Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Energetica e Nucleare

EIT InnoEnergy Master in Environomical Pathways for Sustainable Energy Systems (SELECT)

Politecnico di Torino

Tesi di Laurea Magistrale

Software Sustainability

Assessing the Environmental Impact of the Software Life cycle

Supervisor:
Prof. Massimo Santarelli

Candidate:
Sohel Abbas Chungikar

October, 2024
Academic year: 2023/2024

This page has been intentionally left blank.

Acknowledgment

I would like to express my deepest gratitude to everyone who has supported me throughout my academic journey, culminating in the completion of this thesis.

First and foremost, I am profoundly grateful to Prof. Massimo Santarelli for his invaluable guidance during this thesis. His lectures and research have been a constant source of inspiration, motivating me to delve deeper into the field of energy engineering and pursue a career in this dynamic sector.

A special thanks to Avantika Shastri and Bithika Chakravarty, my managers at Eaton, for providing me with the internship that laid the foundation for this thesis and for their unwavering guidance, support, and time throughout the internship. Their mentorship has been instrumental in shaping both my professional and academic growth.

I would also like to extend my heartfelt thanks to all the professors, colleagues, and friends I have made over the past two years in the Master's program. This journey has been incredibly enriching, filled with unforgettable experiences. My time at EIT InnoEnergy, UPC, and PoliTo has been extraordinary, and I will cherish these memories for a lifetime.

A special note of thanks to Jordi Cusido Roura, my mentor during my Bachelor's thesis. His support and belief in me, even when I had little experience, opened up the world of energy and sustainability to me and ignited my passion for addressing the challenges of climate change. I am deeply grateful for the opportunity he provided.

Finally, my deepest gratitude goes to my parents for their unwavering support, encouragement, and belief in me from the very beginning. Their constant love and motivation have made this journey possible, and I look forward with optimism and excitement to all that the future holds, knowing that their support will always be with me.

This page has been intentionally left blank.

Abstract

With the rapid rise of laptops, smartphones, and artificial intelligence, there has been limited attention paid to their environmental impact. This thesis presents a framework developed after extensive research into the green software domain, aimed at helping corporations with emissions accounting and reporting, while adhering to the Corporate Sustainability Reporting Directive (CSRD) and similar policies. The core metric adopted within the framework is Software Carbon Intensity (SCI), which is measured through system utilization parameters. These parameters are calibrated using two benchmarking tools, Running Average Power Limit (RAPL) and Performance Counter Monitor (PCM).

Two case studies were conducted to demonstrate the framework's practical application. In the first case, representing the software development stage, emissions of approximately 330 gCO_2eq were generated during typical office hours from 09:00 to 18:30, which extrapolates to 9.87 $kgCO_2eq$ per developer per sprint. The second case study, focused on software usage, recorded emissions ranging from 0.054 gCO_2eq to 2.73 gCO_2eq per run, depending on the location of execution and the method of carbon intensity data aggregation.

A reporting method using candlestick charts is also introduced, providing companies a useful tool for Scope emissions accounting.

Keywords:

Software – Green Software – Sustainability – IT – CSRD – Corporate Sustainability – Software Engineering – Software Sustainability

Contents

List of Figures

List of Tables

Listings

List of Acronyms

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
BCG	Boston Consulting Group
CAGR	Compound Annual Growth Rate
CAPEX	Capital Expenditure
CfD	Contracts for Difference
CPU	Central Processing Unit
CSV	Comma-Separated Values
CVML	Computer Vision and Machine Learning
ECL	Energy Consumption Library
EIO	Economic Input-Output
EU	European Union
GHG	Greenhouse Gas
GPU	Graphics Processing Unit
GSF	Green Software Foundation
GW	Gigawatt
ICT	Information and communications technology
IF	Impact Framework
IoT	Internet of Things
IPCC	Intergovernmental Panel on Climate Change
IT	Information Technology
kW	Kilowatt
kWh	Kilowatt-hour
LCA	Life Cycle Assessment
LLM	Large Language Model
ML	Machine Learning
MW	Megawatt
OCR	Optical Character Recognition
OPEX	Operational Expenditure
PAIA	Product Attribute to Impact Algorithm
PAPI	Performance Application Programming Interface
PCM	Performance Counter Monitor
PPA	Power Purchase Agreement
PUE	Power Usage Effectiveness
RAPL	Running Average Power Limit
SDK	Software Development Kit
SQL	Structured Query Language
TDP	Thermal Design Power
TEADS	Thermal Design Power Curve Estimation
WMI	Windows Management Instrumentation
WUE	Water Usage Effectiveness

1

Introduction

1.1 Climate Change

Anthropogenic activities have dramatically altered the planet, affecting not only humanity but also all natural processes. There has been steady growth since the beginning of human civilization, but the Industrial Revolution has significantly amplified this growth. While technological advancements have improved the quality of life, they have had a huge impact on the environment, often to the detriment of all living beings. These activities have disrupted processes that took millions, if not billions, of years to balance and establish.

Technological advancements have come at a significant cost to the well-being of the planet. One of the most observable changes is the rise of greenhouse gases (GHG). These gases trap heat energy that would have been radiated into space, thereby heating the entire planet. This unnatural heating has led to severe disruptions in weather patterns, a phenomenon commonly referred to as *Global Warming* or *Climate Change*. The disruption in weather patterns has proven detrimental to all living species, as they cannot adapt quickly enough to the rapid changes in their ecosystems. These changes also have severe impacts on human beings, and there is an urgent need to correct our past mistakes.

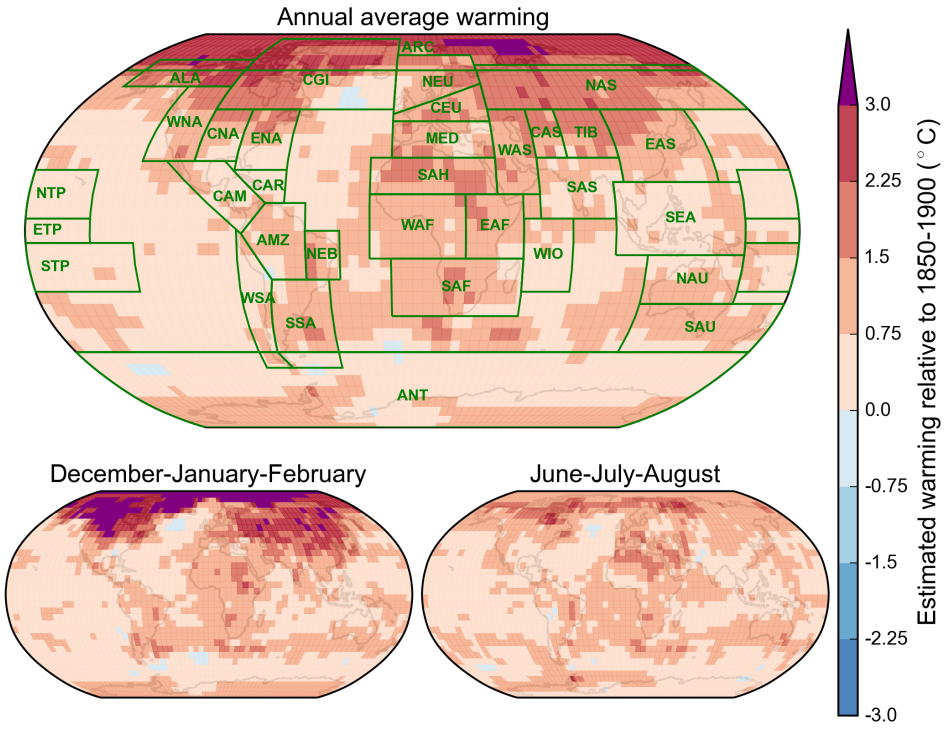

Figure 1: Regional warming in the decade 2006–2015 relative to pre-industrial period [1]

In response, efforts to mitigate these effects are being undertaken by individuals, industries, and

governments alike. A relatively new domain contributing to these efforts is the software industry, which has be further explored in this thesis.

1.2 Global Trends

In addition to the environmental aspects, it is crucial to consider the economic and technical dimensions. These factors provide essential context for evaluating the applicability and potential success of any work being undertaken. This thesis looks at a few indicators that are relevant to the topic of analysis.

Software Sector

One of the largest multi-billion-dollar industries that is expected to grow rapidly over the coming years is the software industry. In 2024, it was estimated that the software industry accounted for more than 700 billion dollars in revenue [2]. The sector shows no signs of slowing down and is expected to have a compound annual growth rate (CAGR) of about 11.9% between the years 2023 and 2030 [3]. These substantial figures highlight the increasing demand the software industry faces and its relevance in the coming years. Hence, this thesis aims to explore and improve the performance of this domain, especially pertaining to the environmental aspects.

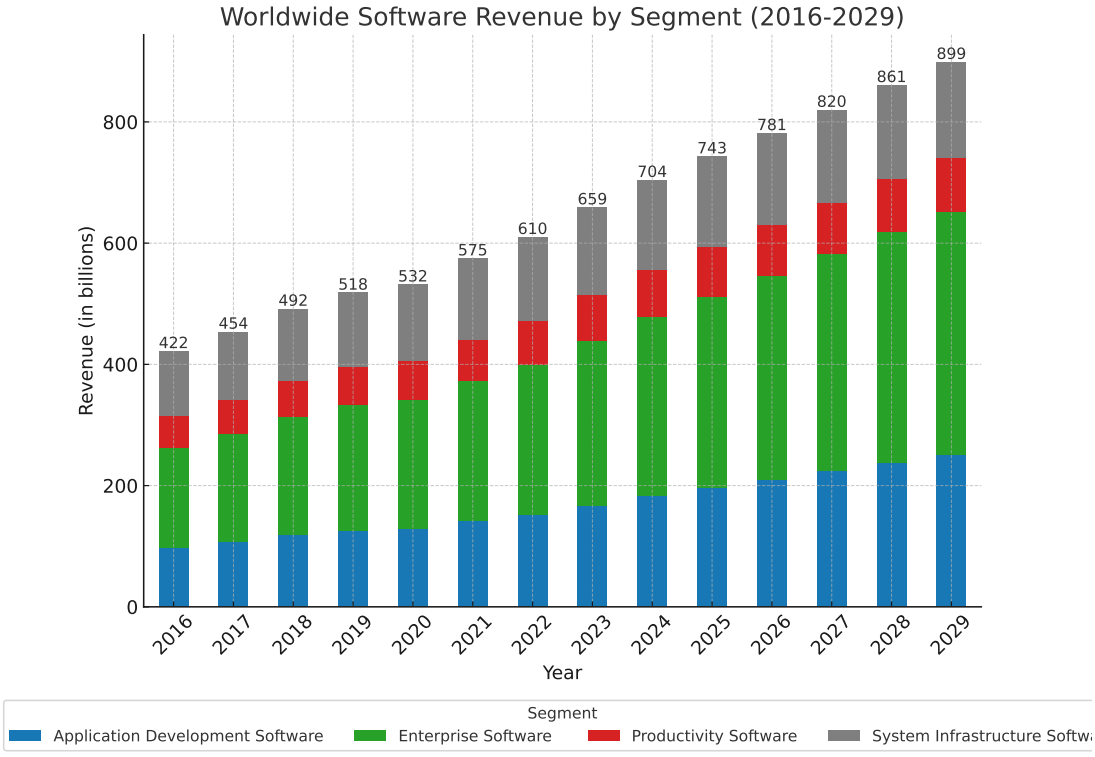

Figure 2: Worldwide Software Revenue [2]

Artificial Intelligence

An emerging and fast-growing sector related to the software industry is Artificial Intelligence (AI). It is an umbrella term that includes technologies such as Machine Learning (ML) and Large Language Models (LLMs). AI relies heavily on processing vast amounts of data to train models, which leads to the resource intensity of energy, water, and computing hardware. For instance, it is estimated that each conversation with ChatGPT-3, consisting of 10 to 50 replies, consumes approximately 500 ml of water [4]. The computational power required for training these models can result in significant energy consumption, leading to substantial CO_2eq emissions.

Additionally, training AI models requires resources such as powerful computational hardware and immense amounts of electricity for the functioning and maintenance of this hardware. This results in the generation of hundreds of tons of CO_2eq emissions, which are bound to increase in the future with advancements in the complexity of these AI models. As presented in Table 1, the carbon footprint as well as the electricity usage of these models is immense, at times larger than the annual footprints of entire cities.

Table 1: Energy and Carbon Expenditure of LLMs [5]

Model name	Number of parameters	Datacenter PUE	Carbon intensity of grid used	Energy consumption	CO_2eq emissions	CO_2eq emissions × PUE
GPT-3	175B	1.1	429 gCO_2eq/kWh	1,287 MWh	*502 tonnes*	552 tonnes
Gopher	280B	1.08	330 gCO_2eq/kWh	*1,066 MWh*	*352 tonnes*	380 tonnes
OPT	175B	*1.09*	*231gCO_2eq/kWh*	*324 MWh*	70 tonnes	*76.3 tonnes*
BLOOM	176B	1.2	57 gCO_2eq/kWh	433 MWh	25 tonnes	30 tonnes

Due to their extremely large footprint, minor tweaks to the codebase or training process of AI models that reduce energy usage could have significant downstream impacts, particularly in reducing emissions. These improvements could help us leverage AI to solve complex and pressing issues without causing significant negative impacts.

1.3 Software

Software refers to any set of instructions provided to a device to execute specific tasks. Unlike physical components, software is intangible, existing as a collection of binary code—1's and 0's—that a computer interprets and executes. It is ubiquitous in modern life, ranging from simple applications, such as turning on a light when motion is detected, to more complex systems, such as the autopilot functionalities that control aircraft. Software is also found in critical applications like pacemakers, where it monitors and regulates heartbeats, as well as in more recreational contexts, such as video games. Software refers to any set of instructions given to a device for it to execute tasks. Unlike traditional

products, it does not have a physical form but consists of instructions that are ultimately executed by the computer in binary format. It has become an important and necessary part of modern life, finding prominence in the simplest of applications, such as switching on a light, to highly complex tasks, such as controlling an entire aircraft through autopilot systems. It can also be found in the most critical areas, such as the operation of a pacemaker.

Given the wide range of uses of software and the significant role it plays in our daily lives, it presents a challenge in creating a universal framework that can assess the societal as well as environmental implications of its development and usage. Nonetheless, it is a pressing issue that needs to be addressed before it leads to major consequences that may not be reversible.

1.4 Scope of the Thesis

The field of Software Sustainability is an emerging and rapidly evolving area within IT, presenting both challenges and opportunities for firms looking to integrate sustainability into their software development processes. Given the nascent nature of this domain, there is a pressing need to establish a robust framework and develop tools that enable organizations of all sizes to begin their journey toward carbon accounting in relation to software development and software products.

The objective of this thesis is to research the current state of the green software domain. Building upon this knowledge of industrial practices and available tools and metrics, a framework is designed that is simple yet open to continuous improvements. The framework is designed to be easily implemented in large corporations and to drive acceptability in the sector. Focus is given to data accuracy while also considering the economic feasibility of implementation.

The results and developed framework in this thesis should provide a basis for companies to adopt and tailor the developed tools to meet their specific requirements with minimal investment of time and financial resources. The framework is intended to be a bridge between software development teams and sustainability teams in a corporation, serving as a common point for the exchange of information and improvements. Additionally, some tests will be carried out to demonstrate the feasibility of the proposed system. The end goal of the thesis is to ingrain sustainability practices into the software industry and help corporations meet their sustainability goals.

2

Why Software Sustainability?

2.1 What is Software Sustainability?

Sustainability is a broad term that has found frequent use in our daily lives and typically refers to the ability to exist or function over an extended period with limited changes. Therefore, when we refer to the term *software sustainability*, it can be interpreted in multiple ways depending on the perspective of the individual.

In the established field of software engineering, the term "sustainability" often refers to technological, economic, and social sustainability [6] [7]. Although these interpretations are widely accepted, as seen in the vast amount of literature on the topic, very few sources address the environmental aspect of sustainability. However, improvements in other aspects of sustainability do tend to have positive implications for the environmental dimension as well.

Technical sustainability refers to reducing software errors and extending its usability and applicability. In contrast, social sustainability involves ensuring that the software meets stakeholder demands without causing adverse social impacts.

Environmental sustainability focuses on the ecological impact of software, considering that software consumes energy not only during its operation but also during its development and deployment phases. Although environmental sustainability may not directly link with other interpretations of sustainability, it often positively influences other aspects.

To address the potential confusion that might arise when referring to the environmental sustainability of software, terms such as *Green Software*, *Green IT*, and *Green ICT* can be used. These terms not only effectively communicate the goals but also provide access to resources for further development and research.

2.2 GHG Protocol

The Greenhouse Gas (GHG) Protocol is a comprehensive and globally standardized framework developed for measuring and managing greenhouse gas (GHG) emissions across private and public sector operations, value chains, and mitigation activities. This protocol has emerged as a widely adopted benchmark in corporate sustainability, facilitating the accurate accounting and reporting of all emissions associated with business operations [8].

Under the GHG Protocol, emissions are classified into three major "scopes," which are determined by the origin of the emissions and the degree of control or influence that a firm exerts over them. This classification not only enables firms to identify where the most significant sources of emissions—often

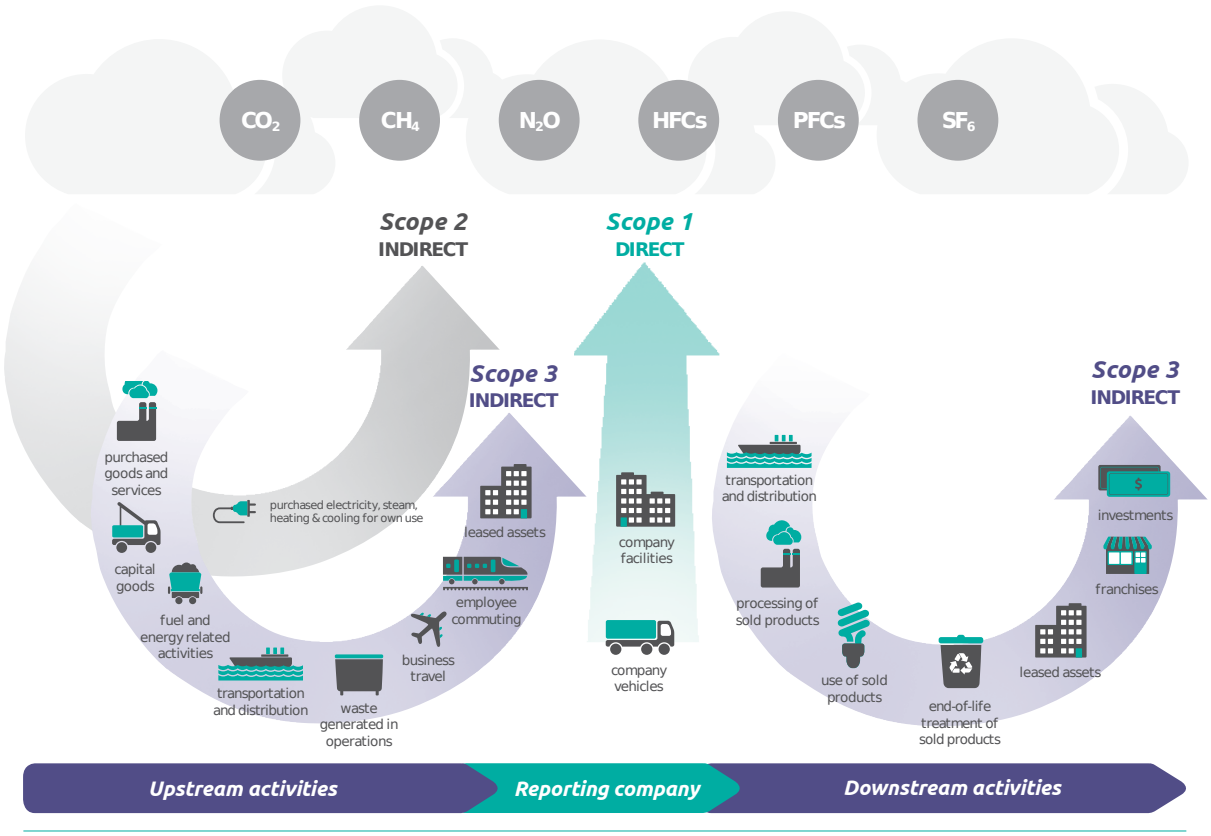

Figure 3: Visual representation of Scope emissions (GHG protocol)[8]

referred to as hotspots—are located but also provides them with the insights necessary to develop effective strategies for managing and reducing emissions. These strategies are designed to minimize the impact on the firm's economic activities. The scope emissions are detailed further below and visually represented in Figure 3.

- Scope 1

  Scope 1 includes all emissions directly resulting from a firm's operations. These emissions are the most controllable and reducible by the firm, yet they typically represent a smaller portion of the firm's overall emissions. Examples include emissions from manufacturing processes, operating a diesel generator for backup power, and leaks of gases or refrigerants.

- Scope 2

  Scope 2 encompasses emissions that result directly from the firm's operations, although the sources of these emissions are not located at the firm's operational site or within its assets. These emissions typically arise from energy sources, such as electricity, steam, heating, or cooling, that the firm purchases. Because these emissions are the result of deliberate choices made by the firm, it has a significant degree of control over them. Examples include emissions generated by the electricity purchased to power office operations or the heating procured to maintain higher

temperatures necessary for industrial reactions.

- Scope 3

  Scope 3 includes emissions associated with both the upstream and downstream elements of a company's supply chain, as well as the emissions generated from the use of the company's products and services. Firms have minimal control over these emissions, making them challenging to reduce. Scope 3 emissions are the hardest to quantify and often constitute the largest share of a company's total reported emissions. Due to the complexity and extensive nature of quantifying these emissions, many assumptions and methodologies based on historical data and economic activities must be employed. Examples include emissions linked to the mining of raw materials used by the firm or the emissions resulting from the use of transmission systems produced by the company.

- Scope 4

  Scope 4 emissions are a relatively new concept and has not yet been officially integrated into the GHG protocol. It quintessentially, refers the the avoided emissions due to improvements in technology or actions being made by the firm under scrutiny. Although not widespread and common knowledge, it is highly relevant in sectors such as that of Renewable Energy as a lot of advancements in this sector replace older technologies that have much higher emissions. Scope 4 emissions can be associated with the "handprint" of the product, which enhances it's value proposition not only from an environmental aspect but also a financial aspect due to the introduction of mechanisms such as *Carbon Taxation* or *Cap and Trade*, such as the EU ETS. This concept is especially applicable to the software/IT industry through the notion of *Sustainability **by** IT*, where IT technologies contribute to improving various sustainability indicators in non IT sectors, ultimately having a positive impact on the world. However, for the purposes of this paper, we focus on *Sustainability **in** IT*, where we introduce metrics and measures aimed at making the IT industry more environmentally friendly and sustainable.

2.3 Emissions of a Software Company

The GHG protocol, first published in the early 2000s, did not directly address the operations of companies whose product lines were predominantly digital. Given the rise of such companies and the widespread use of digital services and technologies, efforts have been made to adapt the GHG protocol for these companies. An extensive breakdown of emission categorization for a software company is provided in Table 7, with a visual representation in Figure 4 [9].

As can be observed in Figure 4, many non-technical aspects are considered, which may be required for environmental reporting by software companies. These ancillary functions are essential to the

proper functioning of a company and the development of its products and, therefore, must be properly managed and reported in ESG disclosures. These observations refer specifically to companies whose sole output is software products and services [9].

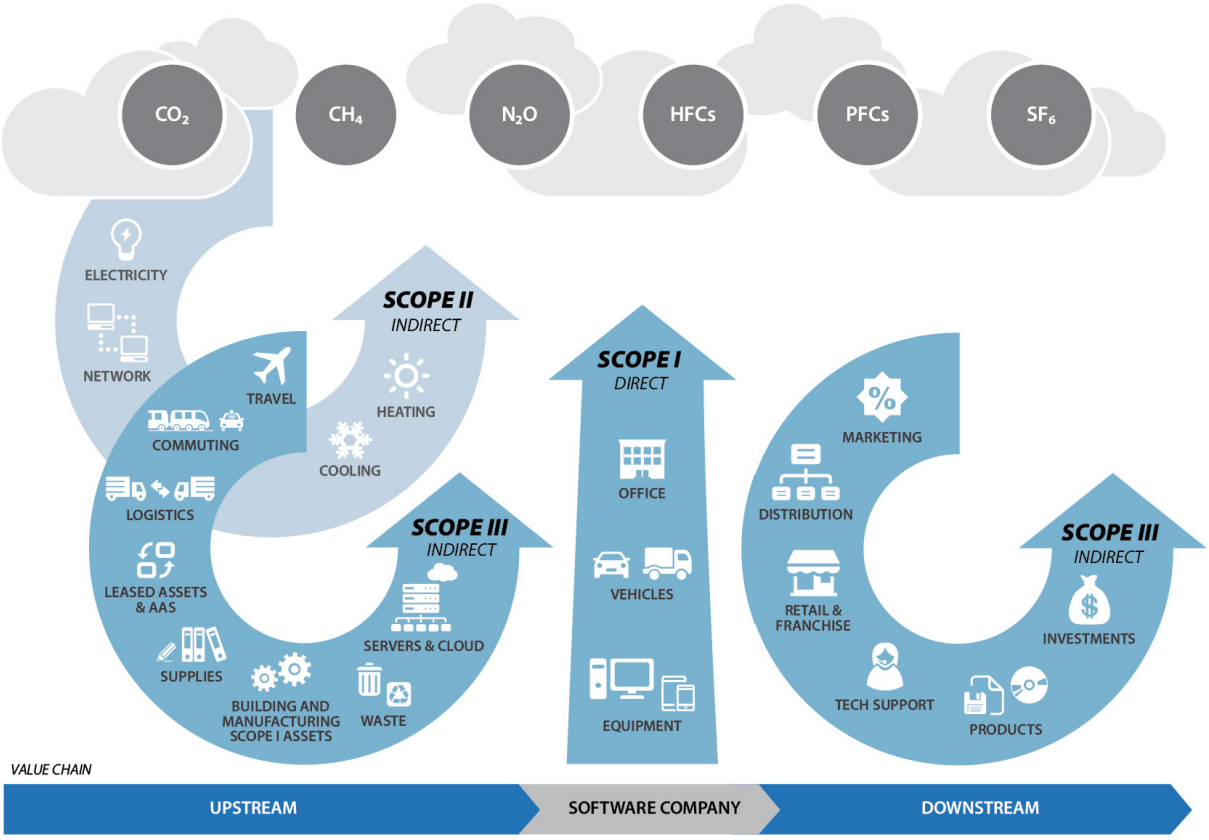

Figure 4: Scope emissions of a Software Company [9]

2.4 Corporate Sustainability Reporting Directive (CSRD)

The Corporate Sustainability Reporting Directive (CSRD) is a policy being implemented in the European Union that is applicable for predominantly large companies and other listed companies. It is an expansion to their previously enforced non-financial reporting requirements that mandated certain companies disclose the societal and environmental impact of their activities [10].

The CSRD is an important way for the investors, customers, and other stakeholders to evaluate the performance of the company in aspects other than its financial aspect, with it serving as an important aspect for the development and advancement of it *European Green Deal*. Making a crucial aspect for companies to have accurate and high quality data in their reporting as they might be subject to audits. These reporting also apply to companies that might have software services as a part of their product line or software is embedded into the hardware the company produces, making good accounting for the software aspect necessary.

2.5 Role of Software Carbon Footprint

Although not apparent at first glance, measuring and quantifying the environmental footprint could have significant implications for the functioning of a company and the value proposition of its products and services. With the rise of eco-conscious consumers and policies such as the CSRD and CSDDD mandating proper accounting of emissions, ESG reporting could lead to major changes that affect the financial operations of a company. In certain markets, firms may also be eligible for incentives and access to funds due to their greener operations. Without measurement, there can be no improvement; therefore, it is crucial to quantify the footprints to establish a baseline for improvements and technological developments.

Additionally, once proper and reliable accounting methods for the software footprint are implemented, its environmental "handprint" could also be calculated. This has the added benefit of helping companies achieve their sustainability goals while simultaneously improving the value proposition of their products. This could positively impact the company, driving growth and innovation through access to funding.

2.6 Green Software Foundation (GSF)

Given the infancy of this field, significant work is needed to bring together collaborative efforts and collective thinking. This not only drives innovation but also reduces opposition for adopters of certain concepts when implementing new ideas.

The Green Software Foundation (GSF) is leading efforts in this domain. They have developed tools and methodologies that cover major aspects of *Green Software*, and their work has been published as an ISO standard, which forms the backbone of the methodology developed and implemented in this thesis.

The goal of GSF is to create a trustworthy ecosystem and influence the software development domain to minimize associated emissions. The steering committee of the GSF comprises well-known and respected firms in the industry, including Accenture, Avanade, BCG X, GitHub, Intel, Microsoft, NTT Data, Siemens, and UBS. An overview of the GSF manifesto is shown in Figure 32.

3

Industrial Metrics & Methods

Misura ciò che è misurabile e rendi misurabile ciò che non lo è
– Galileo Galilei

In any scientific field, a comprehensive understanding of measurement techniques and metrics is crucial for effective communication of ideas and results. In alignment with this philosophy, an overview of various established metrics is presented below. Although these metrics may not be directly applied in the present framework, they serve as an essential entry point into the level of detail and thought process required when applying environmental sustainability concepts to the software domain.

3.1 Hardware Efficiency

Hardware efficiency measures how effectively a particular piece of hardware is utilized for any task. In many instances, hardware is not fully utilized, leaving significant room for improved performance. Considering that the manufacturing of computing hardware is one of the most complex scalable technologies we currently possess, involving elements from almost the entire periodic table sourced from all corners of the globe, the carbon emissions associated with it are substantial. Therefore, it is imperative to utilize this hardware to its maximum extent.

Given the rapid pace of innovation and development in the computing hardware domain, hardware becomes outdated quickly. For instance, a typical data center replaces all of its hardware every 3 to 5 years [11], while consumer-grade hardware is generally designed to last between 2 to 5 years. The concept of Hardware Efficiency is closed linked with embodied emissions and will be presented in detail in subsequent sections.

Figure 5: Comparative Hardware Utilisation[12]

As illustrated in Figure 5, five servers are operating at only 20% of their capacity. The same tasks could effectively be handled by a single machine running at 100% capacity. This consolidation not only reduces the capital expenditure (CAPEX) of the system but also decreases the emissions associated with the manufacturing of 4 additional servers. Moreover, with all tasks being performed by one machine, it can be more efficiently and economically managed thermally, enhancing its operational efficiency from both financial and environmental perspectives.

Additionally, since most modern CPUs are multi-cored, there is a notable impact on how cores are allocated, switched, and how this affects efficiency and execution timings [13][14]. This brings us to an intriguing challenge: maintaining 100% system utilization may not always be the most effective strategy. Instead, a more balanced and optimized approach must be determined, one that dynamically adjusts based on the system's operational demands.

3.2 Energy Efficiency

Energy efficiency is an important metric that measures the amount of energy used to compute a specific task. Due to the complex nature and various approaches to coding, software can be suboptimal in its operation, leading to higher energy usage and degrading energy efficiency metrics. Energy usage is directly linked to carbon emissions, costs, and the need for better thermal management systems and power delivery methods, highlighting the importance of understanding this metric.

A good way to improve the energy efficiency of software is by optimizing the code to reduce run time and power usage. This works well because software can easily be scaled without adding much extra cost, meaning small improvements in the code can have a big impact on how the software operates. When the software is used on hundreds or thousands of devices, the small energy savings from optimization can add up to make a significant difference. However, it's important to consider the context in which the software is being used. If it runs on low-power devices or is used on only a few devices, the energy saved through optimization might not justify the effort and could even be counterproductive in some cases.

This illustrates the importance of making decisions based on more than just technical requirements. Software architects and project managers must balance optimizing the software with the energy required for optimization and the real-world impact these optimizations bring. This will be further discussed in later sections relating to the software life cycle.

3.3 Carbon Efficiency

Carbon efficiency is another crucial metric that is closely linked with both energy efficiency and hardware efficiency. It essentially measures the amount of CO_2eq emissions associated with a task

performed by the software. In the context of sustainable computing, carbon efficiency provides a clear indication of the environmental impact of software operations, making it a vital consideration for developers and IT managers aiming to reduce the carbon footprint of their digital infrastructure.
Improvements in carbon efficiency involve optimizing the software and its associated hardware interactions to minimize energy consumption and system overheads. For example, energy-efficient algorithms would not only require less electricity to run but also less powerful hardware, improving both operational and embodied emissions.
Additionally, carbon efficiency is not simply a function of the direct energy requirements of the software, but also the life cycle and sourcing of the underlying hardware being utilized. The origin of materials, refinement processes, manufacturing, transportation, and subsequent disposal have a huge impact on the carbon footprint of the hardware used. And given that most of these parameters lie outside the control of the users of the hardware, it is important to maximize its usage by extending its lifespan and minimizing unnecessary hardware upgrades. Taking these steps positively impacts carbon efficiency.

3.4 Power Usage Effectiveness (PUE)

Power Usage Effectiveness (PUE) is a widely used metric, particularly in the context of data center operations. As shown in Equation 3.1, it is a dimensionless ratio that serves as a measure of how efficiently a data center uses energy. Specifically, it compares the total energy consumption of the facility to the energy consumed by the IT equipment alone. This metric effectively highlights the proportion of energy used for computational tasks versus the overhead necessary to maintain the proper functioning of compute units, as illustrated in Figure 6.
Most of the world's data centers operate with a PUE value between 1 and 3, with the ideal goal being a PUE of 1, which would indicate no overhead energy usage beyond what is required for computation. Notably, Google has reported achieving a quarterly PUE as low as 1.07 and a fleet-wide quarterly PUE as low as 1.08 [15].

$$\text{PUE} = \frac{\text{Total Facility Energy Consumption}}{\text{IT Equipment Energy Consumption}} \tag{3.1}$$

Although PUE is a highly adopted metric, it has certain limitations that make it less than ideal for some comparative analyses. One of the main shortcomings is that it does not account for the environmental conditions in which the data center operates. For instance, in cooler climates or during colder periods, data centers can utilize free cooling, which is not reflected in the PUE value. Additionally, PUE does not provide any information regarding the efficiency or specifications of the hardware used within the data center.

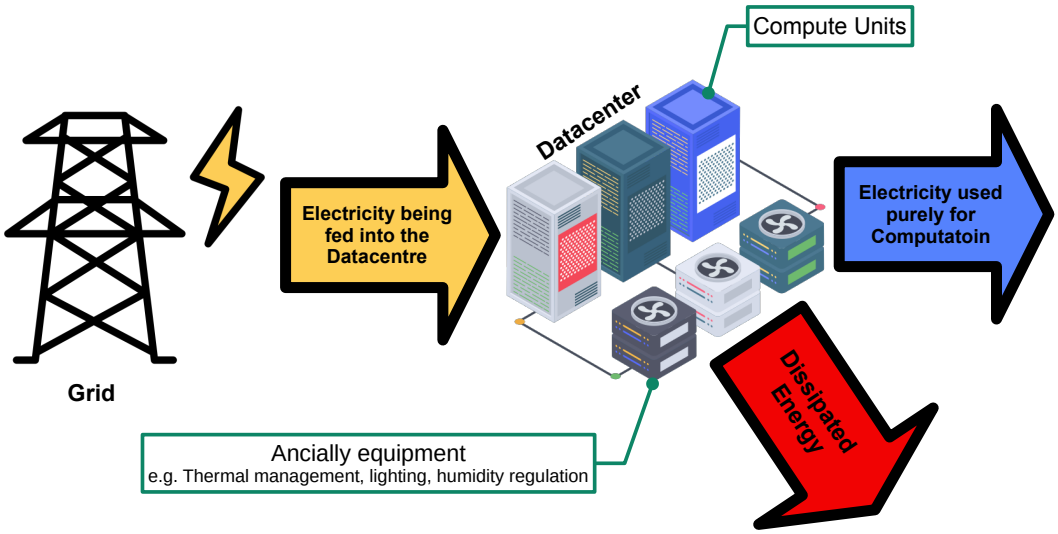

Figure 6: Energy Flow diagram in a Datacenter

As can be seen in Figure 7, the global trend of PUE is declining indicating that data centers are becoming efficient in their energy usage and thermal management strategies. This positive development could play a much bigger role in software sustainability as tasks could be offloaded to these data centers further improving the value proposition of technologies such as cloud computing.

3.5 Water Usage Effectiveness (WUE)

Water Usage Effectiveness (WUE) is an established and widely recognized metric pertaining to data centers. It measures the amount of water consumed by a data center per unit of energy used purely for computation purposes. Due to the highly controlled environment of data centers, water is extensively used in thermal management systems and to regulate the humidity of the internal environment, making water a critical resource for the proper and economical operation of data centers.

$$\text{WUE} = \frac{\text{Data Center Water Consumption (in liters)}}{\text{IT Equipment Energy (in kilowatt hours)}} \tag{3.2}$$

WUE is crucial for the effective management of water resources, especially given that freshwater constitutes less than 3% of the world's total water supply. Alternative water sources, such as seawater or brackish water, require extensive and costly pretreatment before they can be used in a data center, further emphasizing the importance of optimizing freshwater use. Additionally, there are significant

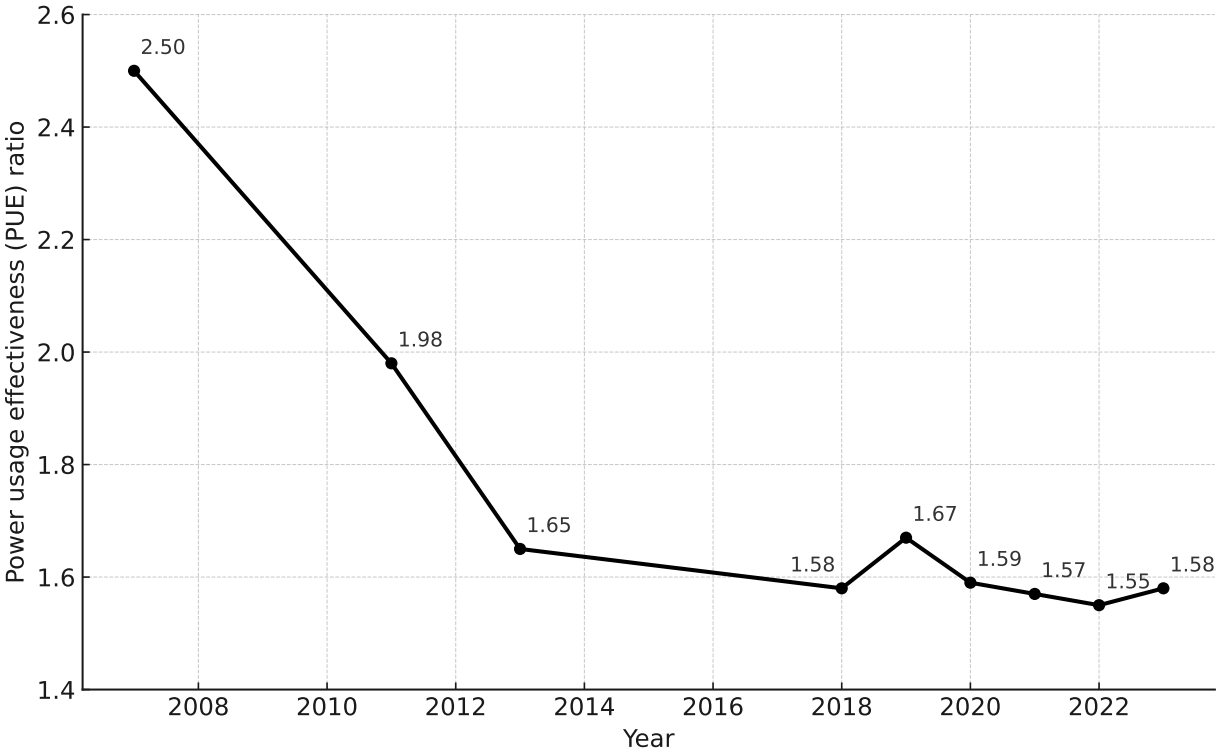

Figure 7: Global PUE Trend [16]

social implications related to water usage and access to clean drinking water.

Although this thesis does not delve into the aspect of water and water related sustainability aspects, it is an important field for major investigations and research. Especially as fresh and potable water becomes scarce, solutions and approaches that incorporate water management could have multiplied implications. Additionally, as data centers continue to evolve and grow in significance in software applications, water management and it's impact will prove an important aspect of design and operations of these systems.

3.6 Carbon Intensity

Carbon Intensity measures the emissions associated with the electricity consumed by software operations. It is closely linked to the technologies used for electricity generation and the types of fuels they utilize. The unit of measurement for carbon intensity is grams of CO_2 equivalent per kilowatt-hour (gCO_2eq/kWh). Typically, organizations have limited direct control over the carbon intensity of the electricity they consume, as it is largely dependent on the grid they are connected to. The carbon intensity of grid electricity fluctuates constantly, influenced by the varying sources of power supply at any given moment.

Unless there is onsite generation, the locational carbon intensity of consumed electricity cannot be

significantly altered. While some market mechanisms, such as Contracts for Difference (CfD), Power Purchase Agreements (PPAs), and Green Certificates, can be utilized by firms to offset their actual emissions, these mechanisms are not fully recognized under the ISO standards governing SCI, which will be discussed in a later section.

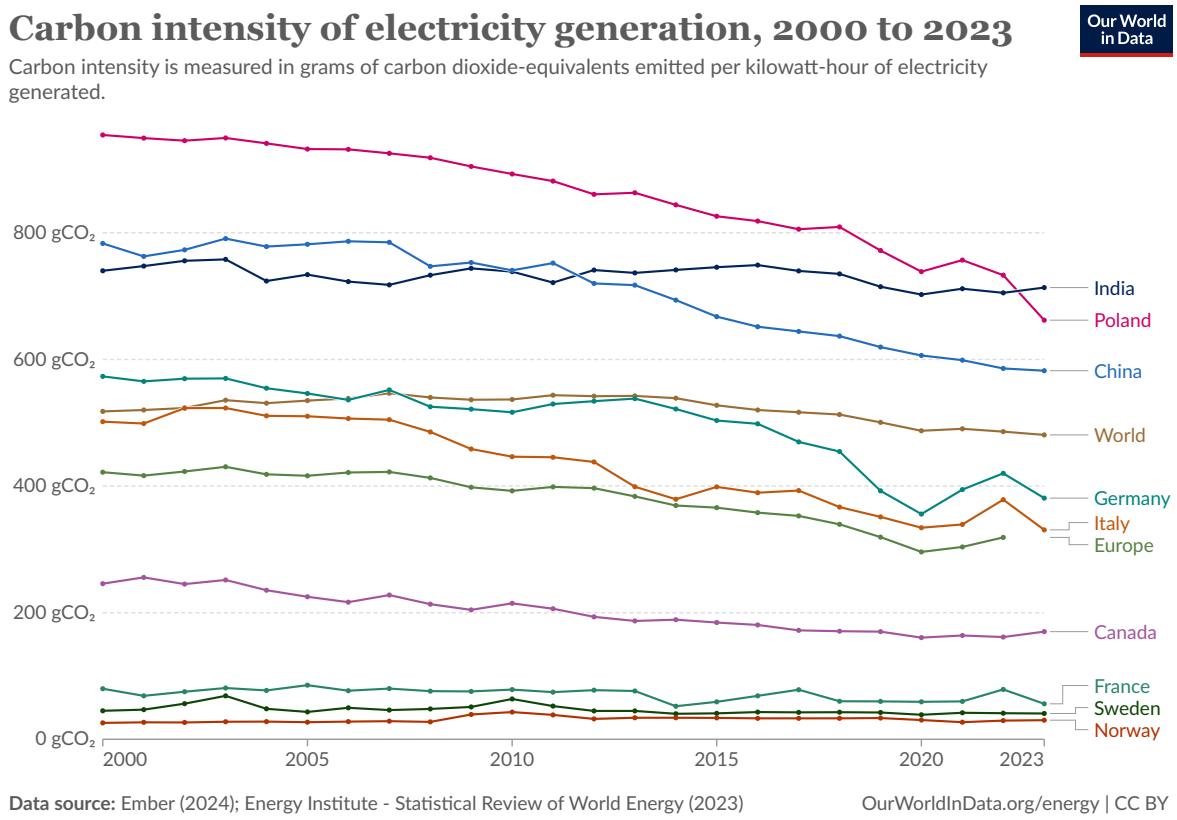

Figure 8: Global Carbon Intensity Trends [17]

Carbon intensity varies significantly with time and location. As seen in Figure 8, carbon intensity differs across regions of the world due to the electricity production technologies and fuels they utilize. Additionally, carbon intensity can vary significantly throughout the day due to the combined effects of changing demand and the availability of wind and solar resources, which can be observed in an example graph in Figure 11.

The global trend looks positive as carbon intensity is declining in most countries. Interestingly, countries like France and Norway have very low carbon intensity in their electricity supply due to the predominant use of nuclear and hydroelectric power, respectively. However, the data also shows a slight uptick in carbon intensity in EU countries in 2022, correlating with the energy disruptions caused by the Russian invasion of Ukraine [18].

3.7 Carbon Awareness

Carbon Awareness refers to the capability of software to adapt its operations based on the carbon intensity of the electricity it consumes or is projected to consume. The goal of Carbon-Aware software is to minimize its carbon footprint by optimizing its operation according to the carbon intensity of the electricity supplied.

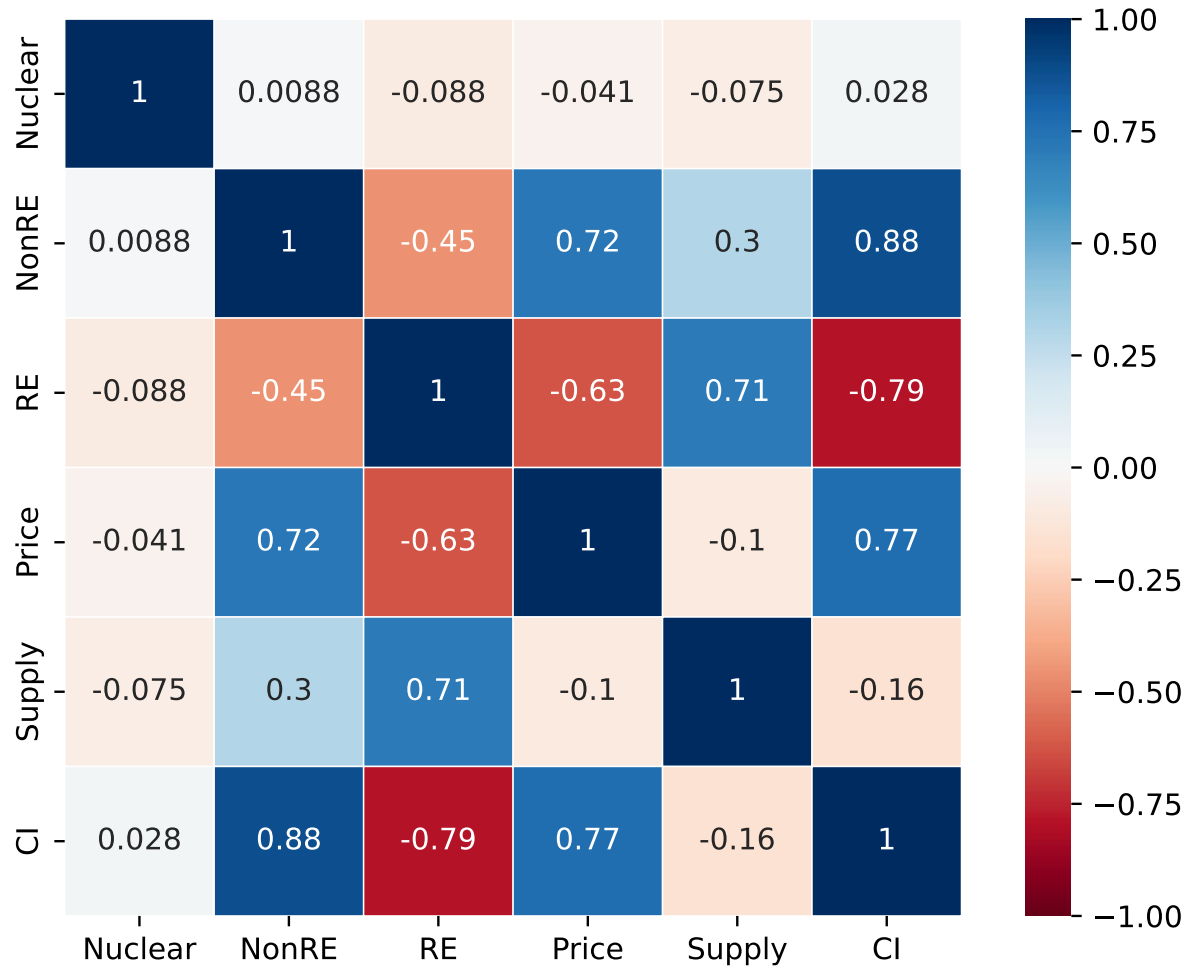

Figure 9: Carbon Intensity Correlational analysis heatmap

To enable software to adapt to the carbon intensity of the electricity mix, it is crucial for organizations to be able to reliably forecast carbon intensity to adjust their computational load accordingly. One approach to achieving this is by using Machine Learning (ML) models to forecast carbon intensity. Although developing ML models for this purpose is beyond the scope of this thesis, a preliminary analysis has been conducted to identify key parameters that could be used in training such models.

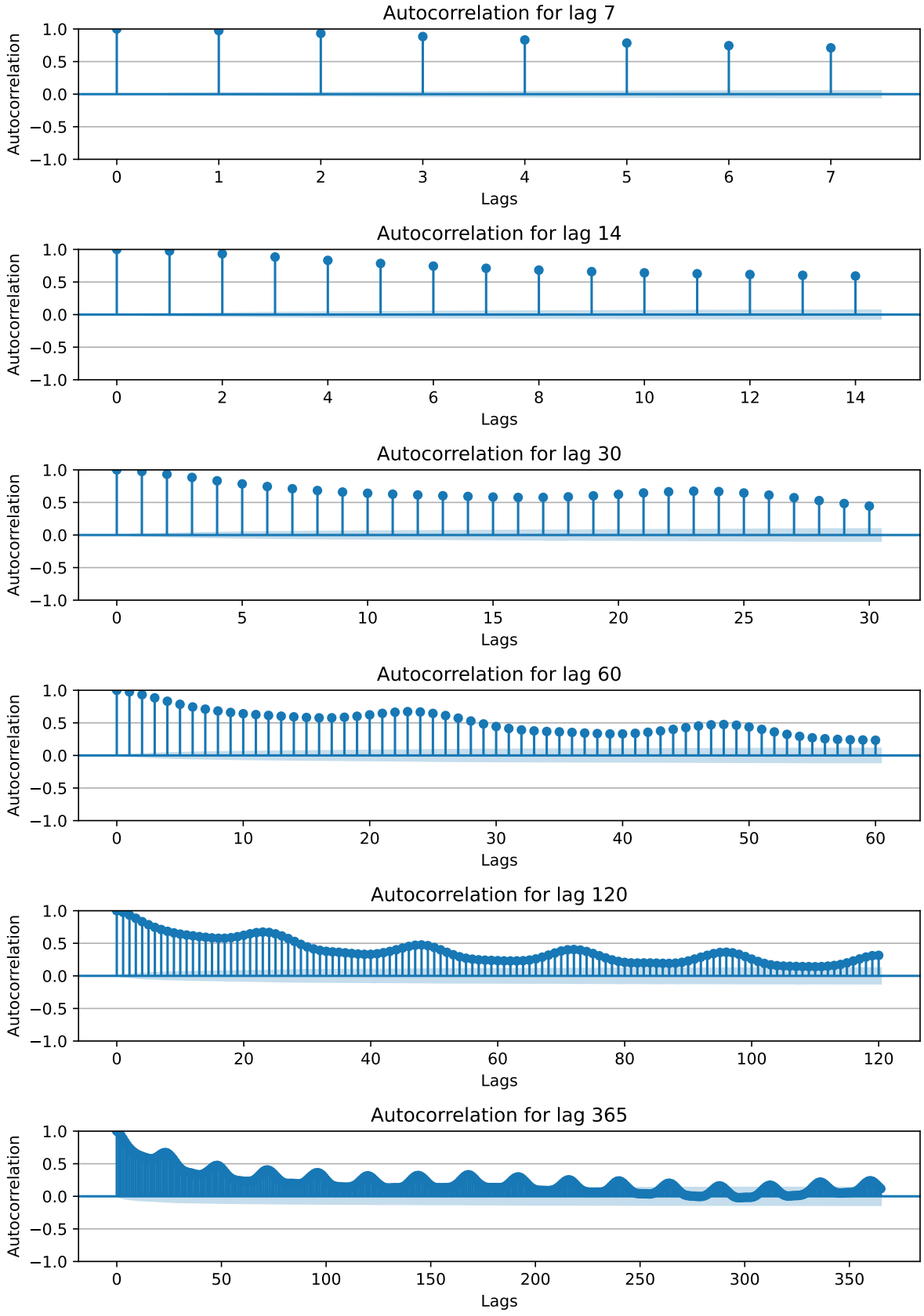

Figure 10: Carbon Intensity Auto-correlational Analysis

Data from various sources, including *Electricity Maps*, *ENTSOE*, and *Energy-Charts (Fraunhofer ISE)*, was collected for the German market due to the availability and accessibility of data. The first step involved performing a correlational analysis between various features such as generation data, renewable energy (RE) capacity, nuclear generation, market prices, and carbon intensity. As illustrated in Figure 9, there are strong positive correlations between carbon intensity and both non-RE generation and electricity prices, and a strong negative correlation between carbon intensity and RE generation. These three parameters could be effectively utilized in an ML model to forecast carbon intensity. Additionally, this analysis suggests that enhancing Carbon Awareness in software could also lead to cost savings, as the strong negative correlation between electricity prices and carbon intensity indicates that lower carbon intensity often coincides with lower energy costs.

Another important preliminary analysis is autocorrelation, which measures the correlation of a parameter with its own historical values. This analysis provides insights into the time horizon over which ML algorithms could accurately predict carbon intensity. Multiple analyses were conducted using various lag values of 7, 14, 30, 60, 120, and 365 days, as shown in Figure 10. The results reveal a high autocorrelation up to two weeks prior, suggesting that an ML model could reliably predict carbon intensity up to two weeks in advance. Additionally, there are notable autocorrelations at approximately 24 and 46 days prior, which could inform medium- to long-term decision-making.

Implementation Techniques

To enhance the working of the software and making it more Carbon Aware, two techniques are presented below. These techniques offer the company the flexibility and control over the functioning of their software product without majorly disrupting the quality and usability of their software. Although these techniques are presented separately, they could be used together complimenting each other limitations whilst still remaining cost-effective and providing a more comprehensive and granular method of mitigating emissions.

Temporal Shifting

As previously demonstrated, there is a high potential for accurately predicting Carbon Intensity using Machine Learning (ML) forecasting models. This capability can be leveraged to schedule tasks and energy-intensive operations during periods when carbon intensity is low. As illustrated in Figure 11, carbon intensity fluctuates throughout the day. By scheduling computational tasks during periods when carbon intensity drops below a predetermined threshold, organizations can optimize energy use. This strategy is particularly effective for high-energy computational tasks, such as training ML or AI models, which may not have stringent deadlines. Temporal shifting can be combined with tools like

Zeus to manage both emissions and operational costs effectively [19].

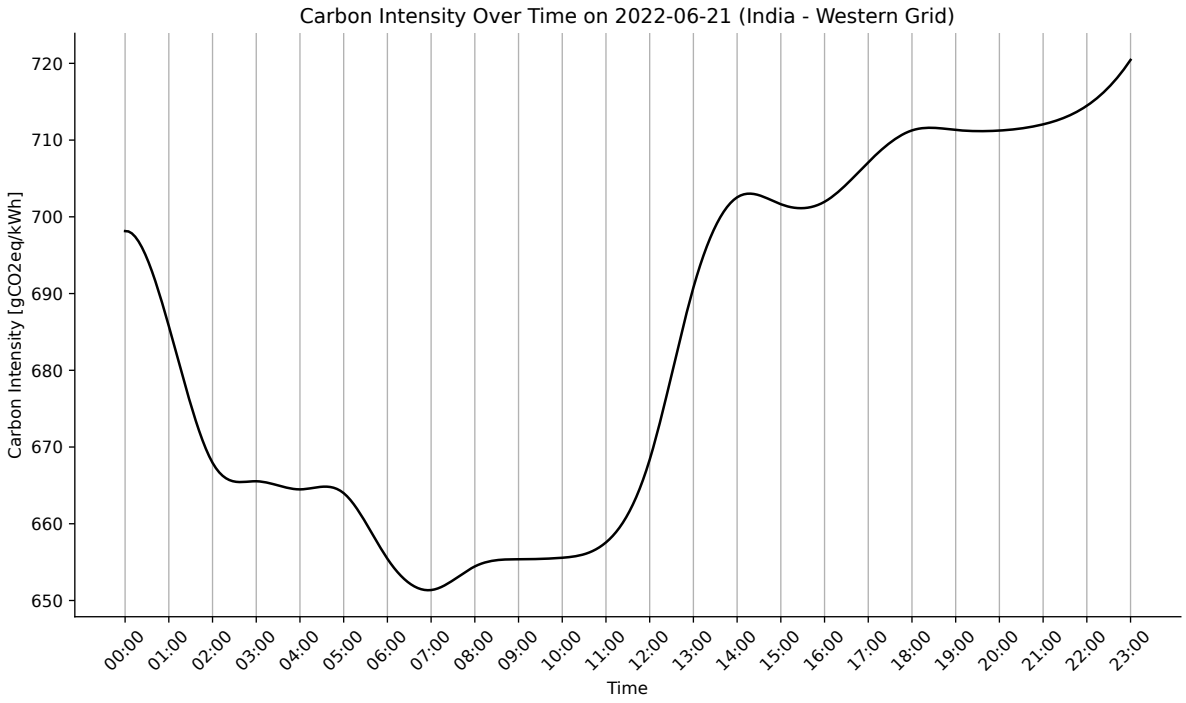

Figure 11: Carbon Intensity throughout the day (India - Western Grid)

Geographic Shifting

Another effective strategy is to shift computational workloads to regions with lower carbon intensities. Although this approach might not be easily accessible to firms with limited resources, it could be highly effective when leveraging cloud computing. As cloud computing becomes more accessible, firms could shift energy-intensive tasks to the cloud, thereby lowering their carbon footprint. By shifting these computations to regions with lower carbon intensity, there could also be a case for optimizing global resources.

This strategy offers the added benefit of minimal delay in task completion. However, it also presents potential challenges, such as increased cybersecurity risks, as data must traverse the internet to reach the destination data centers.

CarbonAware SDK

To support the implementation of Temporal and Geographic Shifting techniques, a tool known as the *CarbonAware SDK* is currently under development by the *Green Software Foundation (GSF)*. This tool provides both real-time and forecasted carbon intensity data for various regions around the world. The data generated by the *CarbonAware SDK* can be used in two primary ways: it can be utilized by

employees within a firm to manually schedule tasks during periods of lower carbon intensity, or it can be integrated directly into the company's software products and tools for automated, carbon-aware operations.

The use of the *CarbonAware SDK* in existing workflows can help organizations optimize their operations to reduce their carbon footprint. It allows for dynamic adjustments to the scheduling of computation loads based on both current and predicted carbon intensity values. Additionally, it can be integrated into the end software product, enabling the user to run the software in a "greener" mode. This would enhance the value proposition of their end products and offer users the choice to be more eco-conscious.

3.8 Software Carbon Intensity (SCI)

Software Carbon Intensity (SCI) is a pivotal metric developed by the *Green Software Foundation (GSF)*. It integrates key metrics such as hardware efficiency and energy efficiency, providing a comprehensive measure of the carbon impact associated with software operations. The SCI is designed to be highly applicable within the software engineering domain, offering a metric that is both straightforward to understand and practical to implement.

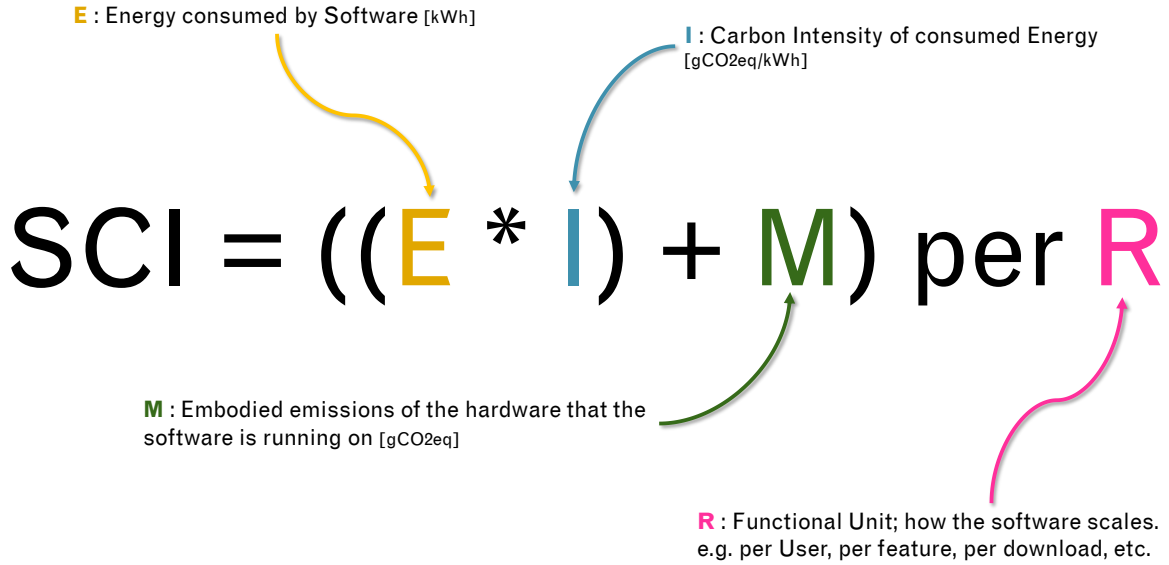

Figure 12: Software Carbon Intensity

The SCI metric is composed of four key variables, each of which will be discussed in detail in the subsequent subsections:

Energy (E)

The energy consumed by the software or the hardware used to develop the software is represented by **E**. This variable is critically important as it can be directly influenced by the company through strategic decision-making and code optimization. Closely tied to energy efficiency and carbon efficiency, **E** offers a scalable and actionable metric for implementing strategies to reduce the carbon footprint of software operations. By focusing on optimizing **E**, organizations can make significant strides in reducing their overall SCI, contributing to both environmental sustainability and cost efficiency. The unit of measurement for **E** is kilowatt-hours (kWh).

Carbon Intensity (I)

As previously discussed in subsection 3.6, carbon intensity is a crucial metric integrated into the Software Carbon Intensity (SCI) framework. Carbon intensity represents the amount of CO_2 equivalent emissions per kilowatt-hour of electricity consumed, typically denoted as **I**. The unit of measurement is grams of CO_2 equivalent per kilowatt-hour (gCO_2eq/kWh).

While the direct control over carbon intensity often lies outside the immediate influence of the company—primarily depending on the energy grid's composition—companies can manage it effectively by adopting Carbon Awareness strategies, as outlined in subsection 3.7. By optimizing when and where computational tasks are executed, organizations can leverage periods of lower carbon intensity, thereby reducing their overall carbon footprint. This approach emphasizes the importance of strategic planning in minimizing emissions, even when direct control over energy sources is limited.

Embodied Emissions (M)

Embodied Emissions, denoted as **M**, represent a significant component of the Software Carbon Intensity (SCI) score. This parameter encompasses the total emissions associated with all computing hardware, including but not limited to laptops, servers, displays, and peripherals such as mice. These emissions originate from every stage of the hardware's lifecycle, beginning with the extraction of raw materials from the Earth's crust, through manufacturing and distribution, and culminating in the final product that runs the software.

Given that computing devices are among the most complex pieces of hardware, requiring extensive global supply chains and a wide range of elements from the periodic table, they inherently carry a substantial carbon footprint. Accurately accounting for these embodied emissions is therefore essential when calculating the overall SCI score. The emissions represented by **M** are measured in grams of CO_2 equivalent (gCO_2eq).

To precisely allocate the emissions associated with the use of a particular hardware item, the concept

of *Amortization* is employed. Amortization involves spreading the embodied emissions of a device over its expected operational lifespan. For instance, consider an office laptop with 250 kg CO_2eq of embodied emissions and an operational lifetime of 4 years. The calculation would proceed as follows:

Laptop embodied emissions = 250 kg CO_2eq

Laptop lifetime = 4 years

Total working days per year = 52 * 5 = 260

Total days of holidays per year = 25

Total number of actual working days = 260 − 25 = 245

Amortized emissions = 250/245 = 1.02 kg CO_2eq per day

Given the complex nature of modern computers, a thorough Life Cycle Assessment (LCA) is critical for accurately estimating embodied emissions. However, conducting such an assessment is often impractical for individual companies due to the rapid pace of technological advancement and the complexity of the supply chains involved. For example, when examining available LCA databases, such as Ecoinvent, only one entry for a laptop was found, which dates back to a model produced in 2005. This laptop, featuring an Intel Pentium 3 processor with a 600 MHz speed, 10 GB of RAM, 128 MB of memory, a 12.1-inch screen, and a total mass of 3.15 kg (including the expansion base), is no longer representative of current technology [20]. Consequently, this outdated data is not suitable for accurately assessing modern hardware.
Fortunately, the Information and Communication Technology (ICT) industry has developed methods and tools for reporting the carbon footprint of both consumer and professional-grade products. One such tool is the Product Attribute to Impact Algorithm (PAIA), which is discussed below.

Product Attribute to Impact Algorithm (PAIA)

Product Attribute to Impact Algorithm (PAIA) is a tool developed and maintained by MIT in collaboration with Quantis, designed to calculate the environmental footprint of ICT devices. PAIA is more than just a tool; it is also a methodology and a consortium of ICT companies aimed at providing an easy-to-use, accessible, and cost-effective method for calculating the environmental footprint of ICT products. The development of PAIA was a collaborative effort involving industry leaders such as Dell, Lenovo, Cisco, IBM, and Hewlett Packard Enterprises, among others [21][22].

Despite its wide adoption, the PAIA methodology lacks publicly available detailed documentation, with most information being confined to conference papers [23]. Upon further investigation, it is found that PAIA is based on a hybrid LCA approach, as detailed in the study "Economic-balance hybrid LCA extended with uncertainty analysis: case study of a laptop computer" [24]. This approach attempts to

calculate the LCA of a laptop by combining historical traditional LCA data with economic methods for components where LCA data is unavailable. For these components, environmental impacts are allocated based on economic factors such as costs and weights, and when no data is available, Economic Input-Output (EIO) tables generated annually by the country are used. This approach ensures the best possible approximation but comes with certain limitations.

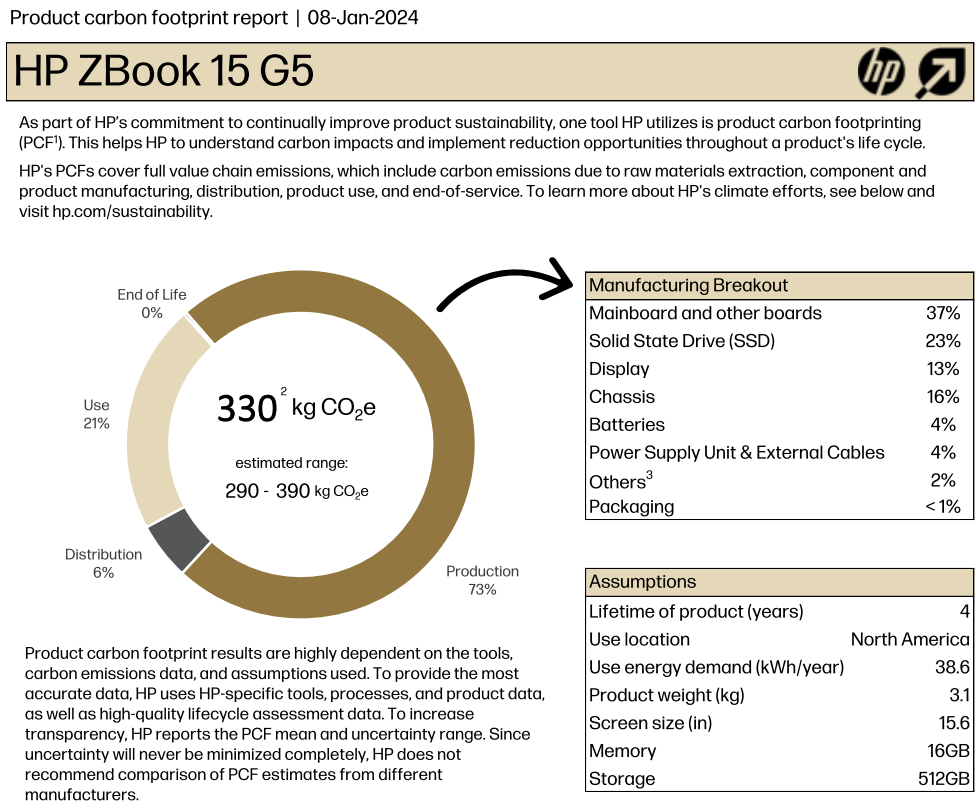

Figure 13: Example excerpt from a PCF report (HP Zbook 15 G5) [25]

With the current rapid pace of technological development, the share of components in a system lacking proper LCA analysis is likely to grow. This may lead to a higher reliance on estimations based on economic parameters. Although this provides a reasonable estimate of the carbon footprint, it could result in a lack of focus on improving the footprint of certain components. Additionally, it would make it more difficult for academic investigations to be conducted on these components due to the lack of reliable data from the industry. Over time, the errors might compound as more reliance is placed on economic accounting for the carbon footprint.

The hybrid methodology is visually shown in Figure 14 [24]. Although PAIA has proven to be an essential tool for estimating the environmental impact of ICT products, it is important to understand its limitations and develop new methods capable of performing more accurate and reliable LCAs on system components, especially by leveraging newer technologies and methods.

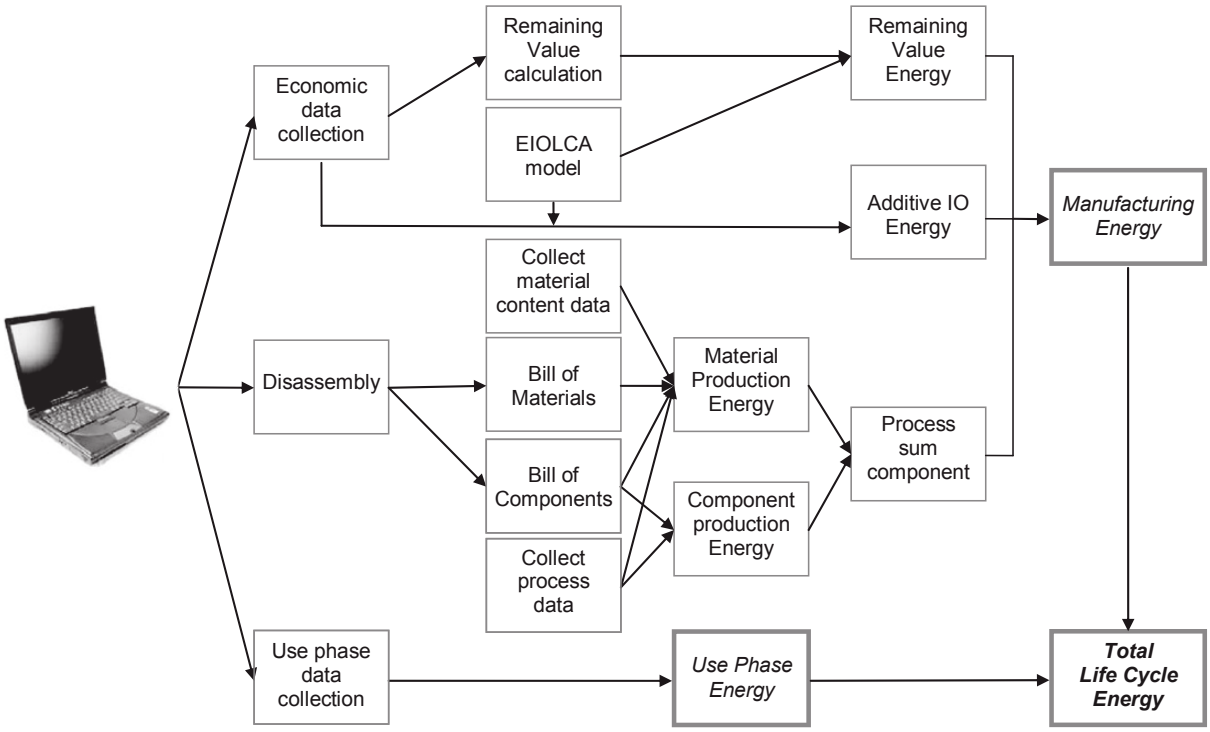

Figure 14: Hybrid LCA methodology [24]

Functional Unit (F)

The final parameter of the Software Carbon Intensity (SCI) score is the functional unit, denoted as **F**. Although it does not require direct quantification or calculation, this parameter is crucial for framing and contextualizing the SCI score, making the information more accessible and comparable across different software applications. Given that software serves a wide range of purposes and its usage can vary significantly, there is no single, universal functional unit that can be applied uniformly across all software types.

An important point of observations is that of the Functional Unit mentioned in the SCI score. Interpretations on the first glance might point to the use of the term "Functional Unit" in the context of Life Cycle Assessment (LCA); especially with people working in the sustainability domain. Pertaining to the SCI, the functional unit refers to how the software scales in it's usage and deployment. For example, if the software is being distributed in a manner where the user purchases it only once, then the functional unit could be the complete development stage of the softwares life cycle. Alternatively, if the software is being distributed as a services (SaaS) or a subscription, then the functional unit would be per API call of the software or per timeframe of the subscription.

Proper discussion and definition of the functional unit is one of the most critical aspects for the SCI score that needs to be undertaken during the design phase of the software. This decision has a direct implication on how the SCI score is calculated and it's wider applicability in the firm to be

used for comparisons with other product lines/services or bench marking purposes. A proper and appropriate functional unit (**R**) ensures that proper interpretation of the SCI score can be undertaken and corresponding actions can be taken for improvement and mitigation.

3.9 ISO Standard

Given the increasing prominence of green software, a new ISO standard has been published that addresses Software Carbon Intensity (SCI). This standard, largely developed by the Green Software Foundation (GSF), was officially released in March 2024. While the publication of this standard represents a significant step forward, it is relatively concise and lacks a comprehensive methodology or a set of tools for effectively implementing SCI in various contexts. Consequently, it is necessary for organizations to adapt the standard to their specific use cases and implementation needs [26].
The ISO standard does not explicitly mention any measurement for the use phase of the software, which might prove as an important area of analysis as large amount of emissions are associated during the usage[27]. The thesis aims to address this potential gap whilst also complying with the ISO standard. The developed approach aims to provide an overall view on the complete lifecycle of the software. The presented methodology will help corporations in the Scope 3 emission calculation and reporting and subsequent mitigation strategies.

3.10 Impact Framework (IF)

The Impact Framework (IF) is a tool and method designed by the Green Software Foundation (GSF) to document and calculate the environmental impact of software. It takes the form of a standard YAML file that is human-readable and does not require any special software to open or edit. Due to its open nature and format, it is highly suitable for auditing and review. An example YAML file can be found in Listing 1, with a breakdown of its components presented below.
In addition to documenting various parameters, the Impact Framework manifest file can be used for computational purposes. This computation is facilitated by the use of Plugins, which are typically open-source or can be developed. The plugins are written in TypeScript, with a template provided at `https://github.com/Green-Software-Foundation/if-plugin-template`. This allows for a low barrier to entry, and software evaluations can be carried out without the need for any coding, using just the plugins provided by GSF or other open-source contributors.
There are three major components of an IF YAML file, which are explained in more detail below:

1. **Metadata**

   The first section contains the metadata of the project, which can be seen in lines 1–2 of Listing 1. This section includes general information about the project and serves as an organizational

tool for users of the YAML file. While it does not play a direct role in calculations, it provides important context for the human user.

2. **Plugin Initialization**

   The second part, which is crucial for the computation and interaction of the manifest file with the Plugins, contains the initialization of the plugins that will be used throughout the project. Global configurations for the plugins are defined here. This section can be observed in lines 3–13 of Listing 1.

3. **Data and Execution Pipelines**

   The third and final part contains the input data on which the plugins will operate and their respective outputs. This section follows a parent-child structure, allowing for multiple child components, each with its own pipeline that defines the order in which plugins are executed. Configuration details for each plugin can also be specified here. The full structure of this block can be seen in lines 15–38 of Listing 1. Individual data entries, such as those in lines 24–28, 29–33, and 34–38, demonstrate the flexibility of the IF. There can be an infinite number of entries, provided standardized variable names are used therefore ensuring compatibility and transparency in the reporting process.

4 Software Lifecycle

Like any other product, software follows a structured lifecycle composed of multiple stages. Each stage in this lifecycle has a distinct impact on the overall carbon footprint and sustainability reporting of the software, the Software Carbon Intensity (SCI) score. Breaking down the lifecycle into concrete categories not only facilitates proper accounting but also aids in identifying hotspots where emissions may be higher and where mitigation strategies can be most effectively implemented.

Figure 15: Generic Software Lifecycle

4.1 Requirements & Design

The initial phases of the software lifecycle are centered on determining the functional requirements and design of the software. These stages are primarily managed by project managers and software architects. Although these phases may not directly contribute to the final SCI score, the decisions made during this stage have a profound influence on the SCI score throughout the software's lifecycle.
While not apparent at first glance, this phase holds the key to having a significant impact on the SCI score, as the company has total control over the decisions made during this phase by the project managers and software architects over the scope, goals, and architecture of the software product and its subsequent life stages. One critical consideration here is the balance between optimizing the software for performance and minimizing its energy consumption. For example, if the software is designed for a low-energy device with minimal carbon emissions, it may not be necessary or efficient to allocate extensive resources toward optimizing its energy performance during the development phase.

4.2 Development & Testing

The development and testing phase is one of the most thoroughly documented and dynamic aspects of the software lifecycle. It can follow linear models like the *Waterfall* approach or iterative models such as *Agile* methodologies. Regardless of the development model employed, the sustainability framework proposed in this thesis can be seamlessly integrated into this phase to track and optimize SCI.
With the growing use of AI, Machine Learning, and other resource-intensive tools, the SCI score during development and testing can vary greatly. For software intended to scale across multiple deployments or be widely distributed, this phase can have a considerable impact on the SCI score.

In these cases, minimizing energy usage and emissions during development can lead to significant long-term sustainability gains. The emissions generated during this phase typically falls under Scope 1 and Scope 2 emissions.

4.3 Deployment

Deployment refers to how the software is delivered to end users, a critical phase in the lifecycle. With the increasing reliance on cloud infrastructure and data centers for software deployment, this stage presents unique challenges and opportunities. Data centers require substantial amounts of energy and water to maintain operations, though, in recent years, environmental and industrial pressures have driven many data centers to adopt more eco-friendly practices.

Despite these improvements, companies still have a medium to high degree of control over the sustainability of this phase. By making eco-conscious choices when selecting cloud providers and ensuring that data centers meet specific environmental standards, firms can significantly reduce the SCI score associated with deployment.

4.4 Usage & Maintenance

The usage and maintenance phase is perhaps the most variable and unpredictable aspect of the software lifecycle, largely because it is influenced by how end users interact with the software. The scalability and versatility of software means that much of this phase is out of the company's direct control. However, design choices made during previous phases of the life cycle can have long-lasting effects on the SCI score.

In terms of sustainability reporting, the emissions generated during this phase fall under Scope 3, as the responsibility for emissions is shared between the producer and consumer. Further discussion on the complexities of accounting for emissions during the usage phase can be found in subsection 5.6.

5

Developed Framework

In line with the requirements of the Greenhouse Gas (GHG) Protocol, companies are obligated to report their emissions categorized as Scope 1, 2, and 3 emissions. To facilitate this process, we have developed a comprehensive methodology that includes detailed measurement techniques, carbon footprint calculations, and benchmarking of the tools utilized. Additionally, a proof of concept is provided, accompanied by the corresponding code, to ensure transparency and reproducibility.

5.1 Framework

To assist corporations in adopting the ISO standard related to the Software Carbon Intensity (SCI) score, we have developed a foundational framework, as illustrated in Figure 16. This framework serves as a straightforward and accessible entry point for companies venturing into the Green Software domain. The framework is designed to be applicable to both the development and usage phases of software, with the core processes remaining consistent while the reporting mechanisms differ.

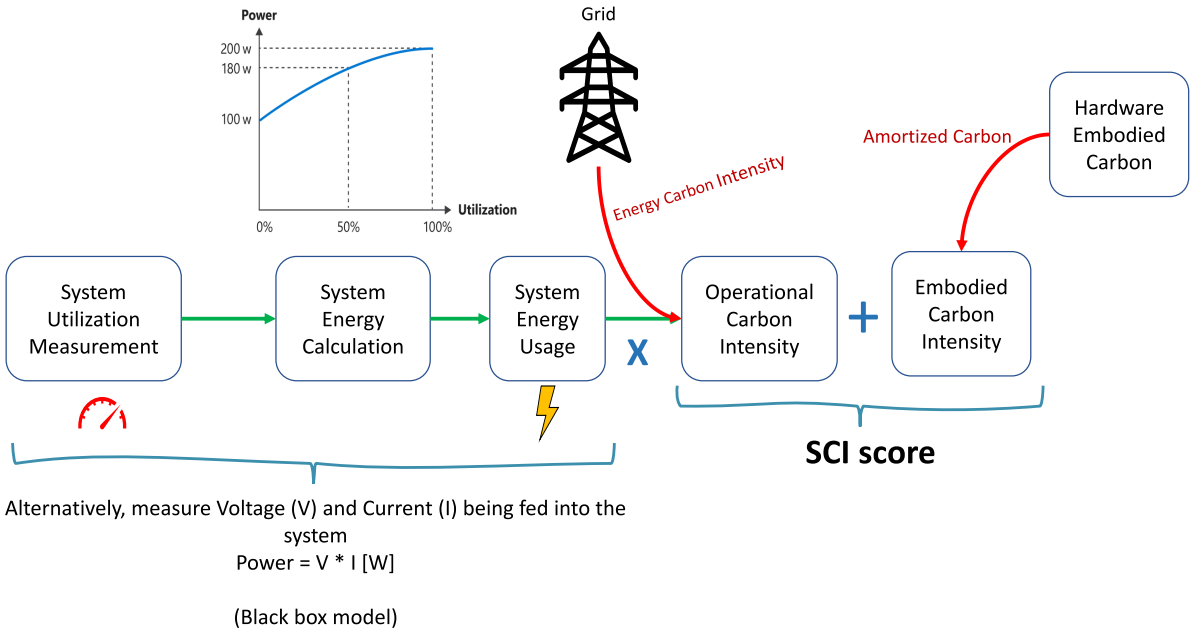

Figure 16: SCI calculation framework

As depicted in Figure 16, the framework begins with the measurement of system utilization parameters, such as CPU and GPU usage percentages. These utilization metrics are critical as they need to be converted into corresponding energy values for each hardware component. Given the absence of a standardized or widely accepted method for this conversion, various tests are conducted in subsequent

sections to determine the most appropriate approach. The chosen conversion method is then used to calculate the Operational Carbon Intensity, which is subsequently combined with the amortized carbon from the embodied emissions of the underlying hardware. The sum of these components yields the Software Carbon Intensity (SCI) value, which can be assigned to a specific functional unit.

For the development phase, the framework leverages the Impact Framework (IF), as introduced earlier in subsection 3.10. This integration not only adheres to a standardized format but also simplifies the validation process for companies by enabling auditing mechanisms that ensure accuracy and compliance in reporting.

For the usage phase, where the software may run on various hardware configurations beyond the company's direct control, certain assumptions and modifications to the reporting system are necessary. As most variables in the usage phase contribute to Scope 3 (downstream) emissions, and are outside the direct influence of the company, it becomes essential to make informed estimates for these emissions. The reporting mechanism for the usage phase is further elaborated in subsequent sections, providing detailed guidance on how to approach these challenges.

This framework, while simple in its design, offers a robust starting point for companies looking to integrate SCI considerations into their operations, paving the way for more sustainable software practices and contributing to broader environmental goals.

Measured Parameters

As illustrated in Figure 16, a variety of system utilization variables must be measured to accurately assess the Software Carbon Intensity (SCI). Since a typical computer consists primarily of components such as the CPU, GPU, memory, networking hardware, and, in the case of laptops, the screen, our focus will be on these key elements.

Certain components, like the screen, have a significant but relatively constant power draw. For these components, we can assume a fixed energy consumption value based on the operational time and power ratings provided by the manufacturer [28]. Given its relatively straightforward nature, this parameter is not covered in detail in this thesis.

Other components, including memory, disk drives, and onboard networking equipment, are also not included in our analysis. This exclusion is due to their comparatively low energy footprint, which, while non-negligible, adds complexity without significantly impacting the overall SCI score.

The primary focus of our measurement efforts is on the CPU and GPU, as these are the most power-intensive components in any computing device. Typically rated at tens to hundreds of watts, the CPU and GPU are crucial determinants of the SCI score. Their power consumption can fluctuate significantly depending on workload, making accurate measurement essential for reliable SCI calculations.

In this section, we explore various tools and methods available for measuring the power draw of the CPU and GPU. These tools are critical for capturing the real-time energy usage of these components, which directly influences the accuracy of the SCI score. By focusing on the major power consumers, we aim to provide a practical approach to calculating SCI that balances precision with feasibility.

5.2 Measurement Techniques

As our goal is to measure the energy being consumed by the hardware on which the software runs on, there are various methods of doing these measurements. Additionally, as the ICT domain has previously dealt with energy efficiency to prolong the runtime and economical operation of their devices, we can leverage these tools in our measurements.

5.2.1 Hardware based

One of the most simplest and widely established methods for measurement of energy, with high accuracy, is the usage of energy or power meters. These devices generally are installed on the current carrying wires that need to be monitored and in the case of monitoring of a computer system, could lie between/on the wall socket and the computer's power inlet. Energy meters typically operate on both single and multi-phase supplies my measuring parameters such as Voltage and Current to calculate the Power and subsequent energy usage.

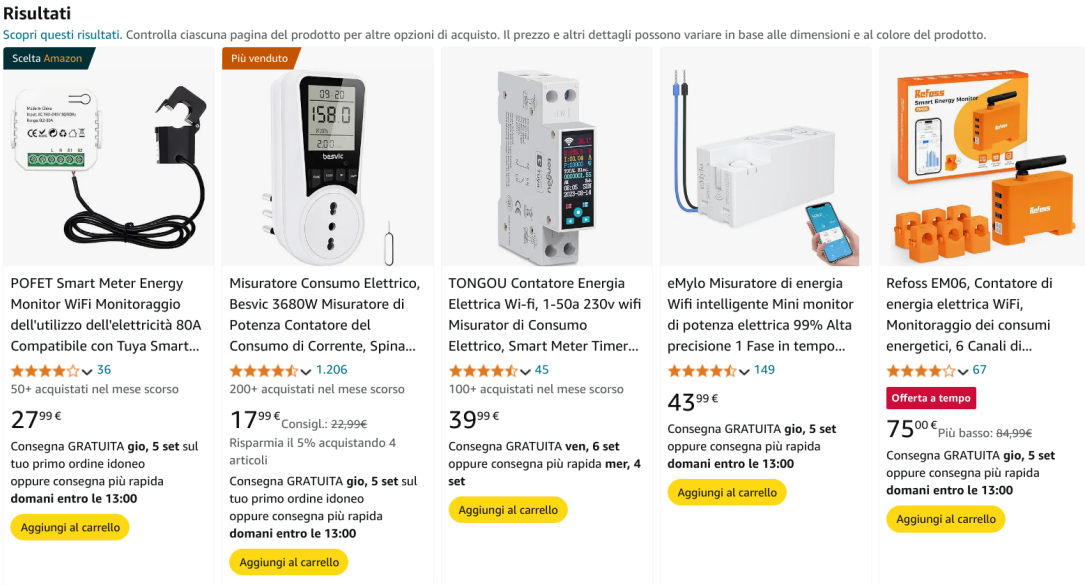

Figure 17: Typical costs of Energy meters

However, while hardware-based energy monitoring provides accurate and direct measurements, it comes with significant drawbacks, particularly in terms of cost. A single energy meter typically costs

upwards of 15 euros, and when there is a need to monitor multiple systems used in software development, the cumulative cost can become substantial. This expense is further compounded by the ongoing costs of maintenance and potential upgrades required for these devices. Moreover, in situations where employees are working remotely, such as from home or during onsite visits, these hardware devices may not be able to capture and report data effectively to a centralized system, posing challenges for consistent energy monitoring.

In addition to the financial costs, there is an environmental impact to consider. These energy meters, being computational devices themselves, contribute to the overall carbon footprint. As previously discussed, the manufacturing, deployment, and operation of computing hardware are associated with significant emissions. Thus, relying heavily on hardware-based monitoring could inadvertently conflict with a company's sustainability goals.

Given these limitations, a more balanced approach may be warranted. One potential solution is to use a single energy meter to monitor multiple computers where feasible, thereby reducing the number of devices required. Another approach could involve using these hardware devices primarily for calibration purposes, verifying the accuracy of other, less resource-intensive tools that are discussed in subsequent sections. This combinational strategy could help mitigate both the financial and environmental costs associated with hardware-based energy monitoring while still providing the necessary granularity and accuracy in data collection.

Given the existence of such limitations, hardware-based energy monitoring is not part of the recommended framework. However, it could be incorporated into the framework in the future due to its higher levels of accuracy. The framework is flexible enough to accommodate hardware-based tools without requiring major changes. The integration of hardware-based monitoring might come at an added cost but could be utilized in parts of the company that require higher levels of data accuracy, such as onsite cloud compute units or high-performance computing applications.

5.2.2 Code based

An alternative approach for measuring energy consumption, particularly during the runtime of a software application—i.e., its usage phase—involves embedding measurement and reporting mechanisms directly into the source code. This method typically requires modifying the codebase with additional libraries designed to quantify energy usage based on metrics such as the number of instructions executed by the processor or other computational models.

While code-based approaches can provide highly accurate measurements, they introduce several challenges. Firstly, integrating these tools into the development process can increase the complexity of the software, making it more prone to errors and potential cybersecurity vulnerabilities. The added layers

of code required for monitoring can also affect the software's performance, potentially leading to slower execution times or unexpected behavior. Despite these challenges, various tools have been developed to assist with code-based energy monitoring, including *Intel VTune Profiler*, *PAPI (Performance Application Programming Interface)*, *PowerAPI*, *PyJoules*, *Energy Consumption Library (ECL)*, and *Joulemeter*. However, it is important to note that most of these tools are tailored to specific programming languages, which can limit their applicability in multi-language or heterogeneous environments.

A significant barrier to the implementation of the framework could be the acceptance by developers, especially when using code-based measurement techniques. The incorporation of these additional lines of code could negatively impact the performance and functionality of the software [29]. These additions could add time and complexity to the development cycle and increase attack surfaces related to cybersecurity threats. These factors, combined with the knowledge gap between software development teams and sustainability teams, could create unnecessary friction within the company, thus causing more problems regarding the adoption of the framework.

The presented challenges could potentially be overcome by integrating and teaching the principles of Green Software in the education received by students planning to become software developers in university. Additionally, current software developers could upskill through training sessions that highlight the implications of sustainability in software engineering. Along with fostering an environment focused on sustainability—similar to how organizations have focused on cybersecurity—this approach could boost acceptance and understanding of these tools while ensuring the performance of the end software is not degraded.

Given the significant challenges and barriers to implementation, this approach may not be practical for all companies. The complexity involved in modifying existing codebases and the potential negative impacts on software performance may deter organizations from adopting green software strategies. In some cases, the difficulties associated with this approach could inadvertently discourage companies from pursuing sustainability initiatives altogether, potentially leading to counterproductive outcomes.

To demonstrate code-based tools, Listing 2 shows a Python script. This code snippet calculates the first 100,000 Fibonacci numbers while using an energy metering tool to monitor and report the associated energy consumption. The tool provides an energy report for each execution of the software, offering valuable insights into the resource efficiency of the algorithm. In our case, the code takes approximately 3.5 seconds to execute and consumes about 5.04 Joules of energy, as reported by the code-based tool, pyRAPL [30].

5.2.3 Software based

There are many software-based tools and methods commercially available for the quantification of energy usage in a computer system. They tend to be more accessible and cost-effective compared to the previously presented methods, making them a good candidate for implementation in a corporate environment. Hence, they hold the most promise of being the tool of choice for implementations in our previously presented framework.

In this thesis, we focus primarily on the energy usage of the CPU, as the CPU is one of the most power-hungry components in a computational device, along with the GPU. The energy consumption of the CPU is important to evaluate as it directly impacts the performance and sustainability aspects of any software.

There are multiple methods of measuring the energy usage of the CPU, which can be divided into two major categories. The first category involves tools such as Running Average Power Limit (RAPL) and Performance Counter Monitor (PCM), which require low-level access to many sensitive parts of the hardware sensors and processes. These tools are typically provided by the hardware manufacturer. As these tools perform low-level actions on the hardware, they could alter parameters that might eventually harm the system, and therefore good knowledge of the tools and their usage is required. However, they provide good quality and reliable data, often with significant levels of granularity.

The second method involves estimating the power draw of the CPU using parameters such as frequency, utilization factors, and operations per second [31]. This approach tends to be simpler to implement but could provide inaccurate data, as CPU performance is based on a myriad of parameters outside the user's control. Nonetheless, it holds promise, as it could be used in conjunction with other approaches to produce a simpler yet refined output.

Several studies and tools have explored various methods for measuring CPU energy consumption. For example, Noureddine et al. [32], Kansal et al. [33], and Schubert et al. [34] provide tools and methodologies for CPU energy measurement that are implemented. Similarly, the work of Islam et al. [35] delves into the feasibility of feature-based monitoring, a technique that will be discussed in subsection 5.6. While these studies contribute valuable insights and offer extensive coverage of the topic, many of the solutions they propose are either too complex or not scalable enough for practical, widespread use in corporate IT infrastructures. This gap highlights the need for more adaptable and scalable methods that can easily integrate into corporate environments without imposing significant overhead or complexity.

To address the issues with data accuracy and access, the following section presents a practical pipeline that could be used in a corporate environment. This pipeline balances accuracy with ease of deployment while keeping scalability and access in mind. This approach is designed to align methods used in the

academic field with the constraints of the corporate environment, enabling cost-effective monitoring of energy usage on individual systems within a company.

5.3 System Utilization Method

In our approach, we have used a simple method to encourage adoption and minimize barriers to implementation. This approach uses tools such as RAPL and PCM to understand the energy consumption of a particular type of computer. This data is then used in conjunction with high-level system parameter monitoring, such as the CPU utilization factor. This avoids the need for constant low-level monitoring—an approach that might not be suitable due to the need for root access to the system, which adds bureaucratic constraints and increases the threat of cybersecurity attacks that could exploit root access. Additionally, we provide another approach that forgoes low-level monitoring but builds on tests conducted by a company on a large set of devices. This approach is presented in subsubsection 5.3.1 and could be used to generate a Proof of Concept (PoC) for the initial steps taken by the company.
In our method, we run a stress test on the system using a custom script written in *Bash* and *Python* that stresses the system to varying levels. We then measure the corresponding power draw for each unit of CPU utilization. While this approach requires root access, it only needs to be performed once for each type of computer the company is using. The data collected is then used to generate a simple relationship between CPU utilization and the corresponding power draw.
These power draw values are then translated into energy consumption, which is further converted into emissions values using appropriate emissions factors. By using this approach, we maintain the balance between ease of implementation and accuracy, providing a scalable solution for measuring and managing energy consumption across different hardware configurations.

5.3.1 TEADS curve

In situations where tools such as RAPL or PCM are not accessible, an alternative method can be employed to estimate energy consumption with reasonable accuracy. This approach is based on the Thermal Design Power (TDP) of the CPU, which is provided by most manufacturers. TDP represents the maximum thermal power that a CPU, GPU, or system-on-chip (SoC) generates, which must be dissipated to ensure the component functions properly without performance degradation. Therefore, TDP serves as a reliable indicator of the energy consumption of a component, as it is designed around this thermal constraint.
As shown in Table 2, at 100% CPU utilization, the component consumes slightly more than its TDP, with a 2% overhead. When idle (0% utilization), the component consumes approximately 12% of the TDP [36] [37]. For ease of implementation and to facilitate computation, a curve-fitting approach

CPU Utilization [%]	Power Draw [W]
0	0.12 × TDP
10	0.32 × TDP
50	0.75 × TDP
100	1.02 × TDP

Table 2: TEADS CPU Utilization vs. Power Draw [36] [37]

was applied to this data. The resulting curve fit has an R^2 value of 0.9975, indicating a strong fit. The following equation can be used to calculate the power draw of a component based on its utilization. The graphical representation of the curve fit is displayed in Figure 18.

$$\text{CPU Power Draw} = [-0.00007 \times (\text{CPU Util})^2 + 0.01603 \times (\text{CPU Util}) + 0.13967] \times \text{TDP}$$

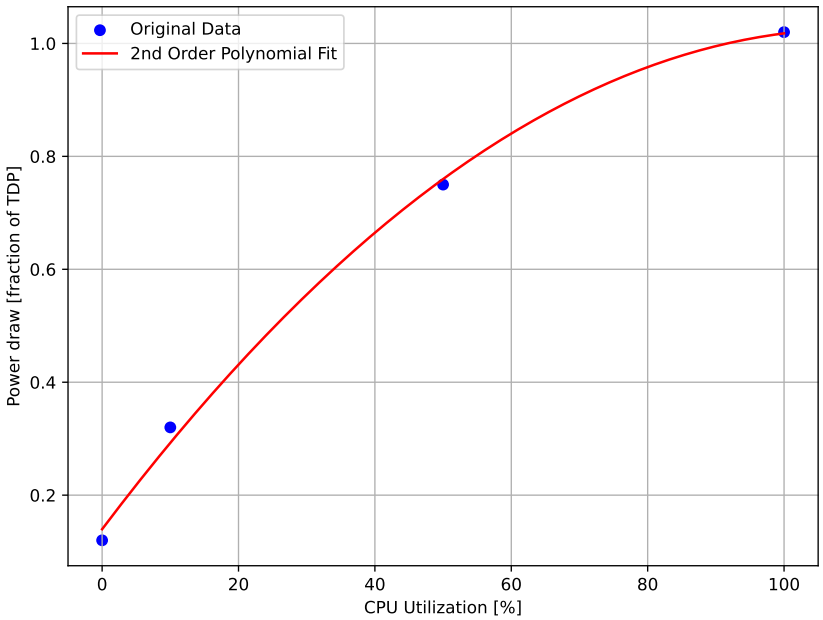

Figure 18: TEADS Power Consumption vs CPU Utilisation

To illustrate the use of the formula, consider a CPU with a TDP of 45 W and a system utilization of 35%. Using the TEADS formula:

$$\begin{aligned}\text{CPU Power Draw} &= [-0.00007 \times (35)^2 + 0.01603 \times 35 + 0.13967] \times 45 \\ &= [-0.00007 \times 1225 + 0.01603 \times 35 + 0.13967] \times 45 \\ &= [-0.08575 + 0.56105 + 0.13967] \times 45 \\ &= [0.61497] \times 45 \\ &= 27.67 \text{ W}\end{aligned}$$

Thus, at 35% utilization, the CPU would draw approximately 27.67 W. This approach offers a simple and effective way to estimate power consumption when more advanced tools are unavailable.

5.3.2 Running Average Power Limit (RAPL)

One of the most well-established and extensively studied tools for power estimation is Intel's Running Average Power Limit (RAPL). While initially designed for Intel processors, RAPL has since been extended to support other x86 CPUs. It is widely integrated into many secondary tools and is primarily used for power management and tuning of the CPU, GPU, and DRAM. Introduced with Intel's Sandy Bridge processors in 2011, RAPL has become a standard tool for monitoring energy consumption in modern computing systems.

RAPL operates by utilizing model-specific registers (MSRs) in combination with various hardware sensors to estimate power usage. The counters in RAPL track a range of events, including CPU instructions executed, memory accesses, and voltage fluctuations, converting these metrics into power estimates [38]. However, it is important to note that RAPL does not directly measure power consumption; rather, it provides an estimate based on these tracked events. This estimation process introduces the potential for reporting errors. Despite this, studies have shown that RAPL's power reporting is generally accurate, with error margins typically within acceptable ranges for most use cases [39] [40] [37].

One of the significant limitations of using RAPL is the lack of support/implementation on the Windows operating system. Although some third-party tools are available to perform this task and run RAPL in the background, they are not advisable as they require root access. One of the well-known official implementations of RAPL on Windows was the *Intel Power Gadget* application. This tool, although not extensive, could provide power usage details. At the time of writing this thesis, this tool has been deprecated and will no longer receive updates. Hence, it is recommended not to use the tool as it may have bugs or cybersecurity vulnerabilities [41].

Additionally, RAPL is not supported on the latest Apple Silicon chips, limiting its applicability in environments that rely on newer Apple hardware. In such cases, the built-in *powermetrics* tool in macOS can be utilized as an alternative. *Powermetrics* supports both Intel-based and Apple Silicon

processors, making it a versatile tool for energy monitoring across different Apple hardware [42]. However, due to the lack of access to Apple hardware, further analysis using the *powermetrics* tool could not be conducted and is therefore outside the scope of this thesis.

While RAPL remains a valuable tool for many x86-based systems, its use is becoming increasingly restricted, and alternatives such as powermetrics, PCM, or other hardware-specific tools may need to be considered depending on the operating system and hardware in use.

5.3.3 Performance Counter Monitor (PCM)

The Performance Counter Monitor (PCM) is one of the most recent tools developed to measure power utilization of system components. This open-source tool, developed by Intel, is compatible with Linux, Windows, macOS X, and BSD, but only on Intel silicon [43] [44]. PCM provides detailed insights into power consumption and performance metrics. It monitors over a hundred different parameters, making it a highly powerful and versatile tool for system analysis.

A valuable aspect of PCM is it's ability to offer comprehensive data on the monitoring of a wide range of hardware, making it valuable tool for tasks such as performance tuning and energy optimizations. This allows it to be integrated into the framework but certain challenges need to be overcome before it can fully adopted.

As PCM was released in 2023 and covers a wide range of hardware options with compatibility on various operating systems, it is a good choice of tool to be used in the framework. However, due to it being extremely new, there have not been in-depth analyses or studies to verify its data accuracy, making it a risky choice for use in the framework. To address this gap, a comparative analysis is conducted between the historically proven RAPL tool and PCM. This analysis is necessary to ascertain any deviations that might exist between the two and to evaluate the accuracy of measurements provided by PCM. If proven similar within a margin of error, it is recommended to be used in the framework due to the lower amount of restrictions it has compared to RAPL.

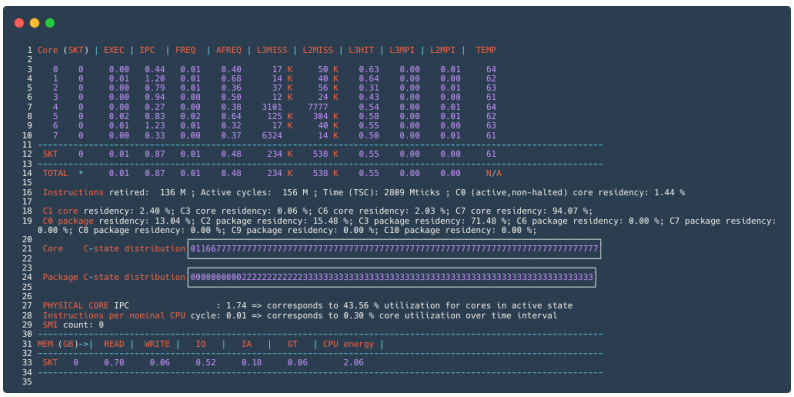

Figure 19: Output of PCM

5.3.4 PCM vs RAPL

Given the limitation that RAPL is not available on the Windows operating system and the relative infancy of PCM, it is crucial to assess the accuracy of PCM's output. To address this, a test was conducted to evaluate PCM's accuracy in comparison to RAPL. The benchmarking process and subsequent results are discussed in detail in the following sections, providing insights into how PCM performs as a reliable alternative for energy measurement.

Benchmarking

The goal of this test is to run the computer under various load conditions to cover the entire operational range of the system. The system used for this test is the Asus ROG GL553VE (manufactured in 2017, making it over 7 years old at the time of writing) with specifications including an Intel i7-7700HQ processor, 16 GB of DDR4-2400 RAM, and an NVIDIA GeForce GTX 1050 Ti Mobile GPU with 4 GB of VRAM, with a total system weight of 2.5 kg [45]. Although the system is equipped with both an integrated and dedicated graphics card, they are not used for this application, and the dedicated GPU is disabled using built-in Linux utilities.

As the stress test focuses on the CPU, it is important to understand the specifications of the CPU under analysis. The system has an Intel i7-7700HQ processor built using a 14 nm lithography process. It has a base clock frequency of about 2.8 GHz and can boost up to 3.8 GHz. It has 4 physical cores and 8 logical threads. The thermal design power (TDP) of the processor is 45 W, with a configurable reduction in TDP to 35 W. This detuned TDP is currently applicable in the laptop under analysis [46]. Additionally, a Linux distribution called Pop!OS (22.04 LTS) is being run at the time of the analysis. Running Linux on the system gives us the advantage of having more control over the operation of the system and provides easy access to a broad range of tools and packages.

One consistent observation across all tests is that the power draw never exceeds 30 W. This could be attributed to either degraded CPU performance over time or a lack of calibration of the measurement tools. Without dedicated hardware measurement devices, further investigation is beyond the scope of this thesis. However, for the purposes of this analysis, we assume the degraded performance hypothesis and consider the current TDP of the CPU to be 30W.

For the stress testing, we use the *stress* package available on most Linux distributions. A Python script is written to facilitate the benchmarking process, but it operates through terminal commands. The CPU load is incrementally increased from 0% to 100% in 10% steps. The benchmarking script for this test is provided in Listing 3 and can be modified as needed.

The script simultaneously runs PCM directly and RAPL via the PowerJoular tool to capture power measurements. PowerJoular is an application designed to monitor power consumption across different

hardware and software, supporting various architectures and configurations, including virtual machines. Due to its ease of use and reliance on RAPL in the background, PowerJoular was chosen for this analysis [47].

The measurements are saved to separate CSV files for further analysis. As shown in Figure 20, both RAPL and PCM follow the same trend over time, with their reported values being closely aligned. This suggests that PCM could potentially serve as a viable alternative to RAPL. However, given that this test was performed on only one CPU, further investigation is warranted, particularly with the introduction of real-time hardware-based power measurement tools.

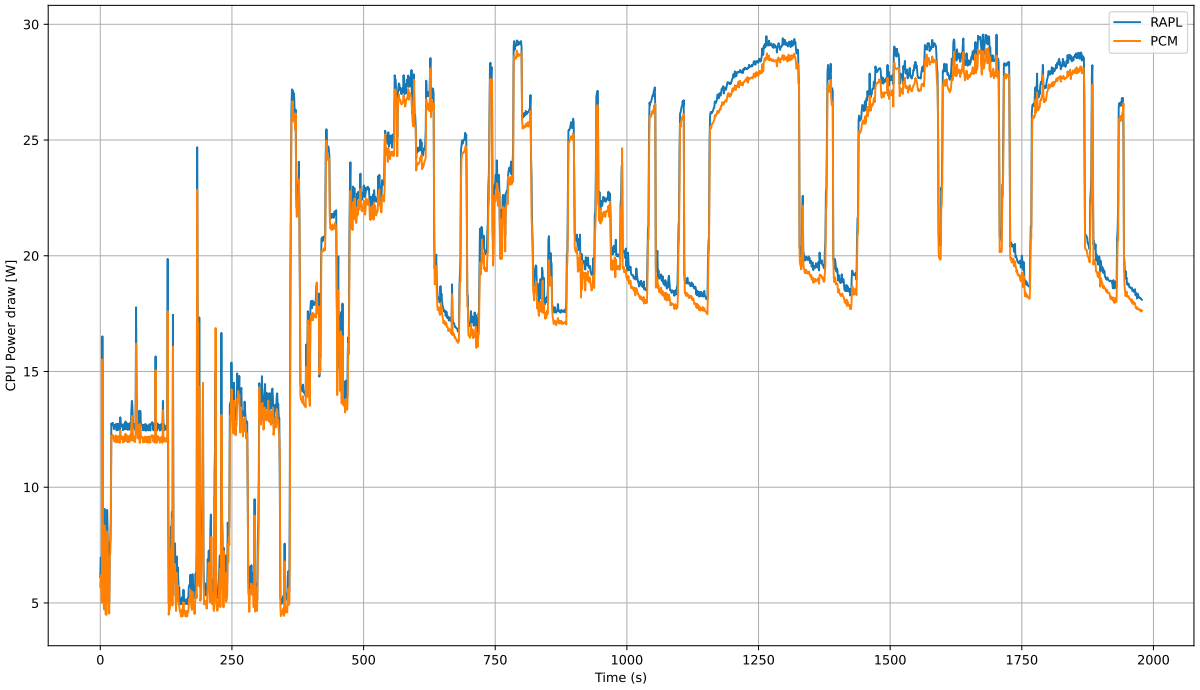

Figure 20: RAPL & PCM reporting during system stress test

A notable observation is that RAPL consistently reports slightly higher power values compared to PCM in most cases. This discrepancy is clearly illustrated in Figure 21, where the red parity line indicates equal reporting from both tools. RAPL's values tend to fall below this line, indicating that it reports slightly higher power consumption than PCM.

Despite this small deviation, the Mean Percentage Error (MPE) between the two reporting tools is only 3.47%, making the discrepancy negligible [48]. As a result, it can be concluded that RAPL and PCM could be used interchangeably, allowing users to select the tool based on their specific needs and the limitations of each tool.

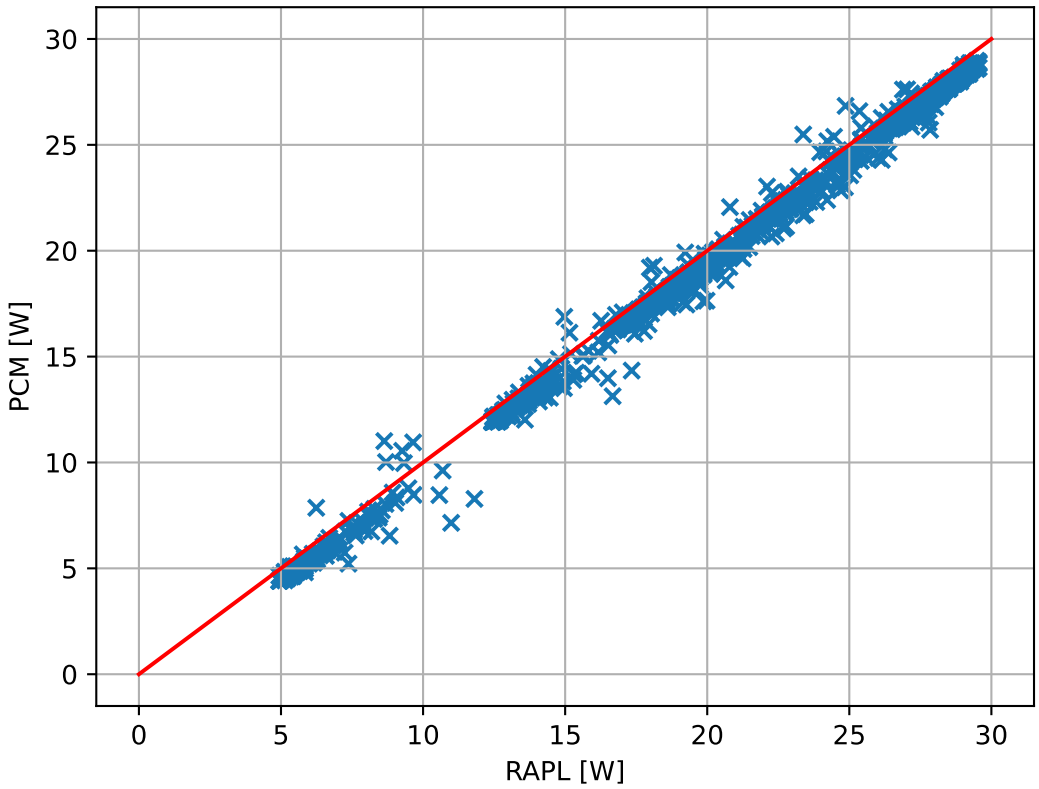

Figure 21: RAPL vs PCM reporting comparison

PCM-RAPL curve fitting

Given that we have established both RAPL and PCM as reliable tools for measuring CPU power consumption, and shown that they can be used interchangeably, we proceed to build a relationship between CPU utilization and power draw. This relationship is crucial for enabling the calculation of power consumption based solely on CPU utilization data, which simplifies energy analysis in various scenarios.

With the vast amount of data points generated during the benchmarking process, the results are grouped into 10 utilization bins to streamline the analysis. These bins range from 0% to 10%, 10% to 20%, and so on, up to 100%. For each bin, the corresponding PCM and RAPL power draw values are averaged. This binned data is then used to perform a 3rd-order polynomial curve fitting, which provides an equation that models the relationship between CPU utilization and power draw. The accuracy of the fit is validated using the R^2 value.

As shown in Listing 4, the code performs this curve fitting operation, generating the equations for both PCM and RAPL. The results are depicted in Figure 22 and Figure 23, where the fitted curves

demonstrate show good relation with the data. The R^2 values for RAPL and PCM are 0.9220 and 0.9232, respectively, indicating a strong fit. Additionally, as seen in Equation 5.1 and Equation 5.2, the coefficients of the fitted curves for PCM and RAPL are very similar, providing additional proof for the possiblity of interchangeable applications between the two.

$$\text{CPU Power}_{\text{PCM}} = 0.0001 \times (\text{CPU Util})^3 - 0.0145 \times (\text{CPU Util})^2 + 0.8326 \times (\text{CPU Util}) + 8.01 \quad (5.1)$$

$$\text{CPU Power}_{\text{RAPL}} = 0.0001 \times (\text{CPU Util})^3 - 0.0144 \times (\text{CPU Util})^2 + 0.8307 \times (\text{CPU Util}) + 8.61 \quad (5.2)$$

In subsequent sections, the derived equations will be used to estimate CPU power draw based on utilization. For the purposes of this analysis, PCM is selected as the preferred tool due to its compatibility with a broader range of computer systems and its status as the most up-to-date power measurement tool.

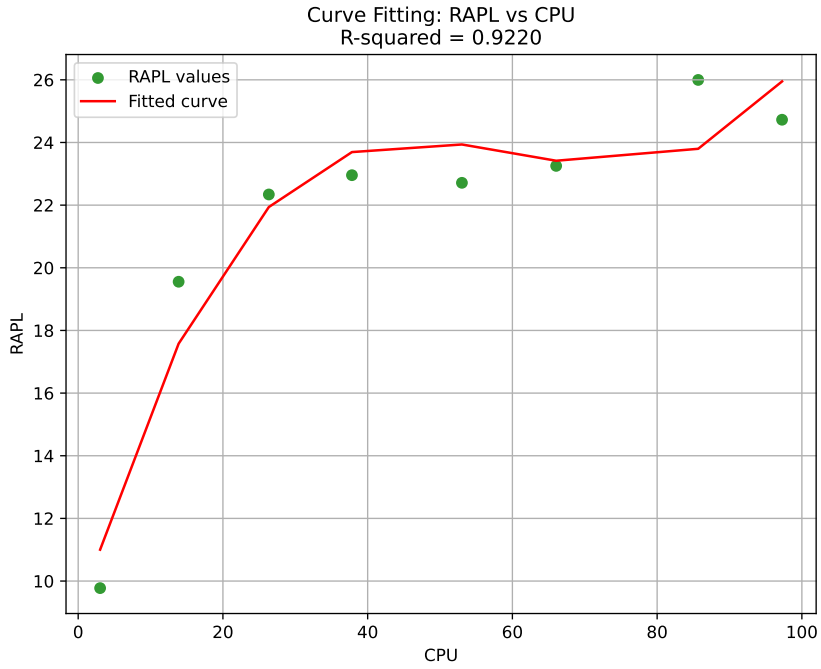

Figure 22: RAPL vs CPU fitted curve

5.4 AMD μProf

While much of the previous discussion has focused on tools for Intel hardware, there is also a need for robust tools that can handle performance and power profiling on AMD systems. Although RAPL can technically be used for both Intel and AMD processors, it is not readily available on the Windows operating system. For AMD hardware, μProf offers a comprehensive alternative to both RAPL and

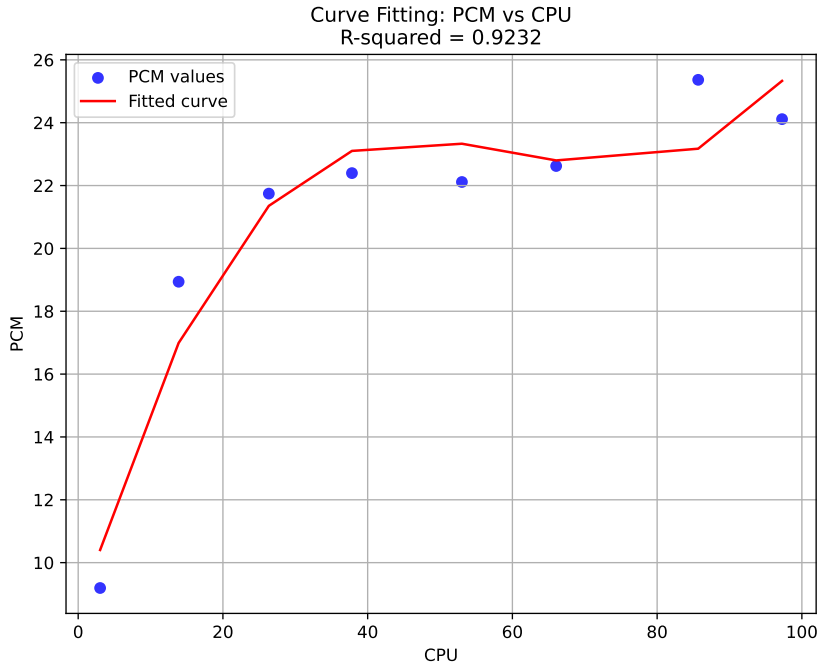

Figure 23: PCM vs CPU fitted curve

PCM. AMD μProf is compatible with Linux, Windows, and BSD [49]. However, due to the lack of access to AMD hardware, further analysis of this tool is beyond the scope of this thesis.

5.5 NVIDIA Management Library (NVML)

In addition to the CPU, the GPU is another major power consumer in a computer system. The laptop under analysis features an Nvidia 1050Ti Mobile GPU with 4GB of GDDR5 memory, built on a 14 nm process. This GPU has a thermal design power (TDP) of 75W, making it more power-hungry than the CPU in the system.

For analyzing GPU power consumption, NVIDIA provides the NVIDIA Management Library (NVML), a C-based API that can be accessed via the *nvidia-smi* tool. NVML enables comprehensive monitoring and management of NVIDIA GPUs, including power draw, temperature, and memory usage. However, due to technical limitations with *nvidia-smi* on this specific laptop, further analysis using NVML is not possible.

The methodology used for CPU benchmarking can be extended to GPU benchmarking by employing tools such as *nvidia-smi* alongside stress testing software like the *GPU Stress Test* tool available at `https://github.com/NVIDIA/GPUStressTest`. This approach would offer deeper insights into the power consumption patterns of GPUs under load, contributing to more accurate power profiling. Given that machine learning and artificial intelligence tasks heavily rely on GPUs, such an analysis

becomes increasingly important for understanding and optimizing GPU performance in high-compute environments.

5.6 Use phase accounting

The previously presented methodology is highly suitable for the software development phase, which accounts for Scope 1 and Scope 2 emissions. However, since Scope 3 emissions are a significant part of the company's reporting, the use phase of the software must also be considered. Software is run on a wide range of devices, each with varying hardware configurations, making it challenging to account for these emissions precisely. Nonetheless, customer surveys or available market data, such as the *Steam Hardware & Software Survey* (`https://store.steampowered.com/hwsurvey/`), can be used to estimate the most common systems on which the software will likely run. After determining an optimal hardware configuration, the software can be tested on this system, and the previously introduced methodology can be applied to compute the Software Carbon Intensity (SCI) score.

Though the emissions accounting methodology remains largely the same, the functional unit may need to be adjusted. It is critical to define this new functional unit, as it differs from the one used in the development phase. This functional unit can then be utilized by customers to calculate their own Scope 1 and Scope 2 emissions associated with running the software in their environments.

Below are several potential functional units that could be used during the software's use phase:

- **Hourly usage**
  This is a high-level functional unit that offers a basic approximation of the software's carbon footprint. A comprehensive test case can be designed to run all the software's features, and the total runtime can be averaged. This approach can also help detect hotspots in system utilization during certain functions, which could provide opportunities for optimizing the code in future development cycles.
  While this method may not offer precise results, it provides a good starting point. Many assumptions, such as customer location, usage patterns, and hardware configurations, may need to be considered.
- **Per subscription or user**
  This is another broad functional unit that can assist the software producer in assessing Scope 3 emissions. It is particularly useful when software is provided as a service (SaaS), where the majority of operations are conducted in cloud environments. Given the extensive use of cloud computing and data centers, numerous methodologies and studies have been published in this field [50] [51] [52].

- **Per API call**

  This functional unit is highly applicable to cloud computing environments. A large number of communications occur between software running on local hardware and cloud-based servers. Both software producers and users can benefit from using this metric, as it provides insight into data transmission, pre-processing, post-processing, and other operations.

  The work of Baliga et al. (2011), in their paper *Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport*, provides a methodology for calculating energy consumption in cloud environments, offering valuable insights for this analysis [52].
- **Per function of the software**

  This is a more detailed but potentially cumbersome functional unit. It can be especially useful in detecting runtime hotspots in software that operates in energy-constrained environments, such as pacemakers or other field instruments.

  The work by Islam et al. introduces a tool called *Parallelized Observation-based Slicing tool* (PORBS), which uses system-level observations to measure the energy consumption of individual software features [53][54]. Although slicing every feature may be impractical, grouping related features together for analysis could provide useful insights for energy optimization.

5.7 Developed measurement and calculation tools

For the subsequent implementation of the previously discussed methodology, a set of custom tools was developed to capture system specifications and monitor the utilization of various components. The code is predominantly written in Python, ensuring broad accessibility and ease of use.

The overall codebase is divided into three primary components. The first component handles the logging of system utilization data and hardware information. This is the most comprehensive part of the code, as it continuously runs in the background, designed to minimize its own computational footprint. The captured data is stored efficiently in an SQLite database for easy access and analysis. The timestamp is stored in UTC, this is to make computation easier, especially across different timezone. This data logger is detailed in Listing 5. An example of logged data can be seen in Table 3.

The second component is responsible for converting the collected utilization data into power draw and energy consumption values. In the example code shown in Listing 6, the TEADS curve approach is demonstrated; however, the equations derived from RAPL and PCM, presented in Equation 5.2 and Equation 5.1, can also be employed by modifying the relevant function in the code. To ensure data integrity, this script reads from the original SQLite database and creates a duplicate table for any modifications, preserving the original dataset.

The third component, illustrated in Listing 7, calculates the emissions associated with the recorded

energy consumption. In this example, a single emission factor from the *Ecoinvent* database is used, corresponding to the India-Western Grid. However, the code is flexible and can be easily adapted to include other static emission factors or dynamic carbon intensity values, depending on the geographic or operational context.

Timestamp	CPU %	GPU %	Mem %	Net Sent	Net Recv
2024-07-31 06:19:28	19.6	1	24.1	1485	10513
2024-07-31 06:19:29	45.3	0	24.2	1485	10513
2024-07-31 06:19:30	29.9	1	24.3	1485	10513
2024-07-31 06:19:31	46.6	3	24.5	1485	10513
2024-07-31 06:19:32	39.3	1	24.2	1485	10513

Table 3: Snippet of logged data

6

Proof of Concept

Building upon the research, methodologies, and tools discussed in previous sections, this chapter seeks to validate the approaches by conducting analyses through two distinct case studies. The first case study focuses on the environmental impact during the Software Development phase, while the second explores the carbon footprint associated with the Software Usage phase of an example application. The case studies offer insights into the potential emissions and energy usage across different stages of the software lifecycle. To deepen the understanding and applicability of these findings, the analysis is further expanded to different applicable scenarios, which evaluates the implications of various operational conditions that a software might face. This analysis helps formulate recommendations for improving both reporting mechanisms and carbon accounting methods, particularly for Scope 3 emissions — those that occur indirectly, such as through the usage of a company's software by external parties.

6.1 Case 1: *Software Development*

The first demonstration of the developed methodology is applied to the software development cycle by monitoring the system's utilization data, similar to what is presented in Table 3. This case study reflects the typical activities of a software developer during a standard workday, which begins at 09:00 and concludes at 18:30. The data was collected in an office setting located in Pune, India. To ensure accuracy in the calculation of emissions, region-specific emissions factors were applied to this analysis, making the results more representative of real-world conditions.

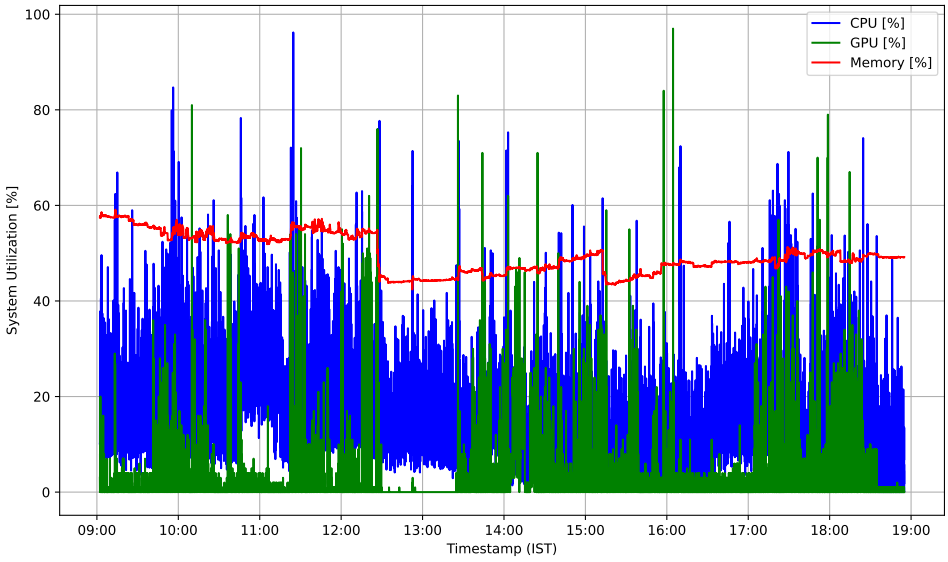

Figure 24: Typical System Utilization during a workday

The system's utilization throughout the workday is depicted in Figure 24. Throughout the day, a range of activities were performed, including web browsing, reading PDF documents, software development tasks, media playback, Microsoft Teams calls, and creating presentations using Microsoft PowerPoint. One noteworthy observation is the near-zero GPU utilization recorded between 12:30 and 13:30, which corresponds to the lunch break. During this period, although the screen was turned off, background processes continued running, explaining the ongoing but reduced system activity.

Results

Using the logged utilization data, the power draw for both the CPU and GPU is calculated based on the methodologies discussed in previous sections. These power values are then converted into energy consumption, which is subsequently used to estimate the associated emissions. For one workday, the total emissions were calculated to be approximately 329.03 gCO_2eq.
To extrapolate these findings for a longer period, such as a typical Agile software development sprint (which usually lasts 30 days [55]), the emissions can be scaled accordingly.

$$\begin{aligned} 329.03\,gCO_2eq/day \times 30\,days &= 9,870.9\,gCO_2eq \text{ per developer per sprint} \\ &= 9.87\,kgCO_2eq \text{ per developer per sprint} \end{aligned}$$

This calculation results in approximately **9.87 kg CO_2eq of emissions per developer per sprint**. This extrapolation offers valuable insight into the carbon footprint generated by a single developer during routine software development activities, underscoring the environmental impact of day-to-day tasks. These findings can be further scaled to estimate the total emissions for an entire development team, providing a broader perspective. Additionally, monitoring system utilization on each developer's machine would lead to more precise and accurate results, enhancing the granularity of the analysis.

Table 4: Key Assumptions of Software Development Emission Calculations

Assumptions	Developer working from 09:00 – 18:00
	Only CPU & GPU Utilisation measured
	As office is located in Pune, India; Carbon Intensity of Western Grid, India is used
	Developer is only using the Laptop and no cloud computing or HPCs
	Each Agile sprint is 30 days

6.2 Case 2: *Software usage*

Another critical phase in the Software Lifecycle is its usage phase. This phase is typically difficult to quantify, not just for software but for most physical products as well, due to the fact that many factors affecting the performance and impact of the product are beyond the control of the manufacturer or producer. These external variables, such as user behavior, hardware configuration, and environmental factors, can significantly influence the software's energy consumption and carbon footprint. The challenges associated with quantifying the use phase have been extensively discussed in subsection 5.6. Despite these challenges, a test case software was analyzed to demonstrate the applicability of the developed methodology and tools.

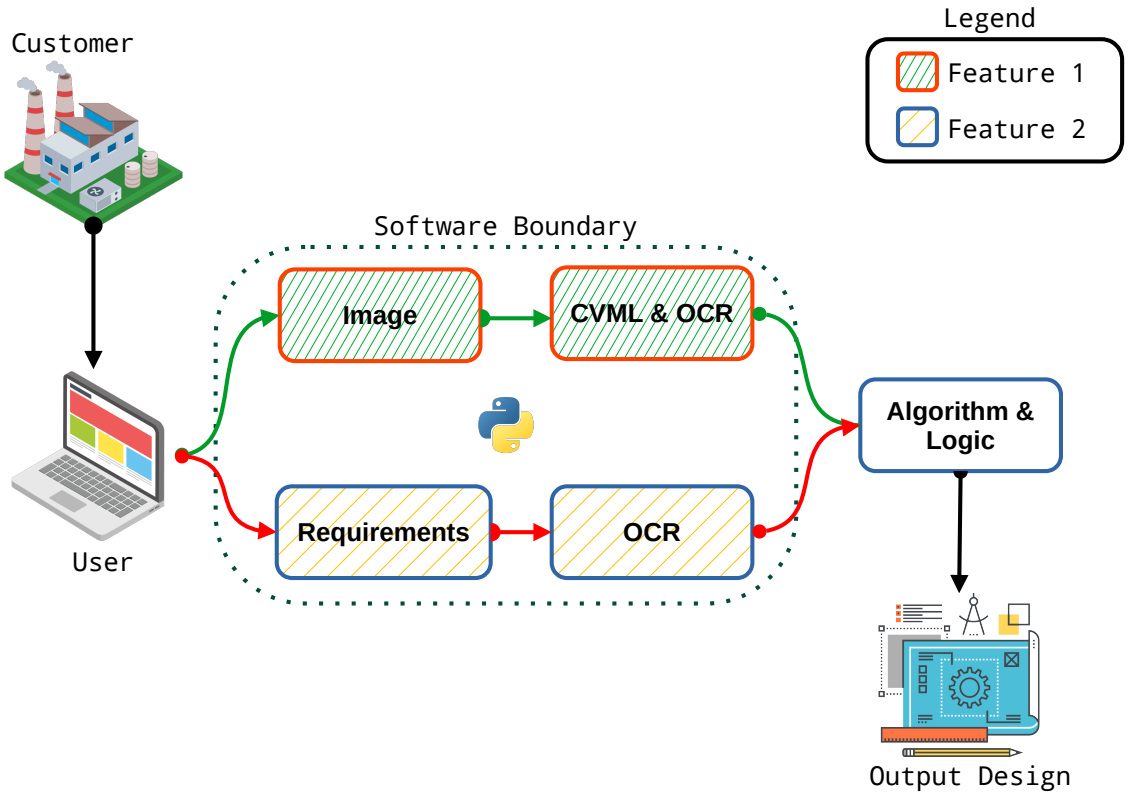

Figure 25: Software Functioning Overview

The function of the software under analysis is to generate a product design based on a basic drawing and a set of customer requirements. The software utilizes advanced techniques such as Computer Vision and Machine Learning (CMVL) as well as Optical Character Recognition (OCR) to analyze the image of the drawing provided by the customer. Additionally, OCR is used to process and extract the customer's specific requirements. This dual functionality divides the software into two core features: one that processes the image and another that processes the textual requirements.

For the purposes of this analysis, each feature of the software was evaluated independently. Although this approach is time-consuming and less practical in a large-scale corporate environment, it offers a high level of granularity and serves as a proof of concept for less complex analysis, as seen in subsection 5.6. The graphical representation in Figure 25 illustrates the software's functioning, with a clear distinction drawn between its two major features—image processing and requirements processing. This level of analysis, though detailed, is highly beneficial for evaluating the energy consumption and performance of individual software components, making it a useful method for more focused or specialized studies.

Test Setup

To ensure the accuracy of the captured data, multiple considerations were taken. Firstly, all the unnecessary background processes were terminated to prevent additional load on the CPU. This would ensure that the captured data reflected the actual implications of running the software on the system. In addition to killing all the unnecessary background processes, the system was isolated from any networks, to prevent the triggering of any other processes, which would in turn impact the results. Additionally, all other external devices such as monitors, mice, Bluetooth peripherals, headphones, etc., were also disconnected to remove any overheads associated with the functioning of these devices. For example, when external monitors are connected, the GPU is generally responsible for rendering tasks. In addition to these considerations, a considerable cool-down period between consecutive tasks was provided to ensure that the system reached its stable state.
Since the code was written in Python, it could initially be executed as a script, provided all necessary dependencies were manually installed by the user. While this method of running the software was functional, it was not particularly user-friendly or practical for widespread use. As a result, the software was packaged into an executable file that could be easily run by users without the need to manually install and manage dependencies. Although packaging the software as an executable limits the ability to modify the source code, it mirrors the typical distribution method of software to end users. This aspect of the test setup presented an interesting comparison—assessing the performance differences between running the software as a Python script and as a packaged executable.

Feature 1 – Script	Feature 2 – Script
Feature 1 – Executable	Feature 2 – Executable

Table 5: Test scenarios - Software Usage

Consequently, four distinct test scenarios were established for analyzing the software, as outlined in Table 5. Each test scenario was repeated five times, resulting in a total of 20 analysis runs. This comprehensive testing framework allowed for a robust assessment of the software's performance,

providing detailed insights into potential variations in CPU utilization and system performance when running as a script versus an executable.

Results - Runtime

The four distinct test cases have been consolidated into two graphs, which are presented in Figure 26 and Figure 27, representing *Feature 1* and *Feature 2*, respectively. Each graph includes the results for both the Script and Executable versions of the software, allowing for direct comparison between the two.

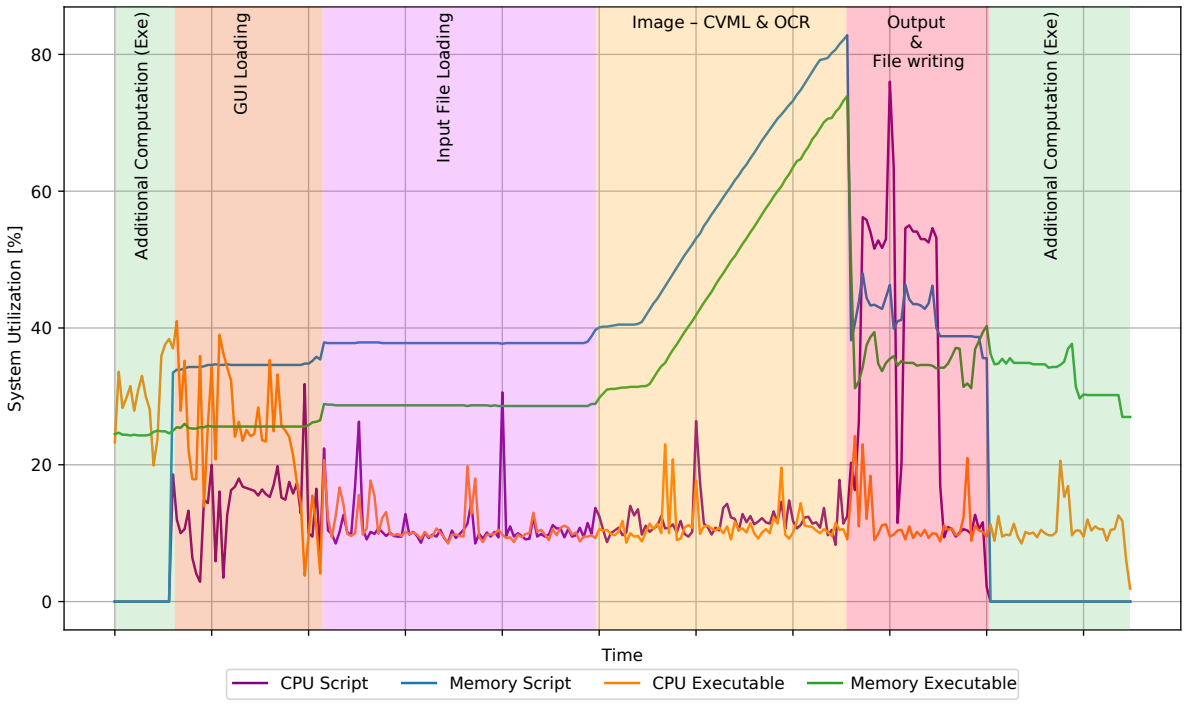

Figure 26: Feature 1 – System Utilization

A more detailed breakdown of the results shown in Figure 26 is necessary due to the wide variety of functions being performed, which allows for easier differentiation between tasks. As mentioned earlier, the power usage of the system memory is not a significant factor in overall energy consumption; however, observing memory utilization helps in identifying the functioning stages of the software. Memory usage is employed to delineate four distinct phases of the software's operation, highlighted in Orange, Magenta, Yellow, and Red in Figure 26. These phases correspond to specific tasks: loading the GUI, reading and processing the input file, performing image processing using CVML and OCR, and generating outputs and writing files, respectively. Additionally, two Green zones at the beginning and end of the graph represent the extra processing time required by the executable to execute the same set of tasks compared to the script version.

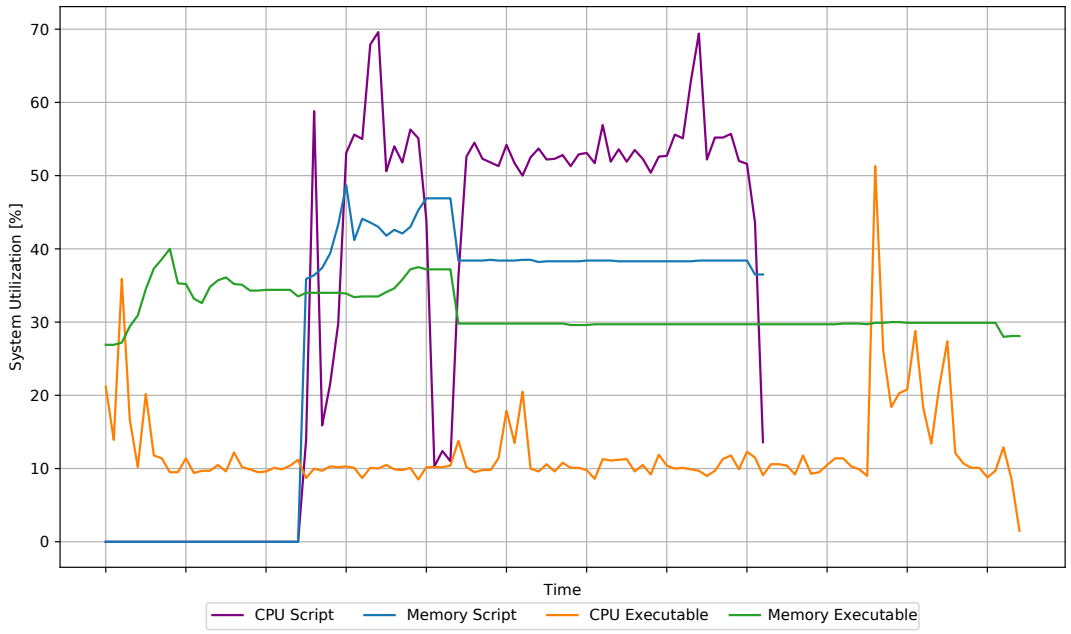

Figure 27: Feature 2 – System Utilization

Several notable observations emerge from the results:

- **Higher memory usage during Script runs:**
  The consistently higher memory usage observed during Script runs is likely due to the fact that all script dependencies are loaded into memory to facilitate quicker access by the software during execution.
- **High initial CPU usage during Executable runs:**
  Since the executable version of the software is packaged along with all necessary dependencies, the initial spike in CPU usage can be attributed to the unpacking and preparation of these dependencies before the core tasks of the software are executed.
- **Stable CPU usage in Executable runs after the initial spike:**
  Once the executable has unpacked the dependencies, it operates using precompiled machine code or bytecode. Unlike the script version, which has to convert each line into bytecode or machine code at runtime, the executable's precompiled code results in a more stable and lower CPU usage after the initial unpacking phase.
- **Erratic and higher CPU usage during Script runs:**
  As Python is an interpreted language, each line of code is converted into bytecode or machine code at runtime. This process leads to periodic spikes in CPU usage whenever new code segments are interpreted and executed, resulting in higher and more erratic CPU utilization during Script runs.

- **Longer computation times for Executable runs:**
  The executable version generally has longer computation times compared to the Script version. This is likely due to the overhead involved in packaging libraries and interacting with OS-specific dependencies. Moreover, there may be opportunities to optimize the packaging process itself, which could help reduce computation times. On average, the executable takes approximately one minute longer to complete the same tasks as the Script version in both Feature 1 and Feature 2.

These findings demonstrate that monitoring software during runtime not only provides valuable insights for emissions calculations but also offers technical insights that can inform code optimization efforts. By analyzing the performance differences between the Script and Executable versions, this study highlights key areas where improvements in execution efficiency can be made, contributing to both environmental sustainability and enhanced software performance.

Results - Emissions

When analyzing the emissions associated with the software's usage, various methodologies can be employed. One of the simplest approaches is to use annual emission factors, which are readily available for different regions through reputable and well-established databases such as Ecoinvent. While this method has the advantage of being straightforward to implement, it risks inflating the estimated emissions, as illustrated in Figure 28 and Table 6. This overestimation could negatively impact efforts to make Software Sustainability reporting more widely accepted in the industry. When using annual aggregated data, emissions can be over-reported by as much as 2.9 times the actual value, as seen in the comparison between the *Ecoinvent (IN-WG)* column and the *India - Day 1* or *India - Day 2* columns in Table 6 or in the corresponding bar graph in Figure 29.

As shown in Table 6, on average, emissions from the Executable version of the software are observed to be approximately 1.1 to 1.2 times higher than those produced by the Script version. This result is noteworthy, given that the CPU utilization for the Script version was consistently higher than that of the Executable. However, the overhead associated with running the Executable—such as unpacking dependencies and additional runtime processes—appears to increase both the emission factor and the overall runtime, ultimately leading to higher emissions.

Another key consideration is that software can be used at any time and in any location, provided certain operational conditions are met. To capture this variability, the software was run in India on two different days and in France on two different days, as shown in Figure 28. In India, emissions were not only higher but also exhibited significant fluctuations during runtime. This variability is likely due to the lower share of renewable energy in the Indian grid and the frequent shifts between different electricity generation technologies. In contrast, the emissions in France were consistently lower and

Emissions [gCO2eq]	Ecoinvent (IN-WG)	India - Day 1	India - Day 2	France – Day 1	France - Day 2
Feature 1 – exe	1.929	0.661	0.685	0.052	0.104
Feature 1 – script	1.754	0.618	0.617	0.045	0.096
Feature 2 – exe	0.804	0.287	0.279	0.015	0.034
Feature 2 – script	0.646	0.223	0.225	0.009	0.024
Total – exe	2.733	0.948	0.964	0.067	0.138
Total – script	2.399	0.840	0.842	0.054	0.120

Table 6: Emissions in gCO2eq for different regions and scenarios

more stable, owing to the country's heavy reliance on nuclear energy, which produces relatively low and steady carbon emissions. All live emissions data was sourced from the *Electricity Maps* platform (`https://www.electricitymaps.com/data-portal`), which provides real-time data on carbon intensity and grid composition.

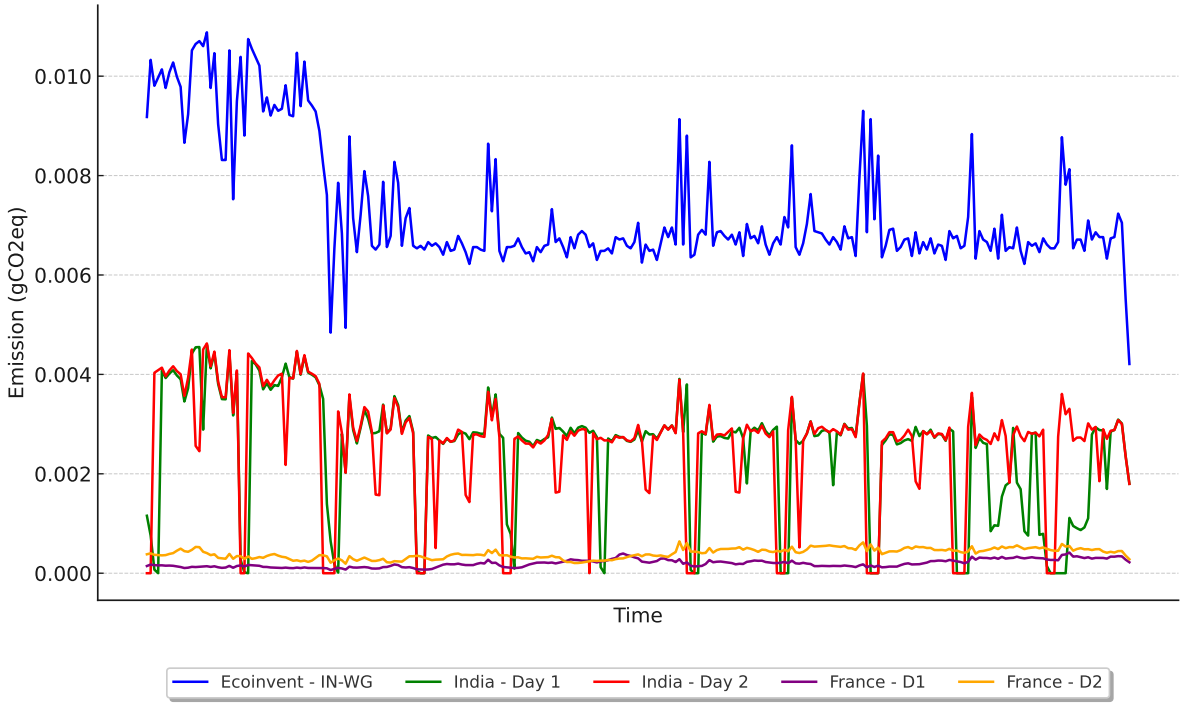

Figure 28: Emission pattern

Emissions Reporting

The reporting of emissions data is critical not only for the software-producing company, as it contributes to their Scope 3 emissions, but also for the firms using the software, as it forms part of their Scope 1 and 2 emissions. Therefore, a clear, transparent, and easily interpretable method of conveying this data is essential.

To address this, we adopt a method commonly used in the financial sector—candlestick charts. These charts are well-suited for emission reporting, as they effectively convey trends and patterns, making it

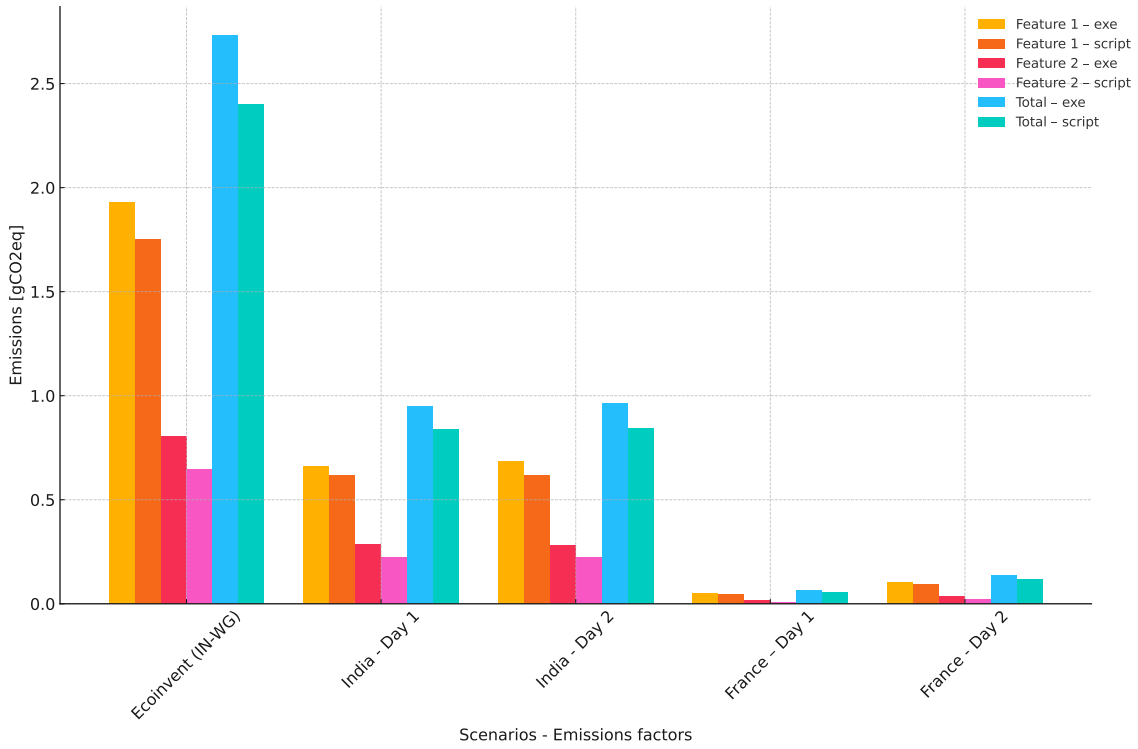

Figure 29: Emissions during Use phase (Scenarios)

easier to communicate both carbon intensity and total emissions associated with the software's usage. Additionally, candlestick charts allow for the representation of data at various levels of granularity, which can enhance both reporting and accounting processes.

For the purposes of this thesis, we present monthly emissions and carbon intensity data using this approach. Unlike the traditional red-green color scheme used in finance, we opt for a black-green theme to align with the environmental context. In this scheme, an increase in carbon intensity or emissions is represented in black, while a decrease is shown in green. A detailed explanation of how to use and interpret candlestick charts is provided in Appendix D.

For our emissions reporting, we assume the software is run approximately 100 times each month. Using hourly carbon intensity data obtained from *Electricity Maps*, we generate Figure 33 and Figure 34 to illustrate carbon intensity trends in India and France. These charts can then be used to calculate monthly emissions, based on the assumption of 100 software runs per month.

As shown in Figure 30 and Figure 31, we calculate the emissions associated with 100 runs of the software each month in both India and France. Due to the flexibility of candlestick charts, it is possible to determine not only the average emissions but also the best-case and worst-case scenarios for each month. These charts also reveal trends and seasonal patterns in emissions, which companies can leverage to optimize their strategies for running the software more sustainably.

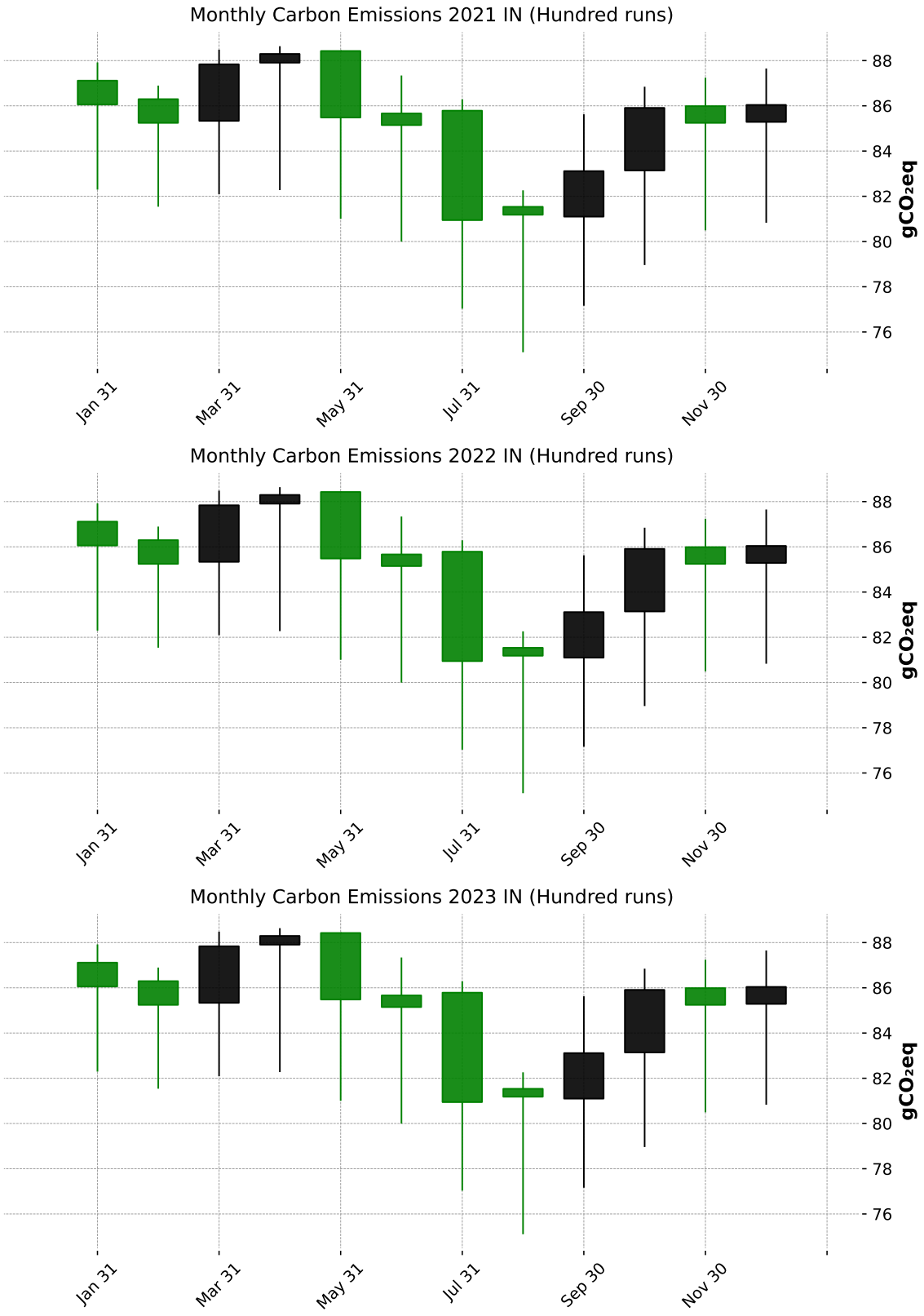

Figure 30: Monthly Carbon Emission (India) per hundred runs

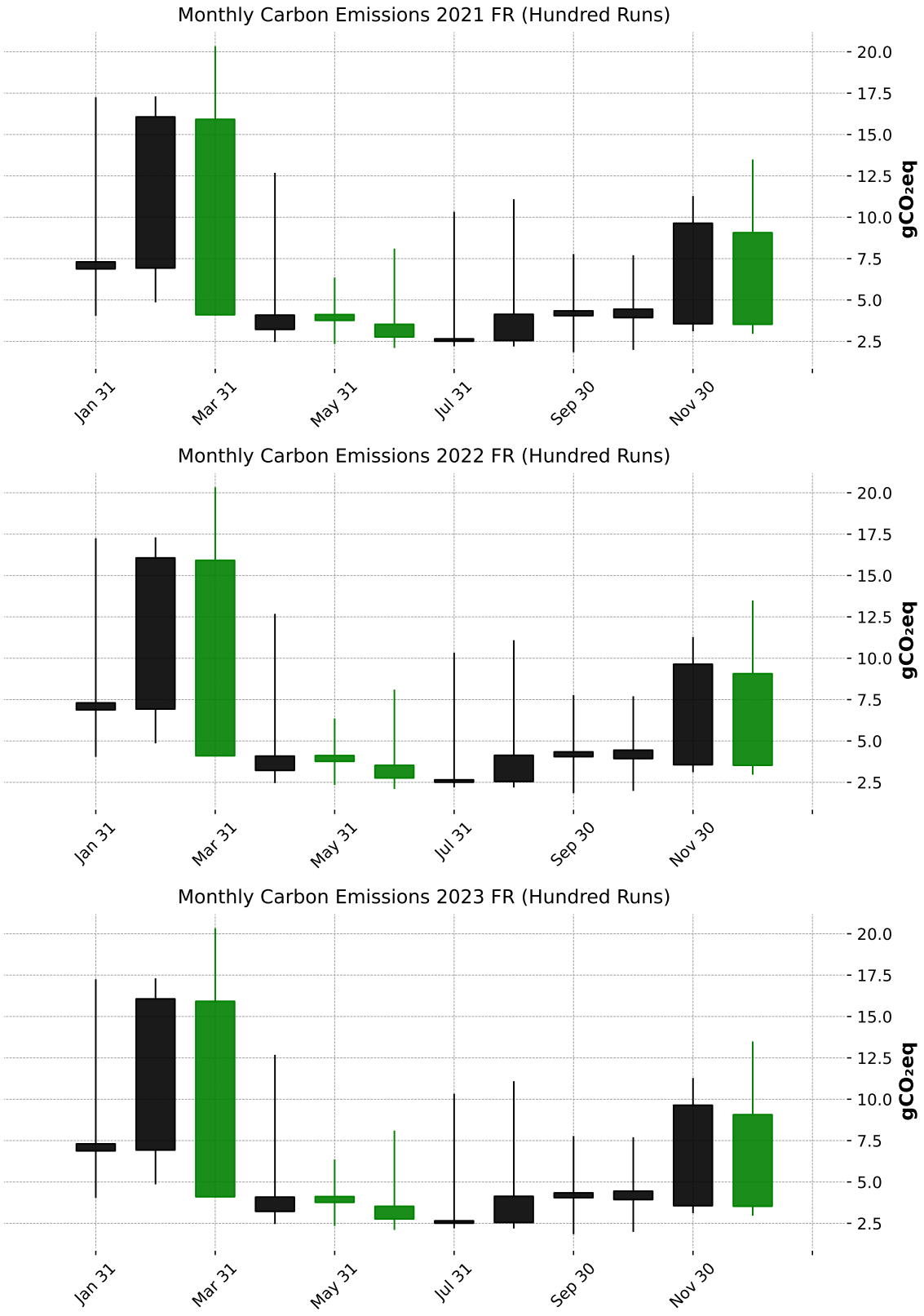

Figure 31: Monthly Carbon Emission (France) per hundred runs

Overall, emissions in India are significantly higher, with peak values approaching 90 gCO_2eq per hundred runs, compared to a peak of around 20 gCO_2eq in France. The data further reveals consistent seasonal trends: emissions in India tend to dip in July, September, and October, which aligns with the end of the summer season. In contrast, emissions peak in January, February, March, November, and December, corresponding to colder periods that increase energy demand, thereby raising carbon intensity in the electricity grid and contributing to higher emissions.

As illustrated in Figure 30, the maximum monthly emissions in India per 100 software runs is less than 90 gCO_2eq, compared to 239.9 gCO_2eq per month when using Ecoinvent database values, which apply a flat annual carbon intensity of 1,588.4 gCO_2eq/kWh. Furthermore, the Ecoinvent data reflects figures are only accurate for the years of 2020 and 2021, which may no longer be accurate and could lead to significant overestimations of emissions in contemporary settings [56].

Using outdated, flat annual data inflates emissions estimates, potentially discouraging companies from undertaking evaluations—particularly in newer, less established fields such as software development and usage. Implementing a candlestick chart-based approach for emissions reporting provides both software producers and consumers with enhanced transparency, greater accuracy, and the flexibility needed for more precise accounting and reporting. This method empowers stakeholders to make more informed decisions and fosters a more sustainable approach to software operations.

7 Conclusion

With the increasing integration of software in our daily lives, it has become an essential part of the global energy landscape. As software grows in scope and complexity, it contributes significantly to energy consumption, making it a critical target for sustainability efforts. Software is not only a consumer of resources but also a powerful tool for driving eco-friendly solutions and supporting global sustainability goals. Therefore, it is crucial to understand its environmental impact to prevent unintended side effects. Corporations are now under increasing pressure to provide accurate and reliable accounting and reporting for their environmental impacts. In response, this thesis has developed a straightforward yet comprehensive framework that integrates established industrial practices and key metrics like Software Carbon Intensity (SCI). This framework offers a methodology for high-level hardware monitoring during software development and deployment phases. Additionally, tools such as RAPL (Running Average Power Limit) and PCM (Performance Counter Monitor) are employed only for the initial calibration of the system utilization method, ensuring that the final measurements are precise and scalable.

This work also references a wide range of accessible data sources, ensuring that companies of various sizes can access relevant and actionable information. A proof of concept was successfully demonstrated for both the development and usage phases, highlighting the variance in emissions and offering insights into optimization opportunities.

A novel emissions reporting method using candlestick charts was introduced, providing a more accurate and user-friendly way to represent emissions data. Moving beyond overly simplistic methods, these charts allow companies to quickly identify trends, peaks, and dips in emissions, enhancing transparency and precision in environmental accounting. This thesis not only contributes to the measurement and reporting of software emissions but also lays the groundwork for broader industry acceptance of these methods.

8

Future work

Given the emerging nature of software sustainability, there are several areas that require further exploration to improve the accuracy of measurements, facilitate broader adoption, and prevent overreporting. Some key areas for future work include:

- **Cross-platform Standardized Tools:** Development of standardized tools for measuring system energy usage across different operating systems without requiring root access. This would increase accessibility and ease of adoption for a broader range of users and developers.
- **Enhanced CPU Metrics:** With recent advances in silicon architecture, CPU utilization alone no longer captures the full picture of energy consumption. A new, architecture-agnostic metric for CPU usage is needed, which would apply universally regardless of the CPU manufacturer or silicon architecture.
- **Transparency in Product Carbon Footprint (PCF) Assessments:** Increasing transparency in the PCF assessments of ICT devices could improve the accuracy of reporting and enhance innovations in reduction of the carbon footprint of hardware components. It would also empower ICT manufacturers to design more environmentally friendly devices.
- **Intel® PCM Validation:** A more in-depth analysis of Intel PCM through hardware monitoring devices and other models could increase the reliability and accuracy of PCM's reporting capabilities, encouraging wider industry adoption.
- **Water Usage:** Whilst having been previously introduced and explained in the thesis, the water footprint of software is an unexplored domain. It could have very high implications especially in regions with lack of water access. A study in this domain would not only improve the environmental implications but also heavily show the social implications.
- **Other Environmental Impacts:** Beyond Water Usage, other environmental impact categories must also be analyzed. While these categories are established in various LCA implementations, they remain underexplored within this domain. This is particularly crucial for the hardware component of the IT sector, which predominantly reports only the carbon footprint. As computers continue to evolve, utilizing a wide array of elements sourced globally, their influence on these impact categories will become increasingly significant, warranting dedicated attention.

This field of study is evolving, and continuous improvements in both methodology and technology will be essential to fully integrating sustainability practices within the software industry.

A

Appendix A

SOFTWARE COMPANY				
DIRECT	SCOPE I	OFFICE		Direct emissions related to an office. Includes office building's energy efficiency defines how much energy its heating and AC use. Its location is also a factor in the commuting for employees.
		VEHICLES		Direct emissions from company's vehicles. Includes company owned vehicles used in its business activities, such as sales and tech support are included here.
		EQUIPMENT		Direct emissions from equipment. Includes all equipment owned by the company and used on premises are included here. This covers the infrastructure and the user devices.
INDIRECT	ENERGY	SCOPE II	ELECTRICITY	Emissions from purchased electricity. Includes the source of the electricity used in a company's own business operations.
			NETWORK	Emissions from network traffic. Includes the amount of traffic and types of connections used in a company's business operations.
			COOLING	Emissions from AC and equipment cooling. Includes the method of cooling offices and equipment in warmer climates and warm seasons.
			HEATING	Emissions from space heating. Includes the method of heating offices in colder climates and cold seasons.
	OTHER	UPSTREAM	TRAVEL	Emissions from work travel. Includes business travel connected to operations, the method of travel used and amount of travelling done.
			COMMUTING	Emissions from employee commuting to an office. Includes the method of commuting used and the amount of days spent working in the office.
			LOGISTICS	Emissions from material logistics. Includes any transport of equipment and materials by a company's order.
			LEASED ASSETS & AAS	Emissions from leased equipment, vehicles and other assets, as well as those purchased as a service.
			SUPPLIES	Emissions from supply chains of office supplies that are needed in everyday business operations. Includes small supplies and fumiture.
		SCOPE III	BUILDING AND MANUF. SCOPE 1 ASSETS	Emissions from the first part of the life cycle of buildings, vehicles and equipment and the relevant supply chains therein.
			WASTE	Emissions from waste processing. Includes all waste produced by any of the company's staff in office.
			SERVERS & CLOUD	Emissions from leased or aaS virtual assets. Includes any third party data centre or cloud service used in the operations of a company.
		DOWNSTREAM	MARKETING	Emissions from marketing activities. Includes any marketing done by the company, whether digital or physical and also third party marketing partners.
			DISTRIBUTION	Emissions from distribution of products. Includes all methods of distribution, whether digital or physical media.
			RETAIL & FRANCHISE	Emissions from retailers and franchise holders. Includes any third party retailer or franchise selling the software.
			TECH SUPPORT	Emissions from tech support activities. Includes all forms of support from automated bots to chat and phone service to support in customer location.
			PRODUCTS	Emissions from product use by the customers.
			INVESTMENTS	Emissions from financial assets held by a company.

Table 7: Software Company Emissions scopes [9]

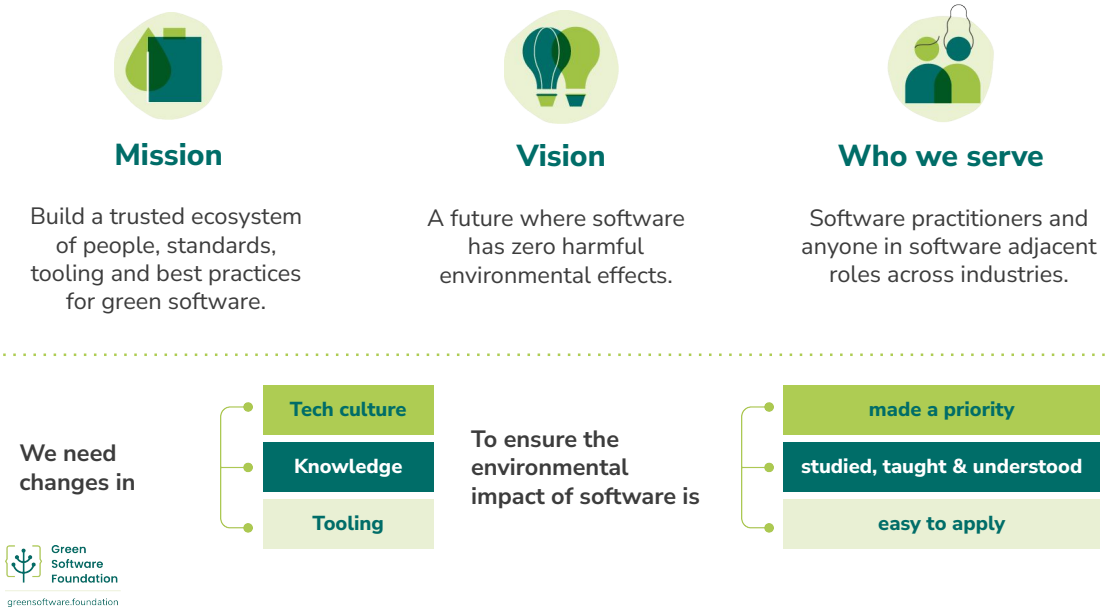

Figure 32: GSF manifesto [57]

A.1 Checklist for Implementing Software Carbon Intensity (SCI)

The following checklist serves as a guideline for companies to incorporate Software Carbon Intensity (SCI) into their software development cycle and emissions accounting processes:

- ☐ Identify the stages of the Software Life Cycle in which SCI scoring will be implemented.
- ☐ Define the Functional Unit, **R**, to be utilized across the various stages.
- ☐ Project Managers and Software Architects should establish specific SCI targets for each phase of the software life cycle.
- ☐ Select the appropriate tools for device monitoring to ensure accurate SCI measurement.
- ☐ Calibrate the chosen measuring software/tools using validated methods, such as hardware monitoring devices or tools like RAPL, to ensure accuracy.
- ☐ Determine the resolution and granularity required for Carbon Intensity and acquire the relevant Carbon Intensity data.
- ☐ Establish reporting methods that facilitate the customer's ability to conduct their carbon accounting efficiently.

B

Appendix B

```yaml
name: demo-project
description: This is a description of the project
initialize:
  plugins:
    interpolate:
      method: Interpolation
      path: "builtin"
      config:
        method: linear
        x: [0, 10, 50, 100]
        y: [0.12, 0.32, 0.75, 1.02]
        input-parameter: "cpu/utilization"
        output-parameter: "cpu-factor"

tree:
  children:
    child:
      pipeline:
        observe:
        regroup:
          -cloud/region
          -cloud/instance-type
      inputs:
        -timestamp: 2023-07-06T00:00
          duration: 300
          cloud/instance-type: A1
          cloud/region: uk-west
          cpu/utilization: 99
        -timestamp: 2023-07-06T05:00
          duration: 300
          cloud/instance-type: A1
          cloud/region: uk-west
          cpu/utilization: 23
        -timestamp: 2023-07-06T10:00
          duration: 300
          cloud/instance-type: A1
          cloud/region: uk-west
          cpu/utilization: 12
```

Listing 1: Example Impact Framework YAML file

```python
import pyRAPL
from pyRAPL import Measurement

def fibonacci(n):
    fib_series = [0, 1]
    for i in range(2, n):
        next_value = fib_series[-1] + fib_series[-2]
        fib_series.append(next_value)
    return fib_series

if __name__ == "__main__":
    # Initialize pyRAPL
    pyRAPL.setup()

    # Create a measurement instance
    measurement = Measurement("Fibonacci Measurement")

    # Measure energy consumption
    with measurement:
        # Calculate the first 1e5 Fibonacci numbers
        fib_series = fibonacci(int(1e5))

    # Print the energy consumption results
    print(f"Energy consumption: {measurement.result}")

    # Optionally, print the Fibonacci series
    print(fib_series)
```

Listing 2: Energy Consumption Measurement (RAPL) for Fibonacci Series Calculation

```python
import os
import time
import subprocess
import signal

def run_stress_test():
    try:
        # Start the pcm command before the stress test and store the process
        pcm_command = "sudo pcm 1 -csv=stress-pcm.csv"
        pcm_process = subprocess.Popen(pcm_command, shell=True, preexec_fn=os.setsid)

        # Start the powerjoular command and store the process
        rapl_command = "sudo powerjoular -f stress-rapl.csv"
        rapl_process = subprocess.Popen(rapl_command, shell=True, preexec_fn=os.setsid)

        time.sleep(1) # Give both pcm and powerjoular a second to start properly

        # Stress test loop from 0% to 100% in 10% increments
        cpu_count = os.cpu_count()
        for i in range(11): # Loop through 0% to 100% load in 10% increments
            load = int(cpu_count * i / 10) # Calculate number of CPUs to stress
            if load == 0:
                print(f"Running at {load}% load for 180 seconds (idle)")
                time.sleep(180)
            else:
                print(f"Running at {i*10}% load for 180 seconds with {load} workers")
                stress_command = f"stress -c {load} -t 180s"
                subprocess.run(stress_command, shell=True)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        # Terminate PCM and powerjoular processes gracefully
        print("Terminating PCM and Powerjoular processes...")
        os.killpg(os.getpgid(pcm_process.pid), signal.SIGTERM) # Kill PCM process
        os.killpg(os.getpgid(rapl_process.pid), signal.SIGTERM) # Kill Powerjoular process

if __name__ == "__main__":
    run_stress_test()
```

Listing 3: Stress test & Benchmarking Script for PCM and RAPL

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

# Load your data from an Excel file
file_path = '/home/sohel/storage/NOTES/Sem-4/Thesis/Software Sustainability/CarbonIntensity/pcm/rapl-pcm-graph.xlsx'
data = pd.read_excel(file_path)

# Create bins of 10 for CPU values
bins = np.arange(0, 110, 10)
data['CPU_bin'] = pd.cut(data['CPU'], bins)

# Group the data by the CPU bins and compute the average PCM and RAPL values for each bin
binned_data = data.groupby('CPU_bin').agg({'PCM': 'mean', 'RAPL': 'mean', 'CPU': 'mean'}).dropna()

# Extract the filtered binned CPU, PCM, and RAPL values
X_filtered_binned = binned_data['CPU'].values
Y_pcm_filtered_binned = binned_data['PCM'].values
Y_rapl_filtered_binned = binned_data['RAPL'].values

# Perform polynomial fits (degree 3) for the filtered binned PCM vs CPU and RAPL vs CPU
degree = 3
coefs_pcm_filtered_binned_deg3 = np.polyfit(X_filtered_binned, Y_pcm_filtered_binned, degree)
poly_pcm_filtered_binned_deg3 = np.poly1d(coefs_pcm_filtered_binned_deg3)

coefs_rapl_filtered_binned_deg3 = np.polyfit(X_filtered_binned, Y_rapl_filtered_binned, degree)
poly_rapl_filtered_binned_deg3 = np.poly1d(coefs_rapl_filtered_binned_deg3)

# Generate predictions for both fits (degree 3)
Y_pcm_pred_filtered_binned_deg3 = poly_pcm_filtered_binned_deg3(X_filtered_binned)
Y_rapl_pred_filtered_binned_deg3 = poly_rapl_filtered_binned_deg3(X_filtered_binned)

# Calculate R-squared values for both fits (degree 3)
r_squared_pcm_filtered_binned_deg3 = r2_score(Y_pcm_filtered_binned, Y_pcm_pred_filtered_binned_deg3)
r_squared_rapl_filtered_binned_deg3 = r2_score(Y_rapl_filtered_binned, Y_rapl_pred_filtered_binned_deg3)

# Plot CPU vs PCM with fitted curve (degree 3) for filtered binned data and save to PDF
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X_filtered_binned, Y_pcm_filtered_binned, color='blue', label='PCM values', alpha=0.8)
plt.plot(np.sort(X_filtered_binned), poly_pcm_filtered_binned_deg3(np.sort(X_filtered_binned)), color='red', label=f'Fitted curve')
plt.title(f'Curve Fitting: PCM vs CPU\nR-squared = {r_squared_pcm_filtered_binned_deg3:.4f}')
plt.xlabel('CPU')
plt.ylabel('PCM')
plt.legend()
plt.grid(True)
plt.savefig('./pcm/PCM_vs_CPU_fitted_curve_filtered.pdf') # Save plot as PDF
plt.show()

# Plot CPU vs RAPL with fitted curve (degree 3) for filtered binned data and save to PDF
plt.figure(figsize=(8, 6))
plt.scatter(X_filtered_binned, Y_rapl_filtered_binned, color='green', label='RAPL values', alpha=0.8)
plt.plot(np.sort(X_filtered_binned), poly_rapl_filtered_binned_deg3(np.sort(X_filtered_binned)), color='red', label=f'Fitted curve')
plt.title(f'Curve Fitting: RAPL vs CPU\nR-squared = {r_squared_rapl_filtered_binned_deg3:.4f}')
plt.xlabel('CPU')
plt.ylabel('RAPL')
plt.legend()
plt.grid(True)
plt.savefig('./pcm/RAPL_vs_CPU_fitted_curve_filtered.pdf') # Save plot as PDF
plt.show()

# Display the fitted curve formulas and R-squared values for filtered binned data (degree 3)
formula_pcm_filtered_binned_deg3 = " + ".join([f"{coef:.4f}*x^{degree-i}" for i, coef in enumerate(coefs_pcm_filtered_binned_deg3)])
formula_rapl_filtered_binned_deg3 = " + ".join([f"{coef:.4f}*x^{degree-i}" for i, coef in enumerate(coefs_rapl_filtered_binned_deg3)])

print(f"Fitted curve formula (PCM vs CPU, filtered binned, degree 3): {formula_pcm_filtered_binned_deg3}")
print(f"R-squared value for PCM (filtered binned, degree 3): {r_squared_pcm_filtered_binned_deg3:.4f}")
print(f"Fitted curve formula (RAPL vs CPU, filtered binned, degree 3): {formula_rapl_filtered_binned_deg3}")
print(f"R-squared value for RAPL (filtered binned, degree 3): {r_squared_rapl_filtered_binned_deg3:.4f}")
```

Listing 4: Script for RAPL & PCM curve fitting and R^2 analysis

C

Appendix C

```
# Importing Packages
import sys
import sqlite3 # Using this package for the sole purpose of storing data directly to the disk as a SQL database
import psutil # Package used to measure most of the parameters of the system
import time
import threading # Threading is to run tasks concurrently
import GPUtil # Package to measure the parameters pertaining to the GPU
import platform # Package to get information about the system to provide more metadata and good accounting of information
import wmi # Package that helps interface with WMI developed by Microsoft to fetch system data
import pandas as pd # Package to handle Dataframes and also to convert databases into CSV
import cpuinfo # Package to get proper CPU information


class SystemDataCollector: # This class contain all the methods required to fetch information of workings of a system
    def __init__(self, db_path=":memory:", interval=1) -> None: # Default database is in memory for easier troubleshooting during development
        self.db_path = db_path
        self.interval = interval

        # Connecting to the SQL database
        self.conn = sqlite3.connect(self.db_path, check_same_thread=False)
        self.cursor = self.conn.cursor()
        self.create_table()

        self.collecting = False

    def create_table(self):
        # Two tables are being create; one to store information about the hardware that the code is being developed upon
        # The second table is to store the hardware utilisation parameters of the hardware

        with self.conn:
            # This table is for storing the hardware information
            # 5 columns of input here
            # CPU model | GPU Model | Computer Name | Computer Model | Memory/RAM size (GB)
            self.cursor.execute('''CREATE TABLE IF NOT EXISTS HardwareInfo(
                                timestamp_utc DATETIME DEFAULT CURRENT_TIMESTAMP,
                                cpu_model TEXT,
                                gpu_model TEXT,
```

```
                            computer_manufacturer TEXT,
                            computer_model TEXT,
                            computer_name TEXT,
                            computer_os TEXT,
                            memory_size_GB INTEGER)''')

        # This table is for storing the system utilisations that will be
            later used for calcs
        # 5 columns of input here
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS SystemUtilization(
                            timestamp_utc DATETIME DEFAULT CURRENT_TIMESTAMP,
                            cpu_usage_percentage REAL,
                            gpu_usage_percentage REAL,
                            memory_usage_percentage INTEGER,
                            network_bytes_sent INTEGER,
                            network_bytes_received INTEGER)''')

# Data pertaining to the hardware information is logged here.
def log_hardware_info(self):
    try:
        # CPU
        cpu_model = cpuinfo.get_cpu_info()['brand_raw']

        # GPU
        gpus = GPUtil.getGPUs()
        gpu_model = gpus[0].name if gpus else "No GPU detected"
        del gpus

        # PC -User
        computer_name = platform.node()
        computer_os = platform.system() + " " + platform.release()

        # PC model
        c = wmi.WMI()
        this_system = c.Win32_ComputerSystem()[0]
        computer_manufacturer = this_system.Manufacturer
        computer_model = this_system.Model
        del this_system

        #
        sys_info = c.Win32_ComputerSystem()[0]
        memory_size_GB = float(sys_info.TotalPhysicalMemory)/(1e9)
        del c, sys_info

        with self.conn:
            self.cursor.execute('''INSERT INTO HardwareInfo(
                            cpu_model,
                            gpu_model,
                            computer_manufacturer,
                            computer_model,
```

```
                                computer_name,
                                computer_os,
                                memory_size_GB) VALUES(?,?,?,?,?,?,?)''',
                                (cpu_model, gpu_model, computer_manufacturer,
                                    computer_model,
                                  computer_name, computer_os, memory_size_GB))

        except Exception as e:
            print(f"Error Logging hardware info: {e}")

    def collect_data(self):
        self.collecting = True
        while self.collecting:
            try:
                # CPU
                cpu_usage = psutil.cpu_percent(interval=self.interval)

                # Memory
                memory_info = psutil.virtual_memory()
                memory_usage = memory_info.percent
                del memory_info

                # Networking
                net_io = psutil.net_io_counters()
                networking_send = net_io.bytes_sent
                networking_rec = net_io.bytes_recv
                del net_io

                # GPU
                # Added the GPU measurement in the end as it is part of a
                    different package and hence the data above
                # might be more coherent as it belongs to the same package
                gpus = GPUtil.getGPUs()
                gpu_usage = gpus[0].load * 100 if gpus else 0
                del gpus

                # Inserting obtained values into the Database
                with self.conn:
                    self.cursor.execute('''INSERT INTO SystemUtilization(
                                        cpu_usage_percentage,
                                        gpu_usage_percentage,
                                        memory_usage_percentage,
                                        network_bytes_sent,
                                        network_bytes_received)
                                        VALUES(?,?,?,?,?)''',
                                        (cpu_usage,
                                         gpu_usage,
                                         memory_usage,
                                         networking_send,
                                         networking_rec))
```

```
135
136             except Exception as e:
137                 print(f"An error occurred during logging data: {e}")
138                 break
139
140     def fetch_hardware_data(self):
141         with self.conn:
142             self.cursor.execute("SELECT * FROM HardwareInfo")
143
144         return self.cursor.fetchall()
145
146     def fetch_util_data(self):
147         with self.conn:
148             self.cursor.execute("SELECT * FROM SystemUtilization")
149
150         return self.cursor.fetchall()
151
152     def close_connection(self):
153         self.conn.close()
154
155     def start_collection(self):
156         self.collection_thread = threading.Thread(target=self.collect_data)
157         self.collection_thread.start()
158
159     def stop_collection(self):
160         self.collecting = False
161         self.collection_thread.join()
162
163     def conv_db(self, format='csv', output_file_name='output'):
164         output_file = output_file_name + '.' + format
165
166         # Reading the tables into dataframe to convert into csv or excel
167         hardware_info = pd.read_sql_query('SELECT * FROM HardwareInfo', con=self.
                conn)
168         hardware_info = hardware_info.transpose()
169         util_data = pd.read_sql_query('SELECT * FROM SystemUtilization', con=self
                .conn)
170
171         if 'csv' in format.lower():
172             hardware_info.to_csv(f'HardwareInfo-{output_file}')
173             util_data.to_csv(f'UtilisationData-{output_file}')
174
175         elif 'xlsx' in format.lower():
176             with pd.ExcelWriter(output_file, engine='openpyxl') as writer:
177                 hardware_info.to_excel(writer, sheet_name='HardwareInformation',
                        index=True)
178                 util_data.to_excel(writer, sheet_name='UtilisationData', index=
                        False)
179
180         else:
```

```
            print('Invalid File format for conversion.')


# Incase the code is run standalone
if __name__ == "__main__":
    database_path = r'.\Experiments\Img-exe.db'
    collector = SystemDataCollector(interval=1, db_path=database_path)
    output_format = 'xlsx'
    output_file_name = 'Img-exe'
    export_output = True

    collector.log_hardware_info()
    print('Hardware Data Collected!')
    try:
        print('System Utilisation Logging started!')
        collector.start_collection()
        while True:
            time.sleep(1)

    except KeyboardInterrupt:
        collector.stop_collection()
        print("Data collection closed!")
        if export_output:
            collector.conv_db(format=output_format, output_file_name=
                output_file_name)
        collector.close_connection()
        sys.exit(0)

    except Exception as e:
        print(f'Error occurred during data logging: {e}')
        sys.exit(1)
```

Listing 5: Script for Logging system utilisation

```python
import numpy as np
from numpy.polynomial.polynomial import Polynomial
import matplotlib.pyplot as plt
import sqlite3
import pandas as pd

class EnergyCalculator:

    def __init__(self, cpu_tdp, db_path, gpu_tdp):
        try:
            self.TDP = cpu_tdp
            self.db_path = db_path
            self.gpu_tdp = gpu_tdp

            # Database
            self.conn = sqlite3.connect(self.db_path)
            self.cursor = self.conn.cursor()

            with self.conn:
                # Creating a new table in the same database to prevent messing up the original data
                self.cursor.execute('''CREATE TABLE IF NOT EXISTS SystemEnergyUsage
                                    AS
                                    SELECT * FROM SystemUtilization''')
        except Exception as e:
            print(f'Error during Class initialisation occurred: {e}')



    def power_curve_coef(utilisation_array = np.array([0, 10, 50, 100]), power_array = np.array([0.12, 0.32, 0.75, 1.02])):
        utilisation = utilisation_array
        power = power_array
        '''
        CPU Utilisation [%] | Power Draw [w]
                0 % | 0.12 * TDP
                10% | 0.32 * TDP
                50% | 0.75 * TDP
                100% | 1.02 * TDP

        This data collected from the TEADS-curve plugin that was developed for IF framework
        https://github.com/Green-Software-Foundation/if-unofficial-plugins/blob/main/src/lib/teads-curve/index.ts
        '''
        coefs = Polynomial.fit(utilisation, power, 2).convert().coef
        power_func = np.poly1d(coefs[::-1])

```

```
45          print(f'Coefficients:{coefs}')
46          plt.scatter(utilisation, power, color='red', label='Data Points')
47          x = np.linspace(0,100,200)
48
49          plt.plot(x, power_func(x), label='Fitted Curve')
50          plt.xlabel('CPU Utilisation [%]')
51          plt.ylabel('Power Draw fraction')
52          plt.title('CPU Utilisation curve Vs Power Draw')
53          plt.legend()
54          plt.show()
55          return coefs, power_func # Returns the coefficients and function
56
57      def power_calc(self, utilisation, TDP):
58          # The coefficients used here were obtained prior by feeding the values
                that can be observed in the method power_curve_coef
59          # power = lambda x: ((1.2e-1) * (x**0)) + ((2.247777e-2) * (x**1)) + ((-
                2.260333e-4) * (x**2)) + ((1.255556e-6) * (x**3))
60          power = lambda x: ((-7.24371373e-5) * (x**2)) + ((1.60251451e-2) * (x**1)
                ) + ((1.39671180e-1) * (x**0))
61
62          return (TDP * power(utilisation))
63
64      def cpu_power_draw(self):
65          try:
66              with self.conn:
67                  print('Calculating CPU Power draw [W]')
68                  # Adding a column which will store the CPU power draw values
69                  self.column_checker_adder(column_name='cpu_power_draw_W')
70                  # Fetching the relevant data to compute CPU power draw
71                  self.cursor.execute('''SELECT timestamp_utc, cpu_usage_percentage
72                                      FROM SystemEnergyUsage''')
73                  rows = self.cursor.fetchall()
74                  # Iterating through the fetched information to obtain
75                  for row in rows:
76                      timestamp_utc, cpu_utilisation = row
77                      cpu_power_draw = self.power_calc(utilisation = cpu_utilisation
                            , TDP=self.TDP)
78                      # Storing the value into the database by matching the
                            timestamp
79                      self.cursor.execute('''UPDATE SystemEnergyUsage
80                                          SET cpu_power_draw_W = ?
81                                          WHERE timestamp_utc=?''',
82                                          (cpu_power_draw, timestamp_utc))
83          except Exception as e:
84              print(f'Error occurred during calculation of CPU power draw: {e}')
85
86
87      def gpu_power_draw(self):
88          try:
89              with self.conn:
```

```
90              print('Calculating GPU Power draw [W]')
91              self.column_checker_adder(column_name='gpu_power_draw_W')
92              self.cursor.execute('''SELECT timestamp_utc, gpu_usage_percentage
93                                  FROM SystemEnergyUsage''')
94              rows = self.cursor.fetchall()
95              for row in rows:
96                  timestamp_utc, gpu_utilisation = row
97                  gpu_power_draw = self.power_calc(utilisation = gpu_utilisation
                        , TDP = self.gpu_tdp)
98                  # Storing the value into the database by matching the
                        timestamp to prevent erroneous data addition
99                  self.cursor.execute('''UPDATE SystemEnergyUsage
100                                     SET gpu_power_draw_W = ?
101                                     WHERE timestamp_utc=?''',
102                                     (gpu_power_draw, timestamp_utc))
103         except Exception as e:
104             print(f'Error occurred during calculation of GPU power draw: {e}')
105
106     def column_checker_adder(self, column_name):
107         try:
108             with self.conn:
109                 self.cursor.execute('''PRAGMA table_info('SystemEnergyUsage')''')
110                 columns = self.cursor.fetchall()
111                 db_column_names = [column[1] for column in columns]
112                 if column_name not in db_column_names:
113                     self.cursor.execute(f'''ALTER TABLE SystemEnergyUsage
114                                     ADD COLUMN {column_name} REAL''')
115                     print(f'{column_name} added in SystemEnergyUsage')
116
117         except Exception as e:
118             print(f'Error during creation of column {column_name}: {e}')
119
120     def energy_calculator(self, duration_s=1):
121         try:
122             with self.conn:
123                 self.column_checker_adder(column_name='system_energy_Wh')
124                 duration_h = duration_s / (60 * 60) # Converting seconds into
                        hours
125                 self.cursor.execute(f'''UPDATE SystemEnergyUsage
126                                     SET system_energy_Wh = total_power_draw_W* {
                                        duration_h}''')
127         except Exception as e:
128             print(f'An Error occurred during calculation of the energy of the
                    system: {e}')
129
130     def energy_calculator2(self):
131         try:
132             with self.conn:
133                 self.column_checker_adder(column_name='system_energy_Wh')
134                 self.cursor.execute(f'''UPDATE SystemEnergyUsage
```

```
135                                 SET system_energy_Wh = total_power_draw_W *
                                        duration_s'''')
136         except Exception as e:
137             print(f'An Error occurred during calculation of the energy of the
                    system: {e}')
138
139     def csv_converter(self, filename):
140         print('Converting SQL Database to CSV')
141         df = pd.read_sql_query('''SELECT * FROM SystemEnergyUsage''', con=self.
                conn)
142         filename += '.csv'
143         df.to_csv(filename)
144
145     def total_power_draw(self):
146         try:
147             with self.conn:
148                 print('Calculating Total Power draw of the system')
149                 self.column_checker_adder(column_name='total_power_draw_W')
150                 self.cursor.execute('''UPDATE SystemEnergyUsage
151                                     SET total_power_draw_W = cpu_power_draw_W +
                                        gpu_power_draw_W''')
152         except Exception as e:
153             print(f'An Error during computation of total energy consumption took
                    place: {e}')
154
155     def calc_system_energy(self):
156         self.cpu_power_draw()
157         self.gpu_power_draw()
158         self.total_power_draw()
159         self.energy_calculator()
160
161
162 if __name__ == '__main__':
163     EnergyCalculator.power_curve_coef()
```

Listing 6: Script for Energy calculations using TEADS curve

```python
import sqlite3
import pandas as pd

class EmissionsCalculator:
    def __init__(self, db_path) -> None:
        try:
            self.db_path = db_path
            # Database
            self.conn = sqlite3.connect(self.db_path)
            self.cursor = self.conn.cursor()

        except Exception as e:
            print(f'Error during Class initialisation occurred: {e}')

    def emission_calculator(self, emission_factor = 1.588417054):
        '''
        Default emission factor data taken from Ecoinvent database :
        market for electricity, low voltage | kilowatt hour | IN-Western grid
        Value is in kgCO2eq/kWh
        '''
        try:
            with self.conn:
                column_name = 'emissions_gCO2eq'
                self.cursor.execute('''PRAGMA table_info('SystemEnergyUsage')''')
                columns = self.cursor.fetchall()
                db_column_names = [column[1] for column in columns]
                if column_name not in db_column_names:
                    print(f'{column_name} added in SystemEnergyUsage')
                    self.cursor.execute(f'''ALTER TABLE SystemEnergyUsage
                                        ADD {column_name} REAL''')

                self.cursor.execute(f'''UPDATE SystemEnergyUsage
                                    SET {column_name} = system_energy_Wh * {
                                        emission_factor}''')
            df = pd.read_sql_query('''SELECT * FROM SystemEnergyUsage
                SystemEnergyUsage''', con=self.conn)
            df.to_csv('Emission-output.csv')
        except Exception as e:
            print(f'Error occurred during emission calculations: {e}')

if __name__ == '__main__':
    path = r'.\demo-data.db'
    emm = EmissionsCalculator(db_path = path)
    emm.emission_calculator()
```

Listing 7: Script for Carbon Emissions calculations using Energy (Ecoinvent)

D

Appendix D

D.1 How to read a Candlestick chart?

A candlestick graph is a popular chart used in financial markets to display price movements of assets over a specific time period. Each candlestick represents four key data points: the opening price, closing price, highest price, and lowest price during that period. The "body" of the candlestick shows the difference between the opening and closing prices, where a filled or colored body typically indicates a drop (closing price is lower than the opening), and a hollow or uncolored body shows a rise (closing price is higher than the opening). The "wicks" or "shadows" above and below the body represent the high and low prices reached during the period. This type of chart helps traders quickly assess price action and identify trends, reversals, or patterns in the market.

D.2 Carbon Emissions Candlestick charts

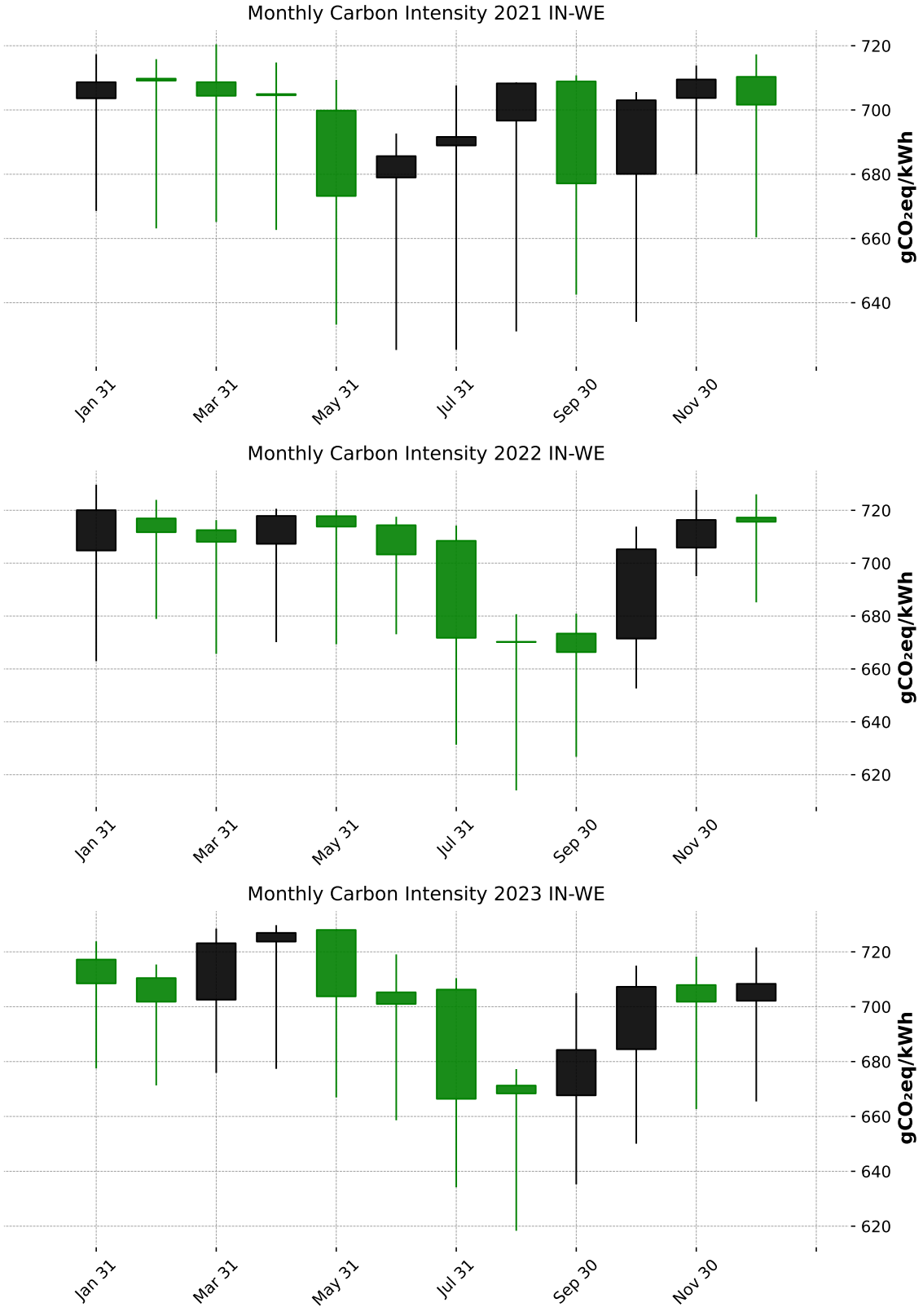

Figure 33: Monthly Carbon Intensity – India (IN-WE)

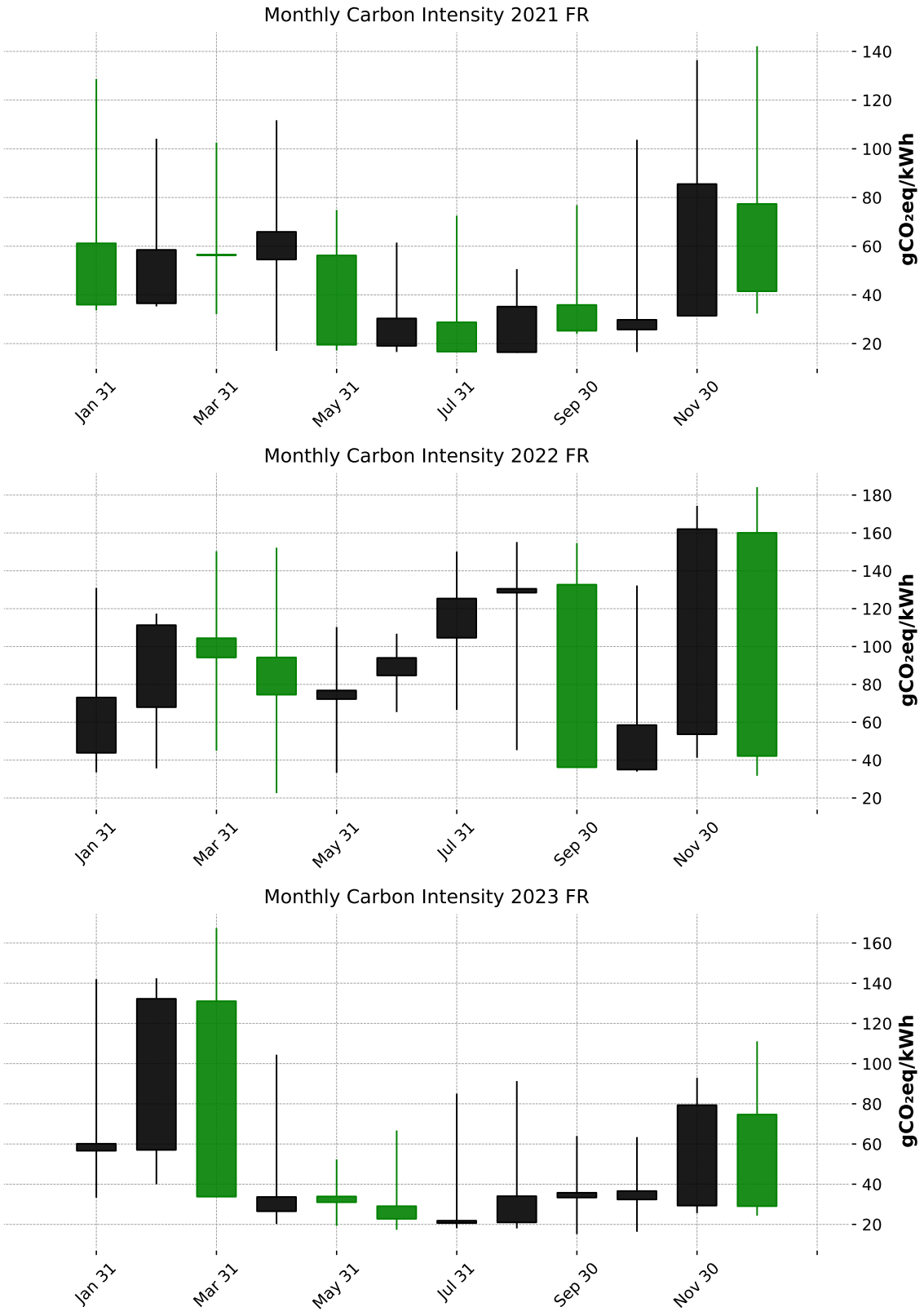

Figure 34: Monthly Carbon Intensity – France

References

[1] Intergovernmental Panel on Climate Change (IPCC). *Global Warming of 1.5°C - Special Report (SR15)*. 2018. URL: https://www.ipcc.ch/sr15/.

[2] *Software - Worldwide*. https://www.statista.com/outlook/tmo/software/worldwide?currency=USD.

[3] *Business Software and Services Market Size, Share & Trends Analysis Report*. https://www.grandviewresearch.com/industry-analysis/business-software-services-market.

[4] Pengfei Li et al. *Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models*. 2023. arXiv: 2304.03271 [cs.LG]. URL: https://arxiv.org/abs/2304.03271.

[5] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. "Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model". In: *Journal of Machine Learning Research* 24.253 (2023), pp. 1–15. URL: http://jmlr.org/papers/v24/23-0069.html.

[6] Colin C Venters et al. "Software sustainability: Research and practice from a software architecture viewpoint". In: *Journal of Systems and Software* 138 (2018), pp. 174–188.

[7] Colin C Venters et al. "Software sustainability: The modern tower of babel". In: *CEUR Workshop proceedings*. Vol. 1216. CEUR. 2014, pp. 7–12.

[8] GHG Protocol Initiative Team and Revision Working Group. *The Greenhouse Gas Protocol: A Corporate Accounting and Reporting Standard*. Revised Edition. Washington, DC: World Resources Institute, 2004. URL: https://ghgprotocol.org/corporate-standard.

Includes guidance for companies on how to prepare a GHG emissions inventory that represents a true and fair account of their emissions.

[9] Antti Sipilä, Laura Partanen, and Jari Porras. "Carbon Footprint Calculations for a Software Company – Adapting GHG Protocol Scopes 1, 2 and 3 to the Software Industry". In: *Software Business*. Ed. by Sami Hyrynsalmi et al. Cham: Springer Nature Switzerland, 2024, pp. 442–455. ISBN: 978-3-031-53227-6.

[10] European Commission. *Corporate Sustainability Reporting*. 2023. URL: https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en.

[11] Aimee Dix. *Cloudflare extends server hardware lifespan to five years, will save 20 million dollars*. Aug. 2023. URL: https://www.datacenterdynamics.com/en/news/cloudflare-extends-server-hardware-lifespan-to-five-years-will-save-20-million/.

Accessed: 2024-03-29.

[12] Green Software Foundation. *Hardware Efficiency*. 2024. URL: https://learn.greensoftware.foundation/hardware-efficiency/.

Accessed: 2024-03-26.

[13] Dimitrios Stamoulis et al. “Capturing true workload dependency of bti-induced degradation in cpu components”. In: *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*. 2016, pp. 373–376.

[14] Christophe Cerin, Hazem Fkaier, and Mohamed Jemni. “Experimental Study of Thread Scheduling Libraries on Degraded CPU”. In: *2008 14th IEEE International Conference on Parallel and Distributed Systems*. IEEE. 2008, pp. 697–704.

[15] Google. *Data Center Efficiency*. 2024. url: https://www.google.com/about/datacenters/efficiency/.

Accessed: June 1, 2024.

[16] Uptime Institute and Upsite Technologies. *What is the average annual power usage effectiveness (PUE) for your largest data center? [Graph]*. Sept. 2023. url: https://www.statista.com/statistics/1229367/data-center-average-annual-pue-worldwide/.

In Statista. Retrieved June 2, 2024, from https://www.statista.com/statistics/1229367/data-center-average-annual-pue-worldwide/.

[17] Ember and Energy Institute. *Carbon intensity of electricity generation – Ember and Energy Institute*. Ember, "Yearly Electricity Data"; Energy Institute, "Statistical Review of World Energy" [original data]. May 2024. url: https://ourworldindata.org/grapher/carbon-intensity-electricity.

Accessed: 23-05-2024.

[18] International Energy Agency. *Russia’s War on Ukraine*. 2024. url: https://www.iea.org/topics/russias-war-on-ukraine.

Accessed: June 6, 2024.

[19] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. “Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training”. In: *USENIX NSDI*. 2023.

[20] ecoinvent. *Ecoinvent Dataset Documentation (ecoQuery)*. 2024. url: https://ecoquery.ecoinvent.org/3.8/cutoff/dataset/2077/documentation.

Accessed: June 16, 2024.

[21] NetApp and MIT. *PAIA Overview Uses & Limitations*. Aug. 2022. url: https://www.netapp.com/es/media/74909-PAIA-Overview-Uses-Limitations.pdf.

[22] Dell Technologies. *Understanding Life Cycle Assessments (LCAs), Product Carbon Footprints (PCFs), and the Uses and Limitations of PAIA*. Nov. 2023. url: https://corporate.delltechnologies.com/en-us/social-impact/advancing-sustainability/sustainable-products-and-services/product-carbon-footprints.html.

[23] Elsa Olivetti and Randolph Kirchain. “A Product Attribute to Impact Algorithm to Streamline IT Carbon Footprinting”. In: *Design for Innovative Value Towards a Sustainable Society*. Ed. by Mitsutaka Matsumoto et al. Dordrecht: Springer Netherlands, 2012, pp. 747–749. isbn: 978-94-007-3010-6.

[24] Liqiu Deng, Callie W. Babbitt, and Eric D. Williams. “Economic-balance hybrid LCA extended with uncertainty analysis: case study of a laptop computer”. In: *Journal of Cleaner Production* 19.11 (2011), pp. 1198–1206. issn: 0959-6526. doi: https://doi.org/10.1016/j.jclepro.2011.03.004. url: https://www.sciencedirect.com/science/article/pii/S0959652611000801.

[25] HP Development Company, L.P. *HP ZBook 15 G5 Product Carbon Footprint Report*. Jan. 2024. URL: https://www.hp.com.

[26] ISO/IEC JTC 1. *Information technology — Software Carbon Intensity (SCI) specification*. en. Standard ISO/IEC 21031:2024. International Organization for Standardization, 2024. URL: https://www.iso.org/standard/86612.html.

[27] McKinsey & Company. *The Green IT Revolution: A Blueprint for CIOs to Combat Climate Change*. 2022. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-green-it-revolution-a-blueprint-for-cios-to-combat-climate-change.

Accessed: July 16, 2024.

[28] Jacob Sorber et al. "Turducken: Hierarchical power management for mobile devices". In: *Proceedings of the 3rd international conference on Mobile systems, applications, and services*. 2005, pp. 261–274.

[29] Haozhou Lyu, Gregory Gay, and Maiko Sakamoto. "Developer Views on Software Carbon Footprint and Its Potential for Automated Reduction". In: *Search-Based Software Engineering*. Ed. by Paolo Arcaini, Tao Yue, and Erik M. Fredericks. Cham: Springer Nature Switzerland, 2024, pp. 35–51. ISBN: 978-3-031-48796-5.

[30] Victor Berger. *pyRAPL: Python package for measuring energy consumption*. 2020. URL: https://pypi.org/project/pyRAPL/.

[31] Qian Diao and Justin Song. "Prediction of CPU idle-busy activity pattern". In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. 2008, pp. 27–36. DOI: 10.1109/HPCA.2008.4658625.

[32] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. "A review of energy measurement approaches". English. In: *Operating Systems Review* 47.3 (2013), pp. 42–49. DOI: 10.1145/2553070.2553077.

[33] Aman Kansal and Feng Zhao. "Fine-grained energy profiling for power-aware application design". In: *SIGMETRICS Perform. Eval. Rev.* 36.2 (Aug. 2008), pp. 26–31. ISSN: 0163-5999. DOI: 10.1145/1453175.1453180. URL: https://doi.org/10.1145/1453175.1453180.

[34] Simon Schubert et al. "Profiling Software for Energy Consumption". In: *2012 IEEE International Conference on Green Computing and Communications*. 2012, pp. 515–522. DOI: 10.1109/GreenCom.2012.86.

[35] Syed Islam, Adel Noureddine, and Rabih Bashroush. "Measuring energy footprint of software features". In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 2016, pp. 1–4. DOI: 10.1109/ICPC.2016.7503726.

[36] Teads Engineering Team. *Teads' CPU Estimation Plugin*. 2024. URL: https://github.com/Green-Software-Foundation/if-unofficial-plugins/blob/main/src/lib/teads-curve/README.md.

Accessed: July 18, 2024.

[37] Benjamin Davy. *Building an AWS EC2 Carbon Emissions Dataset*. 2021. URL: https://medium.com/teads-engineering/building-an-aws-ec2-carbon-emissions-dataset-3f0fd76c98ac.

Published in Teads Engineering, Accessed: July 12, 2024.

[38] Kashif Nizam Khan et al. "Rapl in action: Experiences in using rapl for power measurements". In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3.2 (2018), pp. 1–26.

[39] Muhammad Fahad et al. "A comparative study of methods for measurement of energy of computing". In: *Energies* 12.11 (2019), p. 2204.

[40] Peter Henderson et al. "Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning". In: *Journal of Machine Learning Research* 21.248 (2020), pp. 1–43. URL: http://jmlr.org/papers/v21/20-312.html.

[41] Timothy McKay and Patrick Christian Konsor. *Intel Power Gadget Tool*. 2024. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html.

Accessed: July 19, 2024.

[42] Arne Tarara. *Power Measurement on macOS*. 2024. URL: https://www.green-coding.io/blog/power-measurement-on-macos/.

Accessed: July 19, 2024.

[43] Intel Corporation. *Intel® Performance Counter Monitor - A Better Way to Measure CPU Utilization*. 2022. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html.

ID 778208, Version: Latest, Updated: November 30, 2022, Accessed: July 12, 2024.

[44] Intel Corporation. *Intel Performance Counter Monitor (PCM)*. 2024. URL: https://github.com/intel/pcm.

Accessed: July 12, 2024.

[45] Stefan Hinum. *Asus GL553VE-DS74 Review and Specifications*. 2021. URL: https://www.notebookcheck.net/Asus-GL553VE-DS74.200274.0.html.

Accessed: August 26, 2024.

[46] Intel Corporation. *Intel® Core™ i7-7700HQ Processor (6M Cache, up to 3.80 GHz)*. 2022. URL: https://ark.intel.com/content/www/us/en/ark/products/97185/intel-core-i7-7700hq-processor-6m-cache-up-to-3-80-ghz.html.

Accessed: June 20, 2024.

[47] Adel Noureddine. "PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools". In: *18th International Conference on Intelligent Environments (IE2022)*. Biarritz, France, June 2022.

[48] W. Hildreth. *Case Studies in Public Budgeting and Financial Management*. Jan. 2003. ISBN: 0-8247-0888-1.

[49] Advanced Micro Devices, Inc. *AMD uProf - Performance and Power Profiling Tool*. https://www.amd.com/en/developer/uprof.html. 2024.

Accessed: July 21, 2024.

[50] Ian Schneider and Taylor Mattia. *Carbon accounting in the Cloud: a methodology for allocating emissions across data center users*. 2024. arXiv: 2406.09645 [cs.SE]. URL: https://arxiv.org/abs/2406.09645.

[51] Jens Gröger et al. *Green Cloud Computing: Lebenszyklusbasierte Datenerhebung zu Umweltwirkungen des Cloud Computing*. Tech. rep. Berlin: Öko-Institut e.V., Fraunhofer-Institut für Zuverlässigkeit und Mikrointegration (IZM), Mar. 2021. URL: https://www.umweltbundesamt.de/publikationen.

[52] Jayant Baliga et al. "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport". In: *Proceedings of the IEEE* 99.1 (2011), pp. 149–167. DOI: 10.1109/JPROC.2010.2060451.

[53] Syed Islam, Adel Noureddine, and Rabih Bashroush. "Measuring energy footprint of software features". In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 2016, pp. 1–4. DOI: 10.1109/ICPC.2016.7503726.

[54] Syed Islam and David Binkley. "PORBS: A parallel observation-based slicer". In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 2016, pp. 1–3. DOI: 10.1109/ICPC.2016.7503745.

[55] Vicki-Lynn Brunskill. *Scrum Sprint*. URL: https://www.techtarget.com.

Last updated: August 2019, Accessed: Aug 2, 2024.

[56] Ecoinvent Association. *Market for electricity, low voltage - India, Western grid - electricity, low voltage*. 2023. URL: https://ecoquery.ecoinvent.org/3.10/cutoff/dataset/16623/documentation.

[57] Green Software Foundation. *Introduction to the Green Software Foundation*. Mar. 2024.

This page has been intentionally left blank