Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Investigation on Precise Measurement of CO2 Emissions from AI Applications

Pankhuri Verma

Course of Study:	Computer Science
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dr. Dinesh Reddy
Commenced:	November 28, 2023
Completed:	May 28, 2024

Acknowledgement

I am profoundly grateful to Prof. Dr. Marco Aiello and the Institute of Architecture of Application Systems at the University of Stuttgart for giving me the opportunity to work on this thesis. Prof. Dr. Marco Aiello's guidance and expertise since the start of the thesis have significantly enhanced my understanding and exploration of the subject.

I extend my deepest gratitude to my supervisor, Dr. Dinesh Reddy, who supported and guided me in every stage of this thesis. His knowledge, patience, constructive feedback, and meticulous attention to detail played a very important role in this thesis.

I would like to express my heartfelt thanks to my family—my mother, father, brother, and all my elders—who have always believed in me. Their constant encouragement and belief in my abilities have always motivated me to strive for excellence.

I am also thankful to my friends Vishesh, Deepanshu, Faiz, and Anurima for their support and motivation. Their encouragement has been a source of strength and inspiration during the challenging moments of this journey.

Lastly, I would like to acknowledge everyone who has been a part of my life during my thesis. Their well-wishes and support have been a pillar of strength for me, and I am grateful for every word of encouragement I have received.

Abstract

The exponential growth of Artificial Intelligence (AI) has significantly increased the reliance on Data Centers (DCs), making them crucial for processing and storing vast amounts of data. However, this surge in AI deployment has highlighted an environmental concern of Carbon Dioxide (CO_2) emissions generated by the DCs. These facilities are resource-intensive and demand substantial power to meet the computational needs of AI applications, thus contributing to a high carbon footprint. To address the issue, this thesis explores an innovative approach to measure the CO_2 emissions by introducing a linear regression energy model based on Performance Monitoring Counters (PMCs) such as the total number of instructions and the total number of cycles of the computer processor and the development of energy-efficient AI models by optimising the hyperparameters and architecture of AI models to minimise the impact on the environment.

The operational efficiency and environmental impact of DCs have been estimated based on metrics such as Power Usage Effectiveness (PUE), partial Power Usage Effectiveness (pPUE), and Carbon Usage Effectiveness (CUE). Several types of research have been conducted to optimise hardware such as processor idleness, power supply to the machine, cooling machines for the system, and selecting training locations with low carbon intensity to lower energy consumption. However, such improvements are insufficient since inadequately developed AI models can drastically drain the processor power. Therefore, engineers should focus on developing highly efficient and computationally feasible models. During this thesis, PMCs are used to estimate the computational complexity of AI models running on processors. It has been observed that processor-specific PMCs, like the total number of instructions and the total number of cycles collected during processing, strongly correlate with the processor's energy consumption. They also impose very minimal overhead on energy utilisation, making them ideal for usage with AI applications. Therefore, PMCs have been used to calculate the energy consumption of processors and the DCs they are placed in.

Central to our research is the formulation of an energy model that utilises PMCs to estimate processors' energy consumption and CO_2 emissions. By training various AI models on the Central Processing Unit (CPU), collecting Performance Monitoring Counter (PMC) data, and their associated energy consumption, a linear regression energy model to estimate the energy usage of AI applications is established. Subsequently, the CO_2 emissions of applications running on these Central Processing Units (CPUs) are also calculated. For the simplicity of this research, only CPU and Dynamic Random Access Memory (DRAM) are taken into consideration, as they consume the maximum energy in comparison to other parts of the processor. This linear model produced an error of only 0.158% for CPU and 0.272% for DRAM. Further, the implications of hyperparameter optimisation and model architecture on energy consumption and CO_2 emissions have been studied based on PMCs with a tradeoff in accuracy.

This research will enable the estimation of energy consumption and CO_2 emissions of AI applications based on inbuilt PMCs, and also reduce energy consumption and CO_2 emissions by modifying the model architecture and hyperparameters while maintaining a tradeoff between accuracy and energy consumption.

Contents

List of Figures

List of Tables

Acronyms

ρ Spearman's Rank Correlation Coefficient. 11

R^2 R Squared. 44

AI Artificial Intelligence. 5

AMD Advanced Micro Devices. 34

API Application Programming Interface. 33

APIs Application Programming Interfaces. 33

BERT Bidirectional Encoder Representations from Transformers. 18

ChatGPT Chat Generative Pre-trained Transformer. 9

CO_2 Carbon Dioxide. 5

CPU Central Processing Unit. 5

CPUs Central Processing Units. 5

CUE Carbon Usage Effectiveness. 5

DC Data Center. 9

DCs Data Centers. 5

DL Deep Learning. 31

DNN Deep Neural Network. 17

DRAM Dynamic Random Access Memory. 5

EPA Environmental Protection Agency. 32

GHz gigahertz. 11

GPU Graphical Processing Unit. 17

GPUs Graphical Processing Units. 18

HDD Hard Disk Drive. 20

IPC Instructions per Cycle. 25

IT Information Technology. 29

kg kilogram. 9

KNN K-Nearest Neighbors. 40

KWh	Kilowatt-hour. 47
L2	Level 2. 30
LHS	Left Hand Side. 9
LLMs	Large Language Models. 17
LR	Learning Rate. 7
LTS	Long Term Support. 39
MAC	Memory Access Patterns. 59
MAE	Mean Absolute Error. 44
ML	Machine Learning. 17
MSRs	Model-Specific Registers. 35
MWh	megawatt-hour. 47
NAS	Neural Architecture Search. 18
NLP	Natural Language Processing. 32
NN	Neural Network. 35
OS	Operating System. 30
PAPI	Performance Application Programming Interface. 7
PMC	Performance Monitoring Counter. 5
PMCs	Performance Monitoring Counters. 5
PP1	graphics. 39
pPUE	partial Power Usage Effectiveness. 5
PUE	Power Usage Effectiveness. 5
RAPL	Running Average Power Limit. 7
ReLu	Rectified Linear Unit. 53
RHS	Right Hand Side. 9
RQ	Research Question. 7
SMI	System Management Interface. 68
SSD	Solid State Drives. 20
SVM	Support Vector Machine. 40
TLB	Translation Lookaside Buffer. 32
TPU	Tensor Processing Unit. 17
TPUs	Tensor Processing Units. 32

TWh	terawatt-hour. 9
U.S.	United States. 32
vEC	Virtual Energy Computer. 31
vs.	versus. 25
Wh	watt-hour. 9

1 Introduction

The field of AI is witnessing substantial expansion in high-performance computing clusters and DCs because of their intensive processing requirements and energy consumption. Advancements in AI models and algorithms have significantly improved the capabilities in the domains of machine translation, speech recognition, and object detection. Accelerators such as the Graphical Processing Unit (GPU) and Tensor Processing Unit (TPU) have greatly influenced the progress in the training of large AI models such as Deep Neural Network (DNN) models. However, AI models are computationally demanding due to their large datasets, extensive model sizes, and numerous parameters and weights used for training. Developing these models also requires thorough experimentation with various hyperparameters, resulting in a significant demand for resources during the training. This imposes great stress on the DC processors, leading to more energy utilisation. In addition to being financially demanding, AI applications also have a negative impact on the environment due to the substantial CO_2 emissions generated by the high-end tensor computations on the processor. The training of state-of-the-art models is especially energy-intensive, with certain models, such as Large Language Models (LLMs) requiring weeks or months to complete, resulting in substantial consumption of computational resources and implications for sustainability [MMN+24]. A decade ago, training of most Machine Learning (ML) models was manageable on standard laptops or servers. However, today they require advanced, costly hardware like a GPU or TPU [Jha11].

In the analysis presented by Goldman Sachs [Sac24b], the DC power demand has increased from 1%-2% in 2022 to 3%-4% in 2023. It is anticipated that the demand for power in DCs will increase by 160% between the years 2023 and 2030, as shown in Figure 1.1. The graph shows that in addition to DCs without AI (ex-AI), the power consumption of AI-equipped DCs has significantly increased since 2023 and is expected to continue rising until 2030. They also predict that the growth in power demand in DCs would result in an increase in CO_2 emissions from DCs by more than 100% (almost 215-220 million tons) by the year 2030 compared to the year 2022. While AI offers numerous benefits across various sectors and is crucial in today's world, it is essential to regulate its use due to its carbon emissions and environmental impacts.

DCs rely on many energy sources to maintain their operations. While some energy sources are renewable or compensated through carbon credits, the overall energy demand is still a major concern. Non-renewable sources currently dominate energy generation in several places. In Figure 1.2, we can see that although the DC workload demand is increasing exponentially, the DC power sourced from renewable sources is seemingly less [Sac23; Sac24b].

Goldman Sachs [Sac24a] has forecasted that power demand from AI will increase by approximately 200 TWh from 2024 to 2030, with AI expected to account for around 20% of the total DC power demand by 2030. One of the most computationally intensive AI applications called LLMs are characterised by their vast number of parameters and intricate model structures and therefore require significant amounts of energy and resources to operate. Not only does the training of LLMs

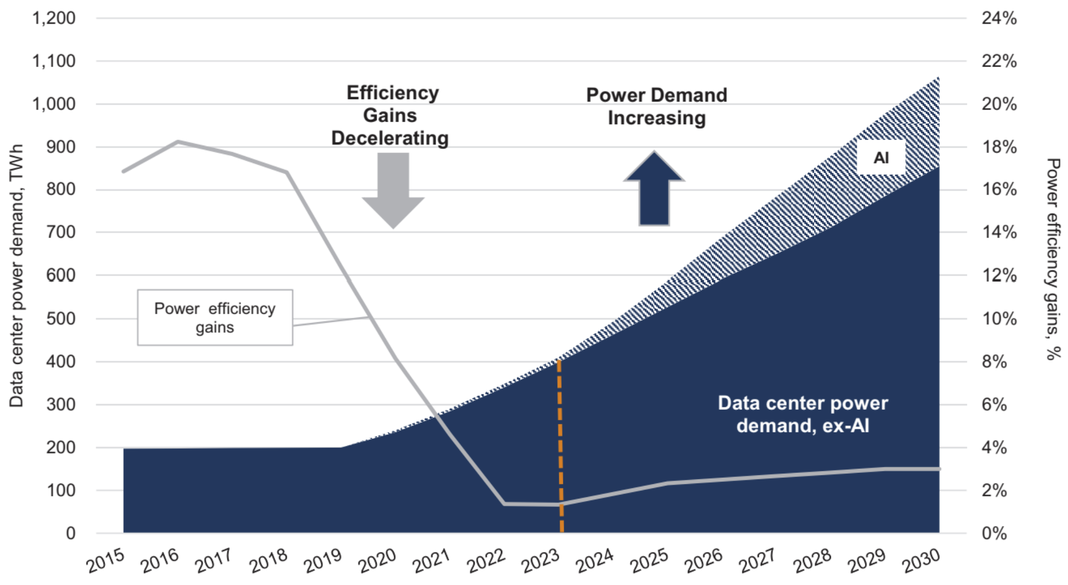

Figure 1.1: Graph of DC power demand in TWh (LHS) and power efficiency gains % (RHS) [Sac24b]

consume substantial amounts of energy, but their inference is also highly energy-intensive. For instance, using models like ChatGPT requires significant computational power each time a query is processed. In Figure 1.3 we can see that a ChatGPT search consumes about 6 to 10 times more power than a traditional Google search [Sac24a]. The carbon footprint of training LLMs like Bidirectional Encoder Representations from Transformers (BERT) on Graphical Processing Units (GPUs) is comparable to the emissions from a New York to San Francisco flight [SGM19]. Research indicates that running Neural Architecture Search (NAS) can result in nearly 626,000 pounds of CO_2 emissions that exceed the lifetime emissions of several cars [SGM19; SLL19; WSS+23]. The substantial carbon emissions generated by LLMs pose a significant risk to the environment. As the usage of these models expands, the environmental impact intensifies, contributing to global climate change. It is imperative to develop and implement strategies to reduce these emissions.

Historically, developers have focused on software qualities such as performance and reliability without considering energy efficiency. This has often led to highly energy-intensive applications. Such power-intensive AI applications can deplete 30% to 40% of the device's battery life [Jha11; SCC+12]. Recent research has indicated a growing inclination towards estimating the energy usage of software programs, highlighting the significance of energy efficiency in software development to reduce the impact on DCs. It is necessary for existing procedures to be modified to include the reporting of training accuracy and energy requirements. This will ensure that all researchers have fair access to computational resources and that the development of energy-efficient models and computer infrastructure is given priority.

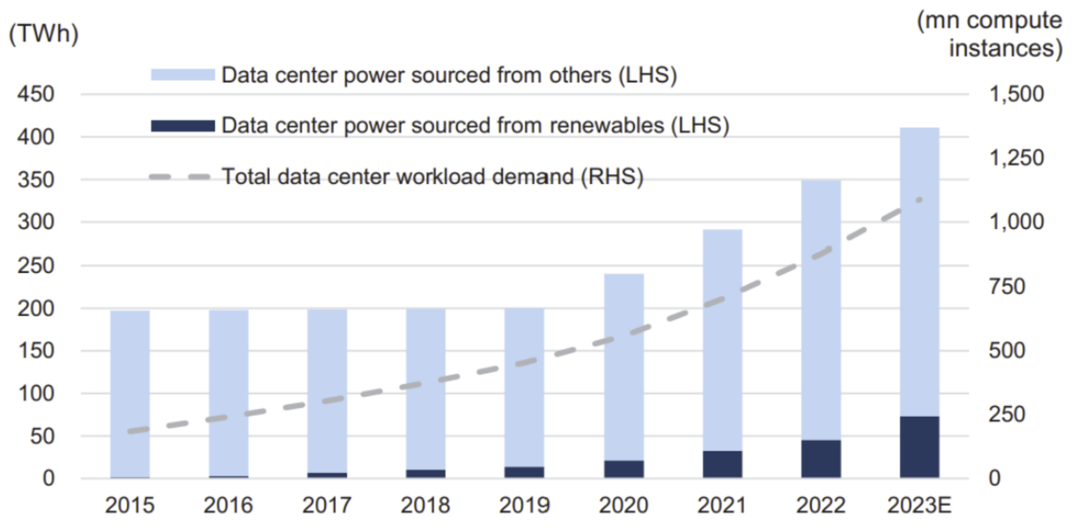

Figure 1.2: Graph of DC workload demand in million compute instances (RHS) and data center power demand in TWh (LHS)[Sac24b]

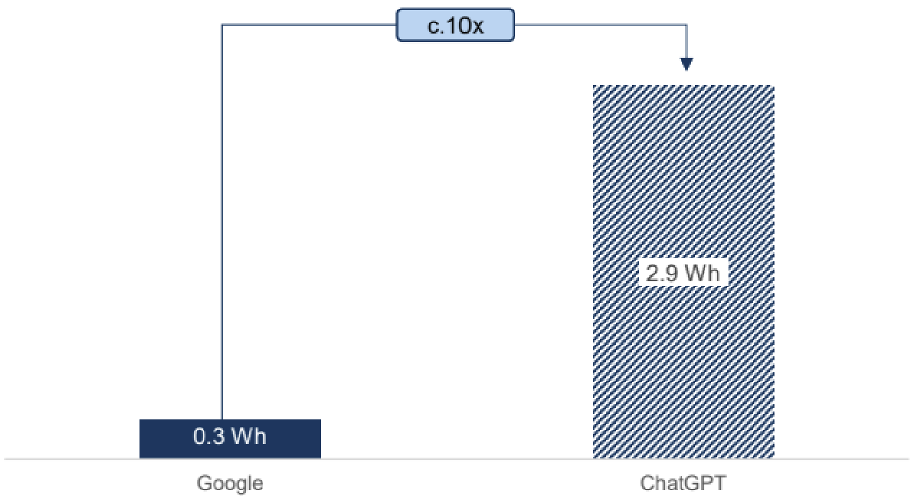

Figure 1.3: Power consumption per query/search in Wh for ChatGPT [Sac24a]

1.1 Problem Statement

In today's technology-driven world, DCs play a crucial role due to their computing capabilities. However, it is essential to recognise their environmental impact. Running AI applications in DCs is progressively increasing the energy consumption. To mitigate CO_2 emissions from DCs, various strategies have been implemented, including advanced cooling technologies, energy-efficient hardware, selecting locations with low carbon intensity, and the adoption of renewable energy sources[HSA22; RSR+17]. However, these measures are meaningless if the AI applications running in the DCs are not efficient. Therefore, highly energy-efficient models should be developed.

While DC energy metrics such as PUE, pPUE, and CUE provide information about the operational efficiency of DCs, a fine-grained analysis is essential. A thorough insight into the energy consumption of AI applications running on DCs is crucial for understanding the specific energy demands at the component level. This thesis aims to meticulously quantify the energy consumption of such AI applications, focusing specifically on the individual components of computing devices such as CPU and DRAM. We concentrate on the CPU and DRAM energy evaluation since they have the largest and most immediate influence on the AI training processes. We do not consider data storage (Solid State Drives (SSD), Hard Disk Drive (HDD)) energy consumption as they do not directly influence the running processes [CM05]. The need for this analysis is crucial as AI applications are becoming more complex in their quest for greater accuracy. This complexity leads to increased energy usage, which is predominantly overlooked when developing new models. Acquiring new hardware to support advancements in this field is both cost-intensive and environmentally unsustainable. Therefore, it is essential to optimise code to mitigate the model training process and increase its energy efficiency.

1.2 RQ

To study how PMCs contribute to the precise measurement of energy and CO_2 emissions, the following research questions have been formulated.

- **RQ1:** Is PMC correlated to the energy consumption of AI applications?
- **RQ2:** How can precise energy consumption of AI applications be measured based on PMCs?
- **RQ3:** How can precise CO_2 emissions of AI applications be measured?
- **RQ4:** How do hyperparameters and architecture of AI applications impact PMCs and energy consumption?

1.3 Thesis Organisation

The organisation of this thesis is designed to ensure a precise investigation of approaches to measure and reduce the energy consumption and CO_2 emissions of AI applications.

The introductory chapter examines the pivotal significance of DCs in the advancement of AI technologies, with particular emphasis on their growing participation in the execution of large AI models. This chapter investigates the substantial energy demands of DCs, which are intensified by the deployment of AI applications. Concepts to precisely measure the energy consumption and CO_2 emissions associated with AI model training are also discussed. The second chapter provides a thorough examination of the current techniques, establishing the foundation for comprehending the high energy consumption in DCs caused by AI applications. It also describes the techniques used to obtain precise energy measurements. This includes the use of PMCs as a dependable approach for tracking energy use in computational tasks.

In the third chapter, we critically examine existing literature that has made attempts to quantify software energy usage. It identifies the current gap in research and prepares for future investigations. The fourth chapter includes a thorough discussion of the various tools and instruments used in the research, highlighting their significance and the reasons behind their selection for the research.

The fifth chapter gives an overview of the methodology used. Chapter six introduces the first methodology of using PMCs to determine the energy consumption of an AI model. It outlines the process of creating an energy model using PMC data and then computing CO_2 emissions. Further, the seventh chapter goes into additional detail about the effects of hyperparameter optimisation and model architecture on PMCs and energy consumption.

The thesis concludes with chapter eight, which discusses the knowledge gathered from the research. It summarises the study's findings and suggests areas for more research, guiding future experiments on GPU that will build on the foundation created by this thesis. This comprehensive method aims to contribute significantly to the discussion on sustainable AI development and the need for energy-efficient computational processes.

2 Background

It is crucial to understand the fundamental aspects of energy usage in processors before examining the energy consumption of AI applications. The below section offers a comprehensive overview of how processors consume power and energy to get a deeper understanding of energy consumption of AI applications. In the context of computing, energy and power are two essential concepts for understanding processor efficiency and performance.

2.1 Energy Consumption

Energy: It refers to the total amount of electrical power consumed over time and is measured in joules (J).

Power: It is defined as the rate at which energy is used and is measured in watts (W). Power has both static and dynamic elements. They are defined below:

- **Static Power:** It is the energy consumed by the processor in an idle state. This type of power consumption occurs due to the leakage of electrical current within the processor's transistors. Therefore, it is also known as leakage power.
- **Dynamic Power:** It is primarily associated with the charging and discharging of capacitive loads when transistors switch states from off to on and vice versa. The amount of dynamic power consumed is influenced by several factors such as the operating voltage, the clock frequency of the processor, and the number of active transistors during a given operation. Dynamic power consumption is significant because it directly correlates with the processor's workload. Reducing dynamic power can be achieved by optimizing software algorithms [GRRG19].

The formula for dynamic power ($P_{dynamic}$) is as follows:

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{2.1}$$

where:

- α denotes the activity factor, which represents the percentage of the circuit that is in an active state.
- C denotes the capacitance.
- V_{dd} denotes the voltage.

- f denotes the clock frequency.

The energy consumed over a time interval can be calculated using the integral:

$$E = \int_0^T P(t)\,dt \tag{2.2}$$

where:

- P(t) denotes power consumption over time dt.
- dt denotes the time interval.

Learning these foundational concepts is crucial for gaining a deeper comprehension of the dynamics of power and energy in processors. Understanding the concept of static and dynamic power in processors is particularly relevant for AI applications as they are often computationally intensive and require significant processor resources. AI algorithms, especially DNN, can cause processors to operate at high levels of dynamic power consumption for long hours.

2.2 Tools to Measure Energy Consumption of AI Applications

In this section, we investigate the various tools used to measure computer system energy consumption. We examine both conventional and more advanced techniques that provide information about the energy consumption of hardware and software components [GRRG19].

- **Traditional Empirical Measurement Methods:** Power meters are one of the traditional ways to measure a computer's power consumption. These gadgets can be put on the motherboard or at a wall outlet to measure energy usage. This conventional method gives precise measurements of actual power consumption at particular periods. However, they are ineffective in identifying program-wise energy consumption. When attempting to optimise software for energy efficiency, this restriction is a significant disadvantage because it offers only limited knowledge about the correlation between code execution and power consumption.
- **Using Simulators for Detailed Analysis:** To attain a more precise evaluation of energy consumption, simulators are used more often. One of the primary benefits of using simulators is that they enable researchers to visually inspect how specific programs interact with each hardware component. This particular capability is of utmost importance in comprehending the interaction between hardware and software in terms of energy consumption. In addition, simulators enable precise measurement of the energy consumption of various programs through the facilitation of instrumentation.

  The main disadvantage of utilising simulators is their intrinsic overhead. The performance of the system is adversely affected by this overhead, making these tools inappropriate for real-time energy measurement. Therefore, despite simulators being indispensable tools for research and development, their utility is constrained in scenarios involving real-time data.

- **Performance Monitoring Counter (PMC):** PMC is another advanced technique used to measure energy consumption. PMCs have been incorporated into the majority of contemporary processors to monitor micro-architectural events within the CPU during processing. PMCs can provide a wide range of information about each core, ranging from cache accesses to Instructions per Cycle (IPC).

By utilising the comprehensive data offered by PMCs, numerous researchers have constructed power models that approximate energy consumption according to the events recorded by these counters. In addition, Intel has implemented an energy model called RAPL, which utilises PMC data to approximate the energy consumption of processors. The methodology employed by RAPL facilitates a more intricate understanding of the energy consumption patterns of various processes by establishing connections between energy usage and specific processing activities.

In this study, the RAPL interface was used to calculate energy consumption. This methodology aided in the direct correlation between energy consumption and performance data acquired from PMCs, thereby furnishing an analysis of the energy efficiency of AI models. AI applications can be optimised for improved energy efficiency by establishing precise correlations between AI applications and their energy implications through the integration of RAPL.

This extensive examination of energy measurement methodologies, spanning from conventional approaches to sophisticated simulative and monitoring techniques, exemplifies the diversity of resources accessible to scientists seeking to improve the energy efficiency of computing systems. Every method presents its own set of advantages and disadvantages, emphasising the significance of choosing the most suitable approach based on the particular demands and restrictions of the research or application setting.

2.3 Approaches to Measure Energy Consumption of AI Applications

This section describes the various methodologies employed to model power and energy consumption in computing systems. These approaches are systematically categorised based on three distinct criteria: type, technique, and level of modelling. Each category offers unique insights and methodologies for understanding and quantifying energy consumption, critical for enhancing the energy efficiency of processors and software [GLG+19].

2.3.1 Type: Empirical versus (vs.) Analytical Models

Type refers to the scientific modelling approach utilised, distinguishing between empirical and analytical models.

- **Empirical Models:** These are developed based on direct observations and measurements from specific types of processors. These models are highly specific and tailored to particular hardware, capturing the unique characteristics and behaviours of the observed system. However, their applicability is generally limited to the type of processor they were derived from, making them less versatile across different hardware platforms.

- **Analytical Models:** In contrast, these models are based on mathematical formulations that describe the power behaviour across various components of potentially multiple processor types. These models use theoretical frameworks and equations to estimate power consumption, offering broader applicability across different types of processors. They provide a generalised understanding of energy dynamics but may lack the precise accuracy of empirical models in specific cases.

2.3.2 Technique: Direct Measurement vs. Simulation

Technique encompasses the methods used to model energy consumption, primarily through direct measurement or simulation.

- **Direct Measurement:** This involves using tools such as PMCs to gather real-time data on processor activities. This approach is advantageous in scenarios requiring immediate feedback, such as machine learning applications where timely data is crucial for optimising processes and reducing energy overhead.
- **Simulation:** This offers a theoretical approach in which hardware behaviour is modelled and executed in a controlled environment. This method allows for extensive instrumentation and the detailed study of how different parts of the hardware are utilised during operations. While simulations provide deep insights and facilitate detailed analysis without risking actual hardware, they often come with significant computational overhead and may not reflect real-world operational discrepancies.

2.3.3 Level: Architecture vs. Instruction

Level pertains to the granularity at which the energy modelling is conducted.

- **Architecture-Level Models:** These models break down the energy consumption into various components of the system architecture, such as the CPU cores, cache, and DRAM. These models are particularly useful for hardware designers and system architects interested in optimising specific hardware components or subsystems.
- **Instruction-Level Models:** These models allocate energy costs to individual instructions executed by the processor. This fine-grained approach is invaluable for software developers and engineers seeking to optimise code, as it identifies which specific instructions contribute most to energy consumption, thereby identifying potential 'energy hotspots' within the code.

This methodology section offers a thorough framework for comprehending and quantifying the energy consumption of computer processors by examining the various approaches and considering their types, techniques, and levels of detail. Because each strategy has unique advantages and disadvantages, it can be used in various application scenarios and research needs. These models provide important insights that will guide developments in software and hardware design to minimise the carbon footprint of AI computing tasks.

In this research, an empirical type of model has been developed that is specific to Intel-based processors. PMCs have been used as a direct measurement technique to generate the architectural level models. An architecture-level model with a focus on CPU and DRAM energy consumption has been developed with PMC tool for measurement.

3 Related Work

In this section, we provide an in-depth exploration of the existing research focused on the energy consumption of AI applications, with a particular emphasis on the utilisation of PMCs to estimate the energy usage of processors. This review highlights significant contributions and methodologies that have shaped the current understanding and approaches towards optimising and monitoring energy efficiency in computational systems running AI applications.

Initially, an extensive review of the DCs hosting these AI applications was performed. Numerous research papers have been published that describe the architecture of DCs, the criteria used to assess their efficiency, and the different strategies employed to reduce their energy usage. These studies provide detailed insights into DC structural design, highlighting advances in hardware and software combinations that improve performance while minimising environmental impact. Reddy et al. [RSR+17] have conducted an in-depth study of various metrics such as energy efficiency metrics, green metrics, cooling metrics, performance metrics, thermal and air management metrics, network metrics, storage metrics, security metrics, and financial impact metrics for sustainable DCs. Avelar, Azevedo, and French [AAF12] researched one of the most important energy efficiency metrics called PUE used to measure the efficiency of a DC in terms of the ratio of total DC facility energy consumption to Information Technology (IT) energy consumption. They concluded that it is also essential to consider the geographical and weather conditions at the DC location. Haghshenas, Setz, and Aiello [HSA22] and Haghshenas et al. [HSBA22] discuss the best practices for optimising energy usage, such as advanced cooling techniques, energy-efficient hardware, using renewable energy sources, suspend-resume scheduling techniques, and selecting locations with low carbon intensity. Through these efforts, the study emphasises the significance of sustainable approaches in addressing the increasing energy demands of DCs with AI workloads. While methods work to improve energy efficiency at a high level, fine-grained analysis is also required. The previous research discussed further in this chapter explains the analysis of energy consumption based on software-level metrics.

The pioneering work by Tiwari et al. [TMWL96] was the first of its kind and marked a significant advancement in understanding how software influences power consumption, especially within power-sensitive environments like mobile and embedded computing systems. This paper introduces a novel, instruction-based power analysis approach aimed at quantifying the power costs associated with software operations. This methodology was groundbreaking because it shifted the focus from traditional hardware-centric power analysis, which predominantly relied on circuit and gate-level assessments, to a more granular, software-oriented perspective. The core of this methodology involved analysing power consumption directly tied to software instructions. They meticulously measured the power usage of individual instructions and their combinations within typical execution scenarios on three different processors. Utilising the data obtained from their measurements, the researchers predicted models of power consumption. These models provided insights into how different instructions and their sequences impact overall power usage, offering a predictive tool for assessing software's power efficiency. A significant outcome of their research was the identification of low-power software implementations. By understanding which instructions or sequences consumed less power, developers could optimise software to be more energy-efficient, which is crucial for extending the battery life of mobile devices and the overall efficiency of embedded systems.

An approach for calculating both dynamic and static power consumption in multicore processors was developed by Goel and McKee [GM16]. They emphasised the importance of uncore energy and the potential energy inefficiencies from using more cores. Alonso et al. [ADMQ14] expanded on this work by proposing a model that takes memory contention and power dissipation into account when calculating the energy consumption of dense matrix factorisations. Basmadjian and Mee [BM12] provided a more precise methodology that takes resource-sharing and power-saving measures into consideration. The research challenged the notion that power consumption in multi-core processors is just the sum of the power of each core. Finally, this research adds to a thorough knowledge of multi-core processor power consumption and offers insightful information for designing and using energy efficiently.

The research by Bellosa [Bel00] significantly improved awareness of using PMCs for calculating processor power consumption. This method was one of the first to incorporate the use of PMCs for power estimation, especially in the context of making Operating System (OS) schedulers more energy-efficient. The research discovers that PMCs integrated into the majority of modern CPUs, could act as effective indicators of power usage. PMCs offer a view into the energy consumption of the CPU by measuring a variety of performance events, including memory references, floating point operations, and Level 2 (L2) cache references. The paper proposes that an OS scheduler can prioritise or manage processes in a way that maximises system energy efficiency by calculating thread power requirements from observable performance events. The foundation for later advancements in power-aware and energy-efficient scheduling algorithms was established by this concept. It has sparked more investigation into how operating systems and other system-level software might use comprehensive performance data to reduce energy usage, particularly in settings like DCs and mobile devices where power economy is critical.

An advanced approach was presented by Joseph and Martonosi [JM01], which used PMC data as proxies for measuring computations. The study suggests creating capacitance-based power models using PMC. These models compute the power consumption based on the known capacitances of circuit components, which change depending on the operational state and utilisation. The power models used are grounded in the physical properties of the processor's circuitry, specifically, the capacitance associated with different functional units. The model provides an estimate of the power usage by establishing a connection between these capacitance values and the activity that PMCs record. This method has a significant drawback because it requires an in-depth understanding of circuit-level implementation, which may not always be possible for all processors. This includes knowledge of capacitance information. The requirement for comprehensive hardware specifications may restrict the model's capacity to be applied and generalised to various processor kinds in situations where such extensive technological data is not easily available.

The study conducted by Singh, Bhadauria, and McKee [SBM09] explores various approaches for measuring CPU power consumption. The authors focus on using PMCs to measure power consumption in real-time accurately. This approach differs from conventional techniques that use costly equipment or power meters to measure power directly. The study uses PMCs to gather data from micro-benchmarks. The research creates analytical models that predict power consumption more accurately across various computing scenarios. The approach divides the analysis into different categories based on the type of processor activity, such as floating point operations and memory access, which are critical in understanding and managing power consumption.

Virtual Energy Computer (vEC), an innovative tool was created to estimate user program energy usage in embedded scenarios. This technology perfectly meets the requirements of systems where energy efficiency is critical. It is considered a substantial development in energy monitoring approaches [KCK+01]. A model to predict the power usage of an OS during runtime was also suggested by Li and John [LJ03]. The model focused on the relationship between power consumption and IPC. Hu, Li, and Kuo [HLK05] created a runtime power consumption model for multimedia application routines in embedded systems, which allowed for precise power usage estimation. Yang et al. [YLL+16] extended these models by proposing a method that emphasises energy optimisation and uses performance events to predict applications' overall system power consumption.

While the above research focused on the energy consumption of computer applications or user programs, García-Martín et al. [GLG+19] researched the energy consumption of ML applications. The research provides a comprehensive examination of energy consumption estimation techniques within the context of ML applications. The research offers a thorough examination of modern approaches for estimating energy usage that is customised for various ML settings. State-of-the-art methods for estimating energy usage, emphasising their applicability and significance in a variety of ML scenarios are studied. This survey emphasises the implications of both new and established methodologies for computational resource optimisation. The energy estimation techniques are categorised in the study based on how well they function in different ML scenarios, like processing big datasets, training DNN, and executing inference models. They discussed the various types of analytical and empirical methods used to measure the energy consumption of ML applications. Simulation and PMC were also discussed as the best practices for measuring energy consumption. This classification aids practitioners in selecting the best methods for their particular machine-learning tasks.

While energy is beginning to be used as a parameter in ML, the majority of research efforts are now directed toward attaining high levels of accuracy with no computational constraints. Their lack of experience with energy assessment techniques explains why they have no curiosity. García-Martín et al. [GRRG19] provide an overview of the many methods for estimating energy usage of ML applications. The goal of this research was to give the ML community helpful recommendations and the basic information they needed to develop specialised energy estimation techniques for ML algorithms. This research showcased the most recent energy estimation software tools, various methods for estimating energy usage, and energy estimation models. They proposed various taxonomies of software and hardware-related energy estimation models.

In their study, Haghshenas, Setz, and Aiello [HSA22] explore the fluctuation of CO_2 signal intensity as it comes from the power grid and illustrate the possible emission reduction that may be achieved by utilising the iterative nature of the training process. Two emission-aware strategies were proposed to accomplish this goal of temporarily shifting the training jobs and migrating them between locations. Experimental data on the power and CO_2 emissions of the training process, as well as delay overheads linked to emission reduction approaches, were provided for a range of sample Deep Learning (DL) models. The results show that emissions may be efficiently decreased by 13% to 57% of the baseline scenarios by following emission signals.

The paper by Strubell, Ganesh, and McCallum [SGM19] analyses the computational and environmental costs of training DNN models for Natural Language Processing (NLP), providing an insightful case study of the full computational resources required for the development and tuning of a recent state-of-the-art NLP pipeline. They show that Tensor Processing Units (TPUs) are more economical than GPUs for some workloads by estimating the cost in terms of cloud computing and carbon emissions for training different NLP models. In addition to encouraging research on more computationally efficient algorithms and fair access to computational resources for university researchers, the paper highlights the significance of publishing training time and sensitivity to hyperparameters. It also provides a formula for power consumption and CO_2 emissions based on power drawn from CPU, DRAM and GPU sockets. This is then multiplied by the average CO_2 produced (in pounds per kilowatt-hour) for power consumed in the United States (U.S.) (Environmental Protection Agency (EPA), 2018) to get the estimated CO_2 emissions. This paper has been used as a basis for CO_2 emission estimation in our research.

Using PMC events, power prediction for Intel XScale processors has been investigated in a number of studies. Power estimation models mapping hardware events to power consumption were created by Qianjie et al. [QTJ+06] and Contreras and Martonosi [CM05]. Contreras and Martonosi [CM05] were able to achieve an average estimated power consumption within 4% of the measured average CPU power usage. They came up with a power estimation model for the Intel PXA255 processor, which is used in embedded systems with strict power requirements. The model used PMCs to estimate CPU and DRAM power consumption, allowing for dynamic adaptation to the device's power consumption. It presents a linear power estimation model that links PMCs such as instructions executed, data dependencies, instruction cache misses, and Translation Lookaside Buffer (TLB) misses with power consumption and demonstrates its accuracy in estimating power consumption within 4% of measured values. The model also allows for parameterisation of power at various voltage/frequency settings and lightweight implementation in C for quick estimation of CPU and DRAM. The results of this paper have been considered the basis for our research and it has shown better results based on PMCs that have a very high correlation with energy consumption of CPU and DRAM.

Previous studies have attempted to estimate the energy consumption of software by utilising PMCs across various benchmarks. Building upon this foundational research, our study advances these efforts by specifically targeting AI applications. We achieve this by running ML applications to develop our energy estimation models. This approach allows for a more precise and tailored understanding of the energy requirements unique to AI workloads.

With this research, we aim to revolutionise the AI industry. Our findings have the potential to significantly impact the AI community by providing valuable insights into the energy consumption patterns of ML models. This step can guide the development of more energy-efficient AI systems. Implementing these energy-efficient practices can lead to reduced operational costs and a lower environmental footprint, making the deployment of AI technologies more sustainable. Our research contributes to the scientific understanding of AI energy consumption and offers practical tools and methodologies that can help AI developers optimise their models for better energy efficiency.

4 Tools Utilised

To conduct the research in this thesis, Linux Application Programming Interfaces (APIs) and Python packages that enabled precise evaluation of processor PMCs and energy consumption of CPU, and DRAM were utilised. The tools explained below provide a structure for gathering comprehensive data that was critical to the research.

By integrating these APIs and Python packages, we were able to establish an automated monitoring environment that recorded the energy consumption and PMCs while various AI models were executed. By following this systematic approach, it was possible to gather precise, real-time data and also facilitate the study of the relationship between code execution patterns and their energy consumption. These observations are of utmost importance to suggest optimisations that may reduce the energy consumption of AI applications, thereby contributing to the larger objective of sustainable computing practices.

4.1 PAPI

The PAPI is a portable way to access hardware PMCs available on most modern processors, independent of the machine and operating system. It enables the tracking of over 100 predefined events through high-level and low-level programming interfaces [MMDH99]. Simple high-level Application Programming Interface (API) functions like starting, stopping, and reading counters for predefined CPU events are its main features. The high-level API comprises eight functionalities. The low-level API, on the other hand, is intended for more experienced developers who require exact control and measurement capabilities. It supports both PAPI presets and native events across all supported components and enables the control of hardware events grouped into user-defined event sets.

PAPI was created to give researchers, developers, and performance engineers a standardised approach and interface to access the low-level and high-level PMCs included in the majority of contemporary microprocessors. These counters offer useful information on the processor's performance, such as the number of CPU cycles used, the number of instructions executed, cache hits and misses, and other metrics that are important for estimating the efficiency of software. PAPI is very useful in the area of performance optimisation, where knowledge of hardware interactions can result in considerable increases in software efficiency. It is frequently utilised by performance analysts seeking to optimise the performance of intricate software systems on diverse hardware configurations, in scientific computing, and for real-time system monitoring. Our research utilises the PAPI high-level events called PAPI_TOT_INS and PAPI_TOT_CYC [MMDH99]. Refer to Appendix 1 for more details.

The Python module known as PyPAPI serves as a wrapper for the PAPI API, enabling Python programmers to utilise the hardware PMCs available via PAPI [Flo24; Pyp24]. PyPAPI is an essential package for researchers and developers to access hardware PMCs from Python code in the context of high-performance computing and performance-critical AI applications. It enables a thorough analysis of how various algorithmic decisions, model designs, and hyperparameter settings translate into hardware-level operations. With this tool, researchers can make more knowledgeable choices about the design and implementation of AI models by measuring these elements and establishing clear links between algorithmic efficiency and hardware usage. PyPAPI's PMC measurement capability can also help clarify how energy-efficient AI algorithms are. PyPAPI contributes to this thesis on AI applications in various ways, especially to investigate the effective use of computing resources.

Key Features of PAPI:

- **Hardware Abstraction:** Developers can design portable code that runs on different architectures more easily by using PAPI, which offers an abstraction layer over the hardware-specific specifics of performance counters. This eliminates the need to customise code to match the unique performance monitoring capabilities of each processor.
- **Portability:** The interface guarantees that applications utilising PAPI can be deployed across different environments without requiring major modifications by supporting a broad range of platforms and processors.
- **Two Levels of API:** PAPI provides both a high-level and a low-level API. The high-level API is meant for customers who require basic performance data without fine-grained management. It makes routine activities like starting, stopping, and reading counters simpler. The low-level API offers comprehensive access to sophisticated users who need accurate performance statistics and the capacity to manage and modify different event sets.
- **Preset and Native Events:** In addition to native events unique to individual hardware models, users can access more than 100 preset events that are typical of many hardware platforms.
- **Multiple Programming Languages:** PAPI can be used in a variety of development situations and applications because it supports multiple programming languages, such as C and Fortran.

4.2 RAPL Interface

The RAPL interface is a feature found in modern Intel and some Advanced Micro Devices (AMD) processors that allow for monitoring and controlling the power usage of various components within the CPU and other parts of the chipset. RAPL is part of a broader suite of energy-efficiency technologies aimed at optimising power consumption and thermal outputs across different computing environments, from servers to desktops and laptops [Mai22; MWKJ17].

Python package PyRAPL is designed to measure computer program energy consumption, with an emphasis on the power used by a system's CPU and DRAM. RAPL is a capability of contemporary Intel CPUs that enables software to track energy consumption. This capability is used by PyRAPL to give Python applications a means of recording and reporting energy consumption statistics. It can measure the energy consumption of the CPU socket package, DRAM, and GPU [Lil18; Lil19].

Developers can monitor the energy used for Python program execution by using the pyRAPL Python package. PyRAPL has fine-grained control over what is measured; it can be configured to track the energy consumption of particular code blocks or the whole program execution. With ML and other computationally demanding tasks, where knowing the energy footprint is essential to creating affordable and sustainable solutions, this feature is extremely useful. PyRAPL can be used, for example, to identify the most energy-intensive operations in Neural Network (NN) training or large-scale data processing tasks, directing efforts to improve algorithms for increased energy efficiency.

PyRAPL is a necessary instrument in the framework of this research that looks into exact measurements of the energy consumption of AI applications. Such data can then guide plans to reduce the carbon footprint of AI applications, balancing technical progress with the sustainability of the environment.

Key Features of RAPL:

- **Power Monitoring:** The RAPL interface is a feature available in modern Intel and AMD processors that enable monitoring and controlling the power usage of the computing unit. RAPL is part of a broader suite of energy-efficiency technologies aimed at optimising power consumption and thermal outputs across different computing environments like servers, desktops and laptops.
- **Power Capping:** RAPL enables the real-time monitoring of energy consumption in various domains of the processor, such as the CPU cores, the uncore, the DRAM, and the package as a whole. Besides monitoring, RAPL allows the system to enforce power limits on these components. This feature is critical in situations where energy efficiency must be maintained, such as DCs.
- **Programmatic Access:** Developers and system administrators can access RAPL through several interfaces, including specialised system calls in the Linux kernel, via the python pyRAPL package and direct access via Model-Specific Registers (MSRs) for those needing granular control. RAPL can be used to dynamically manage power usage in server environments, balancing energy use and performance to increase efficiency and save operating costs. RAPL can be used by developers to fine-tune an application's performance, particularly to comprehend power-performance trade-offs in software architecture.
- **Data Centers:** RAPL allows dynamic power consumption management in DC environments, balancing energy and performance to optimise efficiency and save running costs.
- **Performance Tuning:** Developers can utilise RAPL for fine-grained performance tuning of applications, especially to understand power-performance trade-offs in software design.
- **Research and Development:** RAPL is extensively used in power-aware computing research, both academic and industry, to quantify the energy impact of various algorithms, system configurations, and workload types.

5 Methodology

This research aims to find the correlation between hardware PMCs such as the total number of instructions, the total number of cycles, IPC and the energy consumption of AI applications. These PMCs are crucial for understanding comprehensive performance statistics and enhancing software and hardware efficiency. Monitoring at a detailed level enables accurate fine-tuning of software and system settings, resulting in improved performance and decreased resource usage. PMCs offer a vital role in this research for assessing the operational efficiency of computing devices and determining their energy-associated CO_2 emissions. The correlation between energy consumption and CO_2 emissions is intricate and influenced by the composition of energy sources during the period of utilisation.

The first methodology proposes the use of CO_2 emissions to precisely measure the efficiency of AI applications. The primary objective is to develop an energy estimation model that utilises CO_2 to determine the energy usage of the CPU and DRAM during the training of ML models. We focus primarily on the training process of AI applications, as it is the most resource-intensive part of model development [HSA22]. By collecting PMC data and corresponding energy usage, a regression model is formulated to estimate the energy requirements of AI operations. In addition, we quantify the CO_2 emissions generated by the CPU and DRAM when executing AI models.

Furthermore, the second methodology investigates the influence of hyperparameter optimisation and model architecture on energy usage. We study the influence of various hyperparameters and model architectures like epochs, LR, number of neurons, activation functions and so on on the energy consumption of AI applications.

In conclusion, this thesis aims to make a valuable contribution to the field by providing approaches that improve the sustainability of AI research and guarantee that these advancements are also ecologically mindful.

6 PMC Based Energy Estimation Model

In this thesis, we present a highly portable and efficient methodology that utilises PMCs to measure energy and performance accurately. These counters are standard features on the majority of contemporary general-purpose processors. In particular, we use the widely-known RAPL interface for energy measurement and PAPI for PMC monitoring to estimate energy consumption and measure PMCs on processors.

The RAPL interface is structured into several domains, including the processor (CPU), graphics (PP1), and memory (DRAM), depending on whether the platform is a client or a server. In our study, we concentrate on the CPU and DRAM domains, which are typically the largest consumers of energy within a system.

6.1 Evaluation of Linux PMCs

Our methodology incorporates dynamic analysis, leveraging PMCs to gather event data in real-time as the ML model code is executed. The Intel® CoreTM i7-8565U CPU @ 1.80 GHz (142, 0x8e) running Ubuntu 22.04.3Long Term Support (LTS) and the Linux kernel version 6.5.0-28-generic served as the experimental platform for this study. This processor provides a strong foundation for comprehensive performance data collection by granting access via PAPI to 59 different PMCs. These PMCs are essential for developers, system administrators, and researchers looking to improve hardware and software efficiency through in-depth performance analytics.

Under Linux, the PMCs keep track of a variety of hardware events, including CPU cycles, instructions executed, cache hits and misses, and branch predictions. By measuring the impact of system modifications on performance, these counters play a crucial role in the diagnosis of performance problems in systems and applications. These counters also aid the alteration of software and system configurations with extreme precision, improving performance and reducing wasteful resource usage. These tools are invaluable in high-performance computing environments like DCs, where optimising operational efficiency is crucial.

We chose performance events that show a strong correlation with power consumption to avoid redundancy. The selection process involves determining which PMCs are most likely to affect the processor's energy-intensive functional units. The statistical method named Spearman's Rank Correlation Coefficient (ρ) was used to identify the relationship between PMCs and energy. This demonstrates the strength and direction of the correlation (positive or negative) between a given PMC and energy usage. For this coefficient, a value of +1 denotes a strong positive correlation, and a value of -1 denotes a strong negative correlation. A comprehensive list of PMCs supported by the Intel® CoreTM i7-8565U CPU can be found in Appendix 1.

6.2 Energy Consumption and PMC Correlation

This section aims to answer our first research question.

RQ 1: Is PMC correlated to the energy consumption of AI applications?

We answer this question by finding the correlation between PMCs and Intel processor (CPU and DRAM) energy consumption by executing various ML model programs. Detailed information on PMC activity and CPU and DRAM energy consumption is collected while model training.

Similar research has been done in the past on the relation between PMCs and energy use, but it has used traditional benchmark programs without ML [CM05]. Unlike other approaches, our work is the first of its kind to use ML applications for evaluation, and the results have been impressive. Our goal is to accurately model the energy consumption patterns, specifically for ML model training, which differs from traditional applications in terms of computational demand and energy usage. Therefore, specific AI models have been used to run tests and gather datasets.

To make this analysis easier, we used the RAPL interface to track CPU and DRAM energy consumption and the PAPI to collect PMC data. The PAPI interface offers the benefit of low overhead since data collection happens in parallel with running processes, allowing for the accurate recording of real-time event data during the execution of particular ML training code snippets [GLG+19]. Similarly, the RAPL interface offers a practical way to gauge energy usage without significantly interfering with already running computational processes [GRRG19].

The ML models employed in this study are listed below. The dataset was collected by training these ML models, giving a solid foundation to our predictive analysis.

- Linear Regression
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Decision Trees
- Neural Networks

We employ these models to obtain the energy and performance of only the model training stage. This facility is not provided by ML benchmarks currently. To guarantee that a thorough experimentation was carried out, a dataset comprising approximately 2.5k data points was gathered. Each model was executed with a range of dataset sizes, hyperparameters, and features to represent the various operational profiles of ML applications. The experiments gathered values of various PMCs offered by Intel® CoreTM i7-8565U CPU @ 1.80GHz (142, 0x8e) and the corresponding CPU energy and DRAM energy. After gathering the data, the patterns and strength of the correlations between the energy consumption metrics and the PMC data were examined using ρ.

The results in Table 6.1 and Table 6.2 show a list of the top 20 PMCs that have a strong correlation with CPU and DRAM energy consumption respectively. The total number of instructions executed and CPU cycles have a very high score as compared to other PMC events. It can be observed in Table 6.1 that total CPU cycles (PAPI_TOT_CYC) and CPU have a ρ of 0.922 and total instructions

(PAPI_TOT_INS) and CPU energy have a ρ of 0.861. Other PMC events do not perform very well in terms of correlation and have a ρ of around 0.55 and less. Table 6.2 shows that total CPU cycles (PAPI_TOT_CYC) and DRAM have a ρ of 0.869 and total instructions (PAPI_TOT_INS) and DRAM have a ρ of 0.751. Other PMC events exhibit poor correlation, with ρ values of 0.55 or lower. Therefore, these two PMCs have been used to model the CPU and DRAM energy consumption. In PMC terminology, instructions are the individual operations carried out by a CPU according to the program. The number of clock cycles that the CPU uses to complete tasks is measured by CPU cycles. Approximately one CPU cycle is required to do a single simple operation, such as adding two numbers. As a result, these two PMCs were determined to be crucial indicators for creating our energy model. Based on the above results, RQ1 has been answered and we can conclude that PMCs have a very strong correlation with the energy consumption of AI applications. Supporting the above correlation values, Figure 6.1, Figure 6.2, Figure 6.3, and Figure 6.4 depict the perfect linear behaviour between the two PMCs and the CPU and DRAM energy.

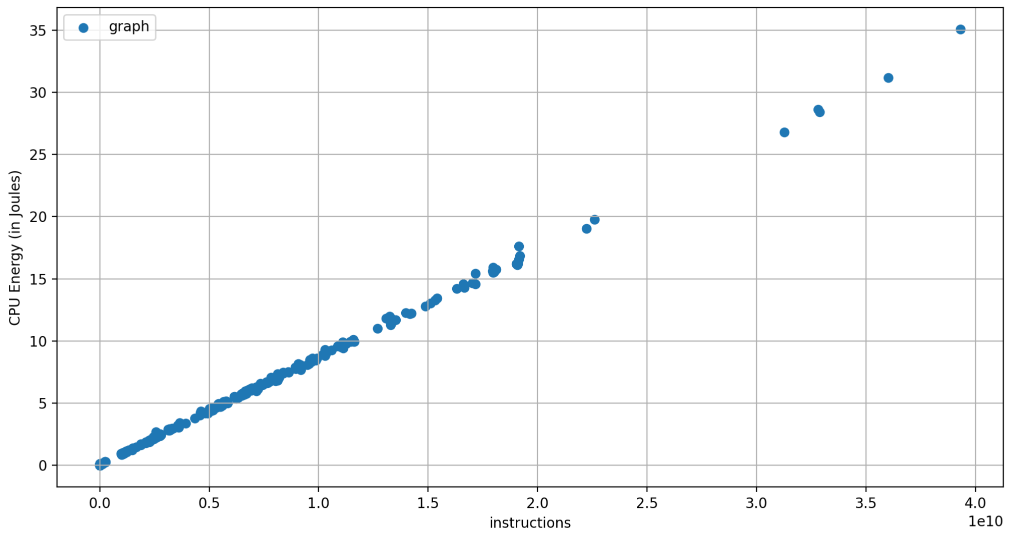

Figure 6.1: Graph of instructions and CPU Energy (in Joules)

6.3 Linear Regression Energy Model

This section aims to answer the second research question.

RQ 2: How can precise energy consumption of AI applications be measured based on PMCs?

In this section, we create a regression model to forecast the energy usage of different AI models operating on Intel processors. To achieve this goal, the total number of instructions, the total number of cycles, and the related CPU energy and DRAM energy were gathered from our generated dataset because they have a strong correlation with energy consumption of CPU and DRAM. Figure 6.5 describes the detailed process of creating the energy models.

The process involved generating the dataset by running the ML models to collect PMC and energy data. This dataset was passed through the data preprocessing stage, where the extreme outliers were removed and the dataset was normalised using Min-Max Scaling [dev07] to bring all the features on

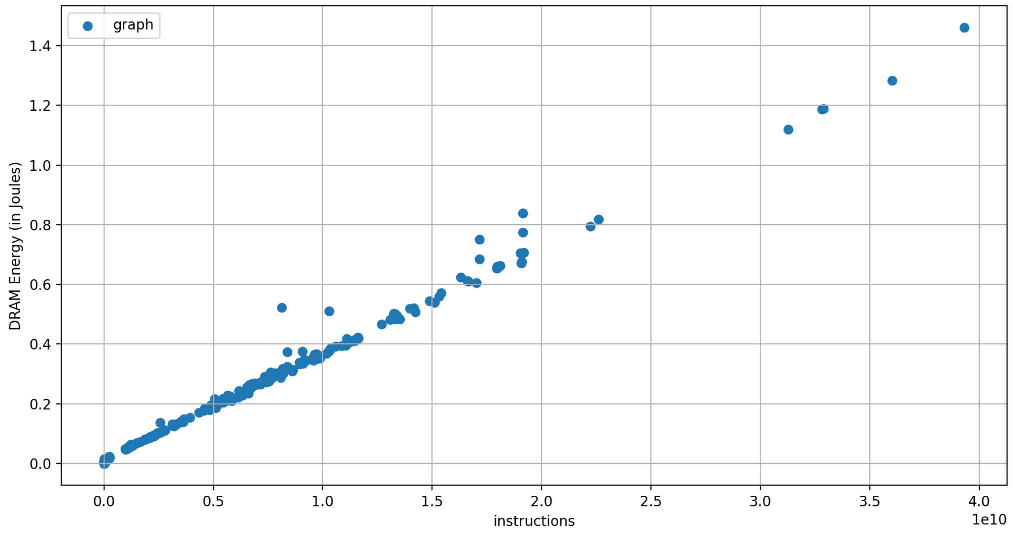

Figure 6.2: Graph of instructions and DRAM Energy (in Joules)

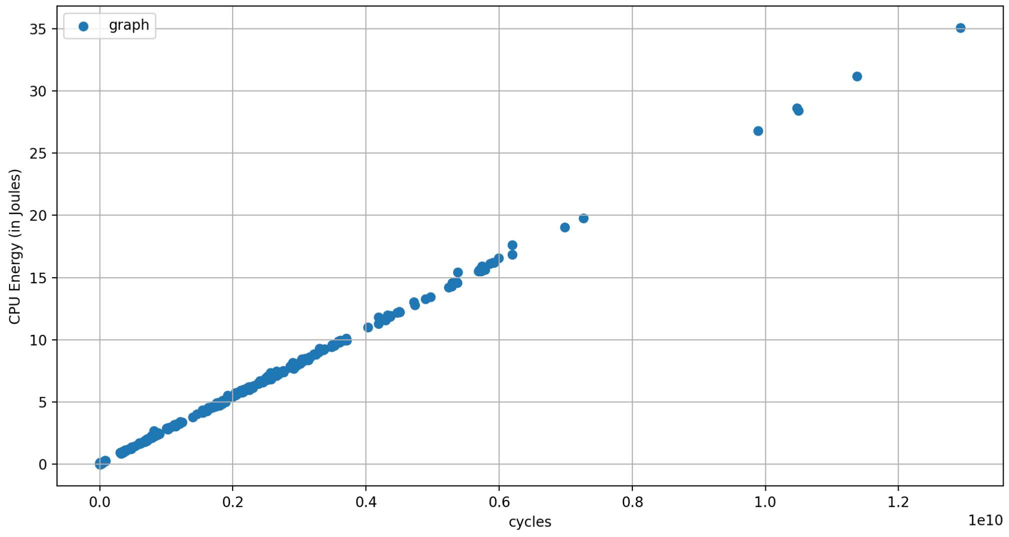

Figure 6.3: Graph of cycles and CPU Energy (in Joules)

a uniform scale [0,1]. Consequently, the dataset was divided into training (55%), validation (25%), and testing (20%). A linear regression algorithm was employed to generate the energy models using total instructions and total cycles as independent variables and CPU and DRAM energy as dependent variables. The model was fine-tuned using the validation dataset to achieve the highest accuracy and test data was utilised to test the model's accuracy.

The following equation represents the model used to predict the CPU energy consumption:

$$CPUEnergy = b_{1,0} + (b_{1,1} \times tot_{\text{ins}}) + (b_{1,2} \times tot_{\text{cyc}}) \tag{6.1}$$

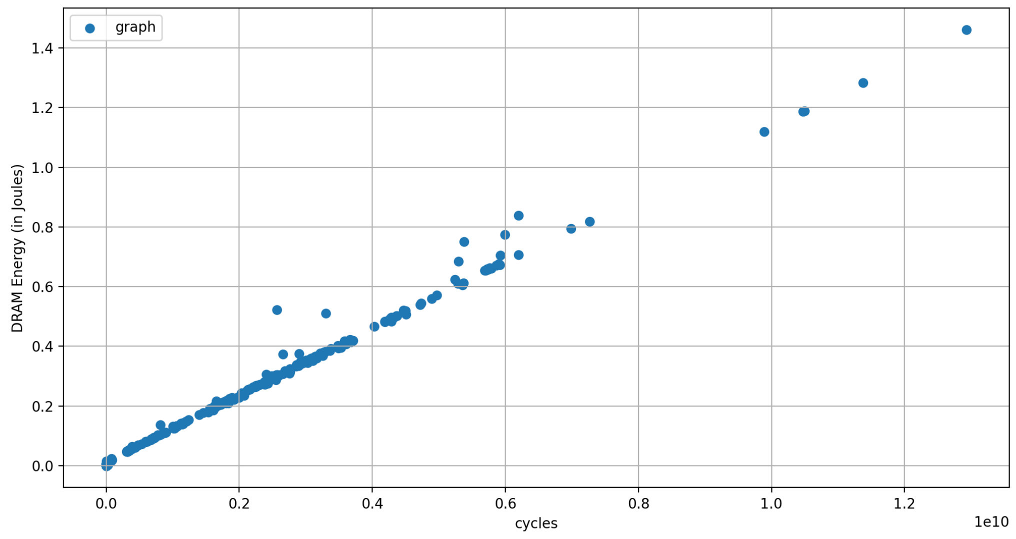

Figure 6.4: Graph of cycles and DRAM Energy (in Joules)

where:

- tot_{ins} denotes the total number of instructions executed.
- tot_{cyc} denotes the total number of CPU cycles executed.
- $b_{1,0}$, $b_{1,1}$, and $b_{1,2}$ represent the regression coefficients of the CPU model.

The following equation represents the model used to predict the DRAM energy consumption:

$$DRAMEnergy = b_{2,0} + (b_{2,1} \times tot_{\text{ins}}) + (b_{2,2} \times tot_{\text{cyc}}) \tag{6.2}$$

where:

- tot_{ins} denotes the total number of instructions executed.
- tot_{cyc} denotes the total number of CPU cycles executed.
- $b_{2,0}$, $b_{2,1}$, and $b_{2,2}$ represent the regression coefficients of the DRAM model.

RQ2 can be effectively answered based on the above formulation. We can estimate the precise value of energy consumption by running any AI application based on the total number of instructions and total number of CPU cycles.

6.4 Results

The energy models devised in the previous section can be used to estimate the energy consumption of CPU and DRAM. Table 6.3 and Table 6.4 show the values of all the coefficients of the CPU and DRAM energy models respectively.

Table 6.1: ρ of PMCs and CPU Energy in Descending Order

PMC Type	ρ
PAPI_TOT_CYC	0.922
PAPI_TOT_INS	0.861
PAPI_SP_OPS	0.555
PAPI_VEC_SP	0.555
PAPI_STL_CCY	0.514
PAPI_BR_CN	0.510
PAPI_BR_PRC	0.510
PAPI_BR_NTK	0.510
PAPI_FUL_ICY	0.508
PAPI_MEM_WCY	0.504
PAPI_REF_CYC	0.504
PAPI_SR_INS	0.493
PAPI_STL_ICY	0.492
PAPI_BR_INS	0.491
PAPI_LST_INS	0.489
PAPI_FUL_CCY	0.488
PAPI_LD_INS	0.488
PAPI_BR_TKN	0.483
PAPI_VEC_DP	0.477
PAPI_BR_MSP	0.410

By evaluating the performance of the developed models on test data, the results demonstrated in Table 6.5 and Table 6.6 were observed. The achieved level of accuracy is notably superior compared to previous research in this domain, which utilised more features in the regression model but achieved a lower accuracy rate of around 4%. The reduction in complexity and the improvement in predictive accuracy to 0.158% and 0.273% for CPU and DRAM respectively underscores the idea of focusing on PMCs such as total instructions and total CPU cycles. Both the energy models perform exceptionally well with a CPU Mean Absolute Error (MAE) of 0.00060 and DRAM MAE of 0.00255. The R Squared (R^2) scores of 0.9998 and 0.9925 for CPU and DRAM respectively also support the argument that these models are a good fit for the ML scenario.

The graph in Figure 6.6 shows the accuracy of our CPU energy estimation model based on the total number of instructions and the total number of cycles. The blue data points represent the predicted CPU energy (in Joules) and the red data points represent the test CPU energy (in Joules). It can be observed that most of the data points coincide, validating the accuracy of our model. A similar trend can be seen in Figure 6.7 for CPU energy based on the total number of cycles. This model achieves an exceptional error rate of only 0.158%.

Table 6.2: ρ of PMCs and DRAM Energy in Descending Order

PMC Type	ρ
PAPI_TOT_CYC	0.869
PAPI_TOT_INS	0.751
PAPI_SP_OPS	0.526
PAPI_VEC_SP	0.555
PAPI_STL_CCY	0.483
PAPI_BR_NTK	0.483
PAPI_BR_CN	0.483
PAPI_BR_PRC	0.483
PAPI_FUL_ICY	0.480
PAPI_REF_CYC	0.477
PAPI_MEM_WCY	0.465
PAPI_SR_INS	0.464
PAPI_STL_ICY	0.462
PAPI_BR_INS	0.461
PAPI_LST_INS	0.457
PAPI_FUL_CCY	0.457
PAPI_VEC_DP	0.456
PAPI_LD_INS	0.456
PAPI_BR_TKN	0.456
PAPI_BR_UCN	0.428

Table 6.3: Values of CPU Regression Model Coefficients

Coefficient	**Value**
$b_{1,0}$	0.00042871
$b_{1,1}$	0.84030965
$b_{1,2}$	0.16071597

Similarly, the graph in Figure 6.8 shows the accuracy of our DRAM energy estimation model based on the total number of instructions and the total number of cycles. The blue data points represent the predicted DRAM energy (in Joules) and the red data points represent the test DRAM energy (in Joules). Here it can be observed that the model shows high accuracy with an error rate of 0.273%. A similar trend can be seen in Figure 6.9 for DRAM energy based on the total number of cycles.

The streamlined approach used in our methodology not only simplifies the energy prediction process but also enhances the reliability of the predictions, making it a valuable tool for those involved in the development and deployment of AI technologies. This energy model, based on a thorough analysis of hardware PMCs, sets a new standard in the field of energy management in AI applications, providing a replicable method for future research and practical implementations in the field of sustainable computing.

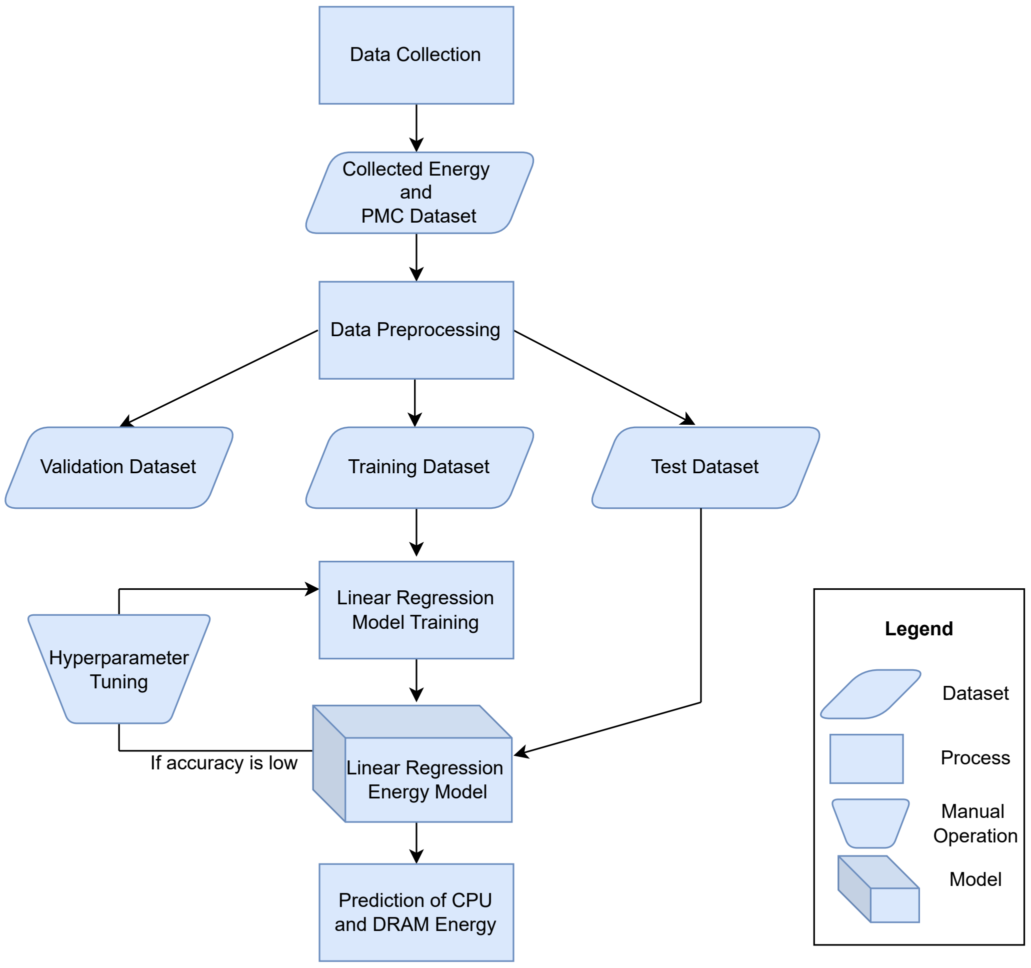

Figure 6.5: Flow diagram for the energy model creation process.

Table 6.4: Values of DRAM Regression Model Coefficients

Coefficient	Value
$b_{2,0}$	0.00153388
$b_{2,1}$	0.84543874
$b_{2,2}$	0.18049153

6.5 CO_2 Emission Estimation of AI Applications

This section aims to answer the third research question.

RQ3: How can precise CO_2 emissions of AI applications be measured?

Table 6.5: Metrics for Accuracy Evaluation of CPU Energy Model

Metrics	Value
CPU MAE	0.00060
CPU R^2 Score	0.9998
CPU Error%	0.158%

Table 6.6: Metrics for Accuracy Evaluation of DRAM Energy Model

Metrics	Value
DRAM MAE	0.00255
DRAM R^2 Score	0.9925
DRAM Error%	0.273%

The next step in our methodology is the assessment of CO_2 emissions resulting from the AI applications. It is important to recognise that there is a non-strict relationship between CO_2 emissions and energy usage. Numerous factors, such as geographic location, the kinds of fuels used to generate energy, and the degrees of economic and technical development, affect the differences in emissions from different nations [Cod20a].

6.5.1 Carbon Intensity

We use the carbon intensity in our estimates to accurately represent the geographical variations in emissions. This coefficient shows the weight of CO_2 emissions, expressed in kg, for each Kilowatt-hour (KWh) of electricity produced. The carbon intensity is determined by the energy mix of a region that includes fossil fuels and renewable energy sources like solar power, hydroelectricity, biomass and more [Cod20a].

Equation 6.3, provides the formula to calculate the cumulative carbon intensity based on the weighted average of all the energy sources.

$$CarbonIntensity = \sum (f_i \times C_i) \tag{6.3}$$

where:

- i represents individual energy sources such as solar power, hydroelectricity, and biomass.
- f_i denotes the fraction of each energy source used within a specific region in %.
- C_i denotes the carbon intensity (kg/megawatt-hour (MWh)) for each energy source.

Hence, the overall carbon intensity for a region is a weighted sum of the intensities for each energy source, accounting for their respective proportions within the regional energy profile. In areas with a high proportion of renewable energy sources, the overall carbon intensity is relatively lower, indicating a cleaner energy system. In contrast, a higher carbon intensity is associated with a higher level of emissions from electricity generation [Cod20a].

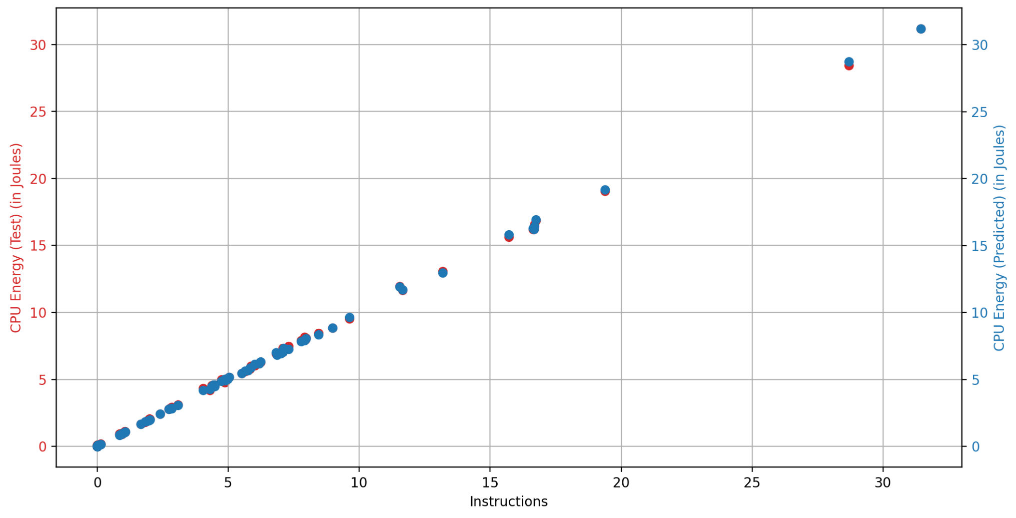

Figure 6.6: Predicted and Test CPU Energy (in Joules) for different number of instructions

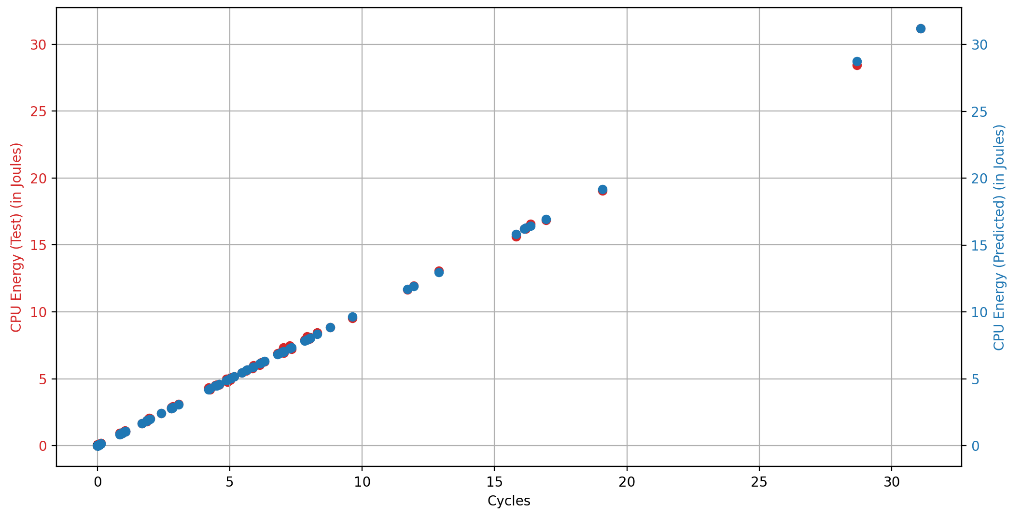

Figure 6.7: Predicted and Test CPU Energy (in Joules) for different number of cycles

6.5.2 Precise CO2 Emission Estimation of AI Applications

In cases where the estimation of CO_2 emissions is not possible via energy mix, the carbon intensity of geographical region is considered [Cod20a]. As of April 29, 2024, the coefficient for carbon intensity for the German region is 385.389, according to Codecarbon [Cod20b]. Since our experiments are conducted on a locally running Intel machine in Germany, the carbon intensity of Germany will give the precise value of CO_2 emissions from any AI application. The aggregate carbon footprint produced throughout the training of AI applications is determined by multiplying the cumulative energy used by the CPU and DRAM with the carbon intensity value of Germany.

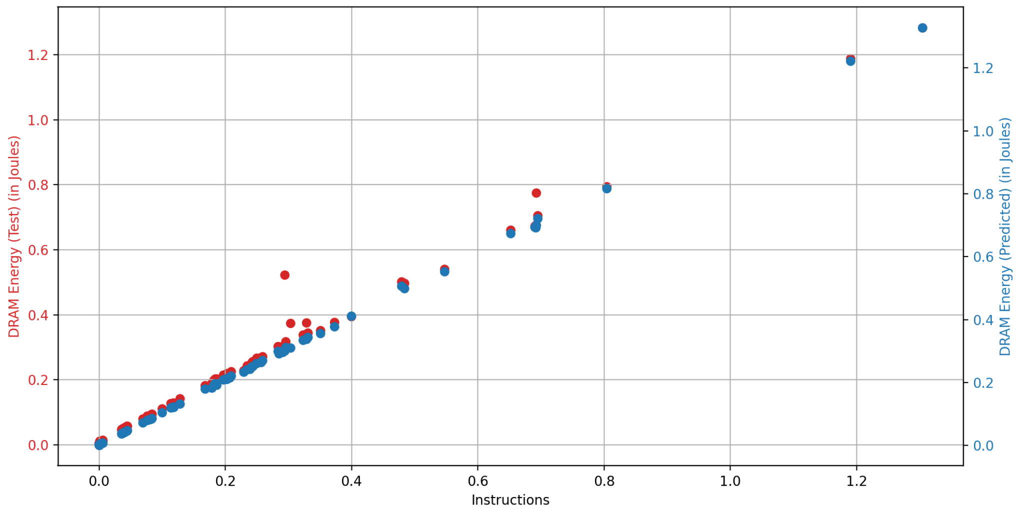

Figure 6.8: Predicted and Test DRAM Energy (in Joules) for different number of instructions

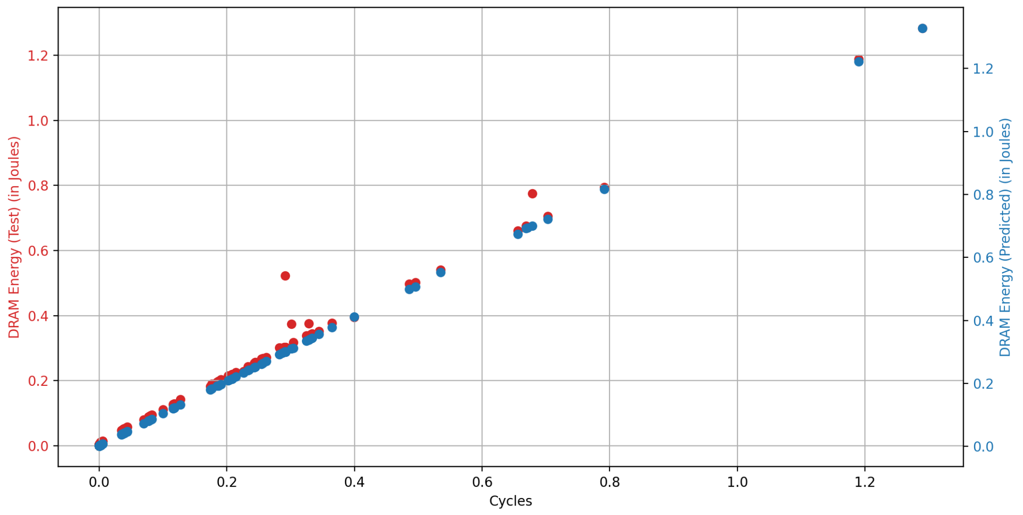

Figure 6.9: Predicted and Test DRAM Energy (in Joules) for different number of cycles

Equation 6.4 depicts the way to precisely calculate the CO_2 emission of a computing unit by taking the sum of energy consumption of CPU and DRAM and multiplying it by the carbon intensity of the region in which the computation is taking place.

$$TotalCO_2Emission = \sum(Energy_i) \times CarbonIntensity \tag{6.4}$$

where:

- i represents CPU and DRAM
- $Energy_i$ denotes the energy consumed by the CPU and DRAM.

- $CarbonIntensity$ denotes the carbon intensity of the corresponding region.

Based on Equation 6.4, the estimated values for CPU and DRAM energy are added together and multiplied by Germany's unique carbon intensity. The proportion of error in our calculations is then calculated by comparing this predicted emission value with the actual CO_2 emissions. Similar to energy percentage error, the CO_2 emission error rate by CPU is 0.158% and CO_2 emission error rate by DRAM is 0.273%. This comparison aids in evaluating how well our model predicts the environmental effects of executing particular AI applications.

Figure 6.10 and Figure 6.11 show the accuracy of our model in terms of the true and predicted CO_2 emissions. The graph shows a linear trend between the true and predicted values, indicating that the model can perfectly fit the data points with high accuracy.

The above formulation, answers RQ3 and precisely measures the CO_2 emissions of AI applications.

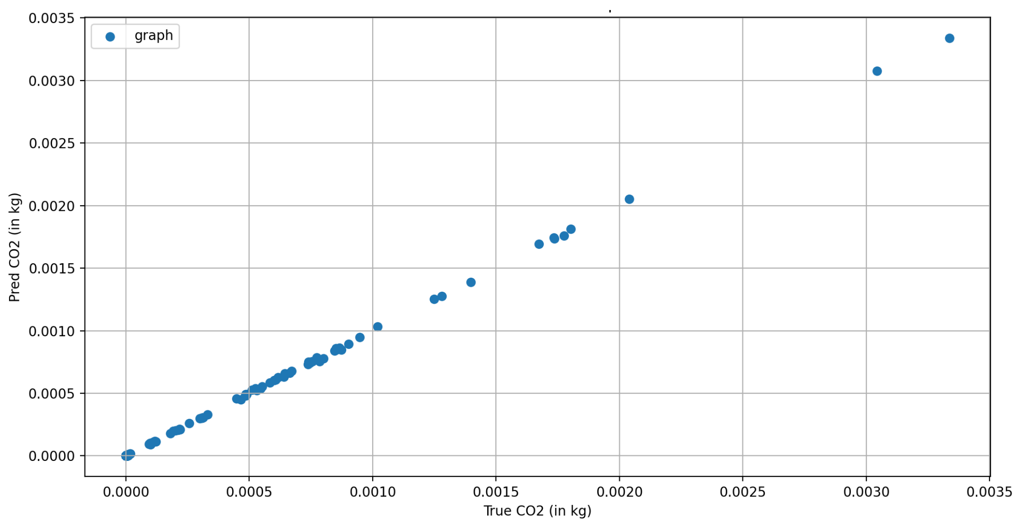

Figure 6.10: Predicted and True CO_2 emission (in kg) for CPU

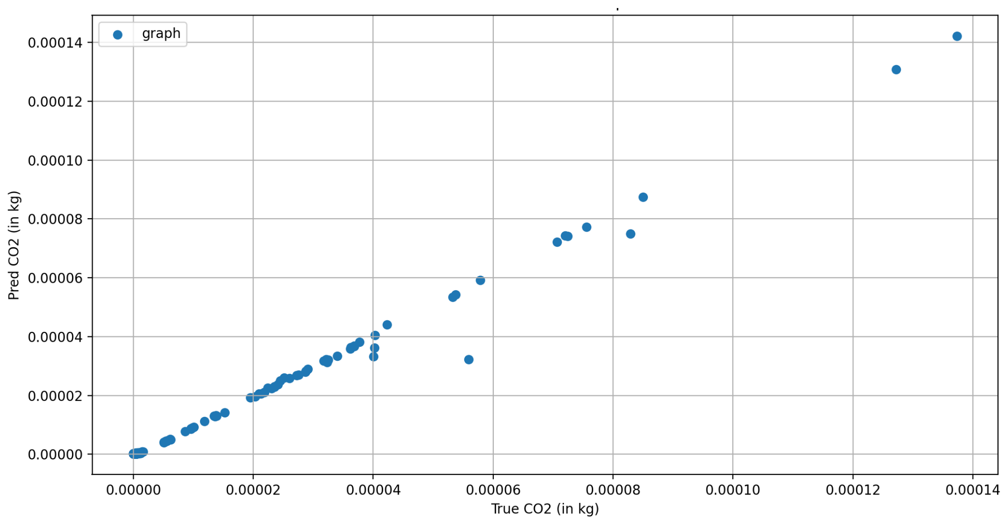

Figure 6.11: Predicted and True CO_2 emission (in kg) for DRAM

7 Hyperparameters and Architecture Optimisation for Energy Conservation of AI Applications

This section aims to answer the fourth research question.

RQ4: How do hyperparameters and architecture of AI applications impact PMCs and energy consumption?

While optimising the efficiency of AI applications, it is crucial to also consider the software optimisation in addition to the hardware aspects to minimise energy consumption. Effective hyperparameter tuning and architectural design play a significant role in reducing the computational burden of ML models, which in turn decreases energy use [CM15; WMV20].

This research aims to thoroughly evaluate the impact of various hyperparameters and architectures on the model's computational demands, specifically focusing on instructions, cycle counts, CPU and DRAM energy consumption. The analysis aims to understand the trade-off between these metrics and the accuracy of the model. It presents a comparative analysis of these metrics to explain how changes in hyperparameters and model architecture can impact the energy usage and accuracy of the model.

By refining hyperparameters, we can significantly lower the carbon footprint of AI training processes. This aligns with the development of more sustainable AI practices, where energy efficiency is as critical as computational performance. To experiment with this ideology, a classic NN model with sequential dense layers has been used for classification using the sklearn breast cancer dataset. The dataset has a sample size of 569 with two classes. All experiments have been performed on CPU and the energy consumption of CPU and DRAM has been recorded.

The various hyperparameters and model architectural designs that have been studied are explained below:

7.1 Epoch

In this section, we analyse the behavioural change in the instruction count, cycle count, CPU energy, DRAM energy and accuracy of the NN model as the number of training epoch varies from 30 to 50. The NN model consists of an input layer, two hidden layers and an output layer. The Rectified Linear Unit (ReLu) activation function has been used in the hidden layer, followed by the sigmoid activation function in the output layer for binary classification. The model utilises adam optimiser with a LR of 0.001 and binary cross entropy loss function. The aim is to understand how changes in epochs affect various aspects like instruction count, cycle count, model accuracy, CPU energy consumption, and DRAM energy consumption of NN training.

Instructions vs. Epochs: The graph of instructions vs. epochs in Figure 7.1 indicates that as the number of training epochs increases, the number of executed instructions also increases, suggesting that the model does more computations during this time. This trend implies that the model performs additional computations as it is exposed to more data throughout each growing training epoch. Essentially, this implies that the model is using data multiple times to learn and modify its parameters according to the input data.

Cycles vs. Epochs: The cycles vs. epochs graph in Figure 7.1 exhibits a consistent upward trend with occasional dips. The number of CPU cycles increases as the model goes through each training epoch. These cycles are the fundamental units of time that a CPU spends processing data. This trend in CPU cycles resembles the instruction count trend. This suggests a direct relationship between the intensity of computations required and the training process. As the model learns and adapts from more data during the epoch, it needs more processing power and a higher number of CPU cycles.

CPU Energy vs. Epochs: In the CPU energy vs. epochs graph in Figure 7.2 it can be observed that, after the 40th epoch ends, CPU energy usage starts to rise noticeably. The instructions and CPU cycles can be connected to the observed increase in energy consumption, which indicates a higher degree of computational activity in later epochs. As the training progresses, the complexity and the volume of the calculations required by the model training also increases, which results in the CPU performing more cycles and executing more instructions. This leads to a rise in the CPU energy consumption.

DRAM Energy vs. Epochs: In the DRAM energy vs. epochs graph in Figure 7.2, the DRAM energy usage also shows an increasing trend with an increase in the number of epochs. This indicates that memory access requirements increase when the number of epochs is increased during training time. The memory access requirement rises with each increasing epoch because the model is handling more data and operations as training progresses. In every training epoch, the model passes through the whole dataset it is being trained on and modifies the weights to increase accuracy. In this process, data is retrieved multiple times from the DRAM resulting in more memory usage.

Accuracy vs. Epochs: The accuracy vs. epochs graph in Figure 7.3 displays an accuracy range of 94% to 99% for epochs ranging from 30 to 50. The accuracy of the model plateaus at 99.12% after the 44th epoch. This suggests that there is no noticeable rise in accuracy as the number of epochs increases and the model is either overfitting or has plateaued. The lack of improvement in accuracy despite additional epochs is a signal to review the training process. Measures like adjusting the model's parameters, considering different models, dataset quality checks, or introducing techniques like early stopping to halt the training process once the model's performance ceases to improve significantly should be employed. The development of efficient, effective, and generalised ML models requires an understanding of when a model plateaus or starts to overfit. This understanding facilitates the optimisation of training epochs and resource-saving.

Conclusion: The data gathered from these five graphs suggests that there is a precise correlation between the number of epochs, resource consumption and accuracy of the NN model. As training advances, the upward trajectory of both instructions and CPU cycles signifies a rise in computational requirements, thereby demanding increased energy consumption from both the CPU and DRAM. Therefore, an evaluation of the model epoch count is needed to identify a trade-off between accuracy and energy consumption and the best epoch needs to be identified by ML developers that do not consume large energy.

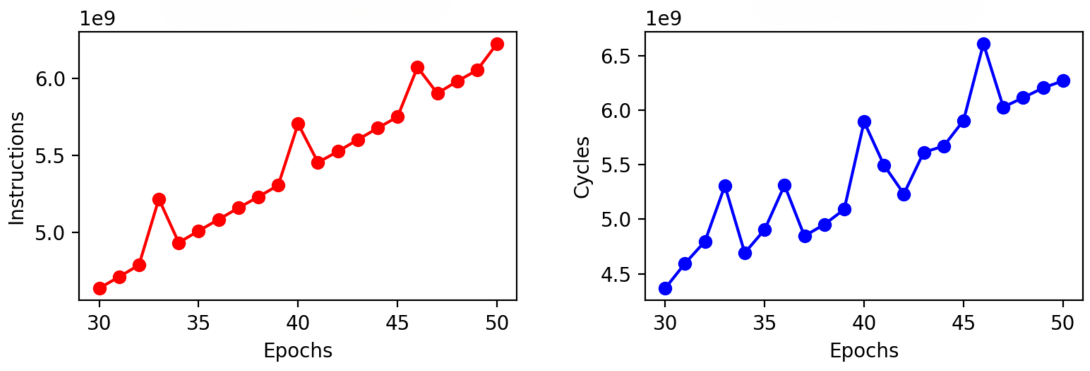

Figure 7.1: Graph of PMCs for different number of epochs.

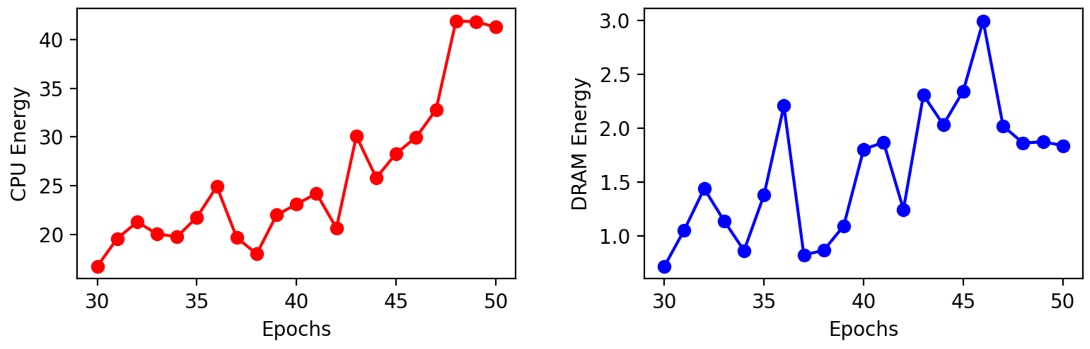

Figure 7.2: Graph of CPU and DRAM energy in Joules for different number of epochs.

7.2 Activation Function

In this section, we analyse the variation in the instruction count, cycle count, CPU energy, DRAM energy and accuracy of the NN model for different activation functions namely - tanh, softmax, sigmoid, and ReLu. Similar to the previous experiment, this NN model also consists of an input layer, two hidden layers and an output layer. The activation functions listed above have been used in

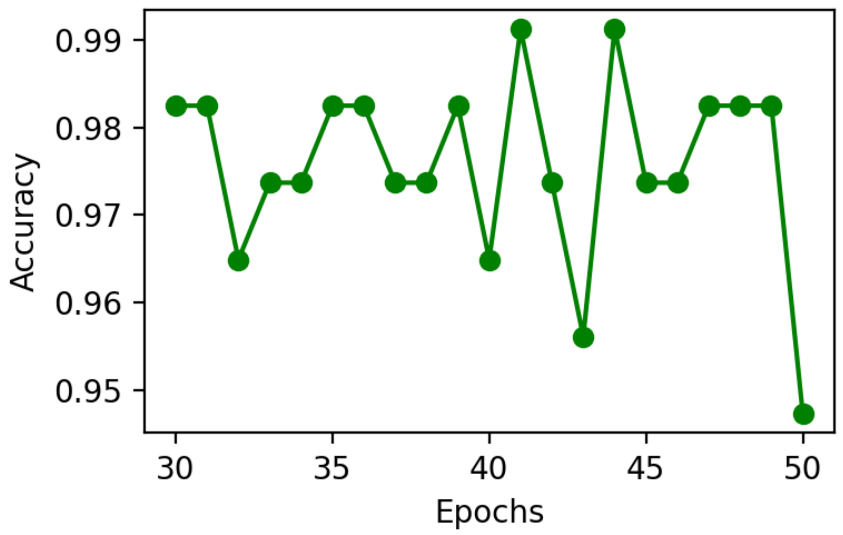

Figure 7.3: Graph of accuracy for different number of epochs.

the hidden layers followed by the sigmoid function in the output layer for binary classification. The model uses adam optimiser with a LR of 0.001 and a binary cross-entropy loss function. The aim is to understand how the different activation functions affect the instruction count, cycle count, model accuracy, CPU energy consumption, and DRAM energy consumption of the NN training.

Instructions vs. Activation Functions: According to instructions vs. activation functions graph in Figure 7.4, the tanh function starts with the highest number of instructions, followed by sigmoid and softmax and ends with ReLu, which shows the least number of instructions. This indicates that tanh requires more complex computations due to its complex mathematical operation, and ReLu function's basic computation requires fewer instructions.

Due to its mathematical complexity, the tanh function leads this list. Tanh is a non-linear function that transitions between -1 and 1. Extensive calculations are needed to map inputs to outputs making it computationally costly. Positive and negative values require multiple processing steps due to the function's S-shaped curve. In contrast, the ReLu function uses a simple threshold technique to map all negative values to zero and maintain positive values as they are. Due to its simplicity, ReLu requires fewer instructions than tanh. Like tanh, the sigmoid function outputs values between 0 and 1. Since it is positioned around the origin it is a little less computationally intensive than tanh. Softmax is generally employed in the final layer of a classification network to represent class probabilities using exponents and normalisation operations, placing it between sigmoid and ReLu in terms of the total number of instructions executed.

Cycles vs. Activation Functions: The trend seen in instruction count is also reflected in the CPU cycles count. The cycles vs. activation functions graph in Figure 7.4 shows that the tanh function consumes the most CPU cycles among the activation functions compared. Tanh function is followed by softmax, sigmoid and ReLu activation functions. This suggests as these functions perform more calculations per input value, more CPU cycles are needed to compute the outcomes.

ReLu's exceptional results in this area prove its computational efficiency. The ReLu function outputs the input directly if it is positive and zero otherwise. This simplicity allows ReLu function to process inputs in fewer CPU cycles than tanh, softmax and sigmoid functions.

CPU Energy vs. Activation Functions: The CPU energy vs. activation functions graph in Figure 7.5 suggests that tanh consumes the most CPU energy. This is consistent with its high instruction and cycle counts, indicating its correlation with computation cost. In terms of CPU energy consumption, ReLu function performs the best. The level of complexity of the operations in activation functions increases the computation cost and the CPU energy. It can be inferred that energy consumption can be reduced by utilising less complex activation functions.

DRAM Energy vs. Activation Functions: In the DRAM energy vs. activation functions graph in Figure 7.5, we observe that the DRAM energy consumption decreases as we move from more complex to simpler activation functions in NN. Starting with the tanh function followed by softmax, sigmoid, and ReLu functions, it can be concluded that activation functions with higher computational complexity consume more energy from memory operations.

The tanh function requires more memory resources due to its mathematical complexity and the smooth, continuous shape of its output curve. This is because tanh needs to calculate more complex gradients leading to more memory demands. These operations are energy-intensive, as they require the frequent reading and writing of values to and from the memory, increasing the energy used by DRAM.

Similar to tanh, the softmax function also requires a high level of computation and memory access due to its exponentiation and normalisation processes. Sigmoid function maps input values to a bounded range between 0 and 1. It involves calculations that are slightly less intense than softmax. However, they still require significant memory interaction. At the end of the spectrum is the ReLu function that outputs the input if it is positive and zero otherwise, involving minimal computation and memory access compared to tanh, sigmoid, and softmax.

Accuracy vs. Activation Functions: In the accuracy vs. activation functions graph in Figure 7.6, there is no specific trend observed. However, ReLu's position on this graph indicates that, despite being computationally advantageous, its simplicity hinders effectively modelling the data. Considering all the activation functions, the sigmoid function can be a good alternative for classification as it has fewer computational demands in terms of instructions and cycles executed and a higher accuracy.

Conclusion The insights gathered from these graphs underscore a complex interplay between the choice of activation function, computational efficiency, and model performance. Tanh, despite its computational expense, offers the best accuracy at the cost of energy efficiency. ReLu, while energetically favourable, might not achieve the highest accuracy, thus compromising the model performance. The softmax and sigmoid functions present a balanced compromise. Therefore, the selection of an activation function within NN models must be conducted with a strategic understanding of these trade-offs, particularly when balancing the computational cost against the performance objectives of the model. This decision is important in the context of energy-efficient AI system design, especially in scenarios with limited energy resources or where energy consumption has substantial economic or environmental implications.

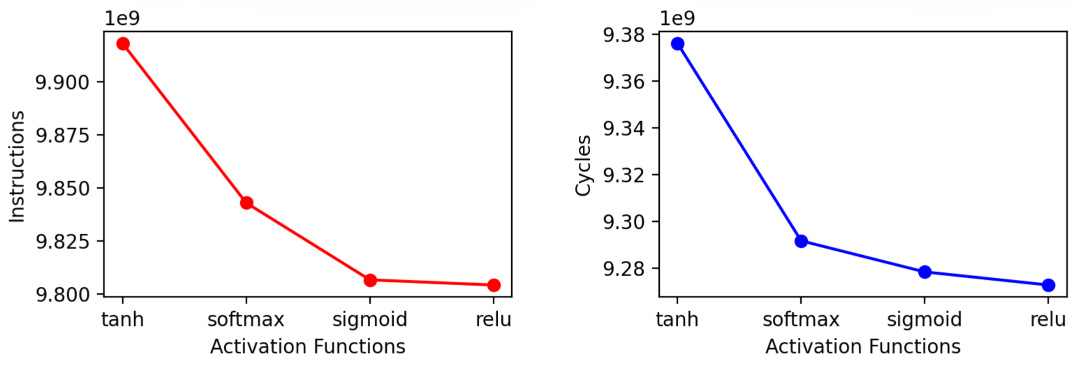

Figure 7.4: Graph of PMCs for different activation functions.

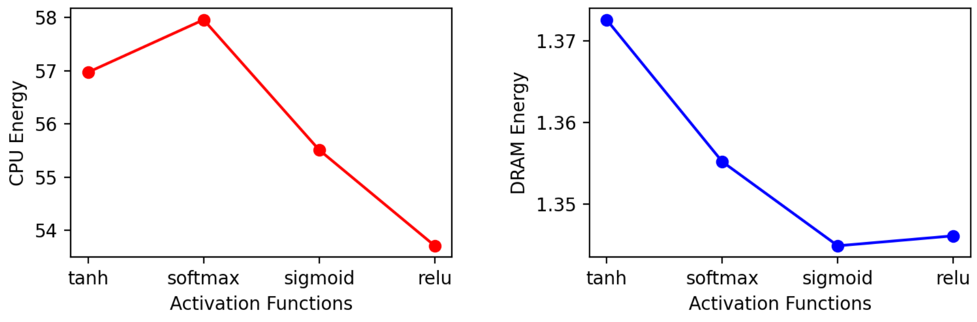

Figure 7.5: Graph of CPU and DRAM energy in Joules for different activation functions.

7.3 Batch Size

In this section, we analyse the variation in the instruction count, cycle count, CPU energy, DRAM energy and accuracy of the NN model for different batch sizes from 20 to 40. Similar to the previous experiment, this NN model also consists of an input layer, two hidden layers and an output layer. The model uses an adam optimiser with a LR of 0.001 and a binary cross-entropy loss function. The aim is to understand how the different batch sizes affect instruction count, cycle count, model accuracy, CPU energy consumption, and DRAM energy consumption of NN training.

Instructions vs. Batch Size: In Figure 7.7 an overall decreasing trend is seen in instructions vs. batch size graph, as the batch size increases. Due to parallel processing capabilities, processing instructions in bigger batches results in more efficient instruction processing. Parallel processing is a method used in computing where multiple processors handle different parts of an overall task

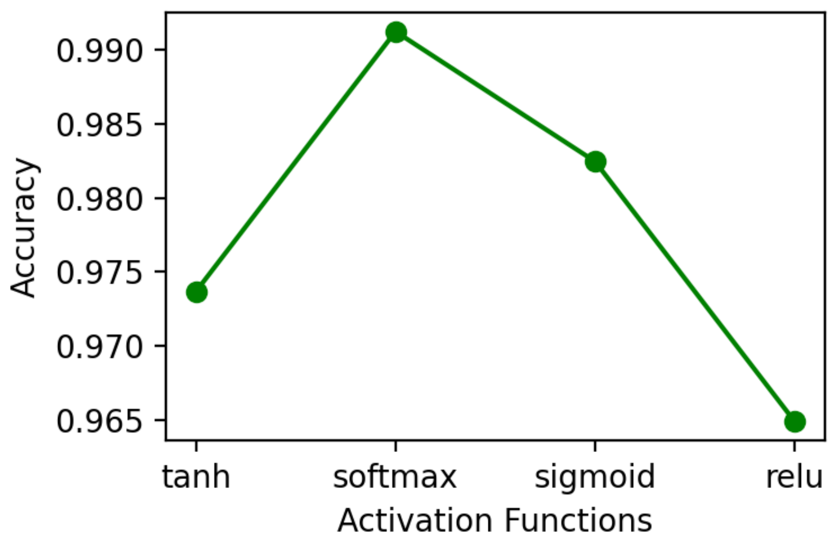

Figure 7.6: Graph of accuracy for different activation functions.

simultaneously. When data is processed in larger batches, more data points are processed at the same time. The capability to handle multiple data points concurrently allows the system to optimise its instruction execution.

Cycles vs. Batch Size: In the cycles vs. batch size graph in Figure 7.7, a similar downward trend is also observed. Larger batches use CPU cycles more efficiently because the cost of loading and processing data is spread over a larger number of examples. When data is processed in larger batches, the initial overhead of loading data, setting up processing tasks, and distributing these tasks across CPU cycles is divided across larger data sizes as compared to several smaller batches. As a result, the CPU maintains a steady state of processing for a longer time, which is more efficient than repeatedly stopping and starting smaller batches.

CPU Energy vs. Batch Size: The CPU energy vs. batch size graph in Figure 7.8 exhibits a steep decrease with increasing batch size. This suggests that when batch size increases, more data is processed in a single batch and increases the parallel computing capabilities. Therefore, data processed in larger batches requires less CPU energy.

DRAM Energy vs. Batch Size: The DRAM energy vs. batch size graph in Figure 7.8, does not show much variation based on batch sizes. This can be the result of the interaction between batch size-dependent caching effects, Memory Access Patterns (MAC), and data loading durations. The stability in DRAM energy usage across different batch sizes suggests the benefits of efficient caching and optimised MAC in ML model training.

Accuracy vs. Batch Size: The accuracy vs. batch size graph in Figure 7.9 does not represent a specific trend. It suggests that there is a nonlinear relationship between batch size and model accuracy. This can be due to the complexity and size of the dataset. The absence of a straightforward trend in model accuracy as batch sizes vary implies that different batch sizes can affect the learning outcome in various ways. Larger batch sizes offer advantages like faster computation and reduced training times due to more efficient processing. However, they have disadvantages such as inaccurate gradient estimation. On the other hand, more frequent weight updates due to smaller batch sizes can result in a more thorough learning process that can capture minute patterns in the data. However, this also results in longer training times and increased computational demands, as the model needs to process many more updates than it would with larger batches. Therefore, a trade-off between accuracy and batch size should be decided based on the energy requirements.

Conclusion: Based on a thorough analysis of these graphs, it appears that larger batches can not always result in better computational efficiency. However, they use fewer instructions and CPU cycles, and lower CPU and DRAM energy. No one batch size works for all cases, as indicated by the accuracy and energy consumption graph. Instead, the batch size should be selected based on the particular model and training environment. This customisation involves finding a trade-off between energy consumption, computational efficiency, and model performance. Because it affects both the practical deployment of NN models and the environmental impact of training procedures, choosing an appropriate batch size is thus an important stage in the design and training of NN models.

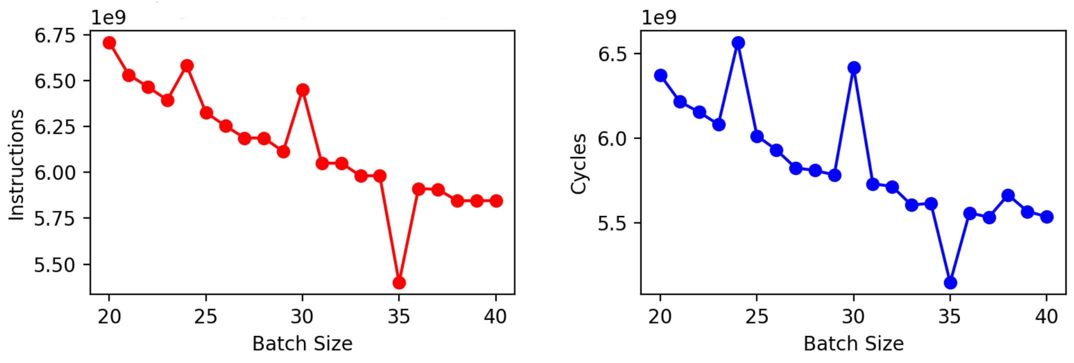

Figure 7.7: Graph of PMCs for different batch sizes.

7.4 Number of Layers

In this section, we analyse the variation in the instruction count, cycle count, CPU energy, DRAM energy and accuracy of the NN model with the number of layers ranging from 1 to 20. This NN model has the same input layer, two hidden layers, and an output layer as the last experiment. The hidden layers utilise the ReLu activation function, and the output layer uses the sigmoid activation function. A binary cross-entropy loss function and an LR of 0.001 are employed with

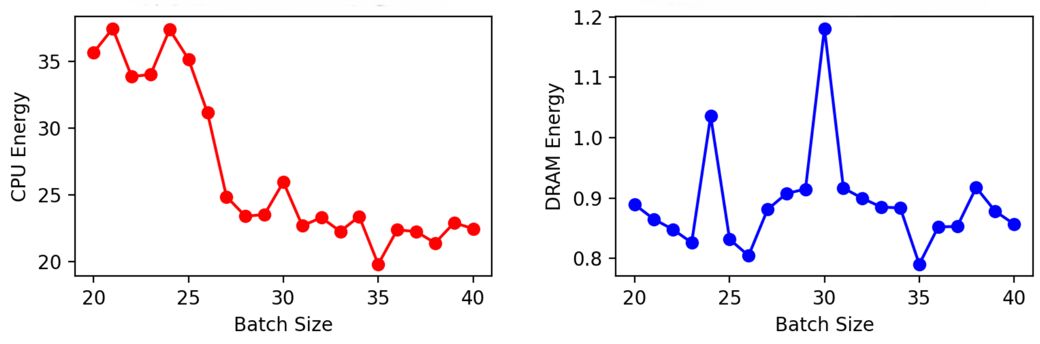

Figure 7.8: Graph of CPU and DRAM energy in Joules for different batch sizes.

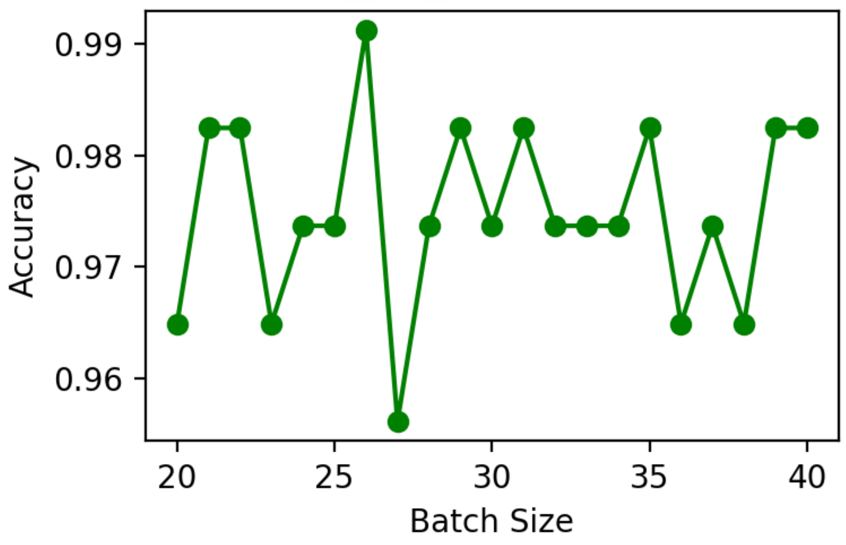

Figure 7.9: Graph of accuracy for different batch sizes.

adam optimiser. The objective is to understand how the number of layers affects the number of instructions, cycles, model accuracy, and energy consumption of the CPU and DRAM during NN training.

Instructions vs. Layers: The instructions vs. layers graph in Figure 7.10 shows a linear relationship between the total number of instructions executed and the number of layers. This is expected since a NN with additional layers will do more operations in both the forward and backward training passes. Given the structure and operation of NN, the linear rise in instruction count with each layer is natural. A set of neurons makes up each layer of a NN, and each neuron in a layer carries out several calculations. These computations have two steps: first, the weighted inputs from neurons in the preceding layer are aggregated, and then these weighted inputs are subjected to a non-linear transformation or activation function. This proves that as the depth of the network increases, the computational burden in terms of instructions also increases.

Cycles vs. Layers: The cycles vs. layers graph in Figure 7.10 shows a generally rising trend with considerable variability. A complex series of computational steps are involved in both the forward pass, where the NN generates predictions, and the backward pass, where the network modifies its weights according to the gradient of the loss function in a NN. The increase indicates that deeper networks need more processing cycles because they have more parameters to update during training and are therefore more complex.

CPU Energy vs. Layers: The CPU energy vs. layers graph in Figure 7.11 shows that the CPU energy increases with the number of layers increasing almost exponentially. The expense of calculating more instructions and CPU cycles for each further layer is reflected in the rise in energy consumption. Each layer of a NN processes inputs and produces output for further layers. Non-linear processes like activation functions and linear transformations like weighted sums are involved in these computations. Additionally, each layer's parameters are changed during the training phase in response to the loss function's feedback. Backpropagation to change the weights based on the gradients of the loss function. The path that these gradients must take increases with the number of layers added and the complexity of calculations also increases. To complete the required computations, more CPU cycles are required which raises the CPU energy usage.

DRAM Energy vs. Layers: The DRAM energy vs. layers graph shown in Figure 7.11 similarly indicates a rise in DRAM energy with the number of layers. This increase is supported by the fact that more data needs to be stored and retrieved during training. The computational requirements of NN grow with the number of layers as more data needs to be processed and stored. Each neuron in the layers has a weight and bias associated with it. Furthermore, intermediate data from each layer needs to be temporarily stored during the forward pass of training and in the backward pass, where the network modifies its parameters based on the error gradient. Inputs, outputs, weights, biases, and gradients are all essential for the model training and increase along with its depth. The DRAM is used to read, write, and access the data which increases energy consumption.

Accuracy vs. Layers: The accuracy vs. layers graph in Figure 7.12 shows peaks and valleys but no discernible pattern, making it noticeably more unpredictable. This implies that while having more layers may improve the model's performance by enabling it to learn more complicated characteristics, it may also increase the risk of overfitting or other optimisation issues that could reduce accuracy. Therefore, a fair trade-off should be maintained between the number of layers and energy usage.

Conclusion: The examination of these graphs demonstrates that as a NN's layers are added, its computing requirements also rise, as evidenced by the rising CPU cycles and instruction counts. The energy requirements for CPUs and DRAM are also raised due to this increased complexity, which emphasises the significance of network architecture optimisation for energy efficiency, especially in bigger models.

Nevertheless, deeper NN do not always mean better accuracy. This suggests that there is an optimal point beyond which adding more layers would not improve the model's performance. This research highlights the need to carefully weigh the model's complexity which maintains a balance between the network's depth and energy expenses. These findings have important applications when using DNNs in energy-constrained settings. Choosing the ideal number of layers is a crucial choice that affects the model's functionality and environmental sustainability.

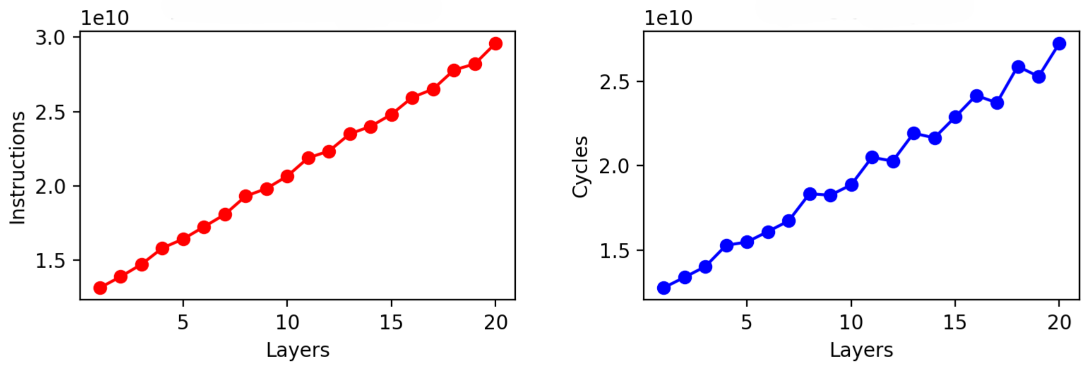

Figure 7.10: Graph of PMCs for different number of layers.

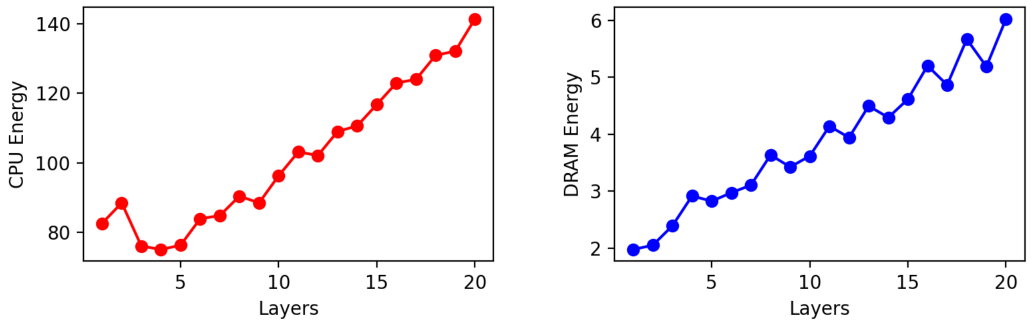

Figure 7.11: Graph of CPU and DRAM energy in Joules for different number of layers.

7.5 LR

In this section, we analyse the variation in the instruction count, cycle count, CPU energy, DRAM energy and accuracy of the NN model with different learning rates such as 0.001, 0.01, and 0.1. This NN model has the same input layer, two hidden layers, and an output layer as the last experiment.

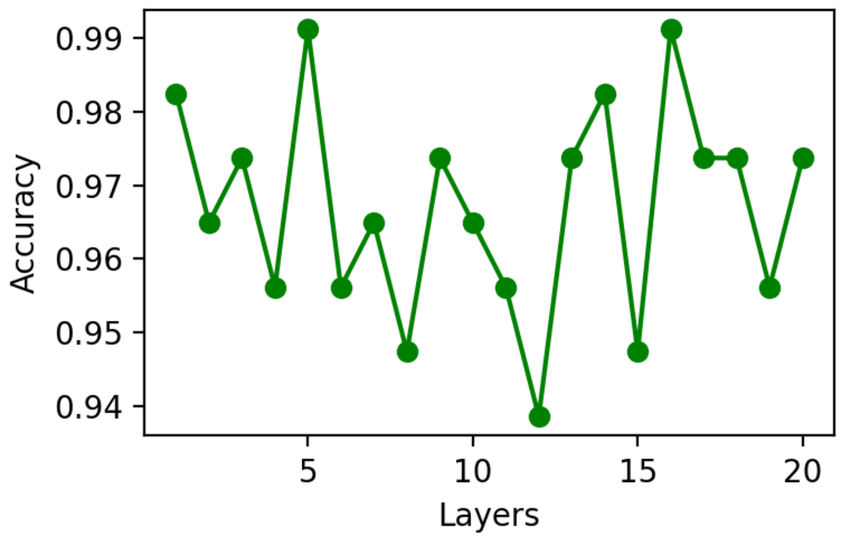

Figure 7.12: Graph of accuracy for different number of layers.

The hidden layers utilise the ReLu activation function, and the output layer uses the sigmoid activation function for binary classification. A binary cross-entropy loss function is employed with adam optimiser. The objective is to understand how different learning rates affect the number of instructions, cycles, model accuracy, and energy consumption of the CPU and DRAM during NN training.

Instructions vs. LR: In the instructions vs. LR graph in Figure 7.13, when the LR is raised from 0.001 to 0.01 there is a sharp drop in the number of instructions, and then it settles out at 0.1. The first decline implies that more instructions are needed for a lower LR because more iterations are required to reach convergence. The LR is a critical parameter that affects how quickly a network learns. The network makes very small updates to its weights at a lower LR, such as 0.001. This results in more number of iterations to converge thereby increasing the total instructions. When the LR is increased to 0.01 and 0.1, larger updates are made to its weights with each iteration. Therefore, convergence is reached faster, reducing the total number of iterations and consequently the total number of instructions. This relationship between the LR and the total number of instructions highlights the importance of tuning the LR for efficient training of NN.

Cycles vs. LR: In the cycles vs. LR graph in Figure 7.13, the CPU cycles show a decreasing trend moving from 0.001 to 0.1. This pattern suggests that the model needs fewer CPU cycles with a higher LR. CPU cycles are a measure of the processing time that the CPU requires for executing tasks. This involves operations such as computing the forward pass, backpropagation, and updating weights in NN. At a lower LR, like 0.001, the weight updates are very small. This slow rate of improvement requires more iterations to achieve convergence for the model. Consequently, more iterations require more CPU cycles, making the training process computationally intensive. As the LR is increased to 0.01 and 0.1, each iteration results in larger updates to the model's weights. These larger updates result in faster convergence, reducing the total number of iterations and CPU cycles.

CPU Energy vs. LR: In the CPU energy vs. LR graph in Figure 7.14, the CPU energy consumption decreases substantially as the LR is increased from 0.001 to 0.01. The most energy-efficient operation occurs at a LR of 0.1 due to fewer instructions and CPU cycles being executed. However, this leads to faster convergence of the model.

DRAM Energy vs. LR: The DRAM energy consumption in the DRAM energy vs. LR graph in Figure 7.14 shows an initial peak at a LR of 0.001 and later a decreasing trend. This is due to the frequent access of DRAM to store intermediate data such as weights, biases, and gradients. With lower LR, more gradients are calculated to achieve slower convergence. This resulted in high DRAM energy due to frequent access.

Accuracy vs. LR: The accuracy vs. LR graph in Figure 7.15 shows that the accuracy fluctuates between 0.964 to 0.973. The accuracy is highest at 0.01 LR and lowest at 0.001. This suggests that smaller updates to the model's weights help it learn more effectively, however, they can sometimes underfit the data. A very large LR can cause overshooting and miss the optimal performance point leading to overfitting. Therefore, the optimal LR should be used that fits the data well and achieves the highest accuracy.

Conclusion: These findings suggest that the connection between computing efficiency and LR, as well as model performance, is not straightforward. Higher instruction counts and CPU cycles are the outcome of a reduced LR. On the other hand, the corresponding rise in energy and computing demands highlights the need to choose a LR that is both resource-efficient and supportive of model accuracy.

The apparent optimal intermediate LR exemplifies a balance between computational expense and model accuracy. These insights can guide the implementation of NN models, especially in resource-constrained environments where both performance and energy efficiency are critical. The choice of LR, therefore, should not be made in isolation but as part of a holistic approach to model training that weighs the trade-offs between accuracy, convergence speed, and energy consumption.

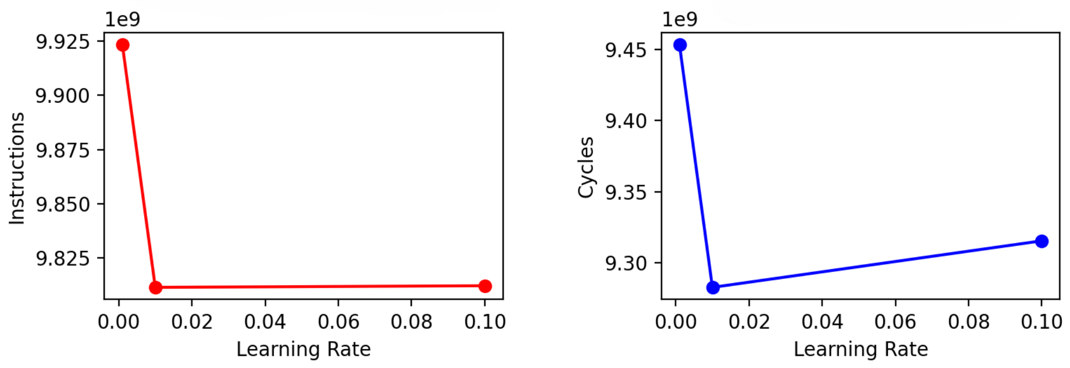

Figure 7.13: Graph of PMCs for different LRs.

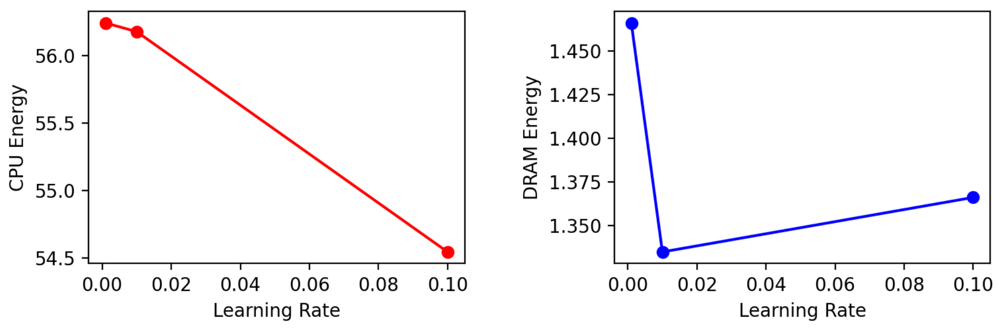

Figure 7.14: Graph of CPU and DRAM energy in Joules for different LRs.

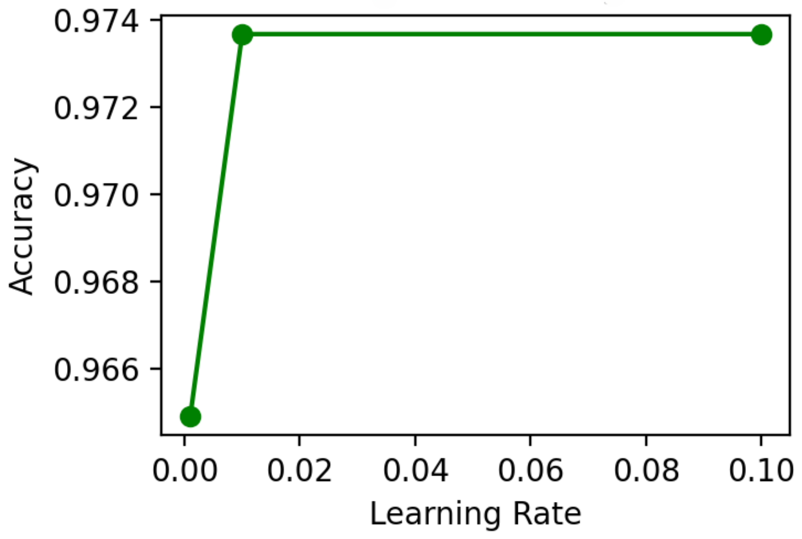

Figure 7.15: Graph of accuracy for different LRs.

The above experiments have enabled the understanding of the impact of hyperparameters and architecture optimisation on the change in PMCs and energy consumption of AI applications and answered RQ4. It can be concluded that the right configuration of hyperparameters is essential for achieving the highest accuracy along with minimal energy consumption.

8 Conclusion

When developing ML algorithms, energy consumption is an important factor to consider. Large AI model development and training account for a considerable amount of greenhouse gas emissions. Nevertheless, the majority of research focuses on improving the predictive performance of algorithms, even though some work is being done to minimise the computations required for ML tasks. Measuring the energy consumption of programs is one of the main challenges today. For NLP based ML models, reporting the time to retrain and the sensitivity to hyperparameters is necessary. Fair access to computational resources is necessary for academic researchers, and developing effective models and hardware should be their top priority. Training time and sensitivity to hyperparameters should be disclosed by the authors. Our ultimate goal is to make energy estimation modelling approaches available to the ML community, as we believe this will help improve energy efficiency in ML.

The thesis delves into the precise measurement of CO_2 emissions from AI applications and critically examines the complex relationship between energy consumption, AI model training, and CO_2 emissions. The idea that processor-specific PMCs and AI applications' energy consumption are directly correlated was supported by our research. A solid framework for estimating the energy consumption and, consequently, the CO_2 emissions of different AI models was provided by the use of a linear regression energy model that integrated PMCs like total instructions and total cycles.

The high accuracy of our energy models indicates their potential utility in real-world applications, particularly in optimising the energy efficiency of AI-driven systems. By applying these regression models, system administrators and developers can predict and manage the energy consumption of AI operations more effectively, leading to significant energy savings and reducing the environmental impact associated with running intensive AI tasks

Furthermore, the study of AI model architecture and hyperparameters marked a significant advancement toward sustainable AI practices. By carefully balancing accuracy and energy consumption, these models are designed to minimise the environmental impact of large-scale computational tasks that are part of AI research and applications. This thesis demonstrates that modifications to model architecture and hyperparameters can significantly reduce energy consumption without sacrificing accuracy. This provides a road map for developing AI systems that are both efficient and ecologically conscious.

In summary, this thesis highlights the dire need for energy-efficient AI practices while also laying the groundwork for future research in this important area. It is imperative that the AI research community start developing models with innovative strategies for reducing AI technologies' carbon footprint in mind.

The research done so far has established a foundational understanding of energy consumption and CO_2 emissions in AI applications, with a particular emphasis on CPU-based models. However, since GPUs have better processing power and efficiency for parallel computations, their use for machine learning tasks has increased. Therefore, a crucial next step is to broaden our research to create GPU-based energy estimation models.

1. **Combining GPU-Based Estimation Techniques:** Subsequent research endeavours should concentrate on utilising and optimising GPU-oriented approximation techniques customised for ML usage. Notably, there are particular opportunities and challenges for energy estimation when it comes to desktop GPUs, which are primarily used in ML research for model training. On desktop platforms, tools like NeuralPower and Synergy have started to meet these needs, but there is no comprehensive framework that covers mobile GPUs [CJSM17; RRL18].
2. **Examining GPU PMCs in Detail:** A thorough investigation of GPU-based PMCs is necessary. These PMCs are essential for deciphering patterns of energy consumption and provide detailed insights into how the device operates. But unlike CPUs, these counters' extraction and interpretation techniques are not as advanced. The main goals of research should be to determine which PMCs have the biggest effects on energy consumption and how to keep an eye on them while AI models are being trained.
3. **Creation of Frameworks and Tools:** GPU's PMCs can be measured with the help of programs like nvprof and Nvidia Nsight Systems. Future development should concentrate on combining these instruments with energy assessment tools such as Nvidia System Management Interface (SMI) to build a unified system that can monitor energy data and PMCs at the same time. This integration will help improve current energy models by enabling a more thorough examination of the relationships between GPU's performance and energy consumption.
4. **Development of Energy Models Using GPUs:** The next stage of the research should focus on creating predictive energy models, especially for GPUs, building on the integrated tools and data. These models would forecast and optimise AI model energy consumption by utilising the insights obtained from GPU's PMCs.
5. **Development of Policies and Standards:** There is an urgent need to create industry standards and policies that direct energy-efficient AI research in comprehension with technological advancements. These guidelines would guarantee that energy efficiency is considered from the start when developing AI models.
6. **Joint Research Projects:** Lastly, encouraging cooperative research projects that unite ML engineers with research scholars working on sustainability can enable the adoption of sustainable AI practices.

Future studies can greatly improve our knowledge and control over energy usage in AI applications by tackling these areas, which will eventually result in more ecologically friendly and sustainable AI technologies.

Bibliography

[AAF12] V. Avelar, D. Azevedo, A. French. *Pue: A comprehensive examination of the metric*. 2012. URL: https://www.thegreengrid.org/en/resources/library-and-tools/20-pue%3A-a-comprehensive-examination-of-the-metric (visited on 05/22/2024) (cit. on p. 29).

[ADMQ14] P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana-Orti. "Modeling power and energy consumption of dense matrix factorizations on multicore processors". In: *Concurrency and Computation: Practice and Experience* 26 (Dec. 2014). DOI: 10.1002/cpe.3162 (cit. on p. 30).

[Bel00] F. Bellosa. "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems". In: Jan. 2000. DOI: 10.1145/566726.566736 (cit. on p. 30).

[BM12] R. Basmadjian, H. de Meer. "Evaluating and modeling power consumption of multi-core processors". In: *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*. e-Energy '12. Madrid, Spain: Association for Computing Machinery, 2012. ISBN: 9781450310550. DOI: 10.1145/2208828.2208840. URL: https://doi.org/10.1145/2208828.2208840 (cit. on p. 30).

[CJSM17] E. Cai, D.-C. Juan, D. Stamoulis, D. Marculescu. "NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks". In: (2017). arXiv: 1710.05420 [cs.LG] (cit. on p. 68).

[CM05] G. Contreras, M. Martonosi. "Power prediction for intel XScale® processors using performance monitoring unit events". In: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. ISLPED '05. San Diego, CA, USA: Association for Computing Machinery, 2005, pp. 221–226. ISBN: 1595931376. DOI: 10.1145/1077603.1077657. URL: https://doi.org/10.1145/1077603.1077657 (cit. on pp. 20, 32, 40).

[CM15] M. Claesen, B. D. Moor. *Hyperparameter Search in Machine Learning*. 2015. arXiv: 1502.02127 [cs.LG] (cit. on p. 53).

[Cod20a] CodeCarbon. *Carbon Intensity*. 2020. URL: https://mlco2.github.io/codecarbon/methodology.html#carbon-intensity (visited on 04/28/2024) (cit. on pp. 47, 48).

[Cod20b] CodeCarbon. *Germany Carbon Intensity Value*. 2020. URL: https://github.com/mlco2/codecarbon/blob/master/codecarbon/data/private_infra/global_energy_mix.json (visited on 04/28/2024) (cit. on p. 48).

[dev07] scikit-learn developers. *MinMaxScaler*. 2007. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (visited on 05/21/2024) (cit. on p. 41).

[Flo24] Flozz. *pyPAPI GitHub*. 2024. URL: https://github.com/flozz/pypapi (visited on 05/22/2024) (cit. on p. 34).

[GLG+19] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, V. Boeva. "How to Measure Energy Consumption in Machine Learning Algorithms". In: *ECML PKDD 2018 Workshops*. Cham: Springer International Publishing, 2019, pp. 243–255. ISBN: 978-3-030-13453-2 (cit. on pp. 25, 31, 40).

[GM16] B. Goel, S. A. McKee. "A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 273–282. DOI: 10.1109/IPDPS.2016.118 (cit. on p. 30).

[GRRG19] E. García-Martín, C. F. Rodrigues, G. Riley, H. Grahn. "Estimation of energy consumption in machine learning". In: *Journal of Parallel and Distributed Computing* 134 (2019), pp. 75–88. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0743731518308773 (cit. on pp. 23, 24, 31, 40).

[HLK05] Y. Hu, Q. Li, C.-C. Kuo. "Run-time power consumption modeling for embedded multimedia systems". In: *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*. 2005, pp. 353–356. DOI: 10.1109/RTCSA.2005.86 (cit. on p. 31).

[HSA22] K. Haghshenas, B. Setz, M. Aiello. "CO2 Emission Aware Scheduling for Deep Neural Network Training Workloads". In: *2022 IEEE International Conference on Big Data (Big Data)*. 2022, pp. 1542–1549. DOI: 10.1109/BigData55660.2022.10020544 (cit. on pp. 20, 29, 31, 37).

[HSBA22] K. Haghshenas, B. Setz, Y. Bloch, M. Aiello. *Enough Hot Air: The Role of Immersion Cooling*. 2022. arXiv: 2205.04257 [cs.DC] (cit. on p. 29).

[Jha11] S. Jha. "Poorly written apps can sap 30 to 40% of a phone's juice". In: *CEO, Motorola Mobility, Bank of America Merrill Lynch 2011 Technology Conference*. 2011 (cit. on pp. 17, 18).

[JM01] R. Joseph, M. Martonosi. "Run-time power estimation in high performance microprocessors". In: *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*. ISLPED '01. Huntington Beach, California, USA: Association for Computing Machinery, 2001, pp. 135–140. ISBN: 1581133715. DOI: 10.1145/383082.383119. URL: https://doi.org/10.1145/383082.383119 (cit. on p. 30).

[KCK+01] I. Kadayif, T. Chinoda, M. Kandemir, V. Narayanan, M. Irwin, A. Sivasubramaniam. "vEC: Virtual Energy Counters". In: June 2001, pp. 28–31. DOI: 10.1145/379605.379639 (cit. on p. 31).

[Lil18] U. of Lille. *pyRAPL 0.2.3.1*. 2018. URL: https://pypi.org/project/pyRAPL/ (visited on 05/22/2024) (cit. on p. 34).

[Lil19] U. of Lille. *Welcome to pyRAPL's documentation!* 2019. URL: https://pyrapl.readthedocs.io/en/latest/ (visited on 05/22/2024) (cit. on p. 34).

[LJ03] T. Li, L. K. John. “Run-time modeling and estimation of operating system power consumption”. In: *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’03. San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 160–171. ISBN: 1581136641. DOI: 10.1145/781027.781048. URL: https://doi.org/10.1145/781027.781048 (cit. on p. 31).

[Mai22] U. of Maine System. *Running Average Power Limit Energy Reporting*. 2022. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html (visited on 05/22/2024) (cit. on p. 34).

[MMDH99] P. Mucci, S. Moore, C. Deane, G. Ho. “PAPI: A Portable Interface to Hardware Performance Counters”. In: (Jan. 1999) (cit. on p. 33).

[MMN+24] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, J. Gao. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL] (cit. on p. 17).

[MWKJ17] A. Mazouz, D. C. Wong, D. Kuck, W. Jalby. “An Incremental Methodology for Energy Measurement and Modeling”. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ICPE ’17. L’Aquila, Italy: Association for Computing Machinery, 2017, pp. 15–26. ISBN: 9781450344043. DOI: 10.1145/3030207.3030224. URL: https://doi.org/10.1145/3030207.3030224 (cit. on p. 34).

[Pyp24] Pypapi. *Welcome to PyPAPI’s documentation!* 2024. URL: https://flozz.github.io/pypapi/ (visited on 05/22/2024) (cit. on p. 34).

[QTJ+06] Qianjie, C. Tianzhou, H. Jiangwei, Qianjie, C. Tianzhou, H. Jiangwei. “Power Estimation for an Application on the Xscale Platform Using PMU events”. In: *International Conference on Digital Telecommunications (ICDT’06)*. 2006, pp. 43–43. DOI: 10.1109/ICDT.2006.61 (cit. on p. 32).

[RRL18] C. Rodrigues, G. Riley, M. Luján. “SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1”. In: (Aug. 2018). DOI: 10.13140/RG.2.2.36489.54881 (cit. on p. 68).

[RSR+17] V. Reddy, B. Setz, G. K. Rao, G. Gangadharan, M. Aiello. “Metrics for Sustainable Data Centers”. In: *IEEE Transactions on Sustainable Computing* 2.03 (July 2017), pp. 290–303. ISSN: 2377-3782. DOI: 10.1109/TSUSC.2017.2701883 (cit. on pp. 20, 29).

[Sac23] G. Sachs. *How quantifying Avoided Emissions can broaden the decarbonization investment universe*. 2023. URL: https://www.goldmansachs.com/intelligence/pages/gs-research/how-quantifying-avoided-emissions-can-broaden-the-decarbonization-investment-universe/report.pdf (visited on 05/21/2024) (cit. on p. 17).

[Sac24a] G. Sachs. *AI, Data Centers and the Coming US Power Demand Surge*. 2024. URL: https://www.goldmansachs.com/intelligence/pages/gs-research/generational-growth-ai-data-centers-and-the-coming-us-power-surge/report.pdf (visited on 05/21/2024) (cit. on pp. 17–19).

[Sac24b] G. Sachs. *AI/Data Centers' Global Power Surge and the Sustainability Impact*. 2024. URL: https://www.goldmansachs.com/intelligence/pages/gs-research/ai-data-centers-global-power-surge-and-sustainability-impact/report.pdf (visited on 05/21/2024) (cit. on pp. 17–19).

[SBM09] K. Singh, M. Bhadauria, S. A. McKee. "Real time power estimation and thread scheduling via performance counters". In: *SIGARCH Comput. Archit. News* 37.2 (July 2009), pp. 46–55. ISSN: 0163-5964. DOI: 10.1145/1577129.1577137. URL: https://doi.org/10.1145/1577129.1577137 (cit. on p. 30).

[SCC+12] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, K. Winbladh. "Towards power reduction through improved software design". In: *2012 IEEE Energytech*. 2012, pp. 1–6. DOI: 10.1109/EnergyTech.2012.6304705 (cit. on p. 18).

[SGM19] E. Strubell, A. Ganesh, A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: (2019). arXiv: 1906.02243 [cs.CL] (cit. on pp. 18, 32).

[SLL19] D. So, Q. Le, C. Liang. "The Evolved Transformer". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri, R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 5877–5886. URL: https://proceedings.mlr.press/v97/so19a.html (cit. on p. 18).

[TMWL96] V. Tiwari, S. Malik, A. Wolfe, M.-C. Lee. "Instruction level power analysis and optimization of software". In: *Proceedings of 9th International Conference on VLSI Design*. 1996, pp. 326–328. DOI: 10.1109/ICVD.1996.489624 (cit. on p. 29).

[WMV20] H. J. P. Weerts, A. C. Mueller, J. Vanschoren. *Importance of Tuning Hyperparameters of Machine Learning Algorithms*. 2020. arXiv: 2007.07588 [cs.LG] (cit. on p. 53).

[WSS+23] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, F. Hutter. *Neural Architecture Search: Insights from 1000 Papers*. 2023. arXiv: 2301.08727 [cs.LG] (cit. on p. 18).

[YLL+16] S. Yang, Z. Luan, B. Li, G. Zhang, T. Huang, D. Qian. "Performance Events Based Full System Estimation on Application Power Consumption". In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2016, pp. 749–756. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0109 (cit. on p. 31).

A List of Intel® Core™ i7-8565U CPU PMC Data

Table 1: List of PMCs offered by Intel® Core™ i7-8565U CPU @ 1.80 GHz (142, 0x8e)

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	Yes	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	Yes	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	Yes	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	Yes	No	Level 2 instruction cache misses
PAPI_L3_DCM	0x80000004	No	No	Level 3 data cache misses
PAPI_L3_ICM	0x80000005	No	No	Level 3 instruction cache misses
PAPI_L1_TCM	0x80000006	Yes	Yes	Level 1 cache misses
PAPI_L2_TCM	0x80000007	Yes	No	Level 2 cache misses
PAPI_L3_TCM	0x80000008	Yes	No	Level 3 cache misses
PAPI_CA_SNP	0x80000009	Yes	No	Requests for a snoop
PAPI_CA_SHR	0x8000000a	Yes	No	Requests for exclusive access to shared cache line
PAPI_CA_CLN	0x8000000b	Yes	No	Requests for exclusive access to clean cache line
PAPI_CA_INV	0x8000000c	No	No	Requests for cache line invalidation
PAPI_CA_ITV	0x8000000d	Yes	No	Requests for cache line intervention
PAPI_L3_LDM	0x8000000e	Yes	No	Level 3 load misses
PAPI_L3_STM	0x8000000f	No	No	Level 3 store misses
PAPI_BRU_IDL	0x80000010	No	No	Cycles branch units are idle
PAPI_FXU_IDL	0x80000011	No	No	Cycles integer units are idle
PAPI_FPU_IDL	0x80000012	No	No	Cycles floating point units are idle
PAPI_LSU_IDL	0x80000013	No	No	Cycles load/store units are idle
PAPI_TLB_DM	0x80000014	Yes	Yes	Data translation lookaside buffer misses
PAPI_TLB_IM	0x80000015	Yes	No	Instruction translation lookaside buffer misses
PAPI_TLB_TL	0x80000016	No	No	Total translation lookaside buffer misses
PAPI_L1_LDM	0x80000017	Yes	No	Level 1 load misses
PAPI_L1_STM	0x80000018	Yes	No	Level 1 store misses
PAPI_L2_LDM	0x80000019	Yes	No	Level 2 load misses
PAPI_L2_STM	0x8000001a	Yes	No	Level 2 store misses

Continued on next page

Table 1 continued from previous page

Name	Code	Avail	Deriv	Description (Note)
PAPI_BTAC_M	0x8000001b	No	No	Branch target address cache misses
PAPI_PRF_DM	0x8000001c	Yes	No	Data prefetch cache misses
PAPI_L3_DCH	0x8000001d	No	No	Level 3 data cache hits
PAPI_TLB_SD	0x8000001e	No	No	Translation lookaside buffer shootdowns
PAPI_CSR_FAL	0x8000001f	No	No	Failed store conditional instructions
PAPI_CSR_SUC	0x80000020	No	No	Successful store conditional instructions
PAPI_CSR_TOT	0x80000021	No	No	Total store conditional instructions
PAPI_MEM_SCY	0x80000022	No	No	Cycles Stalled Waiting for memory accesses
PAPI_MEM_RCY	0x80000023	No	No	Cycles Stalled Waiting for memory Reads
PAPI_MEM_WCY	0x80000024	Yes	No	Cycles Stalled Waiting for memory writes
PAPI_STL_ICY	0x80000025	Yes	No	Cycles with no instruction issue
PAPI_FUL_ICY	0x80000026	Yes	Yes	Cycles with maximum instruction issue
PAPI_STL_CCY	0x80000027	Yes	No	Cycles with no instructions completed
PAPI_FUL_CCY	0x80000028	Yes	No	Cycles with maximum instructions completed
PAPI_HW_INT	0x80000029	No	No	Hardware interrupts
PAPI_BR_UCN	0x8000002a	Yes	Yes	Unconditional branch instructions
PAPI_BR_CN	0x8000002b	Yes	No	Conditional branch instructions
PAPI_BR_TKN	0x8000002c	Yes	Yes	Conditional branch instructions taken
PAPI_BR_NTK	0x8000002d	Yes	No	Conditional branch instructions not taken
PAPI_BR_MSP	0x8000002e	Yes	No	Conditional branch instructions mispredicted
PAPI_BR_PRC	0x8000002f	Yes	Yes	Conditional branch instructions correctly predicted
PAPI_FMA_INS	0x80000030	No	No	FMA instructions completed
PAPI_TOT_IIS	0x80000031	No	No	Instructions issued
PAPI_TOT_INS	0x80000032	Yes	No	Instructions completed
PAPI_INT_INS	0x80000033	No	No	Integer instructions
PAPI_FP_INS	0x80000034	No	No	Floating point instructions
PAPI_LD_INS	0x80000035	Yes	No	Load instructions
PAPI_SR_INS	0x80000036	Yes	No	Store instructions
PAPI_BR_INS	0x80000037	Yes	No	Branch instructions
PAPI_VEC_INS	0x80000038	No	No	Vector/SIMD instructions (could include integer)
PAPI_RES_STL	0x80000039	Yes	No	Cycles stalled on any resource

Continued on next page

Table 1 continued from previous page

Name	Code	Avail	Deriv	Description (Note)
PAPI_FP_STAL	0x8000003a	No	No	Cycles the FP unit(s) are stalled
PAPI_TOT_CYC	0x8000003b	Yes	No	Total cycles
PAPI_LST_INS	0x8000003c	Yes	Yes	Load/store instructions completed
PAPI_SYC_INS	0x8000003d	No	No	Synchronization instructions completed
PAPI_L1_DCH	0x8000003e	No	No	Level 1 data cache hits
PAPI_L2_DCH	0x8000003f	No	No	Level 2 data cache hits
PAPI_L1_DCA	0x80000040	No	No	Level 1 data cache accesses
PAPI_L2_DCA	0x80000041	Yes	No	Level 2 data cache accesses
PAPI_L3_DCA	0x80000042	Yes	Yes	Level 3 data cache accesses
PAPI_L1_DCR	0x80000043	No	No	Level 1 data cache reads
PAPI_L2_DCR	0x80000044	Yes	No	Level 2 data cache reads
PAPI_L3_DCR	0x80000045	Yes	No	Level 3 data cache reads
PAPI_L1_DCW	0x80000046	No	No	Level 1 data cache writes
PAPI_L2_DCW	0x80000047	Yes	Yes	Level 2 data cache writes
PAPI_L3_DCW	0x80000048	Yes	No	Level 3 data cache writes
PAPI_L1_ICH	0x80000049	No	No	Level 1 instruction cache hits
PAPI_L2_ICH	0x8000004a	Yes	No	Level 2 instruction cache hits
PAPI_L3_ICH	0x8000004b	No	No	Level 3 instruction cache hits
PAPI_L1_ICA	0x8000004c	No	No	Level 1 instruction cache accesses
PAPI_L2_ICA	0x8000004d	Yes	No	Level 2 instruction cache accesses
PAPI_L3_ICA	0x8000004e	Yes	No	Level 3 instruction cache accesses
PAPI_L1_ICR	0x8000004f	No	No	Level 1 instruction cache reads
PAPI_L2_ICR	0x80000050	Yes	No	Level 2 instruction cache reads
PAPI_L3_ICR	0x80000051	Yes	No	Level 3 instruction cache reads
PAPI_L1_ICW	0x80000052	No	No	Level 1 instruction cache writes
PAPI_L2_ICW	0x80000053	No	No	Level 2 instruction cache writes
PAPI_L3_ICW	0x80000054	No	No	Level 3 instruction cache writes
PAPI_L1_TCH	0x80000055	No	No	Level 1 total cache hits
PAPI_L2_TCH	0x80000056	No	No	Level 2 total cache hits
PAPI_L3_TCH	0x80000057	No	No	Level 3 total cache hits
PAPI_L1_TCA	0x80000058	No	No	Level 1 total cache accesses
PAPI_L2_TCA	0x80000059	Yes	Yes	Level 2 total cache accesses
PAPI_L3_TCA	0x8000005a	Yes	No	Level 3 total cache accesses
PAPI_L1_TCR	0x8000005b	No	No	Level 1 total cache reads
PAPI_L2_TCR	0x8000005c	Yes	Yes	Level 2 total cache reads
PAPI_L3_TCR	0x8000005d	Yes	Yes	Level 3 total cache reads
PAPI_L1_TCW	0x8000005e	No	No	Level 1 total cache writes
PAPI_L2_TCW	0x8000005f	Yes	Yes	Level 2 total cache writes
PAPI_L3_TCW	0x80000060	Yes	No	Level 3 total cache writes
PAPI_FML_INS	0x80000061	No	No	Floating point multiply instructions
PAPI_FAD_INS	0x80000062	No	No	Floating point add instructions

Continued on next page

Table 1 continued from previous page

Name	Code	Avail	Deriv	Description (Note)
PAPI_FDV_INS	0x80000063	No	No	Floating point divide instructions
PAPI_FSQ_INS	0x80000064	No	No	Floating point square root instructions
PAPI_FNV_INS	0x80000065	No	No	Floating point inverse instructions
PAPI_FP_OPS	0x80000066	No	No	Floating point operations
PAPI_SP_OPS	0x80000067	Yes	Yes	Floating point operations; optimised to count scaled single precision vector operations
PAPI_DP_OPS	0x80000068	Yes	Yes	Floating point operations; optimised to count scaled double precision vector operations
PAPI_VEC_SP	0x80000069	Yes	Yes	Single precision vector/SIMD instructions
PAPI_VEC_DP	0x8000006a	Yes	Yes	Double precision vector/SIMD instructions
PAPI_REF_CYC	0x8000006b	Yes	No	Reference clock cycles

Declaration