version to be as much intuitive and easy-to-use as the GUI one. Figure 2.11b displays the configuration wizard that is executed at the first run of Jolinar or when the configuration file is missing. The GUI and command-line versions both share the same *config.properties* file, giving users and developers two methods to update the settings (they can also edit the file directly).

The command-line tool offers simple commands to update settings or monitor energy with multiple options and flags. For instance, updating the TDP can be achieved by simply typing: *jolinar -tdp 20* to update its value to 20 Watts. Finally, monitoring an application follows a similar straightforward approach as the GUI, users only need to indicate the name of the application to monitor and Jolinar will execute, monitor and then output its energy consumption.

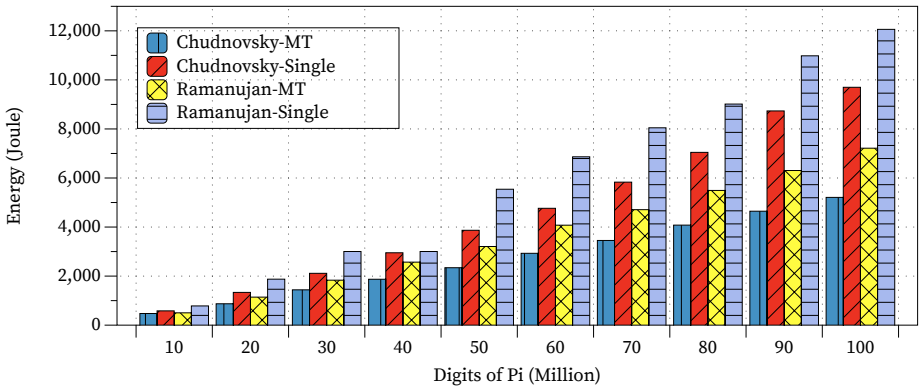


Figure 2.12: Energy consumption of Pi algorithms

We measure the energy consumed by two different algorithm implementations of calculating the digits of the number Pi. We measure the energy consumed by each algorithm to show the energy consumption for operating on the same set of inputs and computing the same set of data. We use y-cruncher [Yee], a software implementing multiple algorithms for calculating many constants - including Pi, to compare Chudnovsky's algorithm and Ramanujan's algorithm implementations on a GNU/Linux operating system. We compare the algorithms using a single thread and in a multi-threaded environment, for calculating digits of Pi ranging from 10 million to 100 million digits. We run the experiments on an HP G70 laptop with an Intel Dual T3400 processor at 2.16 GHz.

The results in Figure 2.12 show that the Chudnovsky multi-threaded implementation (*Chudnovsky-MT*) consumes less than half the energy compared to Ramanujan's single-threaded one (*Ramanujan-Single*). More precisely, across all experiments, the multi-threaded version on average consumes 38% less energy compared to the single-threaded implementation. In addition, Ramanujan's algorithm has a 32% energy overhead, on average, compared to Chudnovsky's algorithm.

In cases where a software engineer is deciding whether to use an implementation of a Pi algorithm as a library feature, Jolinar would help them make an energy-conscious decision thereby reducing the carbon footprint of the software. Profiling and analysis of energy consumption such as this can help developers identify the features that are consuming more energy and use additional information (such as importance and likely frequency of use) to decide on where to focus optimization efforts (*e.g.* optimize code or dedicate energy efficient resources to frequently used features).

Jolinar is a software tool that can estimate the energy consumption of applications in an accurate, lightweight and accessible way to researchers, practitioners and non-technical and users alike. With minimal setup and an automated approach, users only need to choose an application to monitor and Jolinar will provide detailed energy measurements with minimal

effort.
Jolinar development stopped in 2016, and the latest stable version was published in 2015. Jolinar's source code is available at: github.com/adelnoureddine/jolinar.

Epilogue: Following the release of Jolinar, the computing world followed a rapid evolution of hardware (CPUs, GPUs, etc.), a shift in computing usage towards cloud computing and towards connected devices and IoT, and an evolution of software environments and operating systems.
As the development of Jolinar stopped in 2016, and with these evolutions happening, a new approach was deemed necessary to build a comprehensive suite of monitoring software for the new multi-hardware, multi-platform and multi-software environments. The Joular project [17] aims to fill this gap and to advance knowledge in software energy efficiency throughout the life cycle of software, and across a variety of software systems and devices. PowerJoular and JoularJX, described in this chapter, are the initial fruits of this project.
The important user-centric part of Jolinar survives in the form of a new contribution: Demeter, a software architecture for long-term monitoring of end-user applications, which is described in the next section.

2.3.4 Demeter: Long-Term Monitoring of End-User Applications

> The content of this section is adapted from the following publication:
> Demeter: An Architecture for Long-Term Monitoring of Software Power Consumption. Lylian Siffre, Gabriel Breuil, Adel Noureddine, and Renaud Pawlak. In the 17th European Conference on Software Architecture (ECSA 2023). Istanbul, Turkey, September 2023.

Quantifying and long-term monitoring the energy consumption of software in end-user computers is a complex task as computers are heterogeneous in terms of hardware and software configurations. End users also need to visualize their energy consumption and get a per-software feedback about their energy impact to adapt their software usage towards a greener approach.
To address these challenges, we propose Demeter, a monitoring and feedback architecture. Our distributed approach monitors energy consumption per application on runtime, provides end users with immediate feedback through a graphical user interface, and provides delayed feedback through an analysis email notification. Our architecture is capable of long-term monitoring of software usage allowing researchers to study users' energy patterns. We illustrate our approach with a two-week study of software usage of three different user profiles in a corporate environment.
Most recent monitoring software at the component and application levels, such our own PowerJoular [Nou22], are capable of monitoring hardware components for a limited number of applications (typically one application or process, or a select number of predefined applications), and are primarily available on server environments and on Linux systems.
In particular, existing monitoring solutions are limited in capabilities (*i.e.*, only monitoring specific hardware components or a particular software), cannot scale up for multi-days or

[17] noureddine.org/research/joular

weeks of monitoring (*i.e.*, huge data collected, high CPU or energy impact of the monitoring tool, invasive monitoring interface, etc.), or are not tailored towards end users. The latter tend to be less tech-savvy and more reluctant to install monitoring software on their personal or work computers. The difficulty is also in convincing users and administrators alike to deploy a monitoring software, and the advantages in energy reduction that a raised user and corporate awareness provides. Our goal is summarized in two main objectives:

- Provide a long-term monitoring software (days, weeks or months measurements),
- Provide energy feedback to users with fine granularity (per application and per hardware component), along with analysis and recommendations for improvements.

Demeter implements these two objectives and allows practitioners, researchers, and industrial managers to study the energy impact of devices and users, and encourage eco-friendly software usage behavior.

2.3.4.1 Architecture of Demeter

Demeter interacts with the end user (who can view its power consumption and get immediate or delayed feedback) and with applications and the OS in order to collect data for usage and energy monitoring.

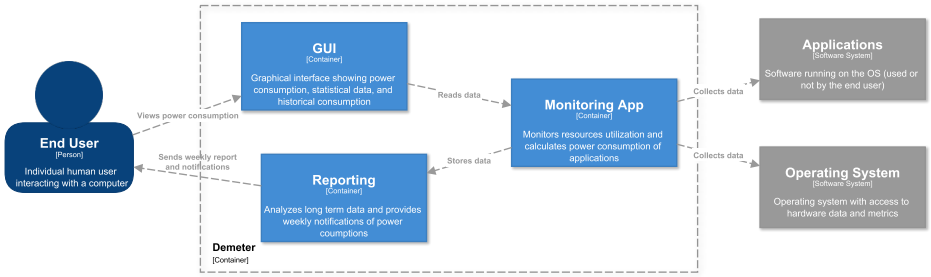


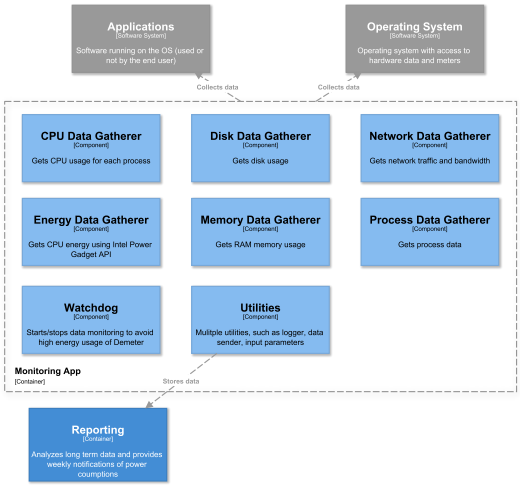
Figure 2.13: Container diagram of Demeter architecture

Figure 2.13 presents the container diagram of Demeter with its three main parts:

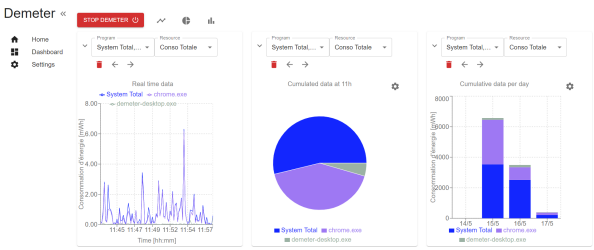
- Monitoring Application: responsible for the usage and power monitoring of every application and hardware component. It collects data from running applications and the OS. Provides the results to the graphical user interface and to the reporting server.
- Graphical User Interface: a graphical interface software allowing per-application visualization to end users, aggregated statistics, and historical power usages.
- Reporting Server: responsible for analyzing long-term power data, providing weekly summaries and notifications to end users about their energy usage and impacts.

Monitoring Application Figure 2.14a presents the component diagram of the Monitoring Application. Each hardware component has a dedicated software component implementing the relevant sensors to collect usage or power data, and its associated power formulas and

models to estimate its power consumption. An additional component is also implemented to collect processes and applications information and usage, and a utility component manages the input/output of the application and communications with the Remote Server.



(a) Component diagram of Demeter's Monitoring Application



(b) Screenshot of Demeter's GUI

Figure 2.14: Component diagram of Demeter and its GUI

CPU power monitoring: In Demeter, the CPU power consumption is collected through Running Average Power Limit (RAPL) and Intel Power Gadget (IPG). They are model-based software and are precise enough to rely on [RNA+12]. We define the CPU percentage usage of a process as in Equation 2.2.

$$U = \frac{t_{\text{user}} + t_{\text{kernel}}}{t_{\text{total}}} \; / \; n_{\text{cores}} \tag{2.2}$$

Where n_{cores} is the CPU's core amount, t_{user} is the duration during which the process runs in user mode, t_{kernel} is the duration during which the process runs in kernel mode and t_{total} is the duration during which the CPU is active and idle. These values are retrieved from PSAPI [18]. In Equation 2.2, the sum between t_{kernel} and t_{user} corresponds to the invested CPU time for a process. To avoid having values above 100%, it is required to divide by n_{cores}.

Hard Drive: As the specifications of hard drives vary from one manufacturer to another, they do not consume the same amount of energy for read and write operations. We use the read and write performance power ratio to calculate the disk energy consumption, in addition to the idle power consumption. This ratio is usually provided by manufacturers of modern SSD drives. In addition, if the sequential read and write performance power ratio is also given, we take this ratio into consideration as it is more aligned with disk operations. In our case, the disk used for the study is NVMe KBG40ZNS512G NVMe KIOXIA 512GB. The

[18] https://learn.microsoft.com/en-US/windows/win32/psapi/psapi-functions

disk has a read power consumption at $0.78\,\mathrm{mW} \cdot \mathrm{MB}^{-1} \cdot \mathrm{s}^{-1}$, and write power at $0.98\,\mathrm{mW} \cdot \mathrm{MB}^{-1} \cdot \mathrm{s}^{-1}$ [19].

Network: Our aim is to probe the energy consumed by the network interface controller. Demeter will only estimate the energy consumed by the computer and its components. Gathering the bandwidth for every process is done using the Npcap library [20]. Npcap sniffs and reads the content of every packet passing through a network interface. Every available network interface of the computer is then opened and sniffed. For each process, the upstream and downstream bandwidths are gathered by our packet parser, using Npcap. The network energy consumption of a given software is the sum of its processes' energy consumption. Packets have to be parsed to account for their costs in bytes. Thus, packets and processes have to be linked together. For a process to send or receive packets, in UDP and TCP, it needs to attach to a port. Our approach is that for each packet, we increase the bandwidth counter of the port it goes through by the size of the packet. The sum of all opened ports' bandwidth for a process results in the process's bandwidth.
We base our model on data from our Ethernet transceiver's data sheet [21]. This transceiver can negotiate three bitrates: $10/100/1000\ \mathrm{Mb} \cdot \mathrm{s}^{-1}$, each with its own energy consumption. For the conversion ratio, we are using 10 Mb/s power consumption, and therefore the upstream and downstream data to energy ratio is $0.068\,\mathrm{mWh} \cdot \mathrm{MB}^{-1} \cdot \mathrm{s}^{-1}$.

WatchDog: Demeter has to control its CPU usage to have the smallest impact on the system. Our CPU consumption regulator is called WatchDog (WD). It will pause Demeter's activity if it becomes too CPU-consuming. The calibration of the WD is based on the CPU average consumption of Demeter over the first hour. It pauses the software for one minute if it detects an over-consumption (*i.e.* if its energy consumption is higher than three times the energy mean value during the calibration hour). Pausing has consequences as data gathering and exporting are stopped. As the network is sniffed by our parser, all data is lost during the pause. The CPU and disk data are not impacted as they are probed by external software.

Immediate Feedback through a Graphical User Interface As Demeter is also meant to be used by non-technical end users. The GUI aims to give immediate feedback to users. A screenshot of Demeter is shown in Figure 2.14b.
The GUI allows end users to monitor the power consumption of all applications. For each application, users can monitor real-time power consumption, the historical power evolution (through time), and aggregated statistics about power usage and other collected metrics. The flexibility of the GUI (per-application and system-wide measurements and statistics), allows end users to customize the interface according to their desires and needs, thus allowing a personalized approach with the tool and in regards to power consumption. We argue that such a relationship where the user feels empowered in freely and easily choosing what to follow might lead to better engagement with power efficiency software usage best practices, and thus to an overall reduction of power consumption.

[19] https://europe.kioxia.com/content/dam/kioxia/shared/business/ssd/client-ssd/asset/whitepaper-cSSD-BG4.pdf

[20] https://npcap.com

[21] https://www.intel.com/content/www/us/en/products/sku/82185/intel-ethernet-connection-i219lm/specifications.html

Delayed Feedback through Cloud Notifications The third part of Demeter is the Remote Server. The server component is responsible for three features: 1) storing all power data sent by the Monitoring Application for all users (for instance, one RS can handle all users in an office, a company, or a building), 2) analyzing per-user and cross-user power consumption patterns and trends and providing an overview of a population's power profiles (for example, through a dashboard), and 3) sending recurrent notifications to users with a summary of their power consumption along with trends and power evolution, and with recommendations for power reductions. For instance, an email notification could be sent to users in a corporate environment with the power analysis of their consumption of the prior week.
In the next section, we present a real-world experiment scenario using Demeter to monitor users' software and energy usage in a corporate environment.

2.3.4.2 Preliminary Study in a Corporate Environment

In our study, we aim to showcase how Demeter can be used to monitor and analyze energy across applications and devices, in a corporate environment with different user profiles.
We showcase the importance of our approach in a two-week experiment, studying the impact of proposed recommended good practices on the energy consumption and sustainability awareness of 3 corporate users. We run Demeter for two weeks on three laptops (Dell Latitude 5420) for 3 different user profiles: user 1 is a project leader in digital marketing, user 2 is a help desk technician, and user 3 is a researcher in Green IT. The three users performed their day-to-day tasks at work as usual. During the first week, no specific instructions have been given to the users. Then, users were briefed about green software good practices before the second week. These good practices include: reducing the quality of streaming videos on YouTube, reducing the quality of music streaming on Spotify, reduce the number of opened tabs in Google Chrome, and deactivating the camera in Microsoft Teams during a video call. Users finally had to report how they used Teams, Spotify, and Chrome during the two weeks.
In Figure 2.15, we gather the energy consumption per day and per hardware component for users 1, 2, and 3, and for Chrome and Teams. We observe that the CPU consumption embodies the total energy consumption. The CPU energy is higher than 99.998% of the total energy for both applications. The upload and download stream is lower than 0.068% of the total energy. The reading and writing phase of the hard disk is the least consuming part of the laptop since their energy consumption is lower than 0.002%.
Figure 2.16 shows the energy consumption of users 1, 2, and 3 during two weeks, with the total energy of all applications (in gray), of Chrome (in blue), of Teams (in purple), and of Windows Explorer (in yellow, a file manager in Windows). Teams and Chrome represent a significant part of the overall energy consumption of a computer for these three users. In the first week, the sum of both applications is 57% (User 1), 48% (User 2), and 90% (User 3) of the overall energy consumption while in the second week, they represent 40% (User 1), 57% (User 2), and 37% (User 3) of the overall energy. Since Teams was less solicited for User 1 (no camera) during the second week, a reduction of energy is detected. During the first week, Teams represent 24% of the total energy consumption while it represents 12% in the second week.
Users 2 and 3 used Teams more during the second week which lead to an increase in energy consumption. It represents 30% (User 2) and 35% (User 3) of the overall energy in the first week while it represents 43% (User 2) and 25% (User 3) in the second. Finally, we see for users 1 and 3 that on 28/11 there is an energy consumption peak, $E_{\text{tot}} = 12.35\,\text{Wh}$ (User

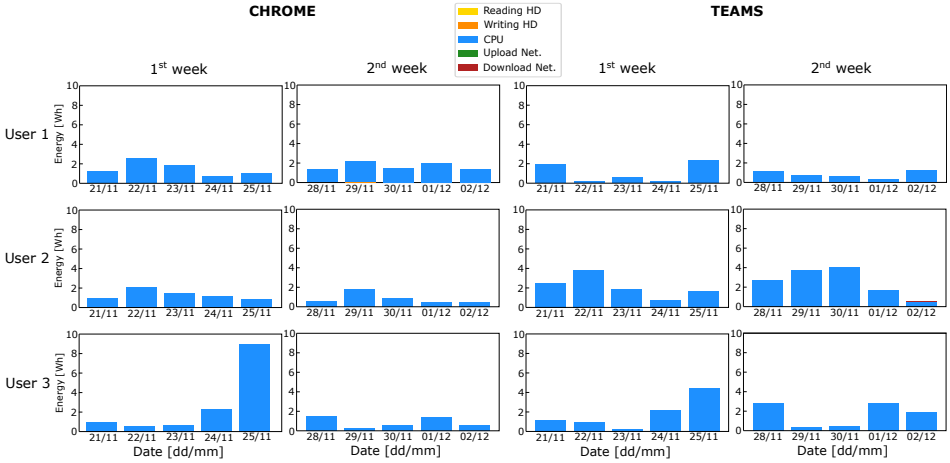


Figure 2.15: Energy consumption [Wh] per component (the reading phase of the hard disk is yellow, the writing phase of the hard disk is orange, the CPU is blue, the upload stream is green, and the download stream is red) of Chrome (on the left) and Teams (on the right) during two weeks for users 1, 2, and 3.

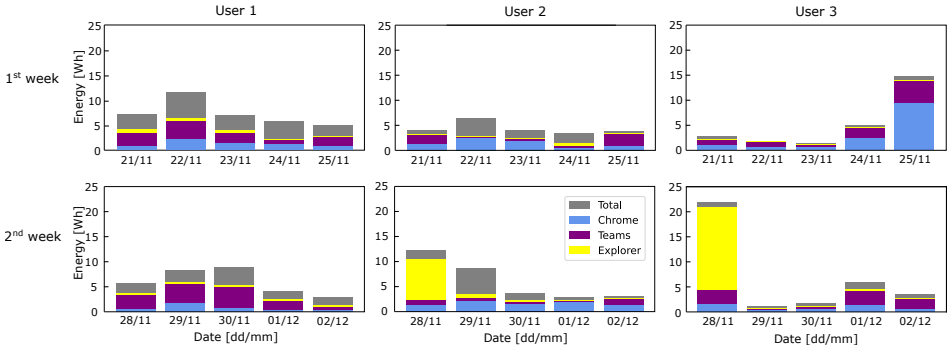


Figure 2.16: Energy consumption [Wh] per application (Explorer is yellow, Teams is purple, Chrome is blue, and the total energy consumption of the computer is gray) during two weeks for users 1, 2, and 3.

1) and $E_{\text{tot}} = 21.76\,\text{Wh}$. (User 3). This is due to the usage of Windows Explorer, while its energy consumption is very low during the other days. As we didn't collect specific and detailed software usage (through our tool or the questionnaire), it is difficult to analyze why we have this peak for a specific application on 28/11. However, Demeter allows users and system administrators to get an insight into software energy consumption for multiple days and weeks, and therefore identify energy leaks or abnormal energy behavior.

Effect on the User Behavior We recommended green good practices for users to modify their software behavior in order to evaluate their impact on the application's energy consumption. All of the recommendations were simple to follow for all three users and did not require significant changes in their workflows. However, some actions were more bothering for users, such as closing the browser's tabs. Since users are not used to frequently close their tabs, they had to constantly be aware and remember they have to close unused tabs. This constant awareness might either fade and users stop applying the recommenda-

tions, or it might turn into a habit. A similar conclusion is brought on the deactivation of the camera which is not a systematic habit for users. Moreover, this action involves other users who would like to keep their camera activated or wish to see the face of the speaker. Finally, we found that the two easiest recommendations to follow were reducing the music and the video quality. Both recommendations required some actions at the beginning (such as changing YouTube's parameters). A longer experiment might give us a wider overview of the impact of good practices recommendations on the software's energy consumption and on user behavior.
Because our study is a preliminary one aimed to showcase the capabilities of Demeter, it has a few limitations, most notably we did not follow proper field studies protocol. The study only consisted of three users, and lasted for only two weeks with the first week serving as a control study, and the second aimed to study the impact of the green recommendations.
Demeter is a software architecture capable of long-term monitoring of the energy consumption of software, and per component (CPU, network stream, and hard disk). We validated Demeter in a a real world preliminary experiment with three users in a corporate environment for two weeks. We observed a significant difference in energy consumption after providing users with green software good practices.
Demeter fills a gap in monitoring power consumption of software in desktop environments while also being tailored towards end users. It allows users but also corporate managers to measure the energy consumption of applications across a park or devices, and analyze these measurements and provide guidance and recommendations for users.

Epilogue: Demeter, as with Jolinar, measures the energy consumption of software in a black-box setup. This is limiting for optimizing software as observing software as whole does not provide a detailed and fine-grained view of energy consumption inside applications. A finer-grain view, across structure (methods, classes) and execution paths (execution branches, call tree), allows a better insight of what's happening inside applications, and provide developers with detailed observations allowing them to then optimize software.
In the next section, I describe JoularJX, my approach to monitor energy inside applications, across its structure and execution, and without requiring any modification to the application's source code.

2.3.5 JoularJX: Source Code Monitoring

> The content of this section is adapted from the following publication:
> PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. Adel Noureddine. In the 18th International Conference on Intelligent Environments (IE2022). Biarritz, France, June 2022. [Nou22]

JoularJX is a Java-based agent for power monitoring at the source code level with support for modern Java versions and multi-OS to monitor power consumption of hardware and software. Java is among the top 10 of the most energy-efficient languages [PCR+21] with its modern on-the-fly optimizations in the JVM, ranking even fifth in the normalized global results for energy.
JoularJX is a Java agent that hooks to the Java Virtual Machine (JVM) on startup along with the monitored application. It runs in a separate thread and collects information about CPU

usage of the JVM process, each thread running in the JVM, and then for each method and for each execution branch of the application.
JoularJX is the successor of Jalen, and the core approach of statistical sampling is based and inspired by the work we did in monitoring energy hotspots in software [NBRS12c, NRS15]. The general architecture of JoularJX is summarized in Figure 2.17:

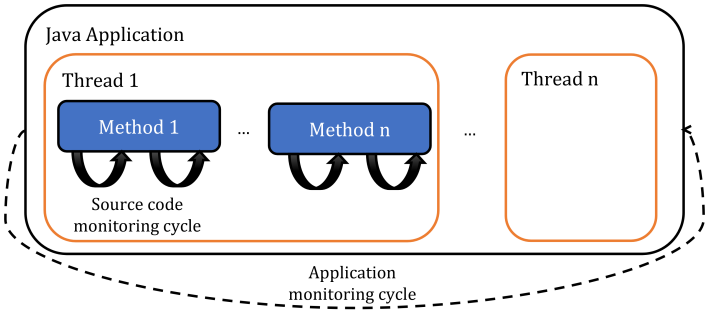
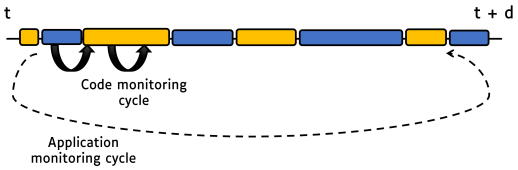


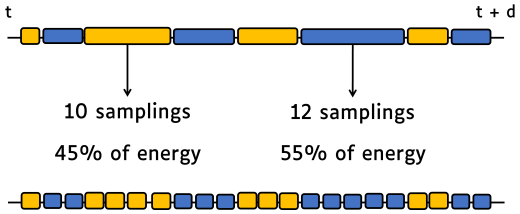
Figure 2.17: General architectural approach of JoularJX

The monitoring process is as follows:

- Every application monitoring cycle (by default, 1 second), JoularJX collects the CPU usage of the JVM and calculates the power consumption of the entire JVM (on Windows, using the PowerMonitor.exe program using Intel API, on x86_64 Linux using RAPL interface in the Linux kernel, on macOS using powermetrics command, and on Raspberry Pi and Asus Tinker Board using our regression power models).
- Then, JoularJX collects the CPU usage of each thread in the JVM using the JDK's `getThreadCpuTime` method, and calculates the power consumption of each thread.
- Every source code monitoring cycle (by default, 10 milliseconds), JoularJX checks, for each thread, the stacktrace and identify the method being executed (the method on top of the stacktrace), and also the execution branch (the methods in the stracktrace). At the end of the application monitoring cycle, JoularJX statistically analyzes the ratio of each method or branch observed in the stacktrace, and allocate the power consumption accordingly as shown in Figures 2.18a and 2.18b.



(a) The monitoring cycles of JoularJX



(b) Allocations power consumption per method in JoularJX

Figure 2.18: Architecture of JoularJX

Usually the method on the top of the stacktrace is a method from the JDK. For example, calling `System.out.println()` from the application's method `Main` method will call other

methods from the JDK (such as buffers, writeln, etc.). JoularJX will verify, when checking the stacktrace, if the method is called, anywhere in its call tree, by a method from the application we wish to monitor, and thus isolate these statistics from methods called by other applications or by the JoularJX agent itself.
During these monitoring cycles, JoularJX not only identify the method being executed, but also its execution branch (all the methods calling it), and can provide power and energy consumption for each execution branch.This is done by isolating every execution branch when analyzing the stacktrave on every monitoring cycle, and calculating its power consumption. For instance, if an application has three methods: A, B and C, with method C being called by A and B, then with JoularJX, we can identify the energy consumption of C when called by either A or B, distinctly. On more complex software, this allows developers to properly understand the power and energy consumption of each method, and where the energy draw is coming from. Consequently, developers can properly apply energy efficiency techniques, and eco-design their software also based on execution branches.
JoularJX will also automatically calculate the power consumption for each method, for each timestamp, thus allowing to trace the power consumption evolution of each method. This also applies to the JDK's methods too, therefore having both data and allowing an even more thorough analysis of energy consumption. Every method (of the application and/or the JDK, depending on the chosen setting in JoularJX) is tracked, its power consumption calculated every second, and can be outputted in a CSV time on runtime, or at the program's execution end.
Understanding the power evolution of each method allows developers to map the impact of external factors on the power consumption of methods. For instance, if a method depends on varying input data through its lifespan (such as reading different files, or changing variable input), then developers can follow and track the power change in real time.
To get power readings, JoularJX uses a custom PowerMonitor program (based on Intel Power Gadget API) on Windows, Intel RAPL (through powercap) on GNU/Linux, powermetrics on macOS, and our accurate power models on single-board computers such as Raspberry Pi devices which we detailed in Section 2.2.2.
Therefore JoularJX is capable of monitoring source code power and energy consumption on multiple architectures and platforms (x86 servers and computers, ARM single board devices), and operating systems (Linux, Windows, macOS). Therefore allowing developers to get a insight about their consumption across platforms, all while having the same feature set in JoularJX.

Features: As JoularJX is aimed towards software developers with a goal of providing in-depth monitoring to cover all use cases, it is built with a wide array of features and configurations.
JoularJX can be configured by modifying the config.properties files, with various options, with the most notable ones:

filter-method-names: list of strings which will be used to filter the monitored methods. JoularJX monitors all methods in a JVM, which also includes the methods of the JDK as these are the ones which will usually be on top of the stacktrace.
With this option, JoularJX can provide the power consumption of the methods of the application itself rather than those of the JDK. This power data is a recalculation done by JoularJX

to provide accurate power monitoring: methods that start with the filtered keyword, will be allocated the power or energy consumed by the JDK methods that it calls. For example, if `Package1.MethodA` calls `java.io.PrintStream.println` to print some text to a terminal, then the power consumption of `println` will be added to `MethodA`'s power consumption. We manage to do this by analyzing the stacktrace of all running threads on runtime.

overwrite-runtime-data: overwrite runtime power data files, or if set to false, it will write new files for each monitoring cycle. This feature is useful for integrating JoularJX into existing development workflows, such as with other tools, GUI, or dashboards. For instance, a GUI can continuously read a file and updates its interface on file rewrites, while avoiding having a large output file.

track-consumption-evolution: generate CSV files for each method containing details of the method's consumption over the time. Each consumption value is mapped to an Unix timestamp. With this feature, JoularJX will automatically calculate and store the power consumption evolution through time of every monitored method, thus allowing to get an overview of power consumption through the runtime of the application.

enable-call-trees-consumption: compute methods call trees energy consumption. A CSV file will be generated at the end of the agent's execution, associating to each call tree it's total energy consumption. When enabling call tree monitoring, JoularJX will identify, on every monitoring cycle (by default, every second), all execution branches of the monitored application, and calculates the power consumption of every branch. It will then output this call tree along with its power consumption in a CSV file as shown in Listing 2.1.

Listing 2.1: Example of call tree output in the CSV file

```
RayCasting.main;RayCasting.contains;RayCasting.intersects;
    RayCasting.printInfo,4.9287
RayCasting.main;RayCasting.contains;RayCasting.printInfo,75.8241
```

Use Case Scenario: To illustrate the functionalities of JoularJX, we analyze a custom version of the RayCasting algorithm in Java. The program has 4 methods: `main, intersects, contains, printInfo`, which are called by each other, including in a recursive way, and each prints multiple values to the terminal.
We run the program in four different platforms (Windows 11 PC, AlmaLinux 9.1 PC, Raspberry Pi 400 64 bits, and a RPi 3B+ 32 bits) including a variety of hardware and CPU architectures (x86, ARM, 32 and 64 bits), operating systems (Windows, Linux), kernels (5.14, 5.15), software components (Windows, AlmaLinux and Raspberry Pi OS), Java compilers, shells (PowerShell, Bash), and terminals. This variety is one of the strengths of JoularJX, allowing to analyze and compare software energy across platforms.
Figure 2.19 shows the total energy consumption of each individual method (in percentage compared to the total energy of the application) of the application on all platforms. This allows a better comparison of the internal distribution of energy in the application, across platforms.

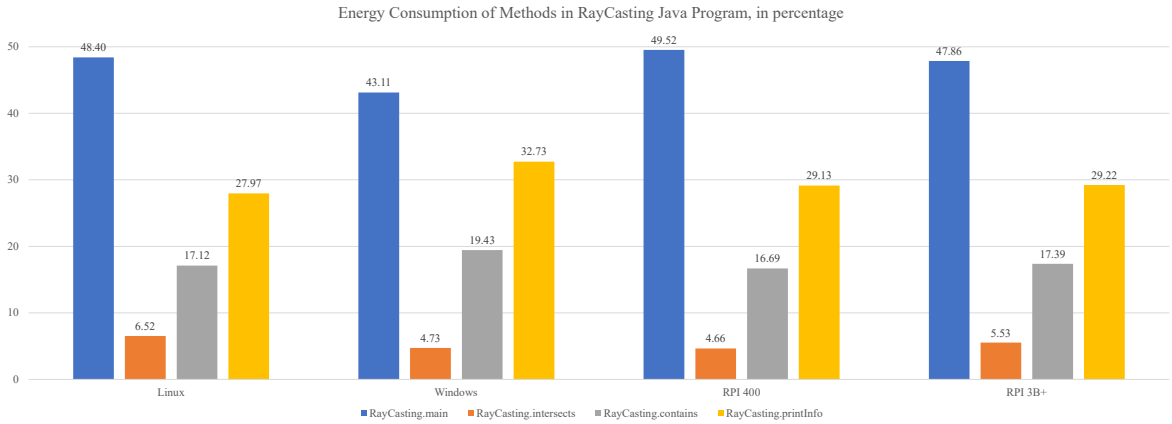


Figure 2.19: Energy consumption of methods in RayCasting Java program, in percentage

As JoularJX can also calculate the energy consumption of the JDK's methods, we observe that on Windows, 96% of the energy is consumed by `java.io.FileOutputStream.writeBytes` method (which is ultimately called when using `println`). In contrast, the number is 82.73% on desktop Linux, and at 86.44% on Raspberry Pi 400, and on Raspberry Pi 3B+, `writeBytes` is at lower 70.01%, followed by `java.io.BufferedOutputStream.flushBuffer` at 9.36%. RPi 3B+ has the lowest specifications of the four platforms and the only one in 32 bits, therefore a lower maximum capacity for addressing RAM memory. This might explain why flushing the buffer is more frequent, and therefore consuming up to 10% of the total energy of the application.

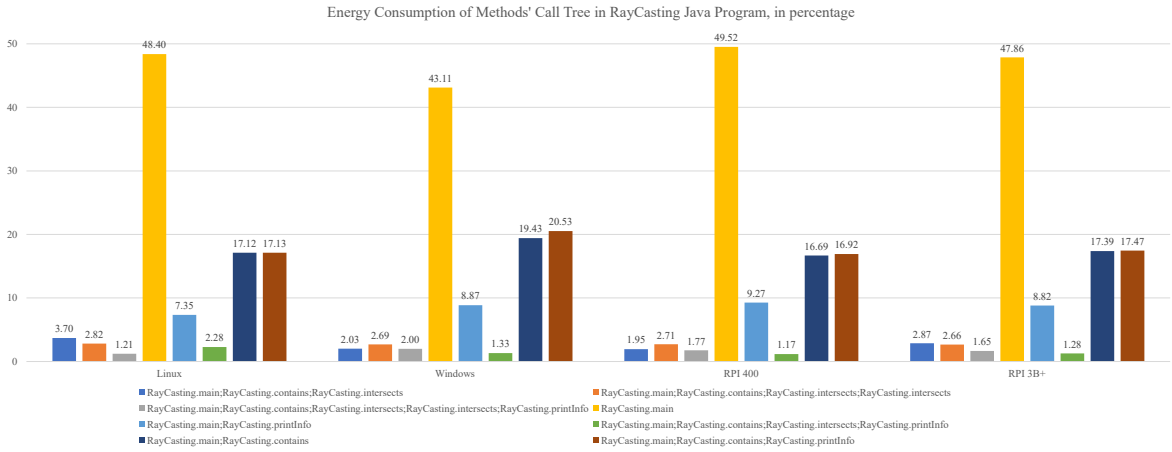


Figure 2.20: Energy consumption of methods' call trees in RayCasting Java program, in percentage

The call tree monitoring capabilities of JoularJX can be seen in Figure 2.20, which outlines the total energy consumption of all call branches of the application (can also monitor those of the JDK too), in percentage and across platforms. As we can observe, the distribution between call trees is stable across Linux platforms. In Windows, `RayCasting.main` call is responsible of 43.11% of total energy (in comparison to 48-49% on desktop and RPi Linux), where the near 5% difference is distributed across the other call branches.

Call trees also allow developers to identify the energy impact of recursive methods, such as the `intersects` method, and isolate the energy consumption for the main calls (`main` → `contains` → `intersects`) and the recursive calls (`main` → `contains` → `intersects` → `intersects`).
JoularJX can also outline the power consumption evolution of each individual method (and those of the JDK) of the application.

JoularJX GUI: JoularJX track all the methods and all the execution branches of the application and the JDK on runtime, every second. As such, it generates huge quantities of data and files. We developed a graphical user interface (GUI) that facilitate viewing, processing and analyzing the energy data of the monitored applications.
The GUI allows viewing all the results of JoularJX and selecting the specific application run to show. Then, it offers two main views:

- Power and energy consumption of methods: allowing to view the total energy consumption and percentage distribution of all methods of the application (including or not those of the JDK), along with the historical view of the power consumption evolution of each method.
- Power and energy consumption of execution branches (call tree): allowing to view the total energy consumption of every execution branch of the application, along with the percentage distribution and a graphical view of the execution branch.w

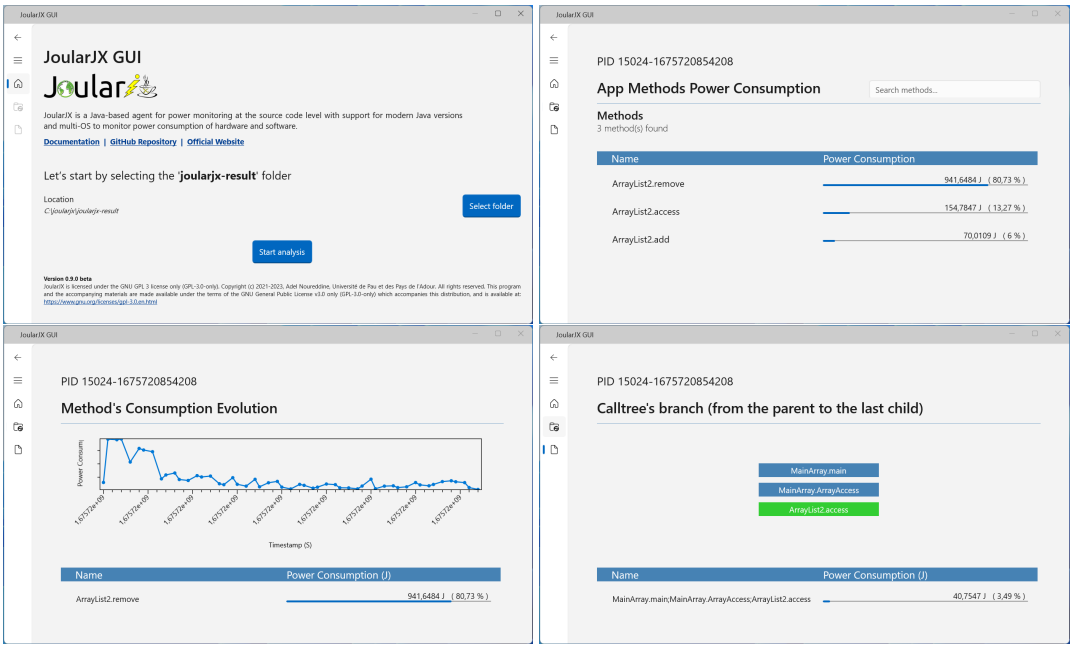


Figure 2.21: JoularJX GUI: a) Main window, b) Methods view, c) Historical method's evolution view, and d) Execution branch view

Figures in 2.21 outlines the main windows of the GUI. The GUI allows developers to easily access the main analysis and results of JoularJX, in an easy-to-access visual format. As developers often resist change in the software process [AS21], it is important to provide them

with easy-to-acquire tools, and automate the process as much as possible. JoularJX and its GUI achieve this by automating the analyzing process, in the Java agent and in the GUI, and by displaying in a easy-to-understand visual format and data (charts, percentages, visual execution branches).
JoularJX is aimed towards software developers in helping them understand and analyze the power footprint of their software and source code, across multiple platforms and devices.

2.4 Epilogue

Observing heterogeneous software ecosystems is complex due to the diversity of devices, components and software, to the rapid pace of software updates and hardware changes, and to the role humans have in these environments (developers or end users).
In my contributions, I proposed approaches and solutions in every layer (from devices to source code), and for each actor (from technical developers to managers and end users). These different building blocks, integrating all actors, help constructing and achieving my vision of a holistic approach to manage energy of software ecosystems.
However, there are still open challenges that require mid- and longer-term research to properly address. For instance, observing software energy is often achieved after development and deployment, and after most of the software development process has been completed. Integrating energy measurement and estimations into development workflows would help observe and understand software energy early on in the development process. Also, observing energy is a necessity not only on the final software product, but the process of developing, deploying and maintaining software is also energy-costly. Measuring and analyzing the energy impact of the entire life cycle of software is an important challenge we need to tackle. In addition, developers and practitioners need to be trained on green aspects, and the impact software has on energy consumption. Finally, measuring energy consumption gives a part of the bigger sustainability picture, and it is primordial for the next decade that we also accurately measure and map other environmental metrics to software. The challenges we face and my perspectives over them are detailed in the perspective section in my concluding Chapter 4.
Observing software is only but a step in the general goal of energy efficiency. Understanding why energy is being consumed will help all actors in adopting energy efficiency strategies and solutions. My contributions on understanding green software, and initial steps to make things better, are described in the next Chapter 3.

Chapter 3

Understanding and Acting Towards Green Software Ecosystems

Contents

3.1 Introduction

One of the first steps of the scientific method is observation. From it, we ask a question and form a hypothesis, then perform experimentations and analysis to test the hypothesis and conclude. My approach adopts the scientific method, and after observing the energy consumption of software ecosystems, and providing models, architectures and tools to do so, it is important to understand and make sens of these observations, and later on take decisions on actions to be applied.
The main question I ask is *why* energy is consumed in software ecosystems, *i.e.*, what are the factors impacting software energy, from technical to human ones. With this knowledge, we can start acting towards making things better, and making software ecosystems more energy efficient.

Table 3.1: An overview of my contributions in understanding and acting towards green software ecosystems

Organizational Issues & Information Systems		Users Green Feedback	
Application Technologies			
Software Methods & Technologies		Design patterns / Code coverage	/ Feature slices
Systems Infrastructure			
Computer Hardware & Architecture	H/W & S/W features	EURECA Tool	Automated Framework

In presenting my contributions in this chapter, I adopt the same layers identified by the ACM Computing Curricula [acm05] in their *Problem Space of Computing* model, that I presented in the introduction and used in the previous Chapter 2.
My contributions are detailed in the following sections, as adaptations of my different scientific publications. Table 3.1 summarize my contributions using ACM's aforementioned model. Many of my contributions target multiple layers, with a main layer (symbolized with a dark circle), and secondary layers (symbolized with an open circle).
In particular, I present two main research axes: understanding the factors impacting software energy, and acting towards greening software ecosystems.
In the first axis, my goal is to provide knowledge and studies in order to answer the *why* question. More specifically, now that we can observe software energy, we need to understand why this consumption is happening, and understand the factors that have an impact on it. Such understanding is key towards optimizing and managing energy. Specifically:

- Software and computing hardware are very complex structures, comprising of various

components, logical designs, and characteristics. In Section 3.2.1, we analyze the impact of over 100 hardware and software features and metrics on energy consumption, including static and dynamic features.

- In Section 3.2.2, we analyze the impact of higher-level design choices, in particular the impact of software design patterns on energy consumption, and propose a transformation approach to reduce energy while keeping the benefits of the design patterns.
- Measuring energy consumption of source code is often done at the methods or functions levels. This vertical measurement and analysis are oriented towards software constructs. In Section 3.2.3, we explore measuring and analyzing software energy at the feature-level, in a more horizontal cross-cutting view, with program slicing.
- After design and programming, one of the latest steps in software developments is testing. In Section 3.2.4, we analyze the impact of unit tests and code coverage on energy consumption, in order to understand if well-tested code is energy-efficient.
- Finally, in Section sec:greenfeedback, we study the impact of the human factor in software energy, as we explore the impact of green feedback on user energy awareness and their behaviors in using software, and conclude on lessons learned and perspectives for improvements.

With better understand of what's happening and why energy is being consumed, we can leverage all the knowledge we build, along with our approaches for observing green software, to start acting towards a greener software ecosystem. In this, I propose two initial approaches aimed to help users make better green decisions, and also in automating such decisions when possible. In particular:

- In Section 3.3.1, we present the EURECA Tool which aims to help data center procurers and technicians to make better green decisions based on state-of-the art assessment and recommendations from the EU Code of Conduct, DCMM and other data center certifications.
- And in Section 3.3.2, we propose an autonomous energy management architecture capable of applying appropriate green decisions in cyber-physical systems, while learning from users' behaviors and respecting their preferences.

My contributions cover different aspects of software ecosystems, with a goal to understand what's happening uniquitously, *everywhere*. From source code and low-level features, to software architecture and design, to cyber-physical systems, and data centers. In the following sections, I present in detail my different contributions in understanding and acting towards green software ecosystems.

3.2 Understanding Technical and Human Factors

My contributions in understanding the technical and human factors impacting the energy consumption of software ecosystems, aim to cover software ubiquitously, *i.e., everywhere*, that is at the different layers of a complex system. First, I present a study on the impact

of hardware and software features on energy consumption, with an aim to relate software energy to the characteristics of software and also to the computing hardware it is running on. Then, I analyze higher-level software constructs, with analyzing the impact of design choices, such as software design patterns on energy consumption, and I provide an approach to make things better without losing the benefits of design patterns. In addition, I propose viewing and analyzing software energy by looking at software horizontally, *i.e.*, at a feature level rather than looking at software constructs. This unique view can lead to better understanding of the energy dynamics inside software. Software development process is never complete without the testing phase, and hence I also try to understand the impact of unit tests and code coverage on energy consumption. These studies cover multiple layers and vary across the different steps of software development and usage. They are not complete without integrating the end user in this loop, and in my final contribution in this section, I analyze the role of end-user software usage with green feedback, with a goal to study whether such feedback can lead to user behavioral change and energy reductions.

3.2.1 Software and Hardware Features

> The content of this section is adapted from the following publication:
> A Study on the Influence of Software and Hardware Features on Program Energy. Ajitha Rajan, Adel Noureddine, and Panagiotis Stratis. In International Symposium on Empirical Software Engineering and Measurement (ESEM'16). Ciudad Real, Spain, September 2016. [RNS16]

Software energy consumption has emerged as a growing concern in recent years. Managing the energy consumed by a software is, however, a difficult challenge due to the large number of factors affecting it — namely, features of the processor, memory, cache, and other hardware components, characteristics of the program and the workload running, OS routines, compiler optimizations, among others. We study the relevance of numerous architectural and program features (static and dynamic) to the energy consumed by software. The motivation behind the study is to gain an understanding of the features affecting software energy and to provide recommendations on features to optimize for energy efficiency.
In our study we used 58 subject desktop programs, each with their own workload, and from different application domains. We collected over 100 hardware and software metrics, statically and dynamically, using existing tools for program analysis, instrumentation and run time monitoring. We then performed statistical feature selection to extract the features relevant to energy consumption. We discuss potential optimizations for the selected features. We also examine whether the energy-relevant features are different from those known to affect software performance. The features commonly selected in our experiments were execution time, cache accesses, memory instructions, context switches, CPU migrations, and program length (Halstead metric). All of these features are known to affect software performance, in terms of running time, power consumed and latency.
Measuring and managing the energy consumed by a software is, however, a difficult challenge due to the large number of factors affecting energy consumed — namely, characteristics of the processor (clock frequency, voltage, pipeline depth), memory (instruction and data cache reads, writes and misses), communication network, GPU, sensors for mobile devices, compiler and optimizations enabled, features of the program and workload running

(number of instructions, input data, number of branches, number of operators and operands, algorithmic complexity, multi-threading) among others. Before we embark on proposing optimizations for software energy consumption, we believe it is imperative to first identify and understand the factors that contribute to energy consumption. Our ultimate goal is to use this understanding to guide techniques in choosing factors to optimize for energy.
Our empirical study does not aim to estimate energy but instead tries to understand the statistical relevance of features, both architectural and program specific when taken together, to energy consumed.

3.2.1.1 Experiment Description

The experiment we conducted is comprised of two phases:

Phase 1: Data Collection: We ran a series of experiments on programs from three different benchmark suites to measure energy consumed and features related to the hardware and program. The programs in our experiments are from a range of application domains such as image processing, biomolecular simulation, networking, automotive, fluid dynamics, operating system, signal processing among others. The programs perform diverse tasks varying in the type and intensity of computation, file I/O, amount and frequency of memory operations, and several other features.
The three benchmark suites are:

- 9 programs from the **SIR** (Software-artifact Infrastructure Repository) repository [DER05]. Programs include GNU projects (gzip 1.0.7 and sed 1.17), two lexical analyzers (printtokens and printtokens2), two priority schedulers (schedule and schedule 2), a pattern matching program (replace), a statistics program (totinfo), and an aircraft collision avoidance system (tcas). Test cases were either provided, or generated using the Make-test-script program and test plans described in universe files. The test cases exercise the parameters and behavior of the programs as described in [DER05].

- 11 programs from the **Parboil** benchmark [SRS+12], which is an open source benchmark suite with benchmarks collected from throughput computing application researchers in many different scientific and commercial fields including image processing, biomolecular simulation, fluid dynamics, and astronomy. Applications in the benchmarks perform a variety of computations including ray tracing, finite-difference time-domain simulation, magnetic resonance imaging, 3-D stencil operation. Parboil provides applications implemented as both serial C++ and OpenMP, along with workloads specific to the programming model. We run both the base and parallel OpenMP versions of these programs.

- 38 programs from the Embedded Microprocessor Benchmark Consortium suite [PLGOC09] (**EEMBC**). EEMBC provides a diverse suite of processor benchmarks organized into categories that span numerous real-world applications, namely automotive, digital media, networking, office automation and signal processing, among others. There are a total of 38 programs: 17 from automotive domain, 6 from consumer, 4 in networking, 5 in office, and 6 from telecom.

In addition, we collected around 100 metrics ranging from 3 main categories:

- **Hardware performance features** reflecting CPU, cache and memory performance. We measure these features since they are known to impact power and energy [BTM00, DP14, DRS09, GHV99]. We use Linux's Perf tool [per] to collect 37 hardware features, in addition to time, ranging from performance features, branch instruction features, cache-related features, and page fault features.

- **Dynamic program features** relating to types of program instructions generated, memory operations performed, and register utilization. Different types of instructions are associated with different base energy costs [TMWL96]. Inter-instruction effects, and operands also affect the energy consumed. Memory operations are also associated with high energy costs. Finally, effective register utilization during machine code translation is shown to help improve energy efficiency [RJ98]. We use the Pin tool [LCM+05] (version 2.14) to collect 42 dynamic program features, ranging from program instructions, memory instructions, and processor registers. Pin is a dynamic binary instrumentation framework where instrumentation is performed at run-time on the compiled binary files.

- **Static program features** such as the number of basic blocks, number of loops, branches, global variables, cyclomatic complexity. We measure these high-level program features since they have been shown to be useful in estimating energy consumed by software [TRLJ01], and are the primary guide in creating and measuring adequacy of test runs (over which execution time and energy are measured) [SWRH10, HWRS08, Raj06]. We collect 25 static program features using Frama-C's [CKK+12] metrics plugin [fra] (Sodium release), ranging from Halstead complexity measures [Hal77], McCabe's cyclomatic complexity [McC76], and other static program features.

Finally, we measure the energy consumed by the computer for each program run using a *Watts Up? .Net* power meter [wat]. The power meter is connected to another computer that collects energy observations at runtime.

Phase 2: Prediction Models: Using the measurements in the data collection phase, we trained our model to select features [22] that are relevant to the energy consumed.
The statistical feature selection technique we use is Lasso regression [Tib96]. Lasso allows to select a subset of statistically relevant features to improve the overall performance of the learning model by preventing overfitting. Feature selection also eliminates the need for measuring features that are not relevant, or are redundant to the energy consumed. Lasso regression is often preferred because it produces sparse models effectively, *i.e.*, the regression coefficients for most irrelevant or redundant features are shrunk to zero, thereby achieving feature selection [LYX05].
In order to validate the accuracy and usefulness of the selected features, we also trained and constructed a linear regression model using all the features. We then compared the energy prediction error of Lasso regression (with the selected features) against that of linear regression using all the features.
We ran our experiments using a desktop computer powered by an Intel Core 2 Duo E8400 processor at 3 GHz, 2 GB of DDR2 memory at 800 MHz, 128 KB of L1 cache and 6 MB of L2 cache. The machine was running Ubuntu Server 14.04 with Linux kernel 3.16.0.33.

[22] Features in our estimation model refers to the hardware and program-related metrics.

Validation: We use cross validation, in particular, Leave-One Out Cross Validation (**LOOCV**) to evaluate the performance of the different prediction models. For each regression model, we compute, using LOOCV, the Root Mean Squared Error (RMSE) which is the square root of the mean of the squared errors over n different predictions. We then normalize the RMSE to the range of the observed response using the minimum and maximum actual response. NRMSE is a popular error metric to compare numerical prediction errors between models. Compared to mean absolute error, RMSE amplifies and severely punishes large errors, such as those from outliers.

3.2.1.2 Experimental Results and Analysis

Each analysis was carried out as follows: we used the Lasso learning technique to select a subset of features from all the hardware and program specific features observed. We assessed the usefulness of the selected features by comparing the NRMSE for energy prediction using Lasso against that of linear regression with all features. We list the relevant features selected by Lasso and their coefficients. The coefficient values of the different features cannot be compared directly since the magnitudes and units of the measurements are not comparable. For instance, execution time is measured in milliseconds, while a Halstead metric gives the number of distinct operators and operands.
We collected observations of energy and features for programs with their respective workloads in the three benchmark families, and also individual analysis of the 9 SIR programs. We did not conduct per program analysis for EEMBC and Parboil benchmarks since the number of test runs available per program was significantly smaller than the number of features measured. Predictions for such programs in our high-dimensional feature space will be highly inaccurate due to the curse of dimensionality [Bel61]. The individual results for the 9 SIR programs are available in our paper [RNS16]. I'll present here a summary of the results within each benchmark family with the NRMSE for all benchmarks in Table 3.2a.

	Linear	LASSO
SIR	0.000971891	0.000827693
EEMBC	0.004287477	0.005514576
Parboil	0.051841246	0.074844556

(a) Normalized Root mean square error (NRMSE) for all benchmarks

Features selected	Absolute Weight
execution.time	66.70251726
cache.misses	0.000154405
cpu.migrations	0.234642543
realloc	0.001386825
calloc	0.001374207
Program.level	0.138383951
Function	0.000190104

(b) Features selected by LASSO for all SIR benchmarks

Table 3.2: NRMSE for all benchmarks and features selected for SIR

SIR Benchmarks NRMSE numbers are low across both prediction models (see Table 3.2a). The features selected with Lasso are marginally more effective in predicting energy than all the features in the linear model.

Table 3.2b shows the features selected by Lasso, and also shows the coefficient values for the selected features. The coefficient value for execution time is 10^2 to 10^5 times larger than coefficient values for other features. Although, coefficient values in such models cannot be directly compared due to the different units of measurements, it provides a hint on the importance of execution time for energy prediction in the SIR benchmarks. To assess the importance of execution time for energy prediction, we compared LASSO prediction errors with and without execution time in the feature set.

	All	All - Time	Only Software	Only Hardware
NRMSE	0.000827685	0.003193454	0.003193454	0.003183806
Features	execution.time, cache.misses, cpu.migrations, realloc, calloc, Program.level, Function	cpu.migrations, Program.level, cpu.clock, context.switches, task.clock, calloc, LLC.load.misses, Exit.point	Program.level, calloc, Exit.point, Free	cpu.migrations, context.switches, cpu.clock, task.clock, LLC.load.misses, minor.faults

Table 3.3: NMRSE and feature selected by LASSO for SIR benchmarks

Table 3.3 lists the NRMSE for LASSO with *All* features, *All - Time*, *Only Software*, and *Only Hardware*. As can be seen, NRMSE increases by 3 times when execution time is removed from the observations. `execution.time` is, thus, a key consideration, and cannot be approximated by any set of hardware or software features for SIR benchmarks.
Table 3.3 also shows that prediction accuracy is approximately the same for *All - Time*, *Only Software* and *Only Hardware* with *Only Hardware* being marginally better. For SIR programs, measuring either only the program characteristics or only the hardware features, along with execution time would suffice to predict energy.
The size of the program indicated by the Halstead metric, `Program.level`, is also prevalent. This implies that the number of distinct operators and operands in the source code for the SIR programs has an effect on energy. `cpu.clock` and/or `task.clock` are selected for all individual SIR programs, but is not seen in the LASSO selection across all programs. The software metrics, `Program.level`, `Function`, and `calloc` have effectively replaced `cpu.clock` and `task.clock`. This is confirmed again in Table 3.3 when we perform Lasso selection with *Only Software* metrics which has the same prediction accuracy as Lasso using *All — Time*.

EEMBC Benchmarks For the 38 programs in the EEMBC benchmark, the Linear regression model performs better with an NRMSE of 0.0042 compared to the Lasso model with an NRMSE of 0.0055 (see Table 3.2a). The prediction errors are, however, comparable and as a result, the features selected can be considered further for optimizations.
Table 3.4a shows the features selected by Lasso. As with the SIR benchmarks, execution time is selected and the coefficient value (without normalizing) is larger than the other features. However, execution time is not as significant for energy prediction as in the case of SIR benchmarks. This is evident from Table 3.5 where the prediction error of Lasso using *All* features is comparable to Lasso using *All — Time* features. *Only Software* and *Only Hardware*

Features selected	Absolute Weight
execution.time	4.834222931
cpu.clock	0.004460450
task.clock	0.004313552
context.switches	0.040315249
cpu.migrations	1.755253195
branch.misses	6.10E-08
bus.cycles	1.38E-08

(a) Features selected by LASSO for all EEMBC benchmarks

Features selected	Absolute Weight
execution.time	58.93799739
cpu.clock	0.001014987
task.clock	0.001215820
L1.dcache.prefetches	6.34E-07
dTLB.load.misses	8.24E-07

(b) Features selected by LASSO for all Parboil benchmarks

Table 3.4: Features selected for EEMBC and Parboil

features also have comparable accuracy in energy prediction. This implies that targeting optimizations at one of these sets of features will be effective in achieving energy efficiency.

	All	All - Time	Only Software	Only Hardware
NRMSE	0.005921402	0.005999306	0.006079861	0.006016273
Features	execution.time, cpu.clock, task.clock, context.switches, cpu.migrations, branch.misses, bus.cycles	cpu.migrations, context.switches, cpu.clock, task.clock	Sloc, basic.blocks	cpu.migrations, context.switches, cpu.clock, task.clock, LLC.load.misses

Table 3.5: NMRSE and feature selected by LASSO for EEMBC benchmarks

`cpu.clock` and `task.clock` are among the other features selected. Contrary to the execution time - which includes wait time, these two clock features monitor time spent using the CPU. The CPU-intensive nature of some of the EEMBC benchmarks drives these observations. Examples of such CPU intensive benchmarks are the image processing, and automotive applications. Branch misses is selected and this may be due to the extensive use of loops in all the EEMBC programs. We believe the feature, `bus.cycles`, was selected due to frequent communications between the CPU and the main memory. Programs in the telecommunications domain, Encoder, Decoder, and Bit allocation, make heavy use of such communications.

Parboil Benchmarks For the Parboil benchmarks, Linear regression model does better with an NRMSE of 0.0518, as seen in Table 3.2a. Lasso has a slightly higher error of 0.07484. Among the 3 benchmark families seen thus far, NRMSE is highest for Parboil across prediction models. We believe there are two reasons for the high NRMSE values for Parboil. (1) The number of observations available for Parboil is fewer than the other two benchmarks (51 for Parboil versus 304 for EEMBC and 270 for SIR), resulting in the dimensional feature

space being much bigger. As a result, the effect of the curse of dimensionality is more significant in Parboil. (2) Parboil benchmarks have both the serial and parallel versions. Number of threads is the only feature in our measurements that reflects the parallel nature of benchmarks. It may be the case that this single feature is inadequate in capturing the effects of parallelism on energy.
Table 3.4b shows the features selected by Lasso and the absolute weights of the coefficients. The coefficient value of execution time is significantly larger than that of the other selected features.

	All	All - Time	Only Software	Only Hardware
NRMSE	0.074839977	0.11318387	0.277245127	0.11318387
Features	execution.time, task.clock, cpu.clock, dTLB.load.misses, L1.dcache.pref	cpu.migrations, task.clock, cpu.clock, dTLB.store.misses, dTLB.load.misses, L1.dcache.pref, LLC.load.misses, branch.misses	Global.variables, Goto, Function.call, rdx.written, Pointer.dereferencing, rcx.written, rdx.read, assignment	cpu.migrations, task.clock, cpu.clock

Table 3.6: NMRSE and feature selected by LASSO for Parboil benchmarks

As can be seen in Table 3.6, prediction error almost doubles when execution time is not included (*All — Time*) in the observations, reflecting its importance for energy estimation over Parboil programs. It is also worth noting that measuring *Only Hardware* features outperforms measuring *Only Software* features. We can safely say that energy consumed by Parboil programs are characterized better by hardware features than software. Many of the Parboil programs are CPU intensive, resulting in `cpu.clock` and `task.clock` being selected. The programs use large input data sets. Frequent accesses to these large data sets without temporal and spatial locality result in data cache misses being an important feature for energy estimation.

3.2.1.3 Discussions

From our experiments, we find that the overall accuracy of the Lasso feature selection technique is comparable to the Linear regression model with all the features. The Lasso technique did not always outperform the Linear model and there was no consistent best model across all experiments. Given the high dimensional feature space, we believe comparable prediction accuracy using the Lasso model is a good start and the selected features can be used as a basis for discussion on potential optimizations for energy. It is worth noting that feature selection using Lasso takes correlation between measured features into account and eliminates redundancy. For instance, features like basic blocks and global variables have high coefficient values in the linear regression model but are not selected by Lasso since they are redundant with respect to execution time. When correlated features are selected, it implies that the features together better predict energy than any one of them (they are not redundant with respect to each other or a combination of other selected features).
We found that some hardware and program features were repeatedly selected across all our

experiments. We list these features (or sets of features) and discuss their potential as energy optimization targets. We also go on to discuss if the selected features and optimizations are any different from those already known for software performance.

Execution Time: Execution time has often been used as a representative for energy [CGSS14, PHZ12, LWG05]. Lasso feature selection over all benchmarks resulted in coefficient values for execution time being 10 to 100 times larger than coefficient values for other features. Although, coefficient values in such models cannot be directly compared due to the different units of measurements, it provides a hint on the importance of execution time for energy. It is clearly valuable to target execution time for energy optimization. However, it is important to bear in mind that energy is dependent on both execution time and power, and reducing one while increasing the other will offset the gains achieved.

Cache and Memory instructions: Our experiments consistently selected cache (L1, LLC, and TLB) features, and dynamic memory instruction features. Missed cache hits, on both the internal and external cache, as well as page faults slow the performance of a program [TMWL96, VKI+00] by significantly increasing memory latency and, therefore, execution time. Note, however, that the cache features being selected by Lasso implies they are *not redundant* with respect to execution time. The selection implies that it is potentially beneficial to optimize cache features in addition to optimizing execution time. For instance, if we have a memory-intensive process whose performance is limited by the available memory bandwidth (cache and memory access speeds are slower than processor speeds), we will gain limited execution time gains from increasing processor clock frequency. The increased clock frequency will result in increased power consumption that may offset the limited execution time gains. Thus, it may be beneficial to move memory-intensive processes to lower frequency cores, while executing less memory-intensive processes on higher frequency cores.
To reduce cache misses and memory latency, one can place related data close in memory and use algorithms and data structures that exploit the principle of locality. We use the Polly data locality loop optimizer [pol] in LLVM over SIR and the automotive EEMBC benchmarks, in order to assess the energy gains from optimizing data locality within loops in these benchmarks.
The energy gains achieved over SIR benchmarks using Polly for loop optimizations is relatively low, maximum being 6.36% (over *printtokens2*. Full results in our paper [RNS16]). This is primarily because in SIR programs, loops are used sparingly with relatively few data and array references within them. As a result, there is not enough opportunity for Polly to improve data locality. For instance, in the case of the *tcas* program where almost no energy gain was observed, loop structures are not present in the program and as a result Polly has no effect. *printtokens*, on the other hand, is a program which accepts a file containing a stream of characters and for each character (outer loop) it performs an expensive search to find the corresponding token (inner loop). This loop structure provides relatively more opportunities for Polly to optimize than in the case of *tcas*, resulting in the higher energy gain of 5.06%.
Energy gains follow execution time gains for most SIR programs. This is because, improved data locality and thus, reduced cache misses, helps reduce execution time and not power. For *schedule* and *tcas* programs, we observe no gain in execution time, however negligible gains in energy because of power is observed. It is difficult to pinpoint the exact reason

for the slight gain in power and can often be simply attributed to small fluctuations in the processor environment.
For the EEMBC benchmarks, we ran the Polly optimism over the automotive domain comprising of 16 programs and associated workloads. In complete contrast to the SIR benchmarks, we observed massive energy gains with Polly, ranging from 45.74% to 83.94%. The large gains can be explained by the program structure of the EEMBC benchmarks, predominantly comprising of statements in for loops and array accesses. Polly's loop tiling and loop fusion optimizations perform extremely well over such program structures. Polly's data locality optimizations resulted in execution time gains that were similar to the energy gains observed, ranging from 44.96% to 83.53%. Power gains over the EEMBC programs were relatively low, and even negative for 3 of the 16 programs. The observations for EEMBC supports our earlier inference from SIR that energy gains are driven by execution time, rather than power, gains when using the Polly optimism.

Context switches and CPU migrations: Both these features are relevant to energy consumption and are known to cause overheads in terms of running time [LDS07, DCC07] and power [DP14]. Switching a process context or migrating it to a different CPU core is a costly task and tends to generate many cache misses (since the process may not find valid data on the new migrated core's cache). CPU scheduling algorithms have been proposed to help tackle this problem to reduce the power and energy consumed [DP14]. Some multi-core programming models (like OpenMP and MPI) allow developers to use environment variables that pin processes to CPU to prevent migration to a different core.

Program Level: This halstead metric was selected for the SIR benchmark family. It is worth noting that for EEMBC, the linear regression model assigned the highest coefficient value to this feature. Lasso did not select this metric since it was, likely, redundant with respect to execution time. We hypothesize that reducing the number of operators, operands and function calls (that are used to compute this metric) in a program will help in reducing energy consumption. In a previous study, we found that optimizing design patterns by reducing function calls improved energy consumption [NR15].

Optimizing for Energy versus Performance: The features picked by our empirical study that affect software energy, namely execution time, cache misses, context switches, CPU migrations, program length (in terms of number of operators, operands and function calls) are no different from those known to affect software performance with respect to running time, memory latency, and power consumed. Datta and Patel propose CPU scheduling algorithms to mitigate performance loss, in terms of power, incurred from context switches, and CPU migrations [DP14]. David et al. and Li et at. have also studied the performance overhead introduced by context switches [LDS07, DCC07]. Cache misses are widely known to impact performance in terms of execution time [YBD01] and existing optimizations for data re-use like Polly achieve significant energy gains by reducing cache misses. Compiler optimizations built into existing compilers, like the GCC optimization flags, attempt to reduce the code size and execution time of the program [KVIY00, PHB15a].
All of the existing optimization techniques proposed for performance, either focus on power or on execution time, but not both. Existing performance optimizations are effective in reducing energy and this has been observed in several studies. It is, however, important to

bear in mind that execution time gains do not always translate to energy gains. An example of this, is frequency scaling in processors that reduces execution time by running at full-clock speed.
Miyoshi et al. observed that scaling CPU frequency, although reducing execution time, can cost more energy than running at a slower clock speed [MLVH+02]. Typical techniques for reducing power are power management software and dynamic voltage frequency scaling in processors. Our study does not consider these techniques since they focus on processors rather than a specific application.
The features affecting energy selected in our study lead us to believe that existing performance optimizations will suffice if one looks to reduce energy by reducing either only power consumed or only execution time. To achieve energy gains that goes beyond existing optimizations, we need to explore techniques that target both power and execution time.

Epilogue: We have studied energy consumption of programs and workloads from three different benchmark families measuring over 100 different software and hardware features. The feature selection technique, Lasso, effectively picks five to seven features for each of the benchmark families that are relevant to energy consumption. The accuracy of energy predictions with the selected features was comparable to energy predictions using a linear regression model with all the features. The features commonly selected in our experiments were execution time, cache accesses, memory instructions, context switches, CPU migrations, and program length (Halstead metric). All of these features are known to affect software performance, in terms of running time, power consumed and latency. As a result, we can use existing performance optimization techniques that reduce power or execution time to also achieve energy gains. We confirmed with the use of Polly, a compiler optimization for data locality, that significant energy gains were possible by reducing latency and execution time. We believe the future in energy optimization lies in techniques that can reduce both power and execution time.
However, optimizing low-level hardware and software features is limited mostly to source code microoptimizations, and most importantly, to implementing such optimizations in compilers. For developers, understanding energy consumption of software and applying optimizations techniques can be effective at the design phase, and in adopting higher-level understanding of their software logics and design. In the next sections, we explore multiple design-phase approaches aimed to provide better understanding and optimizations to software energy consumption.

3.2.2 Design Patterns

The content of this section is adapted from the following publications:
Optimizing Energy Consumption of Design Patterns. Adel Noureddine, and Ajitha Rajan. In New Ideas and Emerging Results (NIER) of the 37th International Conference on Software Engineering (ICSE'15). Florence, Italy, May 2015.

Software design patterns [GHJV95] are time-tested solutions to recurring design problems, and are widely used by practitioners to provide improved code readability, maintainability and reuse. They also facilitate communication between software engineers.

In recent years, energy consumption has emerged as an important design constraint when writing software, especially in the domain of embedded systems where a strict power budget is imposed [PHB15b]. This observation led researchers to evaluate existing design patterns with respect to energy efficiency. Recent studies [BS13, LZCS05, SCG+12] found that some, not all, design patterns negatively impact energy consumption (up to 712% on embedded hardware).

This leads us to the question we explore in this paper, **How can we improve the energy efficiency of design patterns while retaining the essential benefits of improved code readability, maintenance and reuse?**

The answer, we believe, lies through compiler optimizations that have the potential for energy savings with *no changes* to existing software or hardware. Existing studies to improve energy efficiency using compilers rely on optimized use of hardware features such as dynamic frequency and voltage scaling.

Optimization targeting software design patterns for energy efficiency is entirely novel. We believe this approach has the potential to change the current state of practice since 1) Developer coding practices remain unaffected, 2) Benefits of using design patterns are retained, and 3) Energy consumption of software is reduced.

We study, in depth, the Observer and Decorator patterns that consistently performed poorly across all empirical studies, including ours. We propose energy efficient transformations for programs with these design patterns. The program transformations are to be carried out during the compilation stage. As an initial evaluation, we manually transformed 11 programs with the Decorator and Observer patterns and tried to optimize object creations and function calls specific to these patterns. We found our transformations achieved average energy gains of around 10%. Our vision for the future is that given a program with design patterns, we will automatically detect and transform patterns at the compilation stage for improved energy consumption. Our approach addresses the need for energy-aware software while retaining the benefits of design patterns in software engineering.

Empirical Study on Energy Usage

The design pattern examples in our study are written in C++ (14 patterns: Mediator, Observer, Strategy, Template, Visitor, Abstract, Builder, Factory, Prototype, Singleton, Bridge, Decorator, Flyweight and Proxy) and in Java (7 patterns: Chain, Command, Interpreter, Iterator, State, Adapter and Composite). We used OpenJDK 1.7.0_65 to compile and run the Java patterns, and Clang 3.5 to compile the C++ patterns. We used Jolinar (see Section 2.3.3) to estimate the energy consumption of our code at runtime. We ran our experiments on a Lenovo Thinkpad X220, with an Intel Core i5-2540M CPU topping at 2.60GHz. We measured the CPU energy overhead (positive or negative) of a program using a design pattern against a program version without the pattern[23]. We ran each version of each example 100,000 times in a loop in order to get enough execution data for a valid energy estimation, and we repeated each experiment 10 times. The results from our experiments are shown in Figure 3.1. As can be seen in Figure 3.1, the patterns with high energy overhead (>10%) are Observer (30.63%), Decorator (12.24%) and Mediator (26.61%). Our results follow a similar trend to

[23]Memory energy consumption, which according to our model is linear with memory usage was measured but was found to be negligible compared to the CPU energy. Execution time overhead was, in most examples, below 1%, with the exception of Mediator (7.9%) and Decorator (16%) patterns.

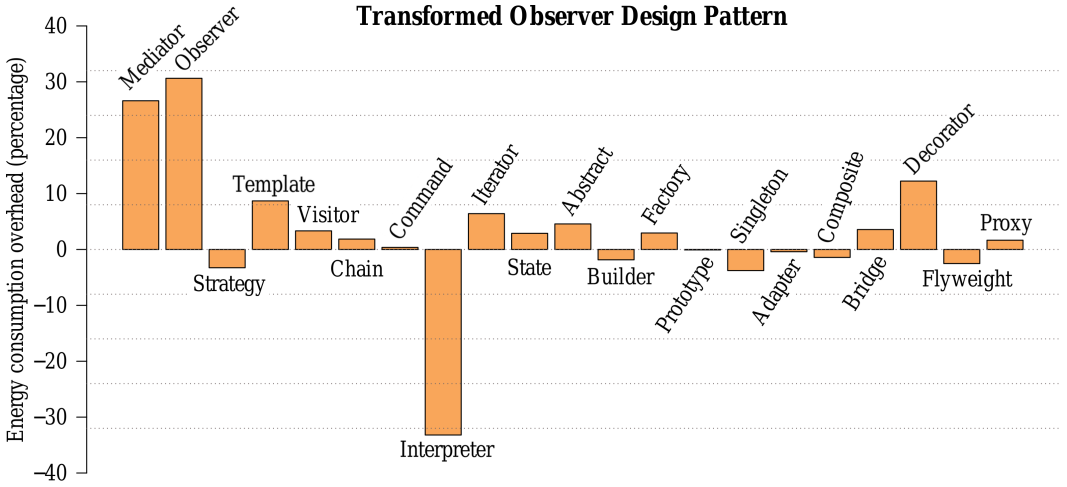


Figure 3.1: The energy overhead of design patterns.

the one presented in [SCG+12]. The exact percentages for energy overhead vary slightly since our experiments were run on an actual end-user computer rather than a specific FPGA board.

Having ascertained the energy usage of design patterns, the next step was to select and optimize design patterns that consistently consumed high energy. We picked the *Decorator* and *Observer* patterns since they have a high overhead (greater than 10% across all studies), and are widely used. We did not pick the *Mediator* pattern because it is less popular, and shares behavior with the *Observer* pattern (*e.g.*, both aim to reduce coupling). In the next section, we further study and optimize the energy footprint of the *Decorator* and *Observer* patterns.

Reducing the Energy Footprint of Design Patterns

In this section, we propose to further analyze two patterns, *Observer* and *Decorator*, and outline our approach to limit their energy overhead. Our ultimate goal is to have energy-aware transformations at the compiler level, therefore keeping the design patterns intact at the source code level while limiting their energy overhead in the compiled code.

We analyze and optimize the Observer and Decorator patterns using the following steps: 1) we ran experiments to check if existing compiler optimizations reduce the energy overhead of the patterns. We do this because in an earlier study we found varying optimization flags of the GCC compiler allowed a reduction in energy consumption [NBRS12b], 2) existing optimizations were found to be insufficient in step 1, so we propose a set of transformation rules targeted at improving energy overhead, and 3) we evaluate our approach over several small examples (discussed in further detail in the following sections).

For each of the examples in our study, we have 3 versions: **without**: a version that does not use the design pattern, **with**: a version that uses the design pattern, and **optimized**: a version where we optimize the *with* design pattern version for energy efficiency.

We iterate the examples 1 million times for each experiment run and repeat the experiment 10 times to mitigate the effect of any hidden bias.

Decorator Pattern: The Decorator design pattern is a structural pattern. The goal in this pattern is to *attach additional responsibilities to an object dynamically* [GHJV95].

Compiler Optimizations: We varied the optimization flags (*e.g.*, O1, O2 and O3) on the Clang compiler for both versions, *with* and *without* the Decorator pattern. Energy consumption while enabling compiler optimizations drops from 9.82 Joules (using O0 flag in Clang) to 9.61 Joules (with O3 flag) on average for the *without* version, and from 11.38 (O0) to 11.11 (O3) Joules for the version *with* the Decorator pattern.
The reduction observed when using compiler optimizations for each version is explained with the number of CPU guest instructions (collected through Valgrind's Lackey tool [lac]). These instructions drop from 3 billion to 2.6 billion for the *without* version, and from 5.5 billion to 4.6 billion for the *with* version when compiler optimizations are enabled (with an average CPU instructions overhead of 77.86%). Clang's optimization *-O* flags activate a number of LLVM's analysis and transformation passes on the LLVM intermediate representation (IR) code. These passes perform low-level transformations such as deleting dead loops or dead code, merging basic blocks or combining redundant instructions. Although Clang's optimizations allow energy reductions within each version, we still observe the same overhead of 15.6% (on average) between the *with* and *without* versions. In other words, existing compiler optimizations are not sufficient to improve the energy consumption resulting from the design pattern.

Transformation Rules: Our transformations for energy efficiency optimize the number of object creations and function calls in the pattern. These instructions cannot be improved by existing compiler optimizations as such transformation require specific knowledge of the pattern.
The transformation rules for the Decorator pattern are presented below and illustrated in Figure 3.1.
The proposed transformation replaces multiple object constructions (an object and its decorators) with a single object creation (an object from the added sub-class).

Listing 3.1: An example of Decorator pattern transformation.

```
// 1. Identify object instantiations using the pattern
Window *decoratedWindow = new HorizontalScrollBarDecorator(
    new VerticalScrollBarDecorator(new SimpleWindow()))

// 2. Create sub-class for the decorated type
class SimpleWindowHorizontalVertical : public SimpleWindow {...};

// 3. Replace with the sub-class constructor
Window * decoratedWindow = new SimpleWindowHorizontalVertical();
```

Empirical Evaluation: We used 6 programs with the Decorator design pattern from GitHub with sizes ranging from 77 to 198 SLOC. We manually transformed each of these programs using our rules. The results, in Table 3.7a, show a marked improvement in the energy overhead after applying our transformation. On average, we achieve a 12.23% reduction in the energy overhead with our transformations, with improvements ranging from

4.71% up to 25.47%. The difference in percentage improvements across our examples is due to the extent to which the pattern is used in them. For example, the *Pizza* program decorates objects with one decorator while the *Sandwich* program has multiple decorators associated with an object. As a result, our transformation removes more object creations in the *Sandwich* program, and thus a larger percentage energy gain, than in the *Pizza* program.

(a) Decorator design pattern

Program	Energy overhead
AXYZ	-9.89%
Window	-9.09%
Sandwich	-25.47%
Pizza	-4.71%
Coffee	-11.69%
Beverage	-12.55%

(b) Observer design pattern

Program	Energy overhead
Content	- 13.02%
Blog	-5.5%
Calculator	-6.22%
Topic	-5.71%
Mod	-4.32%

Table 3.7: Energy consumption overhead of the transformation of the Decotrator and Observer design patterns

In the next section, we study the Observer pattern, and propose a set of transformation rules in order to reduce its energy footprint.

Observer Pattern The Observer design pattern is a behavioral pattern. The goal in this pattern is to maintain a *one-to-many relation between subject and objects* [GHJV95], allowing *observer* objects to be notified and updated automatically whenever the subject changes its state.

Compiler Optimizations We apply Clang's optimizations to both the *with* Observer pattern and *without* versions. Energy consumption drops from 9.85 to 9.36 Joules on average for the *without* version, and from 13.25 to 10.17 Joules for the *with* version due to the reduction in the number of CPU guest instructions. In contrast to the Decorator pattern, Clang's optimizations were effective in reducing the energy overhead between the *with* and *without* versions, from 34.5% down to 8.64%, on average. Nevertheless, a considerable energy overhead remains and we attempt to reduce this in our transformation of the Observer pattern.

Transformation Rules In this pattern, every change in the subject's state is notified to all observers who then execute an update function. It is usually the case that each of those update functions will constitute a *get* of the updated subject's state. This *get* operation is typically repeated across updates in all the observers. We also inline subscribe/unsubscribe function calls.
Our transformation of the Observer pattern, in Figure 3.2, optimizes the repeated *get* operation by querying the subject state once and reusing it in all the observer updates. We hypothesize that transforming multiple function calls and memory accesses into a single call will allow a reduction in the energy consumption. Note that, unlike the Decorator pattern which is structural and creates multiple objects, there is no opportunity to optimize object instantiations in the Observer pattern since it focuses on the communication between objects.

Listing 3.2: An example of observer pattern transformation.

```
// 1. Identify subject using the pattern
MySubject subject;

// 2. Create global collection
std::vector<ObserverInterface *> observersVector;

// 3. Replace subscription with addition to the collection
subject.subscribe(&observerA);
-->
observersVector.push_bash(&observerA);

// 4. Replace calls to notify with direct call to update
long count = observersVector.size();
for (int i = 0; i < count; i++)
    (observersVector[i])->update(msg);
```

Empirical Evaluation We transform 5 different programs using the Observer design pattern and report the energy consumption overhead when applying our transformation rules. The programs are taken from GitHub repositories and their sizes range from 80 to 123 SLOC. The results, in Table 3.7b, show varying improvements in the energy overhead for the different examples. On average, we achieve a 6.95% reduction in the energy overhead with our transformations, with improvements ranging from 4.32% up to 13.02%. As with the Decorator pattern, the percentage improvements vary based on the extent to which the pattern is used in the different examples (*e.g.*, number of observers and number of update notifications).

Our approach aims to improve the energy efficiency of software by optimizing design patterns automatically at compile time. We explored transformations for the *Observer* and *Decorator* patterns and found energy reductions in the range of 4.32% to 25.47%. The proposed approach not only addresses the need for energy efficient software, but also ensures that current coding practices remain unaltered, a feature desirable for industry adoption.

Our vision for achieving energy efficient software is to have compiler optimizations that detect and transform design patterns for improved energy consumption. We aim to leverage existing research for automated detection of design patterns [TCSH06, SO06, SS03], and plan to build compiler transformations in compiler frameworks such as LLVM/Clang. We aim to establish functional correctness of the compiler transformations and perform empirical evaluations investigating energy gains on industry sized software.

3.2.3 Features and Slices

The content of this section is adapted from the following publications:
Measuring Energy Footprint of Software Features. Syed Islam, Adel Noureddine, and Rabih Bashroush. In International Conference on Program Comprehension (ICPC'16). Austin, USA, May 2016.

With the proliferation of software systems and the rise of paradigms such as the Internet of Things, Cyber-Physical Systems and Smart Cities to name a few, the energy consumed by software applications is emerging as a major concern. Hence, it has become vital that software engineers have a better understanding of the energy consumed by the code they write. At software level, work so far has focused on measuring the energy consumption at function and application level.
Current research focuses on measuring the energy consumption of applications at two main granularities. The first consider the energy consumption of a particular function or a set of functions, while the second level of granularity looks at the energy consumed by the entire software stack or process. Work in this area entails creating accurate models to estimate the energy consumption of software, which sometimes includes the use of complex formulas or additional hardware. Such measurements are *vertical* in nature and oriented towards measuring the consumption of software constructs. Although such approaches can aid software engineers identify bottlenecks, hotspots or energy smells in the code, they do not lend themselves to better understand consumption at feature-level, which is ultimately what is being consumed by end users.
We propose a novel approach for measuring energy consumption of end-user features, taking a more *horizontal* cross-cutting view. Our approach will enable newer levels of program comprehension and understanding. On the one hand, it will empower software engineers analyze the consumption of a given feature, which can cross-cut multiple classes, or even programming languages. On the other hand, it will help cloud service providers isolate and better understand energy consumption of their system features, and where applicable, introduce better charging models for end users based on real energy consumption. We show, using an experiment, how the measurement can be done with a combination of tools, namely our program slicing tool (PORBS) and energy measurement tool (Jolinar).

Feature-Level Monitoring Approach

Our approach to measuring software energy consumption at the feature-level has two steps. The first step involves the identification and extraction of an executable subset of the original program that implements the feature of interest (feature slicing). The second involves the measurement of the energy consumed by the feature slice (energy measurement). The latter step is done by using our energy measurement tool, Jolinar (see Section2.3.3).
The first step in our approach is to identify what parts of the system constitutes a feature of the software. Unlike formal functional components of a software system, what constitutes a feature is often down to the granularity at which the software system is modeled. At higher architectural levels, a feature constitutes work done by several modules to achieve some purpose. However, even a small utility function within that same system can be regarded as a feature when taking a more fine-grained view.
Program Slicing is a technique for identifying the influence or dependencies for a specific point of interest within the code, referred to as the slicing criteria. Program Slicing is well researched and has many applications [De 01], including comprehension, testing, debugging, maintenance, re-engineering, re-use, and refactoring.
Although useful for debugging, the original notion of program slicing now known as static slicing is not suitable for feature extraction. The static form is a conservative approach where any possible influences on the slice criterion for all possible inputs are identified. This conservatism over approximates and includes much more dependencies than what a typical

execution of a feature would require. For example, if a static slice was taken with respect to variable s on line 38 of program P (Figure 3.2), the static slice would include the entire program. Several extensions to the original notion of slicing have been proposed over the years that attempt to tackle feature extraction [DFM96, WGHT99, EKS03]. However, most of these techniques do not yield a slice that is an executable program in its own right. Furthermore, none of these techniques can handle multi-language systems.
Recently proposed Observation-based Slicing [BGH+14] is a technique that finds its origin in Weiser's original motivation for slicing [Wei81]: uninteresting statements can be deleted (sliced out) of a program to help focus attention on relevant statements (i.e., the slice). Operationally, PORBS (our observation-based slicer) achieves this by tentatively deleting one or more statements and then observing the behavior of the resulting program. It attempts to find a subset of the program (slice) that preserves the behavior of the original program with respect to a certain set of inputs and the slice criterion.
In greater detail, first a candidate slice is produced by deleting one or more lines of text from a program. Then this candidate is compiled and executed. If the compilation fails or the execution fails to terminate normally, the candidate is rejected. Otherwise, its output, restricted to the slicing criterion, is compared to that of the original program. If the two differ the candidate is rejected. Only if the candidate compiles, executes, and produces the correct output is it accepted and becomes the current slice from which further line deletions are attempted. This process is repeated until no further deletions are possible.
Figure 3.2 shows a program P, which is a utility program with two features, `sort` and unique. The first, reads a file and outputs contents of the file in sorted order, while the second, reads a file and outputs non-repeating contents. The features are triggered by providing the name of the input file and options -s (`sort`) or -u (`uniq`), respectively. We present this simplistic version of the utility tool to explain our approach ignoring details such as error checking. The code has been structured in such a way that the `main`, `readFile` and `writeList` methods will be required for both utilities. Each specific feature is then provided by one of the two methods, namely `sort` by `sort` method and `uniq` by `unique` method.
Running PORBS slicer on the code with an appropriate input file and a feature options causes PORBS to produce an executable slice implementing the certain feature. For example, Column 2 of Figure 3.2 marks lines included in the slice for the feature option -s and Column 3 marks the lines included in the slice for feature option -u. Both slices are executable programs on their own right and will produce the same output as the original program (P) for respective features. PORBS is therefore able to extract executable slices that contains all the code pertaining to the implementation of a particular feature within a program. We are therefore using PORBS in our approach to extract a minimal set of code that implements a certain feature.

Approach Validation

We validate our approach using the use case shown in Figure 3.2 where we identify two executable program slices, the first one is for the `sort` feature and the second one for the `uniq` feature.
Following the extraction of these slices using PORBS, we apply our energy model using Jolinar to estimate the amount of energy consumed by each of the slices when processing a file with 100,000, 1,000,000 and 10,000,000 integer values using a machine running an Intel

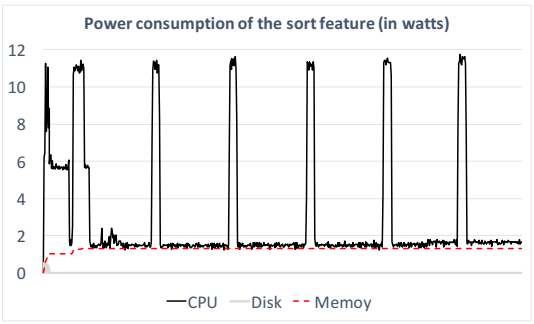
-s -u	Program P

```
1  public class SortUniqeUtility {
2
3    static ArrayList list = new ArrayList();
4
5    public static void main (String[] args)...{
6      readFile(args[0]);
7      if (args[1].equals("-s"))
8        sort();
9      if (args[1].equals("-u"))
10       unique();
11     writeList();
12   }
13
14   static void readFile(String s) ... {
15     FileReader fr = new FileReader(s);
16     BufferedReader br = new BufferedReader(fr);
17     String line = "";
18     while ((line = br.readLine()) != null) {
19       list.add(line);
20     }
21     br.close();
22     fr.close();
23   }
24
25   static void sort(){
26     Collections.sort(list);
27   }
28
29   static void unique(){
30     Set<String> hs = new HashSet<>();
31     hs.addAll(list);
32     list.clear();
33     list.addAll(hs);
34   }
35
36   static void writeList(){
37     for (String s : list)
38       System.out.println(s); // slice on s
39   }
40 }
```

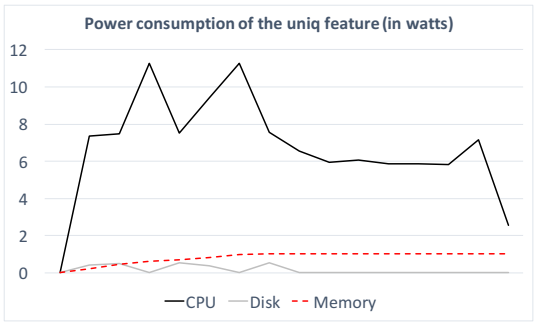
Figure 3.2: Observation-based Slice for Line 38 with option -s and -u.

		sort				uniq			
File Size	Input Size	CPU	Disk	Memory	Total	CPU	Disk	Memory	Total
556K	100000	9.51	0.01	0.23	9.75	5.49	0.01	0.07	5.57
5.4M	1000000	55.6	0.12	7.46	63.18	8.68	0.12	0.22	9.02
54M	10000000	972.3	1.18	441.44	1414.92	54.34	1.18	6.53	62.05

Table 3.8: Energy consumption by the features `sort` and Uniq for various input sizes



(a) Power consumption of the `sort` slice with 10,000,000 inputs



(b) Power consumption of the uniq slice with 10,000,000 inputs

Figure 3.3: Power consumption of sort and uniq slices

Pentium T3400 processor running at 2.16 GHz and a Seagate Momentus 5400.5 SATA hard disk.

Table 3.8 outlines the energy results for the different inputs, detailing energy consumed by the CPU, disk and memory, for both the `sort` and uniq slices. The results show an exponential growth in the energy consumption for the `sort` feature, going from 10 joules to more than 1415 joules when integer inputs are multiplied by 100. The majority of this energy is due to sorting calculations (therefore the CPU), with an increase of memory energy for larger data sets. Nearly a third of the energy (31.2%) consumed by the 10 million integer `sort` feature is due to memory access (read/write) for sorting. In particular, `Collections.sort()`, which is used in the `sort` feature (see line 26 in Figure 3.2), requires a temporary storage of up to n/2 object references. On the other hand, the `uniq` feature exhibits a much lower memory energy consumption as the `unique` method (see line 29, Figure 3.2) only adds input to a hashset and a list.

Figure 3.3a and 3.3b show the power consumption of the `sort` and `uniq` features with 10,000,000 inputs, respectively. Initially, there is an increase in disk power consumption due to disk activities reading the input file (*e.g.*, method `readFile(String s)` in lines 14–23 in Figure 3.2). Memory consumption for `sort` is constant across the experiment as Java allocates a large memory buffer for sorting and did not require additional memory space. More interestingly, the graph outlines spikes in the CPU power consumption which are explained by the sorting cycles in the algorithm and the copying of data in between. With a shorter experiment duration, `uniq` seeks processing power for copying and clearing Java collections objects (a hashset and a list) as seen in Figure 3.3b.

Comprehension of energy consumption by feature can help developers identify the features that are consuming more energy and use additional information (such as importance and

likely frequency of use) to decide on where to focus optimization efforts (*e.g.* optimize code or dedicate energy efficient resources to frequently used features).
Our novel approach aims for better understanding the amount of energy consumed by software at a feature level as compared to traditional methods which analyze energy consumption based on software artifacts (*e.g.*, function or process).
Our validation shows that we are able to identify and measure the energy consumption of features (slices), which could help software engineers conduct a new level of analysis and optimization (*e.g.*, focus on what matters most).

3.2.4 Unit Tests and Code Coverage

The content of this section is adapted from the following publications:
Is well-tested code more energy efficient? Adel Noureddine, Matias Martinez, Houssam Kanso and Noelle Bru. In the 11th Workshop on the Reliability of Intelligent Environments (WoRIE'22)/(IE 2022). Biarritz, France, June 2022.
A Preliminary Study of the Impact of Code Coverage on Software Energy Consumption. Adel Noureddine, Matias Martinez, and Houssam Kanso. In the Second International Workshop on Sustainable Software Engineering (SUSTAINSE)/(ASE'21), Melbourne, Australia, November 2021.

Software energy consumption is a major concern for software developers, practitioners and architects, where they lack tools to monitor software energy, and knowledge in understanding the factors impacting the energy consumption of software [PC17b, MBZ+16b, BWN16]. In particular, the authors of [PC17b] note the *lack of knowledge on how to write, maintain, and evolve energy-efficient software*. The authors also discussed the state of the art of energy-aware software testing, and found that few studies propose energy-aware software testing techniques. These techniques offer new approaches to reduce the energy consumption of test suites [LJS+14], including in Android [JSBM16], or detecting energy bugs through software tests [BCCR14].
Current approaches allow software developers to monitor the energy consumption of their devices' architectures, their applications, and within the source code, thus allowing to detect energy hotspots. With such tools and in-depth software energy knowledge, developers can detect and improve their software. However, the technical and psychological scalability of these approaches (such as resistance from developers to adopt new energy-aware coding behaviors, and the pressure of project deadlines and release) limits their effectiveness, as developers report a lack of proper tools and knowledge as shown in [PC17b].
We argue that leveraging existing, more accepted, and adopted approaches in software development, to guide developers in writing energy-aware software is needed. In particular, we argue for leveraging software testing for energy efficiency. Therefore, guiding software developers in writing better quality code, through software testing, might help in achieving energy optimizations and gains in software with little additional investment to practitioners and current development practices.
The impact of tests quality on the energy efficiency of software is not well understood or studied. In this study, we analyze if software written with unit tests and having good code coverage (along with other test metrics) is more energy-efficient than software with no unit tests or low code coverage.

Research Questions and Methodology

The research question that guides our research is: *Is there a correlation between energy consumption of an application and the quality of its test suite?*
To answer the research question, we set the following hypothesis:

- Null hypothesis $H0$: there is no correlation between a metric that represents test quality and energy consumption of the application of the test.
- Alternative hypothesis $H1$: there is a correlation between a metric that represents test quality and energy consumption.

To address our research question, we first measure the energy consumption of executing an application, and then collect the metrics related to the test cases. Finally, we correlate energy consumption and those metrics.
We focus on the energy consumption of applications from the same domain, all capable of doing a same functionality F: parsing JSON files. This allows us to fairly compare the energy across different implementations of F, and to remove the potential threats of comparing two different functionalities and/or applications with different energy requirements. Note that we measure energy consumption of each implementation of F using the same input values I.
We measure the energy consumption of the workloads using PowerJoular (see Section 2.3.1), which uses Linux kernel's Intel RAPL through the powercap interface. In order to minimize the impact of any noise during the measurement of energy of F, (caused by e.g., system states or background services), we run each test workload in a loop for 200 times and measure its energy consumption. We report the energy consumption per loop by dividing the total energy by the number of loops.
We use SonarQube[24] and JaCoCo[25] to collect test and coverage metrics for the applications under evaluation. We collected the following metrics: branch coverage, SonarQube coverage[26], line coverage, lines to cover, uncovered conditions, uncovered lines, and the number of tests.
Finally, we correlate the energy results with the test metrics using the Pearson correlation method. We also compute the p-value, which allows us to reject or accept our Null hypothesis.
We run our experiments on a Dell Precision 5520 laptop with an Intel Core i7-7820HQ processor, running Fedora 34 with Linux kernel 5.11. We compile and run the Java libraries using openJDK 11.
We focus on a single functionality F: parsing a JSON file for disk for creating a representation of it in RAM.
For our experiments, we use a set of 14 Java JSON libraries that implement this functionality. Those libraries, listed in Table 3.9, were previously studied by [HDBB21]. We focus on JSON libraries because JSON is one of the most used data representation format. For instance, GSON, one of the lib analyzed, is used by more than 331K projects hosted on Github

[24]SonarQube: `https://www.sonarqube.org/`
[25]JaCoCo: `https://www.eclemma.org/jacoco/`
[26]The SonarQube coverage is a mix of Line coverage and Condition coverage: `https://docs.sonarqube.org/latest/user-guide/metric-definitions/`.

As input data I, we use the publicly available dataset of JSON files provided by [HDBB21], conformed by 152 well-formed JSON files. Given this data, our experiment measures the energy that each JSON library requires to parse all those JSON files. This allows us to fairly compare the energy consumption of the 14 libraries by doing a single functionality (JSON file parsing) on the same input data (152 files).

Experimental Results and Analysis

In this section, we present the results of our experimental study and the correlation analysis between energy consumption and testing metrics.

Energy Consumption: Table 3.9 outlines the energy consumption of each JSON library, along with the execution time, and the average power consumption of the CPU.

Table 3.9: Energy consumption for each JSON library workload

Library	Avg Power (Watts)	Energy (Joules)	Time (sec)	Standard deviation for Power
json-lib	46.87	225.43	4.81	5.29
sojo	51.76	76.34	1.475	8.87
flex-json	55.58	37.24	0.67	5.01
corn	53.02	36.32	0.685	4.66
mjson	52.64	26.32	0.5	8.54
jsonij	54.05	25.4	0.47	5.66
jsonutils	38.79	23.27	0.6	4.02
genson	54.88	17.29	0.315	5.39
fastjson	52.11	16.94	0.325	8.48
json-simple	34.79	14.44	0.415	4.17
json	34.08	13.8	0.405	6.41
gson	30.54	10.38	0.34	3.78
jackson-databind	29.35	10.13	0.345	3.72
cookjson	25.77	7.73	0.3	1.71

Our first observation is that energy consumption does not follow execution time. For instance, genson library consumed 17.29 joules on average per workload execution and took 315 milliseconds. In comparison, JSON library took more time (405 ms) and consumed 13.8 joules for processing the same JSON files.
Observing the average power consumption during the experiment for each library sheds a light over their CPU usage and power consumption, as the average power can vary from 25.77 watts to 55.58 watts, a 30 watts difference on a laptop computer. Overall, the energy consumption of the measured 14 libraries shows a big difference with energy varying from just 7.72 joules for cookJSON to 225.43 joules for JSON-lib, more than 2820% increase.

Test Metrics: Table 3.10 shows the metrics extracted from the test suites of our JSON libraries. Branch and line coverage varies greatly from as low as 50.5% and 25,2%, respec-

Table 3.10: Metrics extracted from test cases and the energy consumption from each application.

Library	Branch coverage	Coverage	Line coverage	Uncovered lines
json-simple	50.5	55.7	58	336
mjson	61.9	67.2	71	314
corn	52.1	60.3	64.7	556
flex-json	69.7	72.6	74	445
sojo	95.5	96.7	97.2	82
jsonutils	61.5	67.6	71	879
json-lib	71.3	74.1	75.8	1181
genson	67.3	73.2	76.3	1234
jsonij	55.4	40.4	35.9	4090
cookjson	87.2	55.5	47.8	3406
json	84.4	34.7	25.2	6356
gson	79.1	34.9	27.3	10257
fastjson	78.4	83.9	87.6	3469
jackson-databind	70.3	75.4	78.2	7108

tively, to as much as 95,5% and 97.2% respectively. However, the relation with energy consumption is not straightforward as the highest 2 libraries in branch coverage have the lowest and the 2nd highest energy consumption. Lines covered and uncovered also range from a few hundreds to around 28 thousand lines, and the number of tests varies from 3 tests to just below 5000 tests.

Analysis: We first plot the distribution of values of each test metric in Figure 3.4. We observe that a few libraries have a higher variation of values and are beyond the average range of other libraries. Instead of removing those outliers (as our dataset is limited in this preliminary study), we decide to analyze our data using logarithmic values. This logarithmic transformation allows to decrease the difference between the different values and limiting the impact of outliers while still maintaining the order of values, and it often produces a normal distribution of the studied metrics.
Using the logarithmic approach, we calculate, in Table 3.11, the Pearson's correlation coefficient between different metrics extracted from the test execution (e.g., coverage) and the average power consumed by parsing functionality. We apply a logarithmic transformation to the metrics' value for two reasons: 1) it often produces a normal distribution, and 2) it allows us to decrease the difference between the values, limiting the impact of outliers, and, consequently, avoiding the need of removing them.
We observe that two test metrics exhibit a moderate correlation value with an acceptable P-value: line coverage with 0.46 correlation and a P-value of 0.095 (above the 0.05 range, but within the 0.1 range); and uncovered lines which has a negative correlation at -0.53 and a good P-value at 0.049 (below the 0.5 significance range).
Figure 3.5a shows the correlation with Line coverage where we note that, if we exclude the

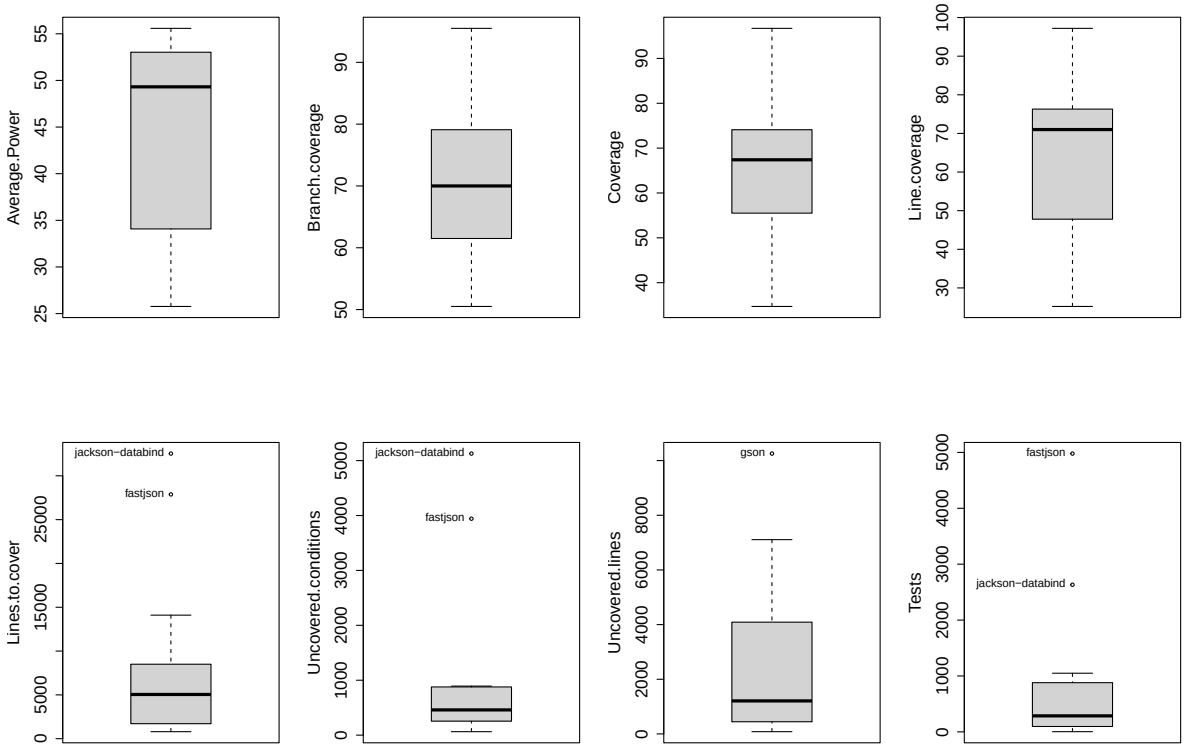


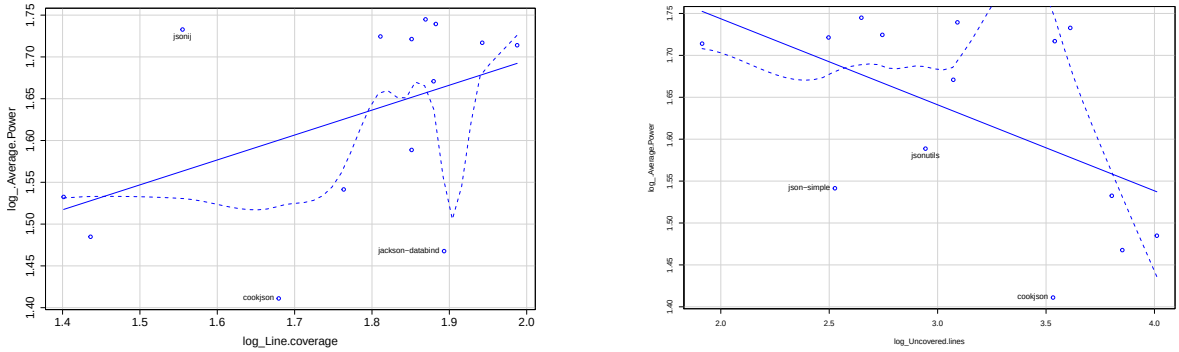
Figure 3.4: Distribution of values for each metric

Table 3.11: Pearson's correlation coefficient between test metrics and average power consumption, using Pearson algorithm on logarithmic values. Last column shows the p-values.

Metrics from tests	Correlation coefficient	P-value
Branch coverage	-0.27	0.342
SonaQube Coverage	0.42	0.133
Line coverage	0.46	0.095
Lines to cover	-0.36	0.211
Uncovered conditions	-0.018	0.95
Uncovered lines	-0.53	0.049
Number of tests	-0.044	0.881

3 out-of-range values, we might have better correlations. We also found the same trend for the correlation of Uncovered line, which has 2 out-of-range values.

We finally plot a principal component analysis (PCA) graph in Figure 3.6, which allows us to synthesize and understand the importance of each metric and to explain the variability of the libraries. Each arrow represents a metric variable: if the arrows are close to each other, then we conclude that a strong correlation exists, while opposite arrows means a negative correlation, and orthogonal means no correlations. We observe that the average power is drawn closer to line coverage, *i.e.*, a possible correlation exists, while the average power is in near-perfect opposite of uncovered lines, *i.e.*, a possible strong negative correlation exists.



(a) Line coverage correlation plot.

(b) Uncovered lines correlation plot.

Figure 3.5: Line coverage and Uncovered lines correlation plot with average power consumption. The straight line is the least squares fitting, while the dashed line is the smoothing model on our dataset.

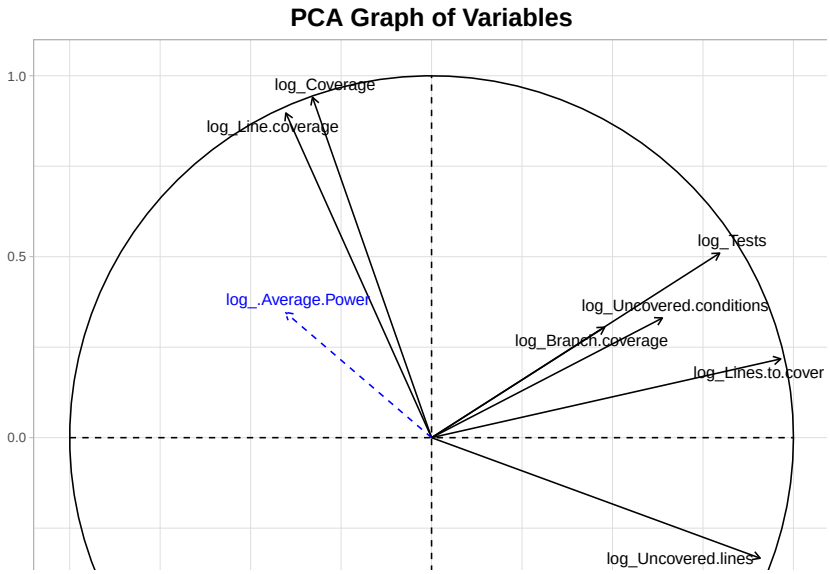


Figure 3.6: Principal component analysis (PCA) of our dataset, using logarithmic values

Discussions: Our statistical analysis, although on a small dataset, sheds the light on potential positive correlation between line coverage and power consumption, and potential negative correlation between uncovered lines and power consumption. That is, the higher lines we cover in our test, the higher the average power consumption of the program might be. In contrast, the higher the uncovered lines are, the lower the average power consumption.

Now, we proceed to accept or reject our hypothesis using the p-values from Table 3.11. Considering a significance level $\alpha = 0.05$, all the p-values related to test metrics are greater than α, which means that we are not able to reject the Null hypothesis. On the contrary, the p-values from the correlation between the metric Uncovered lines is smaller than $\alpha = 0.05$, which means we can reject the Null hypothesis for that metric.

Response to the RQ: Is there a correlation between energy consumption of an application and the quality of its test suite? Our experiment shows that there is a moderate positive correlation between Line coverage and power consumption of the parsing functionality of JSON libraries. However, we do not have enough evidence at the level $\alpha = 0.05$ to conclude that there is a linear relationship in the population of Java JSON libraries between coverage and power consumption.
Our initial results show that a positive correlation between line coverage and power consumption exists, and a negative one between uncovered lines and power consumption. However, the limitations of our study do not allow us to provide a definitive conclusion.
The main takeaway of this research is that applications having a good test suite, expressed in popular metrics such as coverage, do not necessarily exhibit optimal power consumption. Test metrics give users confidence about the software of the application.
However, energy consumption metrics are still hidden for most developers and consumers of open-source software, such as the JSON libraries evaluated in this experiment. This means that an application could be, for instance, well tested in terms of having high coverage but, at the same time, it could not be efficient in terms of energy consumption. The results of this experiment do call to expose non-functional factors such as energy consumption in software.

Epilogue: Studying and understanding the impact of design factors allows developers and practitioners to design and build greener software. However, software life continues in the usage phase where, in many cases, end users interact with software and have an impact on its energy consumption. In the next section, I study the role that users have in the software energy efficiency equation, and whether a green feedback can trigger behavioral change towards reducing energy consumption.

3.2.5 Impact of Green Feedback on Users' Software Usage

The content of this section is adapted from the following publication:
The Impact of Green Feedback on Users' Software Usage. Adel Noureddine, Martín Diéguez Lodeiro, Noelle Bru, and Richard Chbeir. In IEEE Transactions on Sustainable Computing journal (T-SUSC). Volume 8, number 2, pages 280-292. April-June 2023. [NLBC23]

The rise of the energy impact of software systems requires the need to optimize and reduce their energy consumption. One area often neglected is the important role played by users to drive energy reductions. We aim to reduce the energy impact of software by pushing end users to change their software usage behavior, through raising awareness and providing software green feedback. We present a comprehensive and detailed field study of the impact of green feedback on software usage by end users, and the efficiency of green feedback on software behavioral change, using a distributed architecture aimed at providing accurate green feedback in real time. We find that green feedback helps in raising awareness about software energy, and on the willingness of users to apply energy-efficient changes. However, we also find that users lack the knowledge and tools to properly adopt lasting and energy-effective behavioral changes.
Today, most energy management approaches in computing, whether it is software systems, data centers, or cyber-physical environments, focus on technological advancements or opti-

mizations. However, very few approaches integrate end users into the energy optimization equation, and most of these approaches target energy consumption in buildings or smart homes, where energy reductions are achieved in non-computing equipment (such as refrigerators, lightning, HVAC systems, etc.).
We argue that the next major shift in energy reductions needs to involve end users, for instance in relation with reducing the usage of ICT or changing users' behavior regarding ICT devices. In particular, changing the behavior of end users in using smart devices, and in particular software running on these devices, might lead to important energy savings. We decide to focus on software, such as applications or operating systems, which are easier to update than hardware devices. Also, any improvement to the source code of software can scale well and be deployed to billions of devices. In addition, end users essentially interact with software, *e.g.*, the *smart* part of smart devices.
We aim to understand the impact of green feedback on users' software behavior. In particular, we want to study and understand, through a field experiment: 1) *What do users know about energy consumption of software?*, 2) *What perceptions do users have on software and energy?*, and 3) *Will green feedback of software usage lead to behavioral change and energy reductions?*
Many other factors impact energy consumption and/or users behavior. For instance, green feedback might raise awareness for users and progressively change their behavior over time. Users also may lack sufficient knowledge on what to do when energy consumption is high, besides turning off devices or stop using them. In our study, we decide to limit our investigation on two areas: 1) study the short-term behavioral change of end users after providing them with green feedback, and 2) study the perception and user awareness in regard to energy consumption of software.
Our research question is, therefore: **can raising user energy awareness, through live green feedback, drive behavioral change in software energy consumption?** In particular, we want to understand if live green feedback can drive short-term behavioral change, and what type of change users think are appropriate.
Feedback is any information send by software systems to end users, related to the state of software. Energy feedback is specific feedback related to the energy or power consumption of software. An existing definition in [KFS14] explains the difficulties of proposing a clear operational definition, and defines energy feedback as *information about actual energy use that is collected in some way and provided back to the energy consumer*. We extend this definition to cover other metrics and information than those of energy and power, specifically ecological and sustainability metrics (such as CO2 emissions, renewable energy usage, etc.).
In this study, we conduct a field experiment on the impact of green feedback on software usage. We build a distributed architecture to provide live green feedback for end users, using web services to connect to electricity providers to calculate and provide up-to-date and accurate feedback on the power consumption of software and devices, and the price of this consumption and its CO2 emissions.
The field study consists of a power-monitored workload, along with a survey for end users. Our field study provides promising results on the role of green feedback, but sheds light on the challenges related to software behavioral change and the limitations of current tools and feedback visuals. To our knowledge, we believe our field experiment, conducted with a control group, is novel in the green computing and software engineering communities.
It is important to stress that our goal in this study is not to provide guidance or alternative software for users, nor do we aim to tell users what is the right course of actions to reduce energy, nor the correct behavior to adopt. Our study is an observation field experiment

where we aim to observe, analyze and understand what are the factors impacting behavioral changes around software energy, and if green feedback can trigger a behavioral change in users.
Although we have ideas of what can impact software energy and user behaviors, and what users might do to reduce their energy footprint, we have no existing scientific data to back any assumption about end users' behaviors or to provide suggested recommendations for users in this study. We aim to collect and build solid scientific data to validate our research question and hypothesis, and that can be used by the scientific community for future studies.

3.2.5.1 Field Experiment and Methodology

In this section, we describe the methodology we applied in the experiments and user surveys. Our field study consists of two steps: first, users participate in a common experiment where their energy consumption is monitored, then, they fill up a survey to collect their feedback and knowledge on the experiment and green computing.

Participants: In June and July 2020, around a hundred students (95) from computer science and engineering degrees in Lebanon, participated in the field study and survey. The field study was conducted for each student group during one class session (2 hours), with similar protocols for both control group and treatment group.In total, 5 class sessions were needed for the experiment, with 2 classes being the control group (A1), and 3 classes for studying the impact of green visual feedback (A2), with overall a similar number of students in groups A1 and A2. Each class was either part of the control or treatment group (we did not divide a class) in order to avoid students influencing each other during the experiment and voiding the validity of our control group.
Participants were split in two for experiment A1 and A2, with 47 participants in group A1 answering the survey, and 48 for group A2. The majority of participants were males (74.47% for the control group A1, and 83.33% for A2). The average age is 22 years old (youngest at 21 and oldest at 27 years). Participants were students from computer science and engineering degree, with a minority having an additional part- or full-time job (18.9%). The majority of participants, although studying in computing fields, have one or two mobile devices: 33.70% have 1 device, 36.96% have 2 devices, 15.22% have 3 devices, while only 14.13% had four or more devices. Overall, participants are interested in ecology, with a majority of 71.58% choosing the answer that "ecology is a major concern for our planet (pollution and global warming)", with 6.32% saying they are only interested in pollution but not global warming, and 22.11% saying they are not interested in ecology.
Finally, a majority of participants (75.53%) think that radical (43.62%) or gradual (31.91%) actions need to be implemented to reduce energy consumption. 15.96% answer that some actions are needed to maintain energy consumption at current levels, while 8.51% answer that no change is needed as the status quo is fine. Overall, there are no significant differences between participants in the control group A1 and the experiment group A2.

Workload: Every participant had to perform a similar task in order to have a similar workload for comparison. Participants were asked to write a research document of 3 to 4 pages with heavy usage of online resources.

Participants used two sets of software: cloud-based tools, such as the online latex editor OverLeaf, and web browsers to search for research papers, documents, multimedia content, etc., and on-device tools, such as word processing software, PDF or image viewers, etc. Participants had the liberty to choose the software they wish, such as using Google Chrome or Mozilla Firefox for the web browser.

Experimental Setup for power monitoring and feedback: All participants used a laptop running Microsoft Windows, with a few exceptions using a laptop with macOS. We built a power monitoring software, along with a distributed architecture, that collects, every second, a number of useful metrics: CPU power consumption of the laptop, the name and path of software run by participants and the title of the active window (aimed to understand what software or website actively used, and mapped to software usage and energy consumption). With the CPU power consumption, we calculate the equivalent CO_2 emissions based on the country's electricity mix, and calculate the electricity cost.
Our approach first identifies the geographic location of end users, and according to its region, obtains the relevant metrics. Therefore, two users in different countries consuming the same amount of power, would be provided different green feedback for the price and CO2 categories. For the purpose of the experiment, CO2 and electricity data are collected once from the electricity providers, and used throughout the experiment in order to minimize the impact of the network on energy consumption and mitigate slow network connections for certain participants.
Current CO2 emissions and electricity price for the user workload is calculated by multiplying the current real-time power consumption (in watts) by the user location's CO2 emissions (in grams per kWh), or by the price in cents per kWh, and we then adjust the unit metrics.
In addition, we developed a visual feedback tool which showed the power consumption, CO2 emissions and the cost of electricity in real time in a small window. This window, shown in Figure 3.7, cannot be minimized or resized by participants, and stays on top of all other programs at the bottom right of the screen. In addition to textual values of the metrics, a graph is also shown which monitors in real time the power consumption of the device.

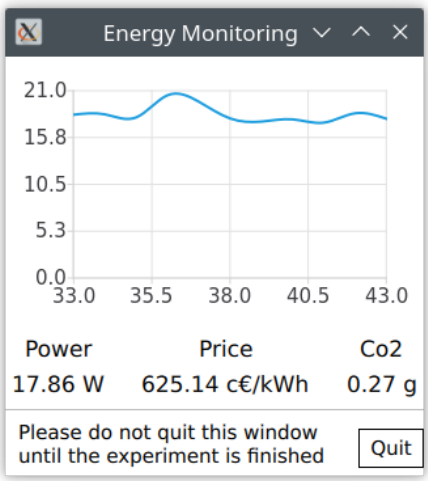


Figure 3.7: The green visual feedback tool used in our experiments

Participants in the control group (group A1) were asked to install the power monitoring software and its dependencies, and run it in the background. This software will silently monitor power consumption of the device and collect software usage. Participants in the experiment group (group A2), were asked to install both software: the power monitoring one (the same as the control group A1), and additionally our green visual feedback tool which will display green feedback in a persistent window on the desktop (cf. Figure 3.7).
As lockdown for universities was still in place during the experiments, participants were asked to send the generated metric data (one CSV file and one TXT file) by email. However, due to technical problems installing the tools, or running and generating the data, and with issues with mapping the collected data to the anonymous survey, we ended up with 52 valid distinct power data files divided into 26 for group A1 and 26 for group A2.

Participants' Hardware and Software Profiles: All participants who send valid monitoring data used a Windows 10 laptop with 4 exceptions: three participants with Windows 8.1, and one with Windows 7, all of which were 64 bit devices. A few users had macOS laptops but none provided valid power data for the experiment.
All laptops were using Intel Core processors: two i3 models, nine i5 models, and the remaining were i7 models. The thermal design power (TDP) of these processors varied from the lowest at 15 watts (for 42 laptops), to up to 95 watts for one laptop using an i7-6700K CPU at 4 GHz, with 18 laptops with a TDP at 45 watts.
In terms of applications, and as the workload is document-oriented, participants mostly used a internet browser, and office software. In particular, the majority of participants used Google Chrome with a few participants using Mozilla Firefox, Microsoft Edge and Opera Browser. The usage of the browser was extensive as participants use it for online search, email and messaging, viewing PDF files, and using the online Latex editor Overleaf. The majority also used Microsoft Teams for the class.
Other applications were used occasionally for specific tasks, such as Microsoft Office software (in particular, Word, Excel and PowerPoint), Windows Notepad for note taking, Adobe Acrobat Reader to view PDFs, Windows Mail for emails, WhatsApp Desktop for messaging between participants, and the Windows Snipping Tool to take screenshots for the assignment.

Survey to collect user feedback: At the end of each experiment, participants were asked to fill in an anonymous online survey. In addition to the power data filename (which we collected to map the survey results to the power consumption data), participants had to answer a set of question regarding sustainability, green computing, and the experiment they are participating in.
We built our survey questionnaire with 4 main blocs around the following themes: behavior (*what we do*), opinion (*what we think*), motives (*why we do*), and identity (gender, etc.). We privileged closed questions as they are easier to answer by participants and to process by us, and to avoid ambiguities. We also avoided tendentious and emotionally charged questions in order to keep the questionnaire neutral. We built our multiple-choice questions to be as precise as possible by asking participants to classify their answers. In order to appreciate an impact of an answer and force a positioning, we built our questionnaire using a Likert-type scale without a midpoint (*i.e.*, without the *Neither agree nor disagree* answer).
We first asked participants questions about their identity: gender, age, education, and work

status. These questions aim to understand whether age, gender or education levels have an impact on their opinion or behavior regarding energy consumption and sustainability in general. We then asked participants on how many mobile devices they own, and their interests in ecology and their view on global warming and if any action needs to be done for energy reductions.
Participants were then asked a few questions on the experiment they participated in. Participants in groups A1 and A2, answered questions on their perception of the energy cost of their experiment. They also had to rate the energy costs (from very low to very high) on a set of software categories (office programs, internet browsers, scientific or programming software, and the operating system itself), and then rank the most consuming categories and individual software.
Participants in group A2, had additional questions about the green visual feedback. In particular, we want to know if the visual tool was helpful or distracting, which form of feedback was most helpful (visual graph or textual numbers), and which metric was most useful (power consumption, CO2 or price). We then asked participants if they changed their software behavior based on the visual tool feedback, and if so, how they changed behavior.

3.2.5.2 Results and Discussions

Summary of Results: The results of our study are extensively detailed in our journal paper in [NLBC23]. I outline here a brief summary of the results.
A majority of participants rated the energy consumption of their session as low (49.47%) or very low (11.58%), with numbers varying between the control group (A1) and the experiment group (A2). Participants who had our live green feedback window, rated their energy session higher with 50% answering *high*, in comparison to only 27.66% in group A1 answering high or very high. Statistically, the difference in answers between groups A1 and A2, is significant. This shows that participants have a different point of view of their power consumption when shown live green feedback.
Participants also rated the operating system as the one with the highest energy consumption (77.17% as high or very high, including 26.09% as very high). These numbers are similar for scientific or programming software (typically, IDEs, compilers, Latex, Matlab, etc.), with 75.79% rating it high or very high. Surprisingly, internet browsers were also rated high, with 66.32% rating it high or very high, albeit a lower number rated it very high (10.53%). In contrast, office programs were the ones least seen as high energy consumers, with 38.79% rating them as high or very high. Even better, office programs had the highest percentage of participants rating them very low (12.63%), while no one rated the operating system very low.
Participants were asked to classify the most energy-consuming software, and they rated three software in the first spot: IDE (Eclipse, Visual Studio, etc.) at 33.33% of participants, Communication (Skype, Teams, Discord, etc.) at 28.89%, and Browser (Chrome, Firefox, Edge, etc.) at 25.56%. The other software didn't pass 7% and no one rated email clients (Outlook, Thunderbird, etc.) at the first spot. The second spot was also taken by IDE (29.47%) and Communication (28.42%). The third place was taken by Browser (30.53%) and, surprisingly, file or PDF viewer at 20%, with presentation programs at 16.84%. The next two spots were shared between word processor and presentation program (21.62% then 29.73% for the former, and 21.05% then 22.11% for the latter). Finally, most people ranked file or PDF viewer as the least consuming with 35.79% ranking it last.

Most participants also consider cloud computing or data centers the most consuming part of the computing industry at 62.11%, with network equipment second at 12.84%, desktop computers third at 12.63%, and finally mobile computing last with 8.42%. This shows that computer-savvy participants are aware of the energy weight of cloud infrastructure (data centers and networking).
Participants in the experiment group A2, had to answer an additional set of questions in the survey, in particular about their perception of the green feedback tool and their usage behavior of software and their devices. A majority of participants agree (58.33%) or strongly agree (33.33%) that the visual feedback was helpful to know when energy consumption is spiking. A similar majority agrees (59.57%) or strongly agree (14.89%) that energy consumption was higher than they thought, while a quarter (25.53%) disagreed. When we asked the opposite question (if energy consumption was lower than what they thought), 39.58% of participants agreed and 15.58% strongly agreed, while 35.42% disagreed and 10.42% strongly disagreed. These numbers show that, overall, participants thought they consumed more or less than their initial perception, therefore showing the importance of feedback in raising awareness. In contrast, around half of participants (47.92%) did not understand what each metric relates to. Finally, a majority of participants saw the window of our green feedback tool as distracting (57.78%).
41.67% of participants answered that the power consumption metric in watts was the most helpful in understanding the green impact of their software and devices. Around a fifth (22.92%) answered that all 3 metrics were helpful (power, price and CO2 emissions), of which a majority (42.86%) stated that power was more helpful than the other two. The price was selected for 16.67%, and 12.5% of participants didn't find any of these metrics helpful. Surprisingly, only 6.25% of participants found CO2 emissions metric helpful. These numbers indicate that participants better associate power metrics to energy-related issues. Other metrics might add a layer of reasoning and comparison, therefore might render them less useful. In contrast, participants were more attentive to the visual chart rather than the numerical metrics with 25.58% of participants considering the visual chart of our tool more helpful, in comparison to 16.28% for the metrics, while a majority of 58.14% answered both. Of the latter, when asked to choose one of the two, a majority of 59.46% preferred the visual chart over the metrics (40.54%).
Participants also overwhelmingly agreed or strongly agreed (91.67%) that the power consumption graphs and metrics correlate to what they think their energy consumption is. However, this agreement does not translate with behavioral change in using software. When asked *"How often did you change your software usage?"*, a majority of participants (62.5%) answered that they did not change their software usage. In addition, a fifth (20.83%) only changed their software usage when they were doing less important tasks (in relation to the workload). Only 16.67% of participants applied some sort of behavior change when they saw a power spike in our visual feedback tool. However, when participants changed their behavior on every spike or temporarily, the applied actions were limited, mostly to a lack of knowledge of what to do. 54.17% answered that they switched software to reduce energy consumption, while 62.5% decided to reduce their usage time of some programs. In addition, 58.33% stated that they wanted to reduce their energy consumption but didn't know how.
These numbers indicate that participants know, essentially, only two ways to reducing their energy consumption: change software or reducing usage time. Both of these dimensions, tooling and temporal, imply that users must be armed with the knowledge of which software

to go to, and for how long should a temporal reduction be applied.

Perception of energy consumption: Software power consumption in mobile devices vary depending on the device itself. A mobile phone running a cloud-based Latex editor will not have the same power consumption as a Raspberry Pi or a laptop, even for the exact same workload. As users tend to use multiple devices (two thirds of participants state having 2 or more mobile devices), we argue that raw metrics may not be the best approach to raise awareness or drive behavioral change.
As applied in buildings [WN07], metaphors may provide better understanding for users about their energy consumption. Such metaphors might be metric-based, such as trees saved or nuclear plants required for electricity production, or might be traffic-light style, such as showing a red symbol or phrase if energy consumption is deemed high. The latter classification of what is high or low energy consumption, is a challenging task for the software research community. Some suggestions may include having a database of average energy consumption per workload, or per category of software, and can be used by an intelligent tool to show warnings to users based on their consumption and the importance of the workload.
From the results, there is a clear majority of participants ranking IDEs as the most consuming software. This finding may be biased to the fact that participants were computer science and engineering students using IDEs very often on not optimal laptop devices. Their perception of energy consumption might be influenced by the perception of slowness of IDEs, their rich feature set, or compilation time of programs. It would be interesting for the research community to extend our study to further understand this perception and what software developers can do to improve it, or to optimize their IDEs.
In contrast, communication software was seen even more consuming than internet browsers. In the context of the experiment (during lockdown in June and July 2020), videoconference software were extensively used, and running audio and video streams requires many CPU cycles and memory usage. Therefore, participants might be biased as they used these programs in addition to performing their class workloads, in comparison to the general public who may use communication software sparsely and with limited multitasking. These software are known to have a high energy impact on energy consumption [ZWC+09]. Therefore, we recommend better optimization for communication software, and switching to less machine-intensive encoding/decoding algorithms to alleviate the load and energy consumption on mobile and computing devices.
Finally, as the dependencies of mobile devices towards cloud computing is rising, the energy consumption of either is therefore close and inseparable. Most mobile devices use cloud-based services to perform their intended features: a smart speaker depends on cloud services to provide accurate and relevant answers to users, or a smart fridge might use cloud services to provide an up-to-date shopping list of missing items, and similar arguments for other devices such as smart TVs or connected security cameras.
We recommend outlining this dependency in green feedback tools, in order to show the impact of the entire chain of software and hardware when users interact with their devices.

Perception of green feedback: Our first observation of the reaction of users to our green feedback tool is that it helps in raising awareness of the actual power consumption of devices and software. Participants either exaggerated or minimized the energy costs, and nearly half

of them did not understand what the metrics mean. Our recommendation follows what we observed in the first part of the survey, that metaphors, such as a traffic-light system, might be more relevant to users.
In addition, participants thought that power metrics were most useful to understand the impact of energy. Without more in-depth investigation with participants, it is not straightforward to conclude that power metrics are easier to understand, and contradicts with a study from 2009 on the usage of energy meters in homes [KG09], that indicated users are more interested in the price metric rather than the power one. However, as our participants were computer savvy (computer science and engineering students) and young, the power metric might actually be more relevant for them, as this metric is common for overclocking, gaming or other hardware-related studies and activities.
In contrast, a small minority of participants, 6.25%, placed the CO2 emissions metric as the most helpful one. We argue that CO2 emissions are seen as an abstract measure, and it is hard for users to grasp how many grams of CO2 emissions is dangerous for global warming. Instead, we recommend abstracting much of these metrics, and use green-related or sustainability metrics. For instance, the number of trees saved, or the estimated degrees lowered or raised by the energy consumption or savings, might have a better effect on users.
Participants were also more impacted by the visual chart than the metrics themselves. The charts in our visual feedback tool is plotted in real time and its Y axis scales to show the maximum plot value, and therefore might had an impact of the users' perception of power consumption. Charts provide a short-term historical view, and most importantly, a comparative indication of an increase or decrease of power consumption. We argue it is important to keep such visual charts as it provides an immediate feedback of whether the actions of users have a positive or negative impact on energy consumption.
Our most interesting finding is that a majority of participants did not change their software usage behavior. And from the few who stated a change, a majority of changes happened when doing less important tasks, and only a minority changed behavior when a spike of energy consumption was seen in the feedback tool. This might be explained by the given workload, and the identity of participants: university students working on a project. We plan on conducting a long-term experiment with diverse profiles, in order to confirm our initial findings.
Participants also miss the tools and the knowledge of what to do to change their behavior, even if they want to. In particular, participants know two dimensions of actions:

- **Temporal:** by reducing the usage time of software, which was the most declared action. However, this answer should be put in perspective of the majority of users who did not change their behavior, or state that the change was not for important tasks. Concretely, we argue that users will rarely reduce the usage of programs or devices, unless they're using them for time-passing entertainment.

- **Tooling:** by switching to a different equivalent program. The main challenge in switching software, is the availability of quality and functionally equivalent ones. Users need to know whether switching to a different program will lead to energy reductions. We argue that energy-related software labeling is crucial, and that such labeling needs to be provided when installing software, and on every run (for example, in the form of a sign on software icons).

However, when we compared the actual power readings of the participants who stated they

changed their behavior, we observe little or no variation in the energy consumption. In Figures 3.8 and 3.9, we outline the power consumption of some participants who stated they changed their behavior when they saw a spike in power consumption, or when doing less important tasks, respectively. The figures show the power consumption data and a cubic spline interpolation (with a smoothing parameter of 0.5). When applying a simple linear model over these numbers, we obtain the intercept and slope.

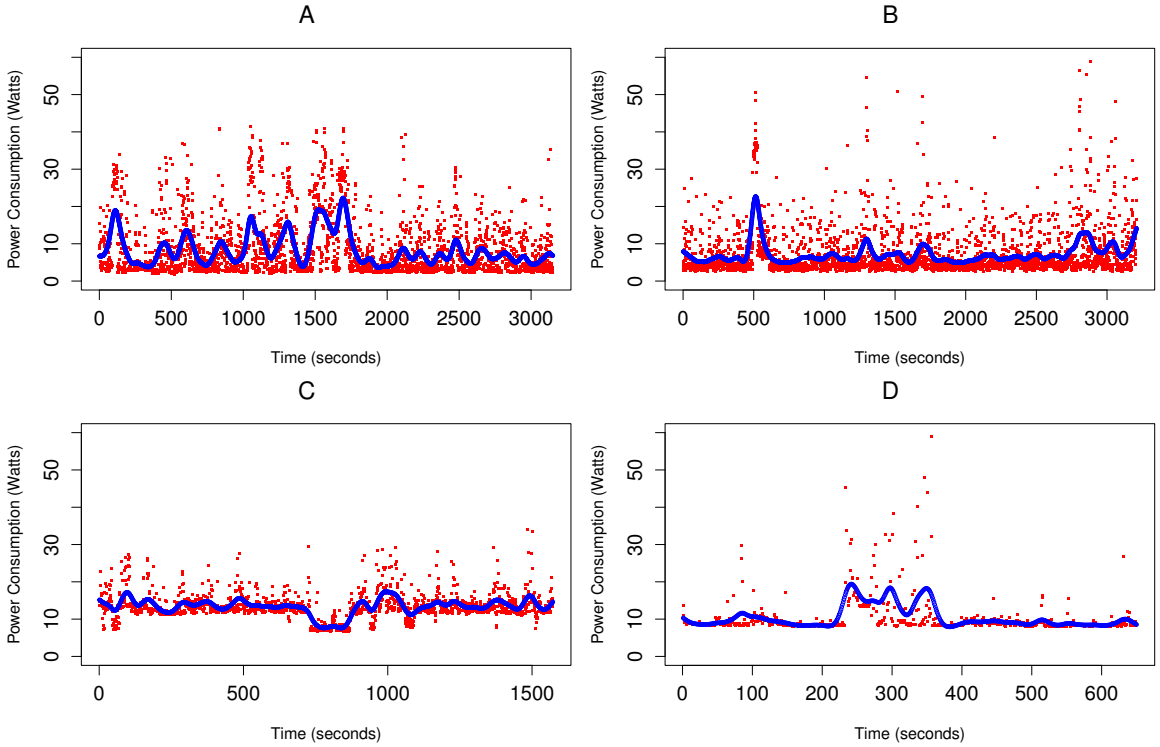


Figure 3.8: Power consumption and cubic spline interpolation (red line) of participants in group A2 who stated a software change every time they saw a spike in energy consumption in the visual feedback window.

Overall, the slope in the linear model, although mostly negative (meaning a reduction of power consumption), is negligible. The only exception was with participant C who had a small reduction after a spike in the middle of the experiment. After checking the program logs, we find that when the spike happened, power consumption rose for a few seconds to 21 then 29 watts, far beyond the average TDP of the CPU model (an Intel i7-6500U). During this event, the participant was using Google Chrome and working on the workload in the online latex editor, Overleaf. As soon as the spike happened, we observe a gradual drop in consumption over the next five seconds, to around 7 watts. This coincides with a switch in software, where the participant opened our monitoring tool for a second, then opened Windows File Explorer, and kept using it for three minutes before switching back to Google Chrome.
Other participants did not apply behavioral changes, or did little changes in software usage with no lasting impact on power consumption. For example, participant A did adopt monitored behavioral changes in software usage (*i.e.*, did not change software, switch tabs, or change the active window). The biggest energy spike (between seconds 1500 and 1700)

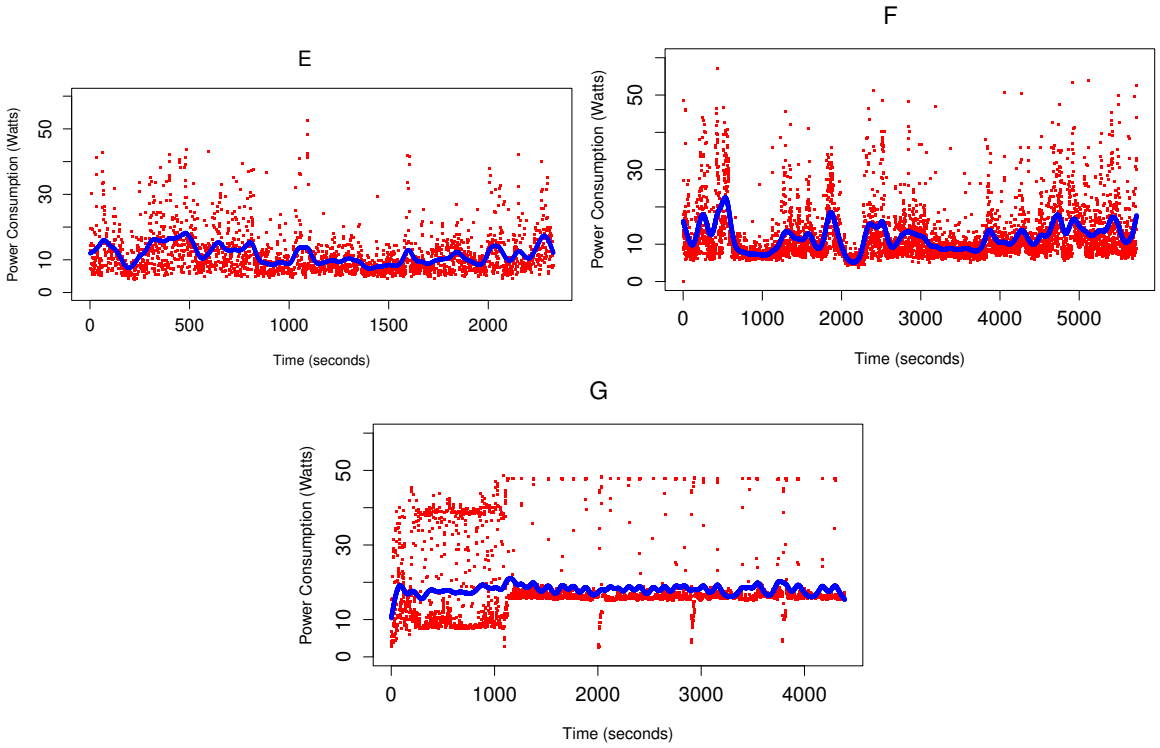


Figure 3.9: Power consumption and cubic spline interpolation (red line) of participants in group A2 who stated a software change only when doing less important tasks.

is associated with using Google Chrome (Overleaf and Google searches), which might be a more intensive work session. Participant B had one major spike around second 500 which is a direct result of accessing a specific web page on the course topic that contained text, images, a video and ads. The spike spanned for around 30 seconds, which might be due to the video auto-play. We did not observe any change in behavior specific to our feedback tool, as the user switched to Overleaf website (as expected from the course workflow) after the end of the power spike. Participant D power spike is explained by opening a PDF thesis online for around 10 seconds, then participant closed the tab which resulted in a minor reduction of energy, and finally switched back to Windows Explorer.
In contrast, users who stated to change behavior when doing less important tasks, did not translate into power reductions. For instance, participant 3 has a consistent usage of software and that translates with consistent power consumption with few spikes. Participant F had a more intensive and diverse usage of his computer (including for checking email and WhatsApp), but we did not observe any behavioral change when not working on the course assignment. Some of the power spikes actually amounts to personal usage of the laptop, with spikes when using WhatsApp desktop application. Finally, participant G also has a consistent power consumption which is due to a continuous intensive usage of software for the assignment and for personal usage (desktop email client, WhatsApp on Google Chrome). From these findings, we can conclude that participants, even when trying to change their behavior, fail to reduce their energy consumption. This might be due to little knowledge of what to do (which 58.33% of participants who wanted to change behavior didn't know how to do so). Or due to the actions applied, such as changing software or reducing usage time,

didn't have the expected effect by users.
Even users who changed behavior, did not correctly or sufficiently apply the changes they state they did. For instance, participant D stated a change in software every time the tool showed a spike. However we did not observe any reasonable switch in software in the log data beyond the normal changes in the workload.
We argue that green feedback needs to be accompanied by runtime propositions of change, along with a sustainability guide. For instance, an up-to-date sustainability guide or documentation might propose a list of equivalent software for the most important tasks on a device (*e.g.*, internet browser, word processing, popular applications such as weather or music), along with an energy-related label or classification. In addition, the green feedback tool needs to propose, on the fly, recommendations on what do to reduce energy usage. These recommendations would be based on the user historical usage, the energy label of software, and the current energy consumption and/or price and CO2 emissions. Recommendations would be clear and optional, and if the user accepts the proposition, the system must apply a smooth, low-energy, and low-latency transition of software without data loss.

3.2.5.3 Lessons Learned

From the field study and the survey, and our analysis, we draw the following lessons:

- **Users underestimate the energy consumption of their devices and software.** As we observed in our survey, users in the control group rated their energy session lower than users who saw their actual power consumption through our green feedback tool.

- **Users perceive the operating system as the most consuming software.** The OS is seen as the most power-consuming software in our experiments, for both the control and experiment groups. High or constant idle power consumption might explain this perception. But also how slow or sluggy systems can be, even though it might be because of other software, most notably background ones.

- **Programming and communication programs are perceived as high-consuming software.** For computing science and engineering students, sometimes using older devices, heavy-duty IDEs can be seen as slow and consuming resources, including power. The same applies for communication software, in particular videoconference programs that might stress the CPU and the device's resources.

- **Technical-savvy users understand the energy weight of the cloud infrastructure.** Most participants in our field study, acknowledge the bigger impact of the cloud infrastructure (*e.g.*, data centers, network) compared to individual end-user devices. However user awareness of the chain of impacts of these devices and software services towards the cloud infrastructure is yet to be properly understood.

- **Live green feedback helps in raising awareness of energy consumption.** Although we did not observe noticeable changes in the overall power consumption between the control group A1 and the experiment group A2, we observed tentative behavioral changes for users in group A2 (with our green feedback tool). Participants wanted to change their behavior but had a few obstacles to do so.

- **Green feedback tools must be minimal and seamlessly integrated to avoid being distracted.** The majority of participants saw our green feedback window as distracting due to its nature (always on top of other windows, and could not be minimized or resized, even though it was small and only covered a limited space in the lower right-end of the screen). The distraction nature of the feedback window could not be scientifically assessed until after we conducted the experiments and collected participants' answers. We argue that green feedback needs to be transparently integrated into the operating system or software systems of end-user devices.

- **Metaphors and evolving charts are more useful than green metrics.** Specific green metrics, such as power consumption, electricity price, or CO2 emissions, do not seem to have an effect on end users. However, charts indicating the evolution of power consumption had a better impact (participants changed behavior following power spikes in the chart). We also argue that metaphors, such as a traffic-light system or the number of trees saved, might lead to better awareness and behavioral change.

- **Users resist behavioral change when the workload is important, or the energy consumption is perceived as low.** Many participants who tried to change their energy behavior, did so when doing less important tasks, or only when an important spike in energy occurred. This indicates that, depending on the workload, users will prioritize the tasks they're doing, over non-functional energy requirements. This might be due to the lack of knowledge of the cost of changing software or reducing usage, or other energy-efficiency actions.

- **Users lack the knowledge and tools to apply software behavioral changes.** A majority of participants who wanted to change their behavior failed to do so because they didn't know what to do. And the remainder who reduced software usage or switched software, did not apply those actions correctly or long enough for any noticeable energy savings. We argue that users need guidance and recommendations in order to know what actions are effective and simple to implement.

Epilogue: In this research, we presented the results of a field study and survey we conducted in order to understand what users know about energy in devices and software, and how can green feedback raise their awareness and push for behavioral changes.
We studied the impact of awareness through live visual feedback using green metrics (power consumption, electricity price and CO2 emissions), and real time evolving charts. Our main conclusion is that green feedback helps in raising awareness, but users lack the knowledge and tools to apply software behavioral changes. Therefore, short-term behavioral changes require previous awareness and guidance so users can change behavior seamlessly. In addition, we observe the importance of the workload, and the tasks when using connected devices and software, as users resist software behavioral change unless the task was less important or its perceived energy is low.
This work opens the way for future endeavors and perspective, such as further investigating the impact of various green feedback approaches on a wider profile of users. In particular, as users are not alike, their reactions to certain forms and modalities of feedback are not similar. Users also evolve as they age or their social or economic life change. Therefore, the challenge is to sustain behavior changes on the long term, which might be achieved through

autonomous and intelligent analysis and decision-making. These perspectives are detailed in my conclusion chapter in Section 4.2.
However, with these understandings of what impacts software energy, we can start acting towards greening software ecosystems and managing their energy consumption, of which my contributions are presented in the next section.

3.3 Acting Towards Green Software

Contributing towards greening software and its ecosystem is challenging as we require first an accurate and complete observation at the energy consumption of such systems, and also a proper understanding of the factors impacting energy. As such, my initial contributions in this section leverage my work I presented so far in this manuscript, to propose two approaches. The first contribution uses assessment tools and methods to observe the energy consumption in data centers, and through knowledge from certifications and guides, we propose a tool aimed for procurers and technicians to make better green decisions regarding their data centers. The second contribution is a first proof-of-concept autonomous management framework that binds observation, understanding and acting together in an initial holistic approach. This framework uses reinforced learning algorithms to continuously learn from the system and from user behaviors, while observing the system, in order to apply the better green actions.

3.3.1 EURECA Tool

The EURECA project is an EU-funded project in the Horizon 2020 program. It started in March 2015 with partners from the United Kingdom (with the University of East London as coordinator), Ireland, The Netherlands, and Germany. The goal of the project is to provide tailored solutions and guidance to help identify energy efficiency and environmental solutions along with cost-saving opportunities, in the procurement choices and process in data centers.
The project aims at tackling the lack of knowledge and awareness on how to identify and procure environmentally sound and sustainable data centers. The work will encompass solutions for pre-commercial procurement (PCP) and procurement of innovative solutions (PPI). This will be achieved by consolidating recognized and emerging benchmark criteria into an easy-to-use tool that can be deployed and used by non-experts.
I worked for 15 months as a research fellow in the EURECA project, where my main roles, along with participation in the project management and workshops, were to design and develop the EURECA tool.
The EURECA Tool was launched in September 2016 at the EURECA workshop in Amsterdam, Netherlands. The tool is a web-based platform that helps public procurers and data center professionals self-assess the energy efficiency and profile of their data centers, and provides improvements and suggestions. This is based on the latest standards, best practices and frameworks as well as research findings and industry input. In particular, the tool maps the Data Centre Maturity Model (DCMM) [Gri] to the EU's Code of Conduct for Energy Efficiency in Data Centres [Com] to provide tailored recommendations for data centers energy efficiency. In addition, the tool provides procurement support with various templates and case studies.

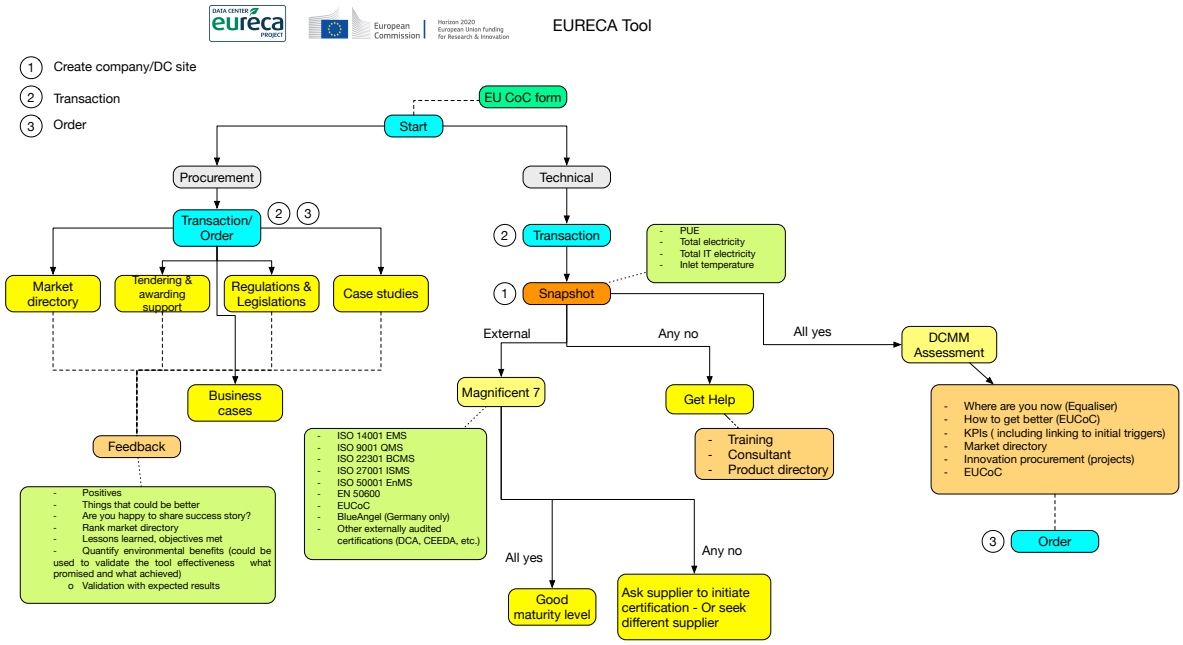
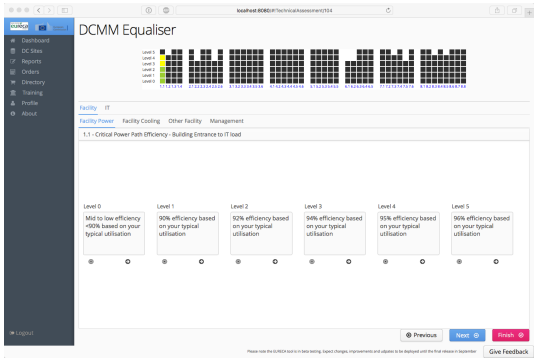
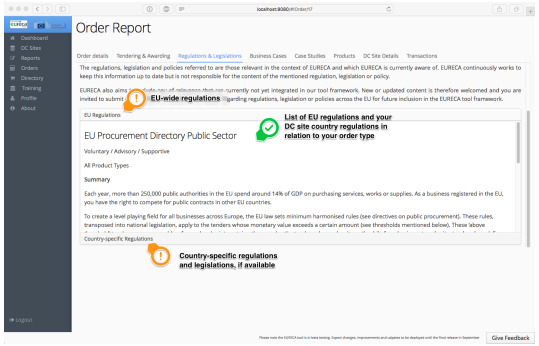


Figure 3.10: General design of the EURECA Tool workflow



(a) DCMM self-assessment in the EURECA Tool



(b) The EURECA Tool order report

Figure 3.11: The EURECA Tool interface

The EURECA Tool workflow is shown in Figure 3.10. Depending on the user (*i.e.*, procurement manager or a technical employee), the tool provides two paths: a technical one, and a procurement one.

The technical path allows users (procurers or technicians) to get tailored assessment reports and sustainability advice about their data center site. The tool first queries the user for technical data about their data center site, by verifying what KPIs are being measured. The KPIs are PUE, inlet temperature, total energy consumption of the data center site, and the energy consumption of the IT equipment of the data center site. This helps assessing the maturity of the data center team in regards to measuring key metrics and KPIs related to energy efficiency.

If all KPIs are measured, and thus the team is mature in its energy-efficiency approach, then the tool opens a self-assessment DCMM view. DCMM is the Data Center Manturity Model [Gri], and the tool provides an interactive and visual view for users to self-assess their data center site according to the DCMM. This allows users to select their current maturity levels, and the target maturity levels they are aiming to achieve 3.11a. According to the current and target levels, the tool then uses the DCMM assessment and the data center site characteristics to provide a detailed and tailored report, including recommended actions for improving the energy efficiency of the data center site. These recommendations are based on a mapping between the DCMM and the EU's Code of Conduct for Energy Efficiency in Data Centres [Com].
However, if not all KPIs are measured, then the team is not mature enough to self-assess their data center using the self-assessment tool. The EURECA tool will then check if the data center site is hosted internally at the user's organization, or externally at a supplier or a cloud provider. If it is hosted internally, the tool will provide a *get help* report, thus recommending users to seek help and support through consultancy services or training materials. If the site is hosted externally, the tool will check if the supplier is certified from a list of recommended certifications (the *Magnificient 7*), and provides a report accordingly: either a *good maturity* report, indicating a satisfactory level of energy efficiency, or a *ask supplier for improvement* report, suggesting to the user to ask their suppliers to initiate certifications, or to seek a different and more energy-efficient supplier.
The procurement path, only available for procurers, and aims to provide advice and guidance in the procurement process. Procurers can either start a new procurement order for their data center site, or start an order based on a technical report (that was already generated by the technician). For each order report, the following guidance information is provided 3.11b:

- Explore data centers use case studies similar to their order,
- Explore business cases with an interactive tool able to generate business cases to help calculate return on investment (ROI) in regards to energy efficiency,
- Explore and read local (country-specific) and European regulations and legislation appropriate to their setup (type of products or services),
- Access a subset of the market directory of hardware and data center services, filtered based on the type of the order and the geographic location of the data center site,
- Access tendering and awarding support, including guidance, energy efficiency indicators, awarding criteria, or tendering templates.

In addition to the tool itself, a training curriculum is provided for procurers and technicians. The aim is to train data center operatives and managers on energy efficiency, and raising their maturity and skills in order to be able to self-assess their data center sites, and improve their energy management. The training curriculum includes procurement modules (PPI for Public Sector Procurers and ICT Managers, Business Case Development, Legislation and Policies, Procurement Strategy, Tendering, and Data Centre Contracts and Risks), and technical modules (Data Centre KPIs and Standards, The EU Code of Conduct for Energy Efficiency in Data Centres, and The Data Centre Maturity Model). Along with the specific training modules, a number of workshop events were also organized (starting in June 2015

up to February 2018), aiming to raise awareness and provide guidance and training about the EURECA project, data center energy efficiency and training participants.
Much work has been done in data center energy efficiency with a technical approach, whether on computing hardware or non-IT equipment, or on software (applications, virtualization, etc.). However, few approaches targets non-technical users and with a holistic view. The EURECA Tool is an important step towards bringing data center energy efficiency towards users, such as procurers, managers or deciders.

Epilogue: The EURECEA tool helps fill a gap in guidance and recommendations for procurers in data centers. However, more complex ecosystems, such as cyber-physical systems, lack personalized recommendations. In addition, end users, even tech-savvy ones, may not have the same level of expertise as data center technicians, or the same level of motivation as procurers, to manage their ecosystems. For this, I present in the next section, an autonomous framework for energy management in cyber-physical systems, leveraging observation techniques and building knowledge of the user behaviors and of energy consumption, to drive effective energy decisions.

3.3.2 Automated Management Framework

> The content of this section is adapted from the following publication:
> An Automated Energy Management Framework for Smart Homes. Houssam Kanso, Adel Noureddine, and Ernesto Exposito. Journal of Ambient Intelligence and Smart Environments (JAISE). 2023.

After observing and understanding the factors impacting software energy and their ecosystems, the next step is applying this knowledge towards managing and reducing energy consumption. Managing energy of complex and heterogeneous software systems requires capturing and monitoring the energy consumption of these systems, understanding the energy impact of each action and decision, and using this knowledge to apply corrective measures to reduce this impact on the long run.
For this, we propose an automated energy management framework , aimed as a first proof-of-concept of our approach, capable for autonomously managing energy in smart homes. The developed energy management system is flexible, scalable, and respects user preferences. It collects data, represents knowledge, and executes optimal actions using ontologies and reinforcement learning techniques. The architecture of this approach is based on three main components: power estimation models, knowledge base, and intelligence module. An implementation of the approach is developed and two validation experiments are conducted in the context of smart homes: (a) a living room with a variety of devices, and (b) a smart home with a heating system. The obtained results show a clear improvement in power consumption while respecting user preferences in both case studies, and offer a first validation for our approach that we aim to expand towards more complex software systems.

3.3.2.1 Framework Architecture

Our approach, presented in Figure 3.12, aims to provide an intelligent power management solution for connected environments by proposing: (1) an automated approach to empir-

ically generate power estimation models (PEMs) for a large set of devices (based on our crowd-sourced modeling in Section 2.2.2), (2) an extension of the SAREF ontology as a knowledge-base contextual understanding component (based on our ontology in Section 2.2.1), and (3) automated knowledge-based reinforcement learning environment and agent generator that generates a decision-making agent responsible of choosing the optimal actions to be executed. It also includes a data repository component formed by a collection of databases that include previously collected metrics, states, applied actions, rewards, power consumption, power estimation models, and potential external data (e.g., outdoor temperature).

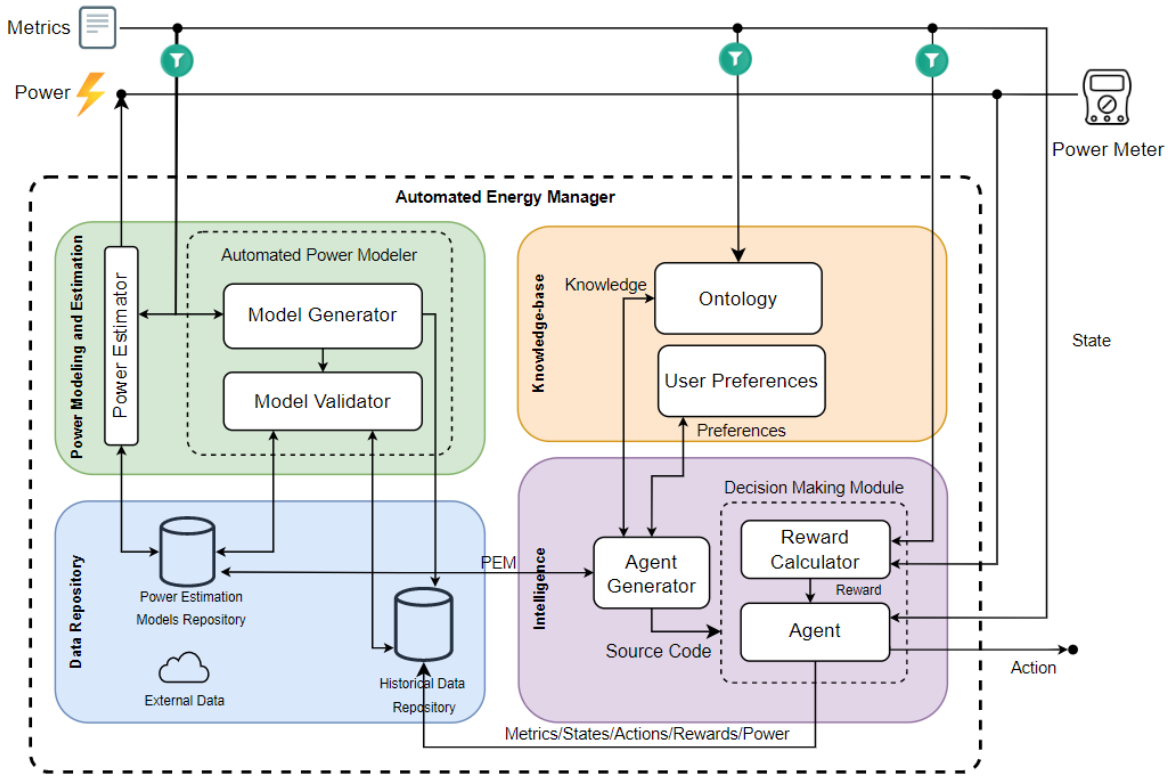


Figure 3.12: Automated Energy Management Architecture

The proposed architecture ensures the following criteria: high autonomy maturity level, flexibility, heterogeneity of devices, general-purpose, and ability to deal with a large variety of metrics. Any changes or updates that occur in the environment are considered and lead to the calibration of estimation models and reinforcement learning agents. It also guarantees a high autonomy level where high-level user-oriented energy policies control the infrastructure operations in an automated manner.
As we already explained in detail, the power modeling and ontology in Sections 2.2.2 and 2.2.1, we will only present here our intelligence module and its reinforced learning algorithm.

Intelligence Module: Figure 3.13 presents the architecture of the reinforcement learning environment and the agent generation process. The agent generator aims to collect intelligence from the ontology and create a virtual environment and a reinforcement learning

model based on the devices present in the environment and the relations between them. First, the agent generator requests necessary knowledge from the ontology and the power estimation models repository for each device. Once this information is retrieved, the agent generator creates the source code of the reinforcement learning environment or modifies the existing one. The generated source code includes all the necessary information needed to optimize energy consumption using reinforcement learning techniques such as states, actions, reward calculation, and power estimation models.

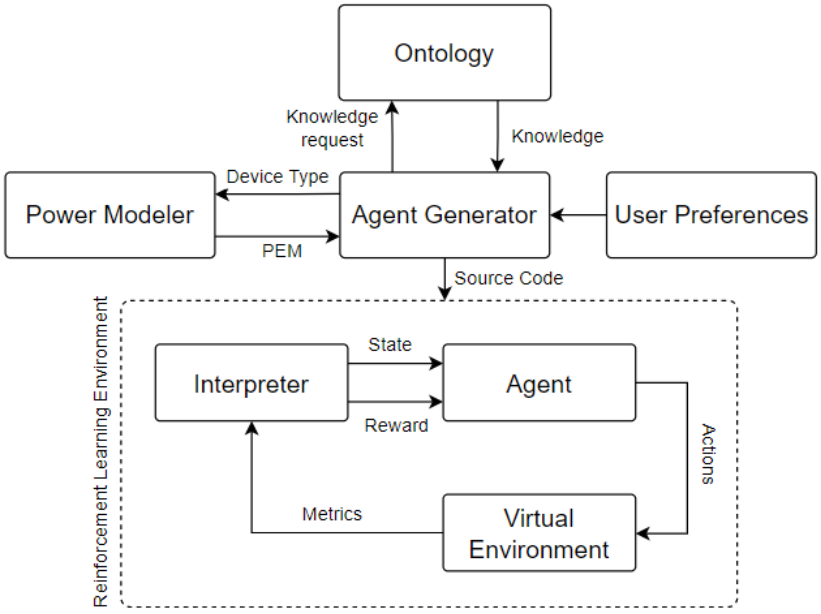


Figure 3.13: Reinforcement Learning Environment Generation Architecture

Reinforcement Learning Implementation The agent generator is responsible for the generation of each section of the reinforcement learning agent based on TensorFlow and Keras-RL, in addition to an environment built using OpenAI Gym. Algorithm 1 provides the pseudocode of how the reinforcement learning environment is generated. First, a series of SPARQL queries are executed to collect knowledge from the ontology using the *request_sparql* function. Graphs containing information concerning actions, metrics, states, and power estimation models of each device corresponding to a unique location are collected.
The collected knowledge is used for the creation of the source code of all the functions that compose the reinforcement learning environment. A function called *generate_code* generates the source code of each of the reinforcement learning environment sections based on the collected ontology knowledge. It generates the source code of six main components: (a) actions, (b) states, (c) rewards, (d) power estimation, (e) initialization, and (f) reset. A template of the source code of the reinforcement learning environment is created. It includes the necessary libraries and fixed chunks of code. However, the automated source code generator writes the majority of the code by using the *write_code* function. It identifies the blank variable sections of the source code that change from one environment to another and writes the correct code in each section. It can generate the source code of any number of properties and devices due to its extensible way and ability to request knowledge and use it.

Algorithm 1: Reinforcement Learning Environment Generation Algorithm

procedure GENERATERLE($ontology$)
    Initialize sparql queries
    $actions, metrics, states, power \leftarrow request\_sparql(ontology, sparql\_query)$
    $actions\_code = generate\_code(actions, states)$ ▷ Generate source code
    $states\_code = generate\_code(metrics, states)$
    $rewards\_code = generate\_code(power, metrics)$
    $power\_estimation\_code = generate\_code(power, metrics)$
    $init\_code = generate\_code(actions, metrics, states, power)$
    $reset\_code = generate\_code(actions, metrics, states)$

$write\_code(init\_code, reset\_code, actions\_code, states\_code, rewards\_code, power\_estimation\_code)$
end procedure

Once the source code is generated, the reinforcement learning environment is ready to be launched. The following explains the role of its main functions:

- *__init__* function defines the observation and the action spaces, i.e., the minimum and maximum acceptable values for each observed metric or action. In addition to a few other attributes, i.e., accepted values for each property and the initial state of each device. This function also initializes states, properties, and power consumption per device. It also encodes and maps properties and states from their initial format to a numerical or boolean format to be accepted by the reinforcement learning algorithm.

- *step* function executes a step in the environment by applying an action and returns the new reward, observation, power consumption per device, and other info. It includes the list of possible actions and the reward calculation based on user preferences and total power consumption. In this function, randomness and external changes impacting the environment can be applied (e.g., outside temperature changes).

- *calculate_power* function contains the power estimation models for each device in the environment. It returns the value of the power consumption of a specific device at a particular time.

- *total_power* function returns the total power consumption of all devices in the environment at a particular time.

The reward value is a rational number bounded between -1 and +1. It is calculated based on three equations (3.1), (3.2), and (3.3). First, it considers user preferences and makes sure all user preferences are respected. If any preference is not respected, the reward will be negative between 0 and -1. It is calculated by counting the number of unrespected preferences and dividing it by the total number of user preferences, as seen in Eq. (3.1), when the user preferences are not measurable.

$$R_t = -1 \times \frac{count(unsatisfied\ user\ preferences)}{count(user\ preferences)} \quad \in [-1, 0] \tag{3.1}$$

However, if the user preferences are measurable (i.e., temperature), the reward is calculated based on the difference between the property value V_t^i at a specific time t and the desired value V_{user}^i , as seen in Eq. (3.2).

$$R_t = -1 \times \frac{|V_{user}^i - V_t^i|}{1 + |V_{user}^i - V_t^i|} \quad \in [-1, 0[\tag{3.2}$$

Once all users basic preferences are respected, Eq. (3.3) is adopted to calculate the reward and ensure a higher reward for each power-aware action. This equation is bounded between -1 and 1. Its value is -1 when the current power consumption is equal to the maximum power consumption and tends to 1 when the power consumption goes down to 0. Reaching 0 as power consumption for the entire environment is difficult due to user preferences but remains possible in some cases.

$$R_t = 1 - 2 \times \frac{P_t^{total}}{P^{max}} \quad \in [-1, 1] \tag{3.3}$$

where: R_t is the reward calculated at time t, P_t^{total} is the total power consumption estimated or measured at time t, and P^{max} is the maximum total power consumption of the environment.

Algorithm 2 shows the model and agent generation pseudocode. First, states and actions are defined based on the previously generated reinforcement learning environment. Then, the model is built using the *build_model* function. The model is based on a Deep-Q network that takes as an input layer the set of states and returns possible actions in output nodes. It has two hidden dense Keras neural network layers along with a ReLU activation. The *build_agent* function specifies the policy and builds the agent based on the previously built model. The agent and model are compiled using the *compile* function, they are later used for training and testing. The *fit* function trains the agent given the environment and the number of training steps.

Algorithm 2: Deep Reinforcement Learning Model and Agent

```
1: procedure CREATERLAGENT(generated_environmentenv)
2:     states = env.observation_space.shape[0]
3:     actions = env.action_space.n
4:     model = build_model(states, actions)
5:     agent = build_agent(model, actions)
6:     agent.compile(model, actions)
7:     agent.fit(env, nb_steps)
8: end procedure
```

3.3.2.2 Case Studies and Discussions

A prototype of the proposed power management approach has been developed to reduce power consumption in smart connected environments. In particular, the following two application case studies have been developed, deployed, and experimented in a smart home for validation purposes.

Living Room Case Study As seen in Fig. 3.14, the first simulated case study is defined as a living room of a smart home with several devices. A LED bulb controls its brightness (LedBrightness) and color (LedColor). A TV controls its brightness (TVBrightness). A Raspberry Pi that measures its CPU utilization and has a light sensor mounted on it that measures the room brightness. The LED bulb, TV, and raspberry pi are considered power consumers and have each a power estimation model that gives their power consumption in real time. A door sensor checks if the door is opened or closed without being able to change its state. The door sensor is not considered as a power consumer because its power consumption is minimal and unchangeable. LedBrightness has a minimum value of 0 and a maximum value of 100 and can be increased or decreased by 1% per minute to ensure a smooth light transition that is not annoying for the users. LightColor can have one of the following values: red, blue, green, and white. Devices states are usually on, off, or sleep. However, they are not limited to these states and can take many other forms. For the living room scenario, user preferences are generated based on [LPC+19]. They are manually adjusted to add randomness that may occur from a day to another, between weeks, and during weekends.

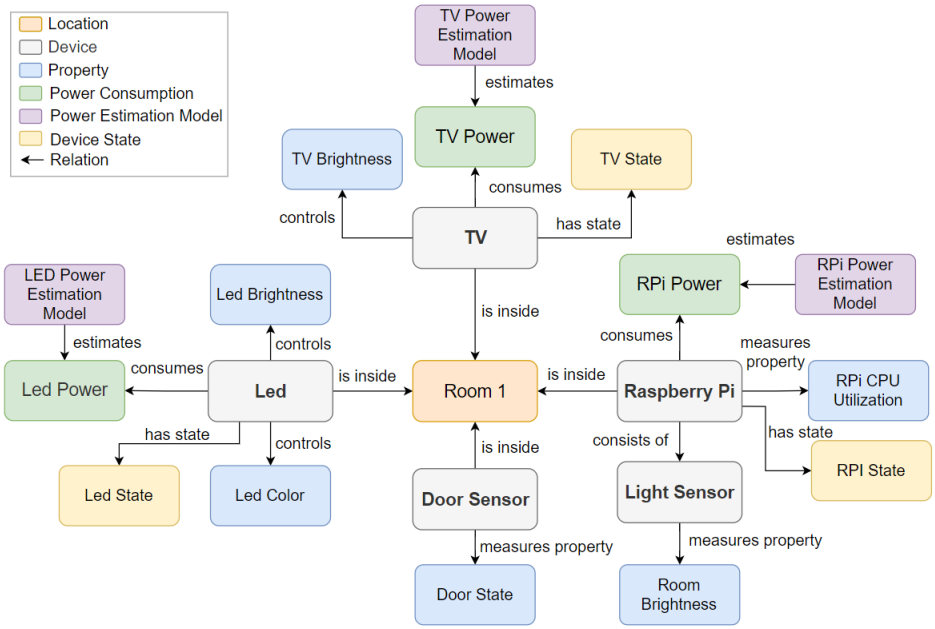
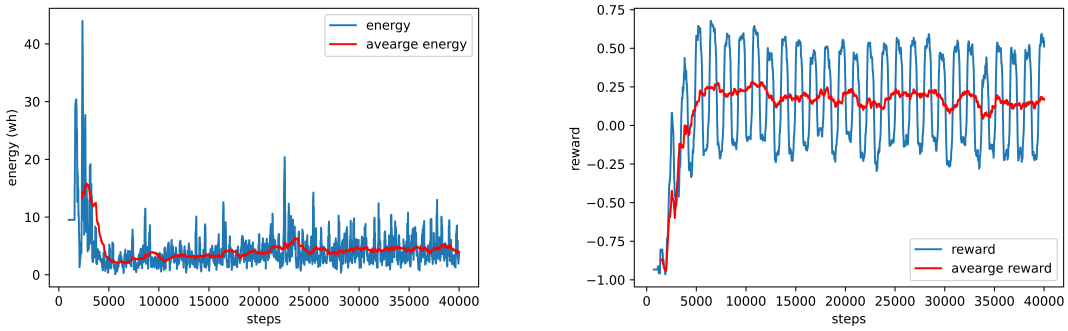


Figure 3.14: Devices, Properties, and Relations Representation of the Living Room Case Study

The experiment has been conducted for 70 episodes (100k steps). Each episode lasts for 24 hours and one step is executed per minute (1660 steps per episode). Once per minute, metrics and states of devices are collected. Then, the agent chooses an action to be executed per device. However, the agent has the possibility to execute a neutral action that keeps all devices in their current state. User preferences for the 28 days are defined (*i.e.*, led state, led brightness, led color, TV brightness, and TV state). Figure 3.15 shows the total energy consumption of the environment per day and the total reward points collected per day by the agent. In such a scenario, energy has converged during a brief period of time (4 days), and the reward has passed from a negative value to a positive one meaning that the user preferences are respected and that the power consumption is at its lower accepted values.

This study case shows that the proposed framework can deal with different types of devices, properties, and actions related to these devices. It validated that using this approach is able to represent knowledge, generate reinforcement learning environments, and train agents. In addition, to the ability to optimize energy consumption in the long term.



(a) Energy consumption per step

(b) Reward point per step

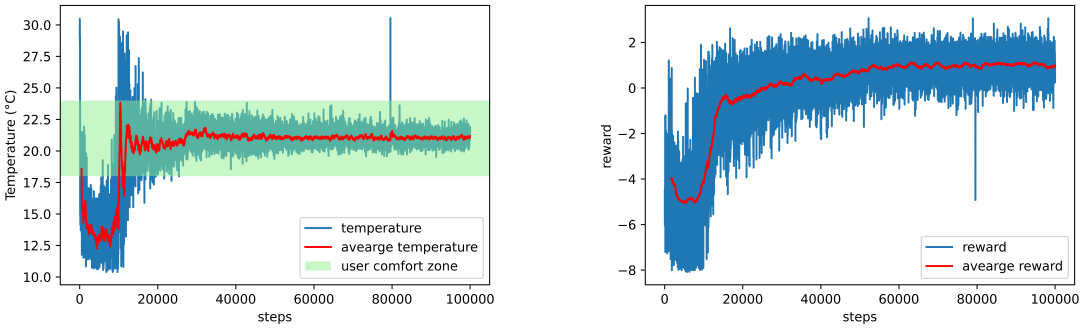
Figure 3.15: Living Room Case Study Experimental Results

HVAC Case Study Another simulated case study is conducted with different devices in a different context. It is based on a HVAC system in a smart home. The room where the HVAC is mounted is equipped with an indoor temperature sensor and a presence sensor. In addition, a temperature sensor is positioned outside the room to capture the outdoor temperature. We model the environment using the previously proposed extension of the SAREF ontology as done in the living room case study. Then, we launch the agent generator to build the environment and the reinforcement learning agent. To simulate indoor temperature dynamics in the environment, the exponential decay model in Eq. (3.4) is adopted [CSL91].

$$T_{t+1}^{in} = \varepsilon \times T_t^{in} + (1-\varepsilon) \times (T_t^{out} \pm \frac{\eta_{hvac}}{A} \times e_t) \quad (+ : heating, - : cooling) \tag{3.4}$$

where T_t^{in} denotes room internal temperature during time t, T_t^{out} is the real outside temperature during time t, $\varepsilon \in [0,1]$ is a thermal time constant, η_{hvac} is the thermal conversion efficiency (heating) or the coefficient of performance (cooling), A is the thermal conductivity that varies based on the insulation of a space, and $e_t \in [0, e_{max}]$ is the electric HVAC system power input at time t (It ranges between 0 when the HVAC system is turned off and e_{max} the maximum power consumption of the HVAC system). We consider $\varepsilon = 0.7$, $\eta_{hvac} = 2.5$, $A = 252\, w/^{\circ}C$ based on the study conducted in [YXX+19], $e_t = \frac{U_t}{100} \times e_{max}$ where U_t is the percentage of utilization of the HVAC system, and $e_{max} = 2000\, w$.
Eq. (3.4) was widely adopted in previous research such as in [YXX+19, ZLSO16, TX12, FPHP13, DZRL16]. This experiment used the outdoor temperature dataset proposed by USCRN/USRCRN [DKP+13] due to its accuracy, low error rates, and high quality of collected data. The dataset is cleaned and filtered mainly to remove missing data indicated by the lowest possible integer, i.e., -9999,0. The used dataset was collected in Santa Barbara, California, the USA

in 2021. User preferences temperatures are extracted from the study conducted in [JND19]. User temperature preferences range between $20^{\circ}C$ and $24^{\circ}C$ during the day. They range between $18^{\circ}C$ and $22^{\circ}C$ during the night when the home occupant is sleeping.
The experiment was conducted for 596 episodes (100k steps). Each episode has lasted one week with a frequency of one step executed each hour (168 steps per episode). User preferences of the minimum and maximum accepted temperatures are defined for an entire week. Results show an increase in the reward per episode, therefore, the user satisfaction and the power consumption, as seen in Fig. 3.16b. User preferences are variable between the day and the night. However, for the purpose of simplicity, Fig. 3.16b graphs the user comfort zone between the lower and higher accepted values. Figure 3.16a shows the room temperature per step and the average room temperature per episode. Results show that in the first part of the training (before the 20 000^{th} step), the indoor temperature is under the user preferences and converges with time to respect user preferences. Therefore, validating the ability of the proposed reward function to lead to a better user experience. Initial room temperature is randomized for each new episode ranging between $12^{\circ}C$ and $18^{\circ}C$, and the initial HVAC level is a random value between 30% and 70%.



(a) Indoor temperature

(b) Reward convergence process

Figure 3.16: HVAC Case Study Experimental Results

To compare the energy consumption of the proposed approach to other control values, we identify three different cases:

- *Case 1:* Represents the results of this papers approach from the knowledge, environment and model generation, and training of the agent.
- *Case 2:* Represents an environment where user preference indoor temperature is set to $21^{\circ}C$. This value is obtained by calculating the average temperature between the minimum and maximum accepted values. The HVAC is turned ON whenever the room temperature goes below $21^{\circ}C$ and turned OFF when it outpaces this value. The obtained values are based on the captured indoor and outdoor temperature readings.
- *Case 3:* Represents an environment with a traditional electric heater that stays ON all the time without capturing and considering any of the indoor or outdoor temperatures.

The mean value of energy for each of the previously defined cases shows that the proposed method has the lowest energy consumption compared to the other two traditional methods.

Case 1 energy consumption equals 932,82 wh, while cases 2 and 3, respectively, are 1083,3 wh and 2000 wh. We consider that the model has already converged before step number 30 000 and energy consumption is almost stable. Therefore, in the 1^{st} case, the average energy consumption is calculated with data corresponding to steps higher than 40 000.

Epilogue We presented an automated energy management framework for smart homes based on knowledge representation and reinforcement learning. It leverages our crowd-sourced power modeling and ontologies, and can use the different tools and approaches we designed to observe energy consumption in Chapter 2. The knowledge collected from observing the environment, and relevant understanding of the actions that have an impact on energy consumption, allowed us to automated energy management with an autonomous approach based on a reinforced learning algorithm. Our initial proof-of-concept and evaluation showed a significant decreased in energy consumption while also respecting user preferences, with a low convergence period.
This proof-of-concept shows that observing and understanding complex environments is key to efficient energy management. This first step in automating energy managing is encouraging and can be used as a base towards my vision and my perspective for a holistic energy management of software which I'll detail in the final chapter of this manuscript.

3.4 Epilogue

After observing energy, the next step is analyzing and understanding the factors impacting energy consumption and efficiency in software environments. The challenges we face come from the heterogeneity of such environments, and to the substantial number of parameters and factors impacting energy. In my contributions, I tried to make sense of all these factors, through rigorous analysis of various factors, from low-level CPU metrics to high-level design choices and human behaviors. The knowledge and lessons we learned from my different studies are additional building blocks in my vision towards a holistic energy management of software environments.
My contributions shed a light in understanding the energy dynamic in software ecosystems, and provide steps in improving this efficiency. However, there are still open challenges we need to tackle. For instance, the role of end users is paramount in driving long-lasting energy efficiency. But current literature is scarce on the role users can play in software energy optimizations. In particular, we need to understand what different social demographics and user profiles understand about software and software energy, how they can be driven to adopt energy-efficient software behaviors, and how to sustain these behavioral changes. Developers also need to be properly trained on how to write energy-efficient code, and design energy-efficient architectures. But writing green code also means being able to detect bad practices and software anomalies and bugs. On the technical spectrum, managing energy in multiple platforms often requires shifting resources and software services in heterogeneous ecosystems. Finally, acting towards green software ecosystem will lead to holistic software energy management, which will require observation, understanding and implicating all actors in the optimization process.
Our adventure in green software environments is not yet over. The challenges we still face are numerous, and I outline ideas and suggestions to address them in my perspectives, detailed in the next chapter in Section 4.2.

Chapter 4

Conclusion and Perspectives

Contents

This chapter concludes the manuscript. I first provide a summary of my contributions and research conducted in the last decade, then discuss the research perspectives for the next decade.

4.1 Summary

In this manuscript, I explored the different facets of green software, ranging from observing the energy dimension of software ecosystems, to understanding the elements at play, including human actors, and steps towards acting on this software energy dimension.
My vision, tackling the challenges I presented in Section 1.2, consists of optimizing and managing software energy ubiquitously. This encompasses optimizing the instructions of software, the hardware it is running on, and also optimizing the needs and requirements.
Towards achieving this vision, we need to first **observe**, and then **understand** software energy. As software today is ubiquitously, we need to observe and understand software energy *everywhere* it is present.
My contributions, so far, towards this vision were aimed at creating building blocks to observe software ubiquitously, and to understand what impacts its energy consumption. This meant building observation approaches and tools to measure, model or estimate energy at all the layers of software ecosystems. Such approaches need to be aimed towards both developers and technical users, and also managers, procurers and end users, in addition to automated and other software systems.
Likewise, my contributions aimed to understand the factors in play in software energy and in complex ecosystems, in order to provide knowledge and analysis to help build more energy efficient systems. This includes understanding low-level hardware and software factors, source code, applications and devices, design choices, and the role of users.
This understanding allows me to provide initial steps towards energy management of these ecosystems, from hardware to software, with recommendation and guidance tools, and an autonomous management framework.
In my contributions, I tried to integrate all actors in the energy efficiency equation, including human ones which are often forgotten. This goes from knowledge and tools aimed to developers, but also guiding procurers and managers, and understanding end users' behaviors.

Observing green software ecosystems: The heterogeneity of software ecosystems makes observing their energy consumption harder. For that, my contributions were aimed at facilitating this observation through models, approaches and tools, starting from the hardware layer up towards the source code of software.
In Section 2.2.1, we modeled the energy dimension of complex cyber-physical environments with an extension of the SAREF ontology with energy-related attributes. This modeling allowed us to map the energy consumption of various heterogeneous devices, capture contextual data relevant to energy, and provide key elements towards building energy-aware knowledge bases useful for autonomous energy management.
In Section 2.2.2, as devices and hardware components rarely have a power model or embedded energy sensors, we proposed a crowd-sourced automated approach to model the energy consumption of heterogeneous devices and hardware components.
We then implemented our approach in Section 2.2.3, for single-board computers, and in particular for Raspberry Pi devices.

And in Section 2.2.4, we adapt our approach to model the energy consumption of non-computing devices and objects.
Along with these modeling approaches, I built architectures, approaches and tools capable of measuring the energy consumption of hardware and software at different granularities, from entire devices, to hardware components, to application, and in source code and execution branches.
In Sections 2.3.1 and 2.3.2, I proposed PowerJoular and PowDroid, aimed to monitor the energy consumption of various devices and hardware components. PowerJoular monitors the energy of computers, servers, and other computing devices, such as single-board computers, all across various CPU architectures (x86, ARM). And PowDroid is capable of monitoring the energy consumption of Android devices with no modifications or installation on the device itself.
In Sections 2.3.3 and 2.3.4, I proposed Jolinar and Demeter, to monitor the energy consumption of applications in a user-friendly way. Jolinar, on Linux, is tailored towards power users and can monitor individual applications with a granularity at the hardware component level. And Demeter, on Windows, is a user-centric software architecture capable of monitoring all running applications on multiple computers, and can provide users with a summary of their energy consumption and recommendations.
Finally, I proposed JoularJX in Section 2.3.5, a multi-platform approach capable of monitoring the energy consumption of software structures (at the methods level) and execution branches.

Understanding and acting towards green software ecosystems: After observing software ecosystems, we need to understand what are the factors, from code to design, from technical to humans, impacting their energy consumption and their energy efficiency. My contributions were aimed at providing knowledge about what's happening (to answer the *why?* question), and initial steps on how to react, manage and improve energy consumption.
In Section 3.2.1, we analyzed the impact on energy consumption of around 100 hardware and software features and metrics, in order to understand the relation between them and their impact on green software design.
In Section 3.2.2, we analyzed the impact of design patterns on energy consumption, and proposed a transformation approach to optimize energy consumption of these patterns.
In Section 3.2.3, we explored measuring and analyzing software energy in a horizontal cross-cutting approach, to study software at the feature-level with program slicing.
In Section 3.2.4, we analyzed the impact of code coverage and unit tests on software energy.
And in Section 3.2.5, we explored the role of end users in the energy efficiency equation, with a study on the impact of green feedback on user awareness and behavioral changes in regards to the energy consumption of their software and computing devices.
Finally, after observing and understanding software energy in complex ecosystems, I proposed approaches aimed to help manage and reduce energy.
In Section 3.3.1, we proposed a design tool aimed to help procurers assess the energy efficiency of their data centers and make better green choices in their procurement process.
And in Section 3.3.2, we proposed a proof-of-concept automated framework, using an autonomous control loop and reinforcement learning approach, to manage energy in cyber-physical systems while learning from user preferences and behaviors.

Epilogue: My vision for green software ecosystems is ambitious in its holistic nature and addressing complex systems. I provided many contributions and building blocks towards achieving this vision, in relations with observing and measuring energy, and understanding the factors impacting energy and initial steps on how we can make things better.
In the next section, I outline my research perspectives and the challenges we still face in order to fully implement a holistic vision to manage energy of these complex software ecosystems, and the next adventure going beyond just energy and towards software sustainability.

4.2 Perspectives

In the climax of our green software story, I outlined initial steps on how we can make green software ecosystems better in terms of energy efficiency. However, the story doesn't end here. My contributions on observing and understanding green software ecosystems, open the path towards new endeavors and challenges, which I outline in the following sections.

4.2.1 Mid-Term Perspectives

4.2.1.1 Users Behavioral Changes

Measuring energy consumption of software ecosystems, and understanding the technical or even the human factors in play, is not sufficient to yield important energy savings. Every actor in the system, technical or human, needs to play its role in this green equation. And users are one of the key actors, often leading long-term changes as the business needs and requirements come from them. Pushing end users into adopting energy-aware and sustainable behaviors in regards to using software services, is an important challenge we need to tackle. Users not only need to know about the energy impact of software and its ecosystems, but also understand that they have the keys in hand, and that their actions can lead change. Providing feedback, as seen in the research I did, can play an important role in raising user awareness about the impacts of IT equipment and software. But feedback needs to be adapted and optimized to the specific profile of users, and tailored towards users' workloads. However, identifying which type and modalities of feedback to provide (normative, injunctive, etc.), and to which demographic, is quite challenging. As I defined in Chapter 1, software is an abstract concept that even tech-savvy users find difficult to comprehend and to understand its energy impact. Therefore, in addition to feedback, users need to comprehend this difficult abstract concept of software, and understand its impact (through green feedback, or other approaches such as gamification).
Then, users need to know what actions they can apply, regarding software ecosystems, to reduce their energy impact while keeping their satisfaction level high. From changing software to a more energy-efficient but equivalent one, to questioning the need for a software solution for their problems or needs, users have a wide landscape of potential actions to discover and apply. Therefore, supporting and empowering users is another major challenge.
The next major direction is integrating end users in energy optimizations in green software optimizations, by studying their behaviors and pushing towards a sustained behavioral change. The research in this area is at its infancy. The ambitious fundamental and applied research project, I am leading, the Behave project (funded by ANR) which started in November 2022, aims to study the role of end users in software energy reductions, and drive users' behavioral changes to achieve that goal. We aim to build knowledge around users behaviors

in regard to software energy, through quantitative and qualitative in-depth field studies. We also aim to build an autonomous approach using artificial intelligence, that drives long-term behavioral change through green feedback and recommendation tools. And finally, we aim to disseminate guides and tools for users to identify the energy cost of software and the best alternative solutions. This Behave research project is one major first step in addressing the challenges of integrating end users in the energy-efficiency process.

4.2.1.2 Detecting Software Energy Anomalies

At the technical side, existing approaches often target software or hardware optimizations at runtime, with few recent approaches aiming toward helping developers write energy-efficient and resource-efficient software, through eco-design approaches. Eco-design solutions, such as ecoCode [LGH23] and other similar approaches, scan the source code for bad practices (from a pre-defined list of patterns) and alert developers, and in some cases provides recommended correction. These approaches often require expert knowledge to identify bad practices, and are limited to select software ecosystems and languages (such as smartphone development). But as software is an abstract concept, and energy conception often depends on hardware optimizations and characteristics, it is manually laborious and near-impossible, to keep track of all bad patterns along with corrective measures, for the huge diversity of software and hardware environments.
My perspective on this is to focus on detecting abnormal software energy behaviors, or software energy anomalies. The detection could use software traces (such as logs) along with metrics collected from the different layers of a computing system (hardware, operating system, virtual machine and application server if available, etc.). This detection needs to be fully automated as the evolution of software ecosystems (changes of hardware, software upgrades) might render some patterns void. It also needs to as fine-grained as possible in order to provide system operators and developers sufficient knowledge to identify the source code culprit and apply corrective measures.
Towards contributions to this challenge, I lead a project, started in January 2023, on software energy anomalies (funded by CCLO and UPPA, with a PhD thesis grant), with an aim to detect these energy anomalies through analyzing software traces.

4.2.1.3 Integration into Development Workflows

Measuring energy at runtime and understanding the factors impacting energy efficiency can go only as far as developers act upon these findings, and when these findings take place. I argue we need to proactively build green software by empowering developers and software architects with energy measurements and understanding at the design, development and testing phases. One approach is by integrating fine-grained energy measurements of applications and source in the continuous integration and continuous delivery (CI/CD) pipelines, but also in the testing process, and in the pre-deployment/staging phase. This will help developers rapidly get an energy insight about their applications, and apply corrective measures, tweaks and optimizations to software before being shipped or deployed in production.
What if we can imagine a specific testing procedure dedicated to energy efficiency? For instance, building green tests at different granularity levels, and measuring or estimating the

energy consumption on similar production hardware? What if we can emulate this process, and thus allow a better scaling and deployment of energy estimation in the testing process? This perspective has its share of challenges, from the heterogeneity of hardware and software configuration, to the difficult comparison of raw energy data between different execution contexts, to the possibility of accurately measure or estimate energy of the final application but during development. These challenges are yet to be fully addressed by the research and practitioner community, and will be important challenges in the coming years as developers are still in the dark in the early phases of development about the energy consumption of their applications. Integrating energy into development workflows, in addition to adequate training of the concepts of measuring and eco-design, are key challenges to sustain future green software and ecosystems.

4.2.1.4 Managing Multi-Platform Software Services

With the increasing usage of cloud services and of cyber-physical systems, managing the energy consumption of software across a wide variety of platform is becoming increasingly difficult. My contributions and those of the community in the last decade have allowed us to observe and measure the energy consumption of hardware and software in heterogeneous systems, and across various layers (from source code to devices).
Software and services can now be moved and migrated across devices and servers. Existing approaches and techniques, such as the migration of virtual machines, containers, tasks or micro-services, are quite effective in leveraging the available hardware in local consolidation to minimize resources usage, or to the edge in users' devices. Although effective, these approaches rarely target a wide range of heterogeneous devices and systems, and are limited to specific use cases.
Therefore, managing software services in multi-platforms, require to leverage the tools and approaches in measuring energy at multiple layers and devices, to leverage the understanding of what impacts software energy at these same layers, and to integrate the human actors and business needs. One way to address these challenges is to collect and analyze energy and CO2 data from all the different devices and software services, in order to provide better optimizations and migration strategies, and propose visualization and recommendations to system managers and deciders.
In order to address this perspective, I co-lead a project, in a cross-borders research collaboration with colleagues from UPPA and the University of Zaragoza (UNIZAR) in Spain, starting in October 2023 with a PhD thesis grant, on load shifting and multi-platform services.

4.2.1.5 Training the New Generation of Software Engineers

My contributions towards green software provides approaches and knowledge to observe, understand and act on software energy. I proposed contributions aimed to integrate the human in the energy loop, including end users but also developers, managers and deciders. Developers and software engineers of today and tomorrow need to be aware of the importance of building green software, and must have the relevant tools and knowledge to do so. Software, as an abstract idea, and a cultural product of human societies, is quite heterogeneous and diverse, across countries, usages and needs. Therefore, the only way to ensure greener software is to train the writers and designers of software on the challenges, approaches and solutions to observe, understand and act towards green software.

This means building green courses and curriculums in software engineering, aimed to teach and raise awareness for future software practitioners about green software. It also means, constructing teaching and academic materials tailored to the various needs of software engineers. And finally, it means popularizing scientific results, tools and approaches.
To this end, I created and still maintain a Green IT master course in the 2019/2020 academic year at UPPA. I also created a new more extended master course, called Sustainable Computing in the 2022/2023 academic year at a different computer science degree at UPPA. In addition, I co-created a yearly summer school on green IT and sustainable computing since 2021.
These first contributions helped kickstart a national dynamic in France, where many of the attendees of my summer school were lecturers and professors who went to create green IT courses at various universities. Although the first outcomes are encouraging, many challenges still need to be addressed to make sure that future software engineering have the right tools and knowledge to build green software, and that these training opportunities are sustainable and available to all universities and students, but also current engineers in industry.
Finally, scattered courses here and there only offer an understanding of one or few facets of green software and green computing. I argue that a full curriculum, such as a one- or two-year master degree, on green computing is needed, as it will encompass the complex dimensions of green software, but also energy consumption and the heterogeneity of their ecosystems.

4.2.2 Long-Term Perspectives

On the long-term, the challenges we face cover four aspects that require strong understanding and analysis of software energy. First, achieving the main objective of seamless energy management in a holistic approach, which will be mostly useful in the deployment, usage and maintenance phases of software. Analyzing, understanding and optimizing the other phases are also important in order to fully improve the efficiency of the entire life cycle of software ecosystems. The third aspect is going beyond energy efficiency, and improving software ecosystems in regard to other environmental metrics. Finally, our quest towards sustainable software will lead us to question the definition of sustainability in the realm of software and ideas and the steps to make this definition a reality. In the next sections, I outline my long-term perspectives.

4.2.2.1 Holistic Software Energy Management

In my research, I studied software-only energy approaches, and explored tentative experimentation on a holistic view and management of the energy efficiency of software ecosystems. In particular, the work we did in Section 3.3.2 showcases the importance of modeling contextual information collected from the whole environment, and applying corrective measures (in our case, through a reinforced learning algorithm) to keep the entire system at low energy while respecting the user's quality of service. Our approach is a first step in a more challenging path where the goal is automatically and holistically manage the energy consumption (and potentially the environmental impacts) of complex software ecosystems.
Software ecosystems are complex: along with software systems themselves, many actors are involved with competing quality of service requirements and, in cases, conflicting needs.

Actors vary from users, to other software systems, to hardware devices and non-computing equipment, and to environmental parameters (such as temperature, weather, congestion, price). Software ecosystems also vary a lot, from servers and data center environments, to smart homes and buildings, to smart and personal computing devices, to industrial settings. Building on our work, we envision a crowd sourced and autonomous approach with machine learning to manage heterogeneous software ecosystems in a holistic approach. Crowd sourced for collecting data, including energy, and signaling contextual changes not easily detected by sensors or analysis of sensors' data. Even if I argue that human actors need to be aware and play a role in the energy-efficiency equation, their role may be integrated in an autonomous approach. Machine learning for helping learn efficient autonomous decisions that are efficient and stays efficient in the course of software evolution and contextual changes. A holistic approach is required to manage an environment as a whole, rather than individual system levels or actors.

4.2.2.2 Life Cycle of Green Software Ecosystems

Optimizing software ecosystems at runtime, or at procurement or design times, are only a few episodes in the long process of software development. When observing the life cycle of software development and its ecosystem, the energy impact of the entire process is more than the cost of runtime, even without observing other environmental metrics. The concept of circular economy needs to be applied to software ecosystems too, because building software shares a similar process to building industrial products:

- Design and conception phase where the product's business needs, functional requirements and specifications are identified and modeled,
- Manufacturing phase where the product is built with raw materials resourced, and quality testing,
- Shipping and deployment phase where the product is shipped to its users and end destination, and deployed to its working environment,
- Usage phase where the product is used to its full specifications, including maintenance and updates,
- Decommission phase where the product is recycled or decommissioned and its end-of-life is managed.

Software shares this similar process, and even though most work today focuses on the usage phase, it is important to *observe, understand and act* on the other phases too. The energy cost of these phases is not negligible, and when calculating and accounting the environmental impact (including impact on GHGE, raw materials, pollution, etc.), software, a cultural abstract concept, becomes quite polluting and has an important impact on energy consumption and on the planet's environment.
In this manuscript, I mostly focused on the energy consumption of the usage phase with some endeavors towards the manufacturing and decommission phases. When factoring the entire life cycle of software, the concept of sustainability takes its full place and the urgency of building a sustainable model of software development is quite clear: we need energy-efficient software, but we also need longer use phases, lower environmental impacts of the

other phases, and sustain this workflow of software development. Issues such as licensing, code source availability and maintainability, production of hardware where software will run on, are important challenges for sustainable software. Even challenges such as maintaining a highly skilled workforce is crucial to build high-quality and long-term software and software systems, such as data center environments and cyber-physical future systems.

How can clients and users be aware of these challenges, and integrate sustainability in their software product's requirements, planning and needs? What is the role of managers and software developers and practitioners in making sure sustainability is sought and its process properly applied? How can we properly calculate the energy and environmental impact of software life cycle, including with unforeseen maintenance updates, and end-of-life? Can we make sure that an energy-efficient and sustainable software, once built, stays sustainable throughout its usage phase with the optimal parameters and configuration, and also throughout software and hardware updates and maintenance?

These are a few challenges that needs to be addressed to guarantee an energy-efficient life cycle for software ecosystems.

4.2.2.3 Going Beyond Energy

In this manuscript, I addressed the challenge and perspectives, past, present and future, of green software. I defined green and software, and worked on the energy efficiency of software.

However, software running on hardware is not only consuming energy, but also using earth's resources in multiple dimensions. First, hardware is built from earth's materials which are finite by nature, and also can consume important amount of energy (electricity but also gas, oil, etc.), water, and other resources in the extraction, manufacturing, and shipping processes. Also, software consumes electric energy which is often generated from non-renewable sources (such as coal, oil or gas), or from sources that can pollute our planet for millennia (such as nuclear). Therefore, it is important to address the environmental impact of software regarding these other resources: CO2 emissions, water consumption, rare-earth metals, etc.

In addition, studying the life cycle of software, as presented in the previous perspective, sheds the light on the importance of software development processes and their environmental impacts.

The difficulties and challenges I presented in this manuscript around software energy, are a drop in the ocean of wider complexities when adding the complexities of other environmental metrics. For instance, how can we measure the water consumption impact of a single function or method in an Android application running in an emulator on top of a Linux distribution in a container inside a virtual machine in a remote data center?

We are in the same lack of knowledge and tools as we were in 2010 when I started my research adventure in green software. It took more than a decade to observe and understand software energy, and if we learn from our past experiences, we should be able to build the missing building blocks for these other metrics more quickly in the next decade. Going beyond energy is challenging, and we currently lack tools, approaches and models to observe these metrics consumption in software, let alone understanding their impact on software code and design.

4.2.2.4 Towards Sustainable Software

Sustainable software is not just about energy-efficient software, or software using less CO2 resources or running on earth-friendly mining operations. Sustainable software is also about software that lasts and survives. This leads me to the analysis I made about software sustainability in the introduction of this manuscript: software sustainability is in the realm of culture.
Software that is consuming an abnormally large amount of energy, or requiring updated devices on each update (therefore, consuming more earth metals and having a bigger impact on the environment), is for sure not sustainable.
But we can say the same thing about software ecosystems that are never updated, or updated for a short duration, and therefore pushing users to change devices and software to continue using their software in a secure, safe and comfortable way. For instance, Android OS is updated every year with a major version and every month with security fixes. However, these updates only support a small number of devices, and ultimately most devices have a couple of years of updates, leaving users vulnerable to attacks, but also running deprecated software that will eventually break. This is also not sustainable.
The advent of agile development and its widespread mis-usage in the industry, along with the acceptance of end users about frequent updates and software bugs, has shifted development from releasing well-tested and bug-free software, towards a mindset of *publish early and fix later*. This leads users vulnerable to software breakage, and slow fixes and updates. Buggy software is not sustainable software.
Many of the software we use everyday rely on crucial software components that are developed and maintained by small groups or individuals. If for any reason the developers stopped maintaining their applications, then the latter are effectively dead, and thus not sustainable. Open-source software might have a second life with new developers, but the pool of competent, available, and willing to freely volunteer, developers is extremely small. Out of the 8 billion humans on earth, only 27 million are software developers [dev23], or 0.3%. This is quite notably not sustainable.
This grim observation and the challenges to overcome it, should push our research community to first provide answers around software of the three main questions that Costanza et al. asked to define sustainability [CP95]: 1) **What system** or subsystems or characteristics of systems persist?, 2) For **how long** they are to be sustained?, and 3) **When** do we assess whether the system or subsystem or characteristic has persisted and has been sustained?
The first step towards building proper sustainable software is to answer these three questions. This seemingly simple task is actually herculean, as software is diversified, its cultural heritage is bound to the beliefs and perspectives of its developers and users, its creation and usage often answer a user need and its continuous support is dependent on business profitability, and most problematically, its own existence is bound to the realm of thoughts and ideas.

Epilogue: In a conclusion summary of my perspectives, my mid-term goals are to further develop knowledge around green software ecosystems, and prepare the necessary building blocks for the long-term challenges.

- For instance, with my ambitious ANR Behave project that started in late 2022, I aim to fully address the role of end users in energy efficiency, through feedback and smart

autonomous recommendations. This will bring essential fundamental research knowledge and proof-of-concept prototype that will help integration users in my long-term perspective for a holistic software energy management.

- Furthermore, identifying and detecting software energy anomalies through my research project that started early 2023 (as my mid-term perspective) is an important step in building knowledge and tools that integrate in the life cycle analysis of software energy (my long-term perspective). For instance, as software and hardware evolve and change (such as in software updates, upgrading the virtual machine or container running the application, or while shifting services - as another of my mid-term perspectives with a project starting late 2023), unforeseen energy anomalies might happen in the maintenance phase of software that need to be detected and repaired.

- The integration of energy measurement and analysis into development workflows, as my mid-term perspective, also follows the same path towards better analysis of the lifecycle of green software ecosystems. My current and future collaboration with industry leaders (such as IDE or application servers' developers, or cloud providers) and academic research groups (in particular around software, cyber-physical systems, and data centers) will provide developers with the necessary tools and approaches to analyze the energy efficiency of their software ecosystems throughout its lifecycle.

- My master courses on Green IT (started in 2019/2020 academic year) and on sustainable computing (started in 2022/2023), along with my summer school on Green IT (started in 2021), are essential steps in training the new generation of software engineering towards building the skills needed for observing, understanding and acting towards green software, hardware and ecosystems. Throughout the last years, my contributions helped many educators build new green curriculums. I'll also be part of the Green Digital Skills project (CMA funded by France 2030) that aims to train thousands on greening IT, and there are many other projects and goals I'm currently building and planning to be part of, such as creating a master 2 program in green computing covering the various aspects of green software ecosystems.

- One of my long-term perspectives also aims to study and optimize the more complex environmental metrics in software ecosystems. The dynamics of these metrics (such as GHSE, CO_2 emissions or water consumption) are much more complex to map to the abstract concept of software. However, software ecosystems have an impact on these resources which, in part, are more crucial to our societies' survival than electric energy. Future collaborations are starting to take shape into this larger problem, as I'll be preparing an inter-disciplinary workshop to address these issues in the future.

- This drives me to my last long-term perspective of the concept of sustainability in software. In the introduction of this manuscript, I proposed my definition of *green* software, as the nuances in definitions and in vocabulary might throw us away from real solutions to our sustainability problems. As a common human society and civilizations where computing and software is playing an integral part of our modern way of life, even in remote and secluded areas, sustainable software begs a proper definition. Algorithms and logic existed before electronics as an early concept of software. They sustained wars and empire collapses, and I argue our modern software might as well

be able to sustain the future challenges of our species. The *what* to sustain and *how* to sustain are the major challenges of the next 30 years in software sustainability.

As a final note, the challenges we face as a society to build a sustainable future are important and difficult to address. The UN's 2030 agenda for sustainable development [UNA] states 5 goals and targets: people, planet, prosperity, peace and partnership. It is crucial for ICT environments, including the abstract concept that is software, to integrate these sustainability goals. ICT is our future, and software are their brains. Their sustainability is our human civilization's sustainability.

Bibliography

[acm05] Computing curricula 2005: The overview report, 2005.

[ADE18] ADEME. Les équipements électriques de nos logements, 2018.

[AL16] Kyle Anderson and SangHyun Lee. An empirically grounded model for simulating normative energy use feedback interventions. *Applied Energy*, 173:272–282, jul 2016.

[APTV15] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and Antonio Vetrò. Understanding green software development: A conceptual framework. *IT Professional*, 17(1):44–50, 2015.

[AS21] Monica Anastassiu and Gleison Santos. Resistance to change in software process improvement - an investigation of causes, effects and conducts. In *Proceedings of the XIX Brazilian Symposium on Software Quality*, SBQS '20, New York, NY, USA, 2021. Association for Computing Machinery.

[ASBSVRSQ16] Fabian Astudillo-Salinas, Daniela Barrera-Salamea, Andres Vazquez-Rodas, and Lizandro Solano-Quinde. Minimizing the power consumption in Raspberry Pi to use as a remote WSN gateway. *2016 8th IEEE Latin-American Conference on Communications, LATINCOM 2016*, (November 2018), 2016.

[BCCR14] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 588–598, 2014.

[BE18] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 177:448–463, 2018.

[Bec78] Lawrence J. Becker. Joint effect of feedback and goal setting on performance: A field study of residential energy conservation. *Journal of Applied Psychology*, 63(4):428 – 433, 1978.

[Bec09] Sean Bechhofer. *OWL: Web Ontology Language*, pages 2008–2009. Springer US, Boston, MA, 2009.

[Bel61] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. 'Rand Corporation. Research studies. Princeton University Press, 1961.

[Bel22] Mohammed Chakib Belgaid. *Green coding : an empirical approach to harness the energy consumption ofsoftware services*. Theses, Université de Lille, December 2022.

[BGH+14] David Binkley, Nicolas Gold, M. Harman, Syed Islam, Jens Krinke, and Shin Yoo. ORBS: Language-independent program slicing. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, FSE 2014, pages 109–120, 2014.

[BGN21] Fares Bouaffar, Olivier Le Goaer, and Adel Noureddine. Powdroid: Energy profiling of android applications. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 251–254, 2021.

[BNRS13] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, 2013(92), 2013.

[BRA15] Kathryn Buchanan, Riccardo Russo, and Ben Anderson. The question of energy reduction: The problem(s) with feedback. *Energy Policy*, 77:89–96, 2015.

[BS13] Christian Bunse and Sebastian Stiemer. On the energy consumption of design patterns. In *2nd Workshop EASED@BUIS Energy Aware Software-Engineering and Development*, pages 7–8, 2013.

[BTM00] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, 28(2):83–94, 2000.

[BWN16] Rabih Bashroush, Eoin Woods, and Adel Noureddine. Data Center Energy Demand: What Got Us Here Won't Get Us There. *{IEEE} Software*, 33(2):18–21, 2016.

[CBB+12] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, dec 2012.

[CCC+14] Marco Couto, Tiago Carção, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Detecting Anomalous Energy Consumption in Android Applications. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8771 LNCS, pages 77–91. 2014.

[CCM20] Emile Cadorel, Hélène Coullon, and Jean-Marc Menaud. Online multi-user workflow scheduling algorithm for fairness and energy optimization. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 569–578. IEEE, 2020.

[CFRS17] Maxime Colmant, Pascal Felber, Romain Rouvoy, and Lionel Seinturier. WattsKit: Software-Defined Power Monitoring of Distributed Systems. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 514–523. IEEE, may 2017.

[CGSS14] Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi. Can execution time describe accurately the energy consumption of mobile apps? an experiment in android. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, GREENS 2014, pages 31–37, New York, NY, USA, 2014. ACM.

[CKK+12] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *Software Engineering and Formal Methods*, volume 7504 of *Lecture Notes in Computer Science*, pages 233–247. Springer Berlin Heidelberg, 2012.

[cnr19] Two french climate models consistently predict a pronounced global warming, Sep 2019.

[Com] European Commission. Code of conduct for energy efficiency in data centres. `https://joint-research-centre.ec.europa.eu/scientific-activities-z/energy-efficiency/energy-efficiency-products/code-conduct-ict/code-conduct-energy-efficiency-data-centres_en`.

[CP95] Robert Costanza and Bernard C. Patten. Defining and predicting sustainability. *Ecological Economics*, 15(3):193–196, 1995.

[CRK+18] Maxime Colmant, Romain Rouvoy, Mascha Kurpicz, Anita Sobe, Pascal Felber, and Lionel Seinturier. The next 700 CPU power models. *Journal of Systems and Software*, 144:382–396, oct 2018.

[Cro20] Matteo Croce. Why you should run a 64 bit os on your raspberry pi4, january 2020.

[CSJS14] Nico Castelli, Gunnar Stevens, Timo Jakobi, and Niko Schönau. Switch off the light in the living room, please! -making eco-feedback meaningful through room context information. In *International Conference on Informatics for Environmental Protection*, 2014.

[CSL91] P Constantopoulos, F.C. Schweppe, and R.C. Larson. Estia: A real-time consumer control scheme for space conditioning usage under spot electricity pricing. *Computers & Operations Research*, 18(8):751–765, jan 1991.

[DCC07] Francis M. David, Jeffrey C. Carlyle, and Roy H. Campbell. Context switch overheads for linux on arm platforms. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, New York, NY, USA, 2007. ACM.

[De 01] Andrea De Lucia. Program slicing: Methods and applications. In 1^{st} *IEEE International Workshop on Source Code Analysis and Manipulation*, pages 142–149, Los Alamitos, California, USA, 2001.

[DER05] Hyunsook Do, Sebastian G. Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal*, 10(4):405–435, 2005.

[dev23] Number of software developers worldwide in 2018 to 2024, 2023.

[DFM96] Andrea De Lucia, Anna Rita Fasolino, and Malcolm Munro. Understanding function behaviours through program slicing. In 4^{th} *IEEE Workshop on Program Comprehension*, pages 9–18, March 1996.

[DKP+13] Howard J. Diamond, Thomas R. Karl, Michael A. Palecki, C. Bruce Baker, Jesse E. Bell, Ronald D. Leeper, David R. Easterling, Jay H. Lawrimore, Tilden P. Meyers, Michael R. Helfert, Grant Goodge, and Peter W. Thorne. U.S. Climate Reference Network after One Decade of Operations: Status and Assessment. *Bulletin of the American Meteorological Society*, 94(4):485–498, apr 2013.

[DNPP+17] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. Petra: A software-based tool for estimating the energy profile of android applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 3–6, 2017.

[DP14] A.K. Datta and R. Patel. Cpu scheduling for power/energy management on multicore processors using cache miss and context switch data. *Parallel and Distributed Systems, IEEE Transactions on*, 25(5):1190–1199, May 2014.

[DRS09] Thanh Do, Suhib Rawshdeh, and Weisong Shi. ptop: A process-level power profiling tool. In *in Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09*, 2009.

[DZRL16] Ruilong Deng, Zhaohui Zhang, Ju Ren, and Hao Liang. Indoor temperature control of cost-effective smart buildings via real-time smart grid communications. *2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings*, pages 0–5, 2016.

[EKS03] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. 29(3), 2003. Special issue on ICSM 2001.

[Ell07] Steve Elliot. Environmentally sustainable ict: a critical topic for is research? In *Pacific Asia Conference on Information Systems (PACIS)*, 2007.

[FFLS10] Jon Froehlich, Leah Findlater, James Landay, and Computer Science. The Design of Eco-Feedback Technology. pages 1999–2008, 2010.

[Fou] Green Software Foundation. What is green software? https://greensoftware.foundation/articles/what-is-green-software.

[FPHP13] Najmeh Forouzandehmehr, Samir M. Perlaza, Zhu Han, and H. Vincent Poor. A satisfaction game for heating, ventilation and air conditioning control of smart buildings. *GLOBECOM - IEEE Global Telecommunications Conference*, pages 3164–3169, 2013.

[fra] Frama-C Metrics plugin. http://frama-c.com/metrics.html.

[Fro09] Jon Froehlich. Promoting energy efficient behaviors in the home through feedback: The role of human-computer interaction. *Proc. HCIC Workshop*, 9:0–10, 2009.

[FRS20] G. Fieni, R. Rouvoy, and L. Seinturier. Smartwatts: Self-calibrating software-defined power meter for containers. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 479–488, Los Alamitos, CA, USA, may 2020. IEEE Computer Society.

[FRS21] Guillaume Fieni, Romain Rouvoy, and Lionel Seiturier. Selfwatts: On-the-fly selection of performance events to optimize software-defined power meters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 324–333, 2021.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GHV99] Tony D Givargis, Jörg Henkel, and Frank Vahid. Interface and cache power exploration for core-based embedded system design. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No. 99CH37051)*, pages 270–273. IEEE, 1999.

[GMNH15] Jiaping Gui, Stuart Mcilroy, Meiyappan Nagappan, and William G. J. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 100–110, 2015.

[Gri] The Green Grid. Data centre maturity model. https://www.thegreengrid.org/en/resources/library-and-tools/438-Data-Center-Maturity-Model.

[Hal77] Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[HDBB21] Nicolas Harrand, Thomas Durieux, David Broman, and B. Baudry. The behavioral diversity of java json libraries. *ArXiv*, abs/2104.14323, 2021.

[HGP16] Thi Thao Nguyen Ho, Marco Gribaudo, and Barbara Pernici. Characterizing energy per job in cloud applications. *Electronics*, 5(4):90, 2016.

[Hin16] Abram Hindle. Green software engineering: The curse of methodology. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 46–55, 2016.

[HLHG13] Shuai Hao, Ding Li, William GJ Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *2013 35th international conference on software engineering (ICSE)*, pages 92–101. IEEE, 2013.

[HWRS08] Mats PE Heimdahl, Michael W Whalen, Ajitha Rajan, and Matt Staats. On mc/dc and implementation structure: An empirical study. In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 5–B. IEEE, 2008.

[ict15] Ict carbon footprint, 2015.

[IEA19] IEA. Electricity Information 2019 – Analysis - IEA, 2019.

[ihs17] The internet of things: a movement, not a market, 2017.

[ipc19] Global warming of 1.5 °c. Technical report, Intergovernmental Panel on Climate Change., 2019.

[ISO93] Information technology - Vocabulary. Standard, International Organization for Standardization, November 1993.

[iso94] Iso/iec 7498-1:1994, 1994.

[Jag17] Erik Jagroep. *Green Software Products*. PhD thesis, Utrecht University, 2017.

[JGTC13] Rishee K. Jain, Rimas Gulbinas, John E. Taylor, and Patricia J. Culligan. Can social influence drive energy savings? Detecting the impact of social influence on the energy consumption behavior of networked users exposed to normative eco-feedback. *Energy and Buildings*, 66:119–127, nov 2013.

[JHvH89] W. Fred van Raaij Jeannet H. van Houwelingen. The effect of goal-setting and daily electronic feedback on in-home energy use. *Journal of Consumer Research*, 16(1):98–105, 1989.

[JND19] Michael G. Just, Lauren M. Nichols, and Robert R. Dunn. Human indoor climate preferences approximate specific geographies. *Royal Society Open Science*, 6(3):180695, mar 2019.

[JSBM16] Reyhaneh Jabbarvand, Alireza Sadeghi, Hamid Bagheri, and Sam Malek. Energy-aware test-suite minimization for android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 425–436, 2016.

[KAK+18] Ki-Dong Kang, Mohammad Alian, Daehoon Kim, Jaehyuk Huh, and Nam Sung Kim. Vip: Virtual performance-state for efficient power management of virtual machines. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 237–248, 2018.

[KDC22] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, pages 1–31, 2022.

[KFS14] Beth Karlin, Rebecca Ford, and Cassandra Squiers. Energy feedback technology: a review and taxonomy of products and platforms. *Energy Efficiency*, 7(3):377–399, Jun 2014.

[KG09] C Kelsey and V Gonzalez. Understanding the use and adoption of home energy meters. In *Proceedings of El Congreso Latinoamericano de la Interaccion Humano-Computadora*, pages 64–71, 2009.

[KGH14] Fabian Kaup, Philip Gottschling, and David Hausheer. PowerPi: Measuring and modeling the power consumption of the Raspberry Pi. *Proceedings - Conference on Local Computer Networks, LCN*, pages 236–243, 2014.

[KHN+18] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(2), March 2018.

[KKN20] Kamar Kesrouani, Houssam Kanso, and Adel Noureddine. A Preliminary Study of the Energy Impact of Software in Raspberry Pi devices. In *29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 231–234, Bayonne, France, 2020.

[KNE23] Houssam Kanso, Adel Noureddine, and Ernesto Exposito. Automated power modeling of computing devices: Implementation and use case for raspberry pis. *Sustainable Computing: Informatics and Systems*, 37:100837, 2023.

[KVIY00] Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Wu Ye. Influence of compiler optimizations on system power. In *Proceedings of the 37th Annual Design Automation Conference*, pages 304–307. ACM, 2000.

[lac] Valgrind's Lackey. http://valgrind.org/docs/manual/lk-manual.html.

[LCM+05] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[LDS07] Chuanpeng Li, Chen Ding, and Kai Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, New York, NY, USA, 2007. ACM.

[Lev15] Dissemination Level. Report on impact analysis of green data centre procurement choices. Technical report, H2020 Data centre EURECA Project, 2015.

[LGH23] Olivier Le Goaer and Julien Hertout. Ecocode: A sonarqube plugin to remove energy smells from android projects. In *37th IEEE/ACM International Conference on Automated Software Engineering*, ASE22, New York, NY, USA, 2023. Association for Computing Machinery.

[LHHG13] Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. Calculating source line level energy information for Android applications. *Proceedings of the 2013 International Symposium on Software Testing and Analysis - ISSTA 2013*, page 78, 2013.

[LJS+14] Ding Li, Yuchen Jin, Cagri Sahin, James Clause, and William GJ Halfond. Integrated energy-directed test suite optimization. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 339–350, 2014.

[LKO+21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. Platypus: Software-based power side-channel attacks on x86. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[LOR+18] Yunbo Li, Anne-Cécile Orgerie, Ivan Rodero, Betsegaw Lemma Amersho, Manish Parashar, and Jean-Marc Menaud. End-to-end energy models for edge cloud-based iot platforms: Application to data stream analysis in iot. *Future Generation Computer Systems*, 87:667–678, 2018.

[LPC+19] Juan Miguel Gonzalez López, Edris Pouresmaeil, Claudio A. Cañizares, Kankar Bhattacharya, Abolfazl Mosaddegh, and Bharatkumar V. Solanki. Smart Residential Load Simulator for Energy Management in Smart Grids. *IEEE Transactions on Industrial Electronics*, 66(2):1443–1452, 2019.

[LTH14] Ding Li, Angelica Huyen Tran, and William G.J. J. Halfond. Making Web Applications More Energy Efficient for OLED Smartphones. *Icse'14*, (CONFCODENUMBER):527–538, may 2014.

[LVBBC+14] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. Mining energy-greedy API usage patterns in android apps: An empirical study. In *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*, pages 2–11. Association for Computing Machinery, Inc, may 2014.

[LWG05] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 37–44, May 2005.

[LWP13] Xue Lin, Yanzhi Wang, and Massoud Pedram. An optimal control policy in a mobile cloud computing system based on stochastic data. In *2013 IEEE*

2nd International Conference on Cloud Networking (CloudNet), pages 117–122. IEEE, 2013.

[LYX05] Fan Li, Yiming Yang, and Eric P Xing. From lasso regression to feature vector machine. In *Advances in Neural Information Processing Systems*, pages 779–786, 2005.

[LZCS05] Andreas Litke, Kostas Zotos, Er Chatzigeorgiou, and George Stephanides. Energy consumption analysis of design patterns. In *International Conference on Machine Learning and Software Engineering*, pages 86–90, 2005.

[MA13] Sara S Mahmoud and Imtiaz Ahmad. A green model for sustainable software engineering. *International Journal of Software Engineering and Its Applications*, 7(4):55–74, 2013.

[mac23] Mac pro power consumption and thermal output (btu/h) information, 2023.

[MBZ+16a] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. An empirical study of practitioners' perspectives on green software engineering. *Proceedings - International Conference on Software Engineering*, 14-22-May-:237–248, 2016.

[MBZ+16b] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 237–248, New York, NY, USA, 2016. Association for Computing Machinery.

[McC76] T.J. McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320, Dec 1976.

[Min07] Simon Mingay. Green it: the new industry shock wave. *Gartner RAS Research Note G*, 153703(7), 2007.

[MLVH+02] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35–44. ACM, 2002.

[Moc06] Dennis Mocigemba. Sustainable computing. *Poiesis & Praxis*, 4:163–184, 2006.

[Mol09] Alem Molla. Organizational motivations for green it: Exploring green it matrix and motivation models. In *Pacific Asia Conference on Information Systems (PACIS)*, 2009.

[Mur08] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(01):24–33, jan 2008.

[Mus15] Mark A. Musen. The protégé project: A look back and a look forward. *AI Matters*, 1(4):4–12, jun 2015.

[MZG15] Haroon Malik, Peng Zhao, and Michael Godfrey. Going green: An exploratory analysis of energy-related questions. In *IEEE International Working Conference on Mining Software Repositories*, volume 2015-Augus, pages 418–421. IEEE Computer Society, aug 2015.

[NBRS12a] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A preliminary study of the impact of software engineering on greenit. In *Proceedings of the First International Workshop on Green and Sustainable Software*, GREENS '12, pages 21–27, Piscataway, NJ, USA, 2012. IEEE Press.

[NBRS12b] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A preliminary study of the impact of software engineering on greenit. In *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pages 21–27, 2012.

[NBRS12c] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. Runtime monitoring of software energy hotspots. ASE '12, page 160–169, New York, NY, USA, 2012. Association for Computing Machinery.

[NDKJ11] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[NIB16] Adel Noureddine, Syed Islam, and Rabih Bashroush. Jolinar: Analysing the energy footprint of software applications (demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, page 445–448, New York, NY, USA, 2016. Association for Computing Machinery.

[NLBC23] Adel Noureddine, Martín Diéguez Lodeiro, Noëlle Bru, and Richard Chbeir. The impact of green feedback on users' software usage. *IEEE Transactions on Sustainable Computing*, 8(2):280–292, 2023.

[NM01] N. Noy and Deborah Mcguinness. Ontology development 101: A guide to creating your first ontology. *Knowledge Systems Laboratory*, 32, 01 2001.

[Nou14] Adel Noureddine. *Towards a better understanding of the energy consumption of software systems*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2014.

[Nou22] Adel Noureddine. Powerjoular and joularjx: Multi-platform software power monitoring tools. In *2022 18th International Conference on Intelligent Environments (IE)*, pages 1–4, 2022.

[NR15] Adel Noureddine and Ajitha Rajan. Optimising energy consumption of design patterns. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 623–626, 2015.

[NRS13] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review*, 47(3):42–49, 2013.

[NRS15] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332, 2015.

[OAL14] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys (CSUR)*, 46(4):1–31, 2014.

[OBR+20] Zakaria Ournani, Mohammed Chakib Belgaid, Romain Rouvoy, Pierre Rust, Joel Penhoat, and Lionel Seinturier. Taming energy consumption variations in systems benchmarking. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 36–47, New York, NY, USA, 2020. Association for Computing Machinery.

[OBR+21] Zakaria Ournani, Mohammed Chakib Belgaid, Romain Rouvoy, Pierre Rust, and Joël Penhoat. Evaluating the impact of java virtual machines on energy consumption. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA, 2021. Association for Computing Machinery.

[oED] World Commission on Environment and Development. Our common future. `https://sustainabledevelopment.un.org/content/documents/5987our-common-future.pdf`.

[OH98] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(1):44–51, 1998.

[PB18] Antonio Paone and Jean Philippe Bacher. The impact of building occupant behavior on energy efficiency and methods to influence it: A review of the state of the art. *Energies*, 11(4), 2018.

[PC17a] Gustavo Pinto and Fernando Castor. Energy Efficiency: A New Concern For Application Software Developers. *Communications of the ACM*, 60(12):68–75, nov 2017.

[PC17b] Gustavo Pinto and Fernando Castor. Energy efficiency: A new concern for application software developers. *Commun. ACM*, 60(12):68–75, November 2017.

[PCL14a] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.

[PCL14b] Gustavo Pinto, Fernando Castor, and Yu David Liu. Understanding energy behaviors of thread management constructs. *ACM SIGPLAN Notices*, 49(10):345–360, oct 2014.

[PCR+21] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Ranking programming languages by energy efficiency. *Science of Computer Programming*, 205:102609, 2021.

[per] Perf tool. https://perf.wiki.kernel.org/.

[PFL16] Giuseppe Procaccianti, Héctor Fernández, and Patricia Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198, 2016.

[PHAH16] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan. What Do Programmers Know about Software Energy Consumption? *IEEE Software*, 33(3):83–89, 2016.

[PHB15a] James Pallister, Simon J Hollis, and Jeremy Bennett. Identifying compiler options to minimize energy consumption for embedded platforms. *The Computer Journal*, 58(1):95–109, 2015.

[PHB15b] James Pallister, Simon J. Hollis, and Jeremy Bennett. Identifying compiler options to minimize energy consumption for embedded platforms. *The Computer Journal*, 58(1):95–109, 2015.

[PHZ12] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Where is the energy spent inside my app? In *Proceedings of the 7th ACM european conference on Computer Systems - EuroSys '12*, page 29, 2012.

[PLCL16] Gustavo Pinto, Kenan Liu, Fernando Castor, and Yu David Liu. A comprehensive study on the energy efficiency of java's thread-safe collections. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 20–31. IEEE, 2016.

[PLGOC09] J.A. Poovey, M Levy, S Gal-On, and T Conte. A benchmark characterization of the eembc benchmark suite. *Micro, IEEE*, PP(99):1–1, 2009.

[POB08] James Pierce, William Odom, and Eli Blevis. Energy aware dwelling: A critical survey of interaction design for eco-visualizations. In *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*, OZCHI '08, pages 1–8, New York, NY, USA, 2008. ACM.

[pol] Polly LLVM library. http://polly.llvm.org/index.html.

[Raj06] Ajitha Rajan. Coverage metrics to measure adequacy of black-box test suites. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 335–338. IEEE, 2006.

[RJ98] Jeffry T Russell and Margarida F Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference on*, pages 328–333. IEEE, 1998.

[RNA+12] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27, 2012.

[RNS16] Ajitha Rajan, Adel Noureddine, and Panagiotis Stratis. A study on the influence of software and hardware features on program energy. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, New York, NY, USA, 2016. Association for Computing Machinery.

[RWB+17] Basireddy Karunakar Reddy, Matthew J Walker, Domenico Balsamo, Stephan Diestelhorst, Bashir M Al-Hashimi, and Geoff V Merrett. Empirical cpu power modelling and estimation in the gem5 simulator. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–8. IEEE, 2017.

[SA20] Anders SG Andrae. New perspectives on internet electricity use in 2030. *Engineering and Applied Science Letter*, 3(2):19–31, 2020.

[SAL18] Claudio Scordino, Luca Abeni, and Juri Lelli. Energy-aware real-time scheduling in the linux kernel. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 601–608, 2018.

[SCG+12] C. Sahin, F. Cayci, I.L.M. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh. Initial explorations on design pattern energy usage. In *1st International Workshop on Green and Sustainable Software, GREENS'12*, pages 55–61, June 2012.

[SG13] MD Samrajesh and NP Gopalan. Component based energy aware multitenant application in software as-a service. In *2013 15th International Conference on Advanced Computing Technologies (ICACT)*, pages 1–5. IEEE, 2013.

[SO06] Nija Shi and Ronald A. Olsson. Reverse engineering of design patterns from java source code. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 123–134, 2006.

[SRS+12] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, vLi Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen mei W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. Technical report, IMPACT Technical Report, IMPACT-12-01, University of Illinois, at Urbana-Champaign, 03 2012.

[SS03] J.M. Smith and D. Stotts. Spqr: flexible automated design pattern extraction from source code. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, pages 215–224, 2003.

[SWRH10] Matt Staats, Michael Whalen, Ajitha Rajan, and Mats Heimdahl. Coverage metrics for requirements-based testing: Evaluation of effectiveness. 2010.

[TCSH06] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T. Halkidis. Design pattern detection using similarity scoring. *IEEE Transactions on Software Engineering*, 32(11):896–909, 2006.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[TMWL96] V. Tiwari, S. Malik, A. Wolfe, and M.T.-C. Lee. Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, pages 326–328, Jan 1996.

[TPS+15] Alain Tchana, Noel De Palma, Ibrahim Safieddine, Daniel Hagimont, Bruno Diot, and Nicolas Vuillerme. Software consolidation as an efficient energy and cost saving solution for a saas/paas cloud model. In *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings 21*, pages 305–316. Springer, 2015.

[TRLJ01] T.K. Tan, A. Raghunathan, G. Lakshminarayana, and N.K. Jha. High-level software energy macro-modeling. In *Design Automation Conference, 2001. Proceedings*, pages 605–610, 2001.

[TX12] Anupam A. Thatte and Le Xie. Towards a Unified Operational Value Index of Energy Storage in Smart Grid Environment. *IEEE Transactions on Smart Grid*, 3(3):1418–1426, sep 2012.

[UNA] Department of Economic United Nations and Social Affairs. Transforming our world: the 2030 agenda for sustainable development. `https://sdgs.un.org/2030agenda`.

[VKI+00] N. Vijaykrishnan, M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pages 95–106, June 2000.

[VOWD12] Iana Vassileva, Monica Odlare, Fredrik Wallin, and Erik Dahlquist. The impact of consumers' feedback preferences on domestic electricity consumption. *Applied Energy*, 93:575–582, may 2012.

[VVHC+10] Willem Vereecken, Ward Van Heddeghem, Didier Colle, Mario Pickavet, and Piet Demeester. Overall ict footprint and green communication technologies. In *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 1–6. IEEE, 2010.

[W+08] Molly Webb et al. Smart 2020: Enabling the low carbon economy in the information age. *The Climate Group. London*, 2008.

[wat] Watts Up? power meter. https://www.wattsupmeters.com.

[Wei81] Mark Weiser. Program slicing. In 5^{th} *International Conference on Software Engineering*, pages 439–449, San Diego, CA, March 1981.

[WGHT99] W. E. Wong, S. S. Gokhale, J. R. Horgan, and K. S. Trivedi. Locating program features using execution slices. In *Application-Specific Systems and Software Engineering and Technology*, pages 194–203, 1999.

[WN07] G. Wood and M. Newborough. Energy-use information transfer for intelligent homes: Enabling energy conservation with central and local displays. *Energy and Buildings*, 39(4):495–503, apr 2007.

[WNLMM18] Yewan Wang, David Nörtershäuser, Stéphane Le Masson, and Jean-Marc Menaud. Potential effects on server power metering and modeling. *Wireless Networks*, pages 1–8, 2018.

[WW16] Huaming Wu and Katinka Wolter. Analysis of the energy-performance tradeoff for delayed mobile offloading. *EAI Endorsed Transactions on Energy Web*, 3(10):250–258, 2016.

[XNLH21] Peng Xiao, Zhenyu Ni, Dongbo Liu, and Zhigang Hu. Improving the energy-efficiency of virtual machines by i/o compensation. *The Journal of Supercomputing*, 77:11135–11159, 2021.

[YBD01] Y. Yu, K. Beyls, and E.H. D'Hollander. Visualizing the impact of the cache on program execution. In *Proceedings Fifth International Conference on Information Visualisation*, pages 336–341, 2001.

[Yee] Alexander Yee. y-cruncher, a multi-threaded pi-program. `http://www.numberworld.org/y-cruncher/`.

[YXX+19] Liang Yu, Weiwei Xie, Di Xie, Yulong Zou, Senior Member, Dengyin Zhang, Zhixin Sun, Linghua Zhang, Yue Zhang, and Tao Jiang. Deep Reinforcement Learning for Smart Home Energy Management. *IEEE INTERNET OF THINGS JOURNAL*, 7:1, 2019.

[ZLSO16] Dong Zhang, Shuhui Li, Min Sun, and Zheng O'Neill. An Optimal and Learning-Based Demand Response and Home Energy Management System. *IEEE Transactions on Smart Grid*, 7(4):1790–1801, jul 2016.

[ZWC+09] Jiucai Zhang, Dalei Wu, Song Ci, Haohong Wang, and Aggelos K Katsaggelos. Power-aware mobile multimedia: a survey. *J. Commun.*, 4(9):600–613, 2009.