Journal of Parallel and Distributed Computing 134 (2019) 75–88

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Estimation of energy consumption in machine learning

Eva García-Martín [a,*], Crefeda Faviola Rodrigues [b], Graham Riley [b], Håkan Grahn [a]

[a] Blekinge Institute of Technology, Karlskrona, Sweden
[b] University of Manchester, Manchester, UK

ARTICLE INFO

Article history:
Received 30 November 2018
Received in revised form 17 May 2019
Accepted 23 July 2019
Available online 21 August 2019

Keywords:
Machine learning
GreenAI
Energy consumption
Deep learning
High performance computing

ABSTRACT

Energy consumption has been widely studied in the computer architecture field for decades. While the adoption of energy as a metric in machine learning is emerging, the majority of research is still primarily focused on obtaining high levels of accuracy without any computational constraint. We believe that one of the reasons for this lack of interest is due to their lack of familiarity with approaches to evaluate energy consumption. To address this challenge, we present a review of the different approaches to estimate energy consumption in general and machine learning applications in particular. Our goal is to provide useful guidelines to the machine learning community giving them the fundamental knowledge to use and build specific energy estimation methods for machine learning algorithms. We also present the latest software tools that give energy estimation values, together with two use cases that enhance the study of energy consumption in machine learning.

1. Introduction

Computer architecture researchers have been investigating energy consumption for decades, especially to be able to deliver state-of-the-art energy efficient processors. Machine learning researchers, on the other hand, have been mainly focused on producing high accurate models without considering energy consumption as an important factor [18]. This is the case for deep learning, where the goal has been to produce deeper and more accurate model without any constraints in terms of computation. These models have grown in computation (typically in the GigaFlops) and memory requirements (typically in the millions of parameters or weights). These algorithms require high levels of computing power during training as they have to be trained on large amounts of the data while during deployment they may be used multiple times. Some awareness in energy consumption is starting to arise, originating from a few machine learning research groups [12,14,47,61] and challenges such as *The Low Power Image Recognition Challenge* (LPIRC) [26]. Thus, we believe that efforts towards estimating energy consumption and developing tools for researchers to advance their research in energy consumption are necessary for a more scalable and sustainable future.

We believe that the reasons why the machine learning community has not shown more interest in energy consumption is because of their lack of familiarity with the current approaches to estimate energy and the lack of power models in existing machine learning frameworks, for example, in Tensorflow [1], Caffe2 [43], PyTorch [56] and others to support energy evaluations. This study addresses this challenge by making the following contributions:

i. We present a literature review of different energy estimation approaches from the computer architecture community (Section 4). We synthesize and classify the papers into high-level taxonomy categories (Section 3) and modeling techniques to enable a user from the machine learning or computer architecture community to decide which estimation model could be used or built for a given scenario. We also present the advantages and disadvantages for each category.
ii. We present the current state-of-the-art approaches to estimate energy consumption in machine learning (Section 6).
iii. We present the currently available software tools and present their characteristics to the user to facilitate building energy consumption models (Section 5). We categorize the tools based on the granularity of the energy estimations, software that is supported, precision etc.
iv. Finally, based on the classification of the surveyed papers we present two use cases from the perspective of a machine learning user that wants to estimate the energy consumption of their machine learning model and reveal insights into the importance of studying energy consumption when designing future machine learning systems (Section 7).

Our survey covers energy estimation models but not direct energy measurements since the latter is based on tools such as watt-meters [47] or power sensors [60] for which most systems

* Corresponding author.
E-mail address: eva.garcia.martin@bth.se (E. García-Martín).

https://doi.org/10.1016/j.jpdc.2019.07.007

lack the necessary infrastructure and requires investment in time to set up the necessary hardware and software support [30]. Moreover, some efforts to build energy estimation models require real-time power consumption measurements from such devices and is thus included in our survey. While other works have surveyed energy estimation techniques on mobile devices [2,38], GPUs [8] and HPC systems [55]; we present, to the best of our knowledge, the first survey of system-level energy estimation approaches in the architecture community that could be applied to machine learning scenarios. We provide a useful guideline to the machine learning community on how to use these well-established methods to estimate energy and build models for their algorithms. This survey extends a previously published paper [25]. The scope of this survey focuses on CPU-based and DRAM-based energy estimations, leaving energy estimation on emerging hardware for future works.

2. Background

This section explains the main concepts and terminology used throughout the rest of the paper. **Energy**, measured in joules (J), is the total power consumed during an interval of time. Power, i.e., the rate at which energy is consumed, is the sum of static and dynamic power. **Static power**, also known as leakage power, is the power consumed when there is no circuit activity. **Dynamic power** is the power consumed by the circuit, from charging and discharging the capacitance load in the circuit [35]:

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{1}$$

where α is the activity factor, representing the percentage of the circuit that is active. V_{dd} is the voltage, C the capacitance, and f the clock frequency. The power consumption of a system during an application's execution is determined by the power consumption of the components themselves as well as the how the components are used. Another metric to evaluate energy efficiency that is used by several surveyed papers is the **energy delay product (EDP)**. EDP is calculated as the product of the energy and the delay (execution time). Since energy is the product of power and time, EDP is the product of power and time squared. This measure is used to give more importance to application runtime, with the goal of making both low energy and fast runtime applications [46].

Finally, performance counters (PMCs) are a set of special-purpose registers in modern processors that count specific event types that are hardware related (e.g. L2 cache misses). Intel has incorporated many PMCs in their modern processors. However, they differentiate their metrics into **core** and **uncore**. Core refers to the components that are in the core, such as the ALU, registers, L1 cache, and L2 cache. Uncore, on the other hand, are those components outside the processor core, such as the DRAM and memory controllers.

3. Taxonomy of power estimation models

Power models are built to design better hardware, design better algorithms or design better software to map these algorithms onto hardware. Following the abstraction levels at the hardware–software stack, there already exists a taxonomy of power models ranging from low-level transistors to high-level system description of hardware components, such as processor, cache, bus and others [73]. In this paper we propose a more detailed taxonomy that better aligns with our goal of mapping energy estimation techniques to machine learning applications. We focus on power estimation models that can be built at the system-level to understand the energy consumption at the application level. We propose the following taxonomy of power estimation models at the system-level:

- Software level: The developers of the model at this level are interested in the energy consumption of the application or software implementation and explore optimization techniques that include designing efficient algorithms or better software implementation of the algorithm.
  - Application-level: At the topmost abstraction level, a power consumption model can be built by relating algorithmic properties of the application directly to the power estimation. Here, the developer of the power models extracts characteristics of the application, for example, kernel sizes in a neural network, and relate it to the energy profile of the application.
  - Instruction-level: At the next level, a power consumption model can be built to understand which specific instructions in the program contribute to the energy consumption. The instruction traces can be extracted using an instruction-set simulator or performance counter profiling, and the cost for each instruction can be added either by known or relative the cost of the instruction or experimental data. This can be applied to estimate the energy consumption of the different functions of machine learning algorithms. This is useful for understanding which parts of the algorithm are consuming most of the energy, to focus the efforts on reducing the energy consumption of such parts [24].
- Hardware-level: The developers of the model at this level are interested in the energy consumption of specific hardware components. They are interested in identifying the hardware components (processor, memory and IO peripherals) that are strongly correlated to the power of the application, also referred to as Functional-level power analysis (FLPA) in [59]. These power models are valuable for the machine learning researchers interested in building specific chips for machine learning computations [45].

There are a number of ways to extract useful features or activity factors from the target hardware. These include simulators and performance counters. More details about these techniques and their connection to the proposed taxonomy categories are given in Section 4.

4. Approaches to estimate energy consumption

The goal of this section is to introduce key approaches to estimate energy to the machine learning expert. First, we give a general overview of the energy estimation field, providing the reader with the basic knowledge. Second, we explain in detail how the energy estimation models are built, to give the machine learning expert or computer architecture researcher the starting point to build or use more specific machine learning energy models.

The papers reviewed in this section are chosen from a more general survey [30] on power measurement, estimation, and management. We also include papers that we discovered from researching the field on ways to estimate the energy consumption that could be applied in machine learning scenarios. This was achieved by searching on online databases (e.g. Google Scholar) and by looking at the references from some key papers in the area.

The surveyed papers can be clustered into four groups: (i) papers that obtain the activity factors with performance counters and use regression or correlation techniques to obtain the power or energy; (ii) papers that use simulation data to obtain the activity factors; (iii) papers providing architecture or instruction

Table 1
Connection between the categories from the taxonomy (Section 3) and the techniques from Section 4.

Taxonomy		Technique
Software-level		
Application-level	→	PMC, Simulation, Real-time power estimation
Instruction-level	→	Instruction-level estimation
Hardware-level	→	Hardware-level estimation

level information; (iv) papers that provide real-time power or energy estimation. Table 1 provides the connection between the general categories from the taxonomy presented in Section 3 and the specific techniques mentioned above (PMC, simulation, architecture or instruction level, and real-time estimation).

Table 2 summarizes the advantages and disadvantages of each technique, together with examples of possible machine learning applications of such techniques. More details are given in Sections 4.1–4.4, where we provide a synthesis of the reviewed papers grouping them in the mentioned categories.

Table 3 categorizes the surveyed papers into: taxonomy category, input, technique, output, validation, model requirements, type of machine, and availability. *Input* refers to what was the input in order to create the model. *Model requirements* refers to the type of activity factors required by the model to output power or energy consumption values. Most papers require either simulation data or data obtained from performance counters.

4.1. Performance counters using regression or correlation techniques

The majority of papers reviewed in this study obtain the activity factors of the computations via performance counters (PMCs), to then build the model using regression techniques. The models that are included in this category are: [4,5,22,27–29,51,58,61,68,70,77,78].

The majority of the approaches [4,5,22,27,29,51,58,68,77,78] derive the power consumption by obtaining the power weights associated to each PMC using linear regression or similar techniques, as presented in Eq. (2) [5].

$$P_{total} = \left(\sum_{i=1}^{n_{component}} AR_i \cdot w_i \right) + P_{static}, \tag{2}$$

where w_i is the weight associated to component i, AR_i is the activity ratio of component i, and P_{static} represents the overall static power of all components.

While not all papers in this category follow Eq. (2), their power modeling approach can be approximated by it with a few modifications. In particular, a few approaches [29,68] present a piecewise linear regression model. They divide the linear function into several segments that better fit the data. To obtain the weights associated to the components, all papers mentioned above except for two [77,78], isolate the power associated by each component by running a specific micro-benchmark that only runs instructions that will stress that component and not others. To choose the appropriate set of PMCs, several papers [4,27,29,68,77] correlate the power consumption to the PMCs, choosing the PMCs with the highest correlation to power. Another study [78], in addition to correlating PMCs to real power measurements, also eliminates the PMCs that show high correlation to other PMCs, choosing PMCs that can isolate power values. They also use a clustering approach to create sets of PMCs, to then choose the most representative PMC from each set. They estimate the total power consumption with regression techniques but adding the frequency and voltage parameters to Eq. (2).

As opposed to previous models, other approaches [28,70] correlate capacitance (C value in Eq. (1)) to PMCs. Their approach is similar to the one represented by Eq. (2), but in this case the authors' goal is to minimize the difference between the predicted capacitance and the real one. By correlating capacitance instead of power, they obtain models that are independent of frequency and voltage Eq. (1). This is different to previous approaches, since they had to create a model for each set of frequency-voltage pair. Rather than using benchmarks that isolate each PMC, they run many benchmarks and obtain the best set of weights from all the runs.

The advantages of using performance counters are many, detailed in Table 2. There is practically no extra overhead of using this technique. Thus it can be used to measure the energy consumption of both training and inference of machine learning algorithms. Since it is available for different operating systems (OSX, Windows, and Linux) it can be used in many different scenarios and in many different platforms. Another advantage is that it can be used for both large-scale datasets and small datasets, which is very useful with the current advancements on Big Data. Although the energy results are broken down for the CPU and DRAM, the main drawback is that there is no energy breakdown per process.

4.2. Simulation

Other models obtain the activity factors via simulation [10,44,49,50,59,64,79,80]. Wattch [10] was one of the first architectural simulators that estimated CPU power consumption. They presented parametrized power models and used analytical dynamic power equations to estimate the power values. Their models are based on capacitance estimations, similar to the papers mentioned above [28,70]. However, instead of correlating capacitance to performance counters, Wattch estimates the capacitance at the circuit level. Another approach that estimates capacitance using simulation is SimplePower [80], which provides a table that characterizes the capacitance for each *input transition*. The total energy for each module is calculated as the sum of the energy consumed by each *input transition* [80]. Furthermore, another study [44] uses PMCs as activity factors, but utilizes Wattch power models to estimate the power consumption. This paper could be also included in Section 4.1, where we define PMC-based models.

An extension [64] to Tiwari et al. [76] correlates simulation data to power using regression based approaches. They propose a pipeline-aware energy model, in contrast to a traditional approach that does not consider the effect of more than one instruction present in the pipeline. Another simulation study that uses regression [49] create piecewise functions to model the non-linearity of the data, similar to the PMC studies already mentioned [29,68]. The novelty of their approach is that they propose a methodology that uses a reduced set of simulation runs to build their model, reducing the complexity of the solution.

The authors of McPAT [50] present a modeling approach that takes simulation data as input, and based on analytical power models, output power estimates at each unit. McPAT is similar to Wattch, but McPAT provides more state-of-the-art, detailed and realistic models for multi-core systems. Finally, a recent study [79] simulates the number of memory accesses at each level of the memory hierarchy.

Table 2
Advantages, disadvantages, and possible machine learning applications of the techniques summarized in Sections 4.1–4.4.

Technique	ML Application	Advantages	Disadvantages
PMC	Energy consumption analysis of any ML model	No overhead. Application independent	No per-processor results
Simulation	Analysis of algorithms behavior on ML specific hardware	Detailed results	Significant Overhead
Instruction-level	Energy consumption analysis of specific layers in a neural network	Detailed breakdown of energy consumption	Not easily available
Architecture-level	Improve programming hardware for ad-hoc ML applications	Detailed view	Usually not generalizable to different hardware platforms
Real-time	Streaming data and IoT	Easily available	Usually not detailed results

Table 3
Energy estimation models.

Model	Taxonomy category	Input	Technique	Output	Model req.	Type of machine	Availability	Validation
[76]	Instruction	Current	Analytical	Energy	Profiling	CPU	Model	Yes
[10]	Hardware	Analytical	Analytical	Power, Energy, Perf	Simu data	CPU	Theoretical	Yes
[80]	Hardware	Analytical	Transition-sensitive	Energy/Power	Simulation	CPU, f units	Theoretical	–
[44]	Hardware	Real	Heuristics	Power	PMC, sim data	CPU	–	Yes
[64]	Instruction	Simulation	Analytical	Energy	Simulation	VLIW Processor	Model	Yes
[4]	Hardware	Real	Correlations	Power, Temperature	PMC	CPU	Model	Yes
[27]	Hardware	Real	Regression	Power	PMC	Intel PXA255	Model	Yes
[58]	Hardware	Real	Correlation	Power, Perf	PMC	Desktop	Model	–
[49]	Hardware	Analytical	Regression	Power, Perf	Simulation	CPU, f units	Methodology	Yes
[22]	Hardware	Real	Liner regression	Power	PMC	Server	Methodology	
[68]	Hardware	Real	Regression	Power	PMC	Desktop	Model	Yes
[50]	Hardware	Analytical	Analytical, empirical	Power, Time, Area	Sim data	Multicore processor	Open source	Yes
[5]	Hardware	Real	Regression	Power	PMC	Multicore processor	Model	Yes
[29]	Hardware	Real	Regression	Power	PMC	CPU	Methodology	Yes
[17]	Hardware	–	–	Energy, power	PMC	Intel processor	Tool	–
[70]	Application	Real	Correlation	Power, Perf	PMC	Processor	Model	Yes
[66]	Instruction	Real	Analytical	Energy	PMC	Intel Xeon Phi	Methodology	Yes
[59]	Hardware	Real	Regression	Power	Simulation	m-CPUs, mem, f units	Theor, Method.	Power data
[48]	Instruction	Compile time	Regression	Perf, energy, EDP	Static code analysis	mCPU	Methodology	Sandy bridge
[77]	Hardware	Real	Regression	Power	PMC, CPU util	mCPUs (2 systems)	Theoretical	Power data
[28]	Hardware	Real	Correlations	Power	PMC	CPU	Model	Yes
[51]	Instr, HW	PMC	Linear regression	Static, Dyn energy	PMC	Multicore processor	Model	Yes
[78]	Hardware	Real	Regression, Analytical	Power	PMC	mCPU	Tool	Power data

The main advantage of using a simulation approach to estimate energy, as shown in Table 2, is that it gives extensive details regarding where the energy is consumed in both the hardware components and at the instruction level. The main drawback is that it introduces a significant overhead, thus big experiments on large-scale datasets are unfeasible. In regard to machine learning, simulation can be used to understand which hardware component is responsible for the highest energy consumption, to then design the algorithm accordingly.

4.3. Instruction-level or architecture-level estimation

Only a few papers give instruction-level energy estimations [48,51,64,66,76]. Most of the approaches run a set of curated micro-benchmarks where each benchmark loops over a target instruction type, to be able to isolate the power of that specific instruction [51,66,76]. In particular, Tiwari et al. [76] propose to the best of our knowledge, the first instruction-level energy estimation model. They profile the execution of the program, instead of using performance counters [66]. On the other hand, Shao et al. [66] model the energy per instruction for an Intel Xeon Phi processor, proposing more modern models that consider multi-core and multi-thread processors. The energy is then estimated as a function of the power consumed during the run of the benchmark, the cycle time, and the frequency. Finally, a recent work [51] presents an approach to model energy consumption both at the instruction and architectural levels.

Architectural models are more useful for micro architecture level exploration [66], for instance, to create more energy efficient accelerators for machine learning tasks. The majority of the papers give details at the architectural-level [5,10,22,44,48–51,59,77,78,80]. These papers provide an energy or power breakdown per component, thus giving information about which components are the most energy consuming (e.g. L1 cache, DRAM, etc.) These models are also useful for researchers interested in overall power consumption.

4.4. Real-time power estimation

All models that obtain the activity factors via performance counters allow for real time energy monitoring. The reason is that accessing those registers does not introduce any significant overhead [4,5,17,22,27–29,44,51,58,61,66,68,70,77,78]. Some models, however, need an offline calibration phase to obtain the parameters of the model, but this is usually done only once [22] for each machine.

Simulation based models, on the other side, do not offer real-time energy or power estimation, due to the introduced overhead and that they need to do a full profile run to get the values [10, 49,50,59,64,76,79,80].

Real-time estimation is useful for areas such as data stream mining and online learning, where the models are built as the data arrives.

4.5. Detailed explanation of the reviewed papers

The previous sections synthesized the surveyed papers by explaining in general the different categories and main techniques used to obtain power estimation values. A more detailed view into each paper is presented in the following paragraphs. They center on how the model was created, and how can energy be estimated from a few processor statistics.

Tiwari et al. (1996) [76]. To our knowledge, this is the first approach to estimate power consumption at the instruction level by assigning an energy cost to each instruction type. Their motivation is to provide an approach to estimate the power consumption of software, which was not done yet at that time, since traditional power analysis tools were not suited for power analysis of software. The total power is given by the current (I) times the supply voltage (V_{DD}). They measure the current directly at the CPU power pins with a meter. Once the current is known, they derive power costs to individual instructions by running specific programs and measuring the current drawn during the execution. They execute a loop that consists of several instances of the same instruction and measure the current and the V_{DD}, obtaining the base power cost for that type of instruction. That base power is multiplied by the number of non-overlapped cycles needed to execute the instruction, obtaining the energy base cost. To the base cost, they add the circuit overhead and the other inter instruction effects such as cache misses. They have applied this methodology on three different commercial processors. They have validated their approach by comparing them with other current measurement setups.

Brooks et al. (2000) [10]. Wattch is an architectural simulator that estimates CPU power consumption. They have created power models for different hardware structures, and then based on the cycle-level simulation data, outputs power consumption, performance (measured as the number of cycles), energy consumption, and energy-delay-product. They have integrated their power models into SimpleScalar, and extended it to obtain more power-related measurements. They validate their models with three approaches. First, they check if they modeled the capacitance levels correctly, by measuring them in real circuits. Second, they compare their power levels with already published results from industry chips. Third, they compare the maximum power of their models against published works.

One interesting aspect of the paper is that they present three case studies where studying power consumption can be useful. In the first case study they vary some architectural parameters (e.g. cache size) and run several benchmarks to compare the power consumption, performance, and energy-delay product. The second case study evaluates the effect of *loop unrolling* on power consumption. The results show how loop unrolling benefits in terms of power and also in terms of energy-delay product, even though after rolling with a factor of 4, the execution time remains the same. This is a clear example where execution time can be the same, however the energy consumption is significantly different. The third case study looks at the effects of *result memoing*, where the inputs and outputs of long-latency operations are stored and re-used the if the same inputs are encountered again. The results show that there is an average power improvement of 5.4%.

Ye et al. (2000) [80]. The authors present SimplePower, a simulator that estimates the power and energy consumption per cycle. At each clock cycle, SimplePower simulates the execution of the instructions and outputs power consumption values based on their power models and the usage of the functional units. Their power models are based on input transitions [52]. They simulate a subset of instructions from SimpleScalar and create the energy models and switch capacitance values for the datapath, memory, and on-chip buses. In order to estimate the power/energy consumption of an application, SimplePower simulates the execution of the set of instructions, using the power models associated to each functional unit that have been activated by the set of instructions.

SimplePower provides switch capacitance tables for the following units: ALU, adders, multipliers, shifter, controllers, register file, pipeline registers, and multiplexors. It also provides a cache simulator based on a modified version of an existing cache simulator [37] and analytical energy models [67]. Finally, they provide a bus simulator, which combines the activity factors from the simulator and an interconnect power model [81] to create the switch capacitance of the on-chip buses.

Joseph et al. (2001) [44]. The authors present the Castle project, that gives runtime power readings from different processor units. To the best of our knowledge, they provide one of the first approaches to use performance counters to estimate power consumption. The present per-unit power breakdowns based on direct measurements, rather than through simulations. They partially validate their model with measurements from a power meter. The power model is created using Wattch [10] together with SimpleScalar [11]. They correlate PMCs with the most power-relevant counts. However, the authors do not provide which PMCs are the most relevant ones.

Sami et al. (2002) [64]. The authors present an energy model for VLIW (Very Long Instruction word) architectures. *A VLIW processor is a pipelined CPU that can execute, in each clock cycle, a set of explicitly parallel operations [64]* Their model estimates the energy consumption for each instruction, but decomposing it for different active components. It is a pipeline-aware model, since it models the power for each clock cycle, splitting the energy of each instruction into the energy of the pipeline stages. They derive an analytical model to estimate the energy consumption, but we simplify the explanation of the model focusing on the main building blocks. For a complete detailed explanation of the model we ask the readers to refer to the original Ref. [64].

The main idea is to estimate the total energy consumption by summing the energy contributions of each instruction. The energy consumption associated with a specific instruction is the sum of the contributions of each pipeline stage. That is, the average energy consumed per stage when executing the instruction, plus the energy consumed by the connections between the pipeline stages.

The energy consumed per stage is estimated as a linear combination of the average energy consumption of such state during an ideal execution in the absence of any exceptions, plus the energy consumed at that stage due to a miss on the data cache, plus the additive energy consumed at that stage due to a miss in the instruction cache. The energy consumed by the connections depends on V_{dd}, the clock frequency, the capacitance, and the switching activity. They are able to estimate the parameters of each equation using linear regression and measuring the power consumption. To validate their model, they obtain real measurements from the Synopsis Design Power tool and compare it to their model by varying the different set of parameters.

Bellosa et al. (2003) [4]. The authors provide a model to estimate power consumption and temperature on-the-fly using performance counters. To get the real power measurements they instrument the motherboard with four thermal resistors attached between the board and the power supply. They correlate the real power consumption to processor-internal events to obtain energy estimations. To know which events account for what fraction of the energy consumption, they have a set of test programs (benchmarks) with specific type of operations, such as ALU and memory operations. Running the set of benchmarks will activate a set of PMCs, and will give power consumption values. From the executions of running all the benchmarks, they estimate the weight of each PMC by using linear regression. The output is an energy/power model based on PMC information. The reason why they have this set of benchmarks is clear, to have different activations of different PMCs. If they had very similar programs executions, it would not be possible to correlate the PMC information to energy consumption, and it would not generalize to

other type of applications. In this manner, they isolate the power consumption of different type of operations, obtaining more generalizable results. They validate their model with another set of benchmarks and real power consumption measurements.

Gilberto et al. (2005) [27]. The authors present a power estimation model that can acquire fast, low-overhead power estimates. They use PMC information to estimation the power consumption of the CPU and memory. To create the model, they correlate real power consumption to five PMCs, using different benchmarks. The goal is to estimate the power weights, that represents the importance of that PMC in relation to the final energy consumption. They also incorporate idle processor power consumption in the power model. To measure the power consumption of the main memory, their initial approach was to use PMCs that monitor memory accesses. However, the processor of their study did not include that information. Thus, they use PMCs related to instruction cache misses and data dependencies to estimate the power consumption of the main memory. They obtain real power measurements from the processor and memory.

Rajamani et al. (2006) [58]. The authors present an application-aware power management methodology. Their methodology incorporates real-time power and performance models for several DVFS frequency-voltage pairs. This is one of the first works that estimates power for different frequency-voltage pairs, since most of the work that we have already presented either focuses only on one frequency level or creates a model for each frequency. The methodology for application-aware power management consists of three phases: (i) power and performance monitoring with PMCs; (ii) estimation and prediction of power consumption and performance (iii) choosing the most optimal voltage-frequency pair. To create the power model for each frequency-voltage pair, they used four micro-benchmarks and one PMC. They believe that the most important counter is *Decoded Instructions per Cycle*, since it correlates highly to measured power. The power estimation model is constructed as a linear fit of measured DPC, minimizing the absolute-value error between the measured power and estimated power.

For the performance model they use the Instructions retired counter, and, based on a set of benchmarks, they create two sets of equations to differentiate between core-bound and memory-bound workloads. To obtain real power measurements to create the models, they have used some meters using a Radisys system board, specified in another study by the same authors [57]. Based on the monitoring and prediction of power and performance, they also present two power management solutions: Performance Maximizer, which sets the most optimal frequency-voltage pair to maximize performance while staying within set power consumption limits. The second solution is Powersave, that provides energy savings while staying within performance set limits. The paper does not give thorough detail on the validation of their models.

Lee et al. (2006) [49]. The authors present a model to estimate power and performance by using regression analysis applied on an initial set of simulation of different micro-architectural components specifications. They first choose a set of 4000 samples of possible specifications, such as L2 cache size and number of special purpose registers, uniformly at random. They choose a subset of the 22 benchmarks to run their simulations. To derive performance and power models they perform regression analysis on the 4000 samples.

The idea is to check, from the set of predictors, which are most valuable to predict performance and power. Predictors that are observed to vary similarly via clustering are merged together. Predictors that do not correlate well with performance or power are removed. Since they use regression, in the end the model consists of a set of weights for each predictor, summed together, to obtain the power or performance, depending on the model. The analysis made on the power model state that predictors that are significant for performance prediction are probably also significant for power prediction. They validate their models with the power and performance estimates from the simulator, which also validated their estimates with existing power models [9,53].

Economou et al. (2006) [22]. The authors present Mantis, a method to model the power consumption of server systems using PMCs. Their approach consists of two well defined phases, namely, the calibration phase, and the power estimation phase. During the calibration phase, they correlate real AC power consumption values to system performance metrics (PMCs), using diverse benchmarks as input and linear regression. The calibration phase is done offline and is run only once for each system. Based on the information obtained from the calibration phase, the model estimates power consumption by using real-time PMC information. The power consumption is broken down into the following components: CPU, Memory, Hard disk, Network and peripherals. They are able to obtain the power measurements of these units separately by measuring the real power consumption of the different components of the board during the calibration phase. Finally, to validate their model, they compare the results obtained from Mantis against real AC power consumption in more than 30 benchmarks.

Singh et al. (2009) [68]. The authors present a model to estimate power consumption by correlating PMCs to observed power consumption. Based on platform constraints, they restrict the number of studied performance counters to four, namely: *L2_cache_miss*, *retired_uops*, *retired MMX and DF instructions*, *dispatch stalls*. They have chosen these four performance counters as the ones with highest correlations to real measured power. They form the power model by assigning weights to PMCs. They do this by running micro-benchmarks, and applying multiple linear regression to real power measured with the Watts Up Pro power meter.

They have validated their model by testing it and comparing it to real power measurements on three different benchmarks. The main difference between this work and others, is that they only use the performance counters that are available at run-time in a single run, allowing for real-time power estimation. In comparison to Goel et al. (2010) [29], Goel et al. (2010) extend this work by including temperature in the model, exploiting DVFS, and validating the model on several more platforms.

Li et al. (2009) [50]. The authors present McPAT, a power, area, and timing modeling framework. Dynamic power is calculated with analytical models to calculate the capacitance, and statistics from simulation to calculate the activity factors (of how the circuit is being used). They use existing [54] analytical models to calculate the power dissipated from switching the circuits. They use MASTAR [65] and Intel's data [3] to model the leakage power. The model outputs also timing and area results, based on improved versions of existing models. One key improvement in comparison to previous models is that they are able to model power-saving techniques. Their model works such that, based on an input from a simulator, in XML form, which provides activity factors of the different hardware components, outputs power, time, and area values. They validate their results on three different processors, and use the published data to compare the values. Power is validated based on peak power data from the processors that was already published.

Bertran et al. (2010) [5]. The authors present an approach to model power consumption using PMCs. Instead of portraying the overall power consumption of the processor, they break down the power distribution per component (floating point unit, integer unit, branch prediction unit, L1 cache, L2 cache, font side bus, and main memory). More specifically, they present a systematic methodology to produce power models for multi-core architectures, focusing on decomposability (per component), accuracy, and responsiveness of the models. Responsiveness they refer to the ability of the model to capture power variations.

Their methodology consists of four steps. First, they define the power components; second, they define the micro-benchmarks; third, they collect the data to train and validate the model; and fourth, they output the model. The power components are a set of micro-architectural components that are grouped together based on levels of activity. For instance, the whole memory subsystem is divided into three power components: L1 cache, L2 cache, and the Front Side Bus (FSB). They match these power components to PMC activity. They design a total of 97 benchmarks with different set of instructions. With the execution of the benchmarks they are able to get the activity ratios of each power component. With power measurements and the activity ratios for each component, they are able to create the power model using multiple linear regression. They validate their models using empirical measurements of the SPECcpu2006 benchmarks [36]. However, it is unclear how they obtain the real measurements to validate or create their models.

Goel et al. (2010) [29]. The authors present a methodology to produce per-core power models in real time using PMCs and temperature sensor readings. As the previous models presented above, they first find the PMCs that correlate strongly with measured power. They sample those PMCs while running micro-benchmarks and then apply linear regression to give a weight to each PMC. The best set of PMCs is unique for each system. Their methodology has the following steps. First, they identify the PMCs that represent the heavily used parts of a core's micro-architecture. They derive the following four categories: Floating point units, Memory, Stalls, and Instructions Retired. The instructions retired refer to the number of instructions that were completely executed by the CPU.[1] The *Stalls* category measures resource stalls to understand how the out-of-order logic contributes to power usage, since if an instruction stalls, that can contribute to an increase of dynamic power. Second, they rank the PMCs by running a set of micro-benchmarks that cover the categories mentioned in the first step. They use the Watts Up Pro power meter [20] to obtain real power measurements. They rank the PMCs based on the correlation to the real power measured using Spearman's rank correlation [69]. They used *pfmon* [23] to collect the data for each PMC. Finally, using multiple linear regression they create the power model to estimate the power for each core, introducing also the core temperature. To validate their model they run four benchmark suites, real power measurements with Watts Up Pro power meter, and sensors to obtain core temperatures.

Intel RAPL (2010, 2011) [17,31]. Intel's RAPL interface is presented in Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B, Part 2, Chapter 14.9 [31]. It is also referenced in a paper about power estimation in memory [17] using RAPL.

RAPL (Running Average Power Limit) is a driver that allows energy and power consumption readings of the core, uncore, and DRAM. RAPL provides a set of specific PMCs with the energy and power readings. However, the RAPL interface is not well documented since the authors have not published how they model power or energy. There have been several studies that validate RAPL's interface [19,32], and several tools available that make use of such interface to provide energy measurements, such as PAPI (Performance API) [75]. Since it gives accurate results, and it is available on any Intel modern machine, we have used this interface in our use case in Section 7.1.

Spiliopoulos et al. (2012) [70]. The authors present a power and performance estimation tool that is per-phase and per-frequency. Their methodology accounts for the different phases of a program, thus being able to understand in more detail which part of the program is responsible for which amount of energy consumption. They provide a performance model based on analytical DVFS models, and a power model based on capacitance correlations. In particular, they collect information from a specific run of an application for each interval/phase using PMCs. To estimate the performance, they use analytical models that are based on the number of stalls and misses for different frequency values. The total power is estimated as the sum of static and dynamic power. The dynamic power is calculated as the frequency (f) times the supply voltage (V_{dd}) squared times the effective capacitance. The effective capacitance is the capacitance times the activity factor. Since they want their model to be applicable to different frequency values, they do not correlate PMC information to power directly (otherwise they would need a model for each frequency), instead, they correlate the core's effective capacitance (which does not depend on frequency) to PMCs. They do this by first running a set of benchmarks in maximum frequency and measuring processor total power consumption, then subtracting the static power, and finally dividing with $f \cdot V^2$. The static power is measured for all frequencies when the processor is idle. They validate their models using real power measurements.

Shao et al. (2013) [66]. The authors present an instruction-level energy model for the Xeon Phi processor using PMCs and analytical models. The total energy is estimated as the energy per instruction (EPI) multiplied by the instruction counts obtained from the PMCs. They collect the PMC statistics using Intel's VTune tool.[2] To calculate the EPI, they use a set of micro-benchmarks that cover all major instruction types. In particular, for each instruction type, the EPI is calculated as the difference in power between the start and the end of the execution of the specific micro-benchmark, times the number of cycles executed by the micro-benchmark, divided by the frequency; and all that term divided by the number of instructions in the micro-benchmark. The power values are real power measurements using the Xeon Phi Beta Software Development Platform. They characterize the EPIs within different number of cores and threads. They show very interesting patterns, such as that power is double for the micro-benchmark that loads a cache line from memory into the local cache, compared to the micro-benchmark that executes an arithmetic operation incurring in no cache misses. Finally, they validate their model using the SHOC benchmark suite [16] and real measurements from their Xeoh Phi card, with 60 cores and four threads.

Rethinagiri et al. (2014) [59]. The authors propose to build a power estimation tool at the system-level for embedded platforms. They combine the power models for different components, such as, processor, memory and functional units to obtain a system-level estimate of power consumed.

They first characterize the system by extracting a set of generic features or activities that can be applicable to most programs.

[1] https://software.intel.com/en-us/vtune-amplifier-help-instructions-retired-event.

[2] https://software.intel.com/en-us/vtune.

These include architectural features such as frequency of the processor, bus, and number of cores, and application features such as instructions per cycle (IPC), cache miss rate and external memory access rate. These performance counter and power information was collected for real systems by running some assembly-level programs to simulate each component. Power values were collected using Agilent LXI digitalizer to obtain static and dynamic power consumption. Subsequently, a regression-based approach was use to map these features to the power values. Unlike the previous approaches, they focus on building power models for mobile CPU. Second, they develop a cycle-accurate simulator for different ARM cores (ARM Cortex A9, Cortex-A8 and ARM9) using the component models provided in gem5 simulator. They built single and dual core power model for the Cortex-A9. They extract activities such as instruction miss rates, read and write miss rates and IPC for a benchmark application, that is a JPEG decoder application. Power estimates are provided at a fine-grained level for the main tasks in the decoder application. They validate their power estimation approach to state-of-the art tools such as McPAT with a Mult2Sim functional simulator and real power measurements.

Laurenzano et al. (2014) [48]. The authors aim to characterize HPC application benchmarks on several ARM processors in terms of performance, energy, and energy-delay product (EDP). To measure the performance of the HPC kernels in the benchmarks they insert timing instrumentation around main phases of the code such as loops. They measure wall power and attribute this system power to the individual cores analytically. They gather data by running multiple experiments with different core counts and extract the system power based on the number of idle and active cores. They use the empirically gathered data to form a system of equations that can be solved by Gaussian elimination to get the power draw for a single core.

To extract features from the application code they use static binary analysis tools such as EPAX toolkit for the ARM binaries and PEBIL toolkit for x86 binaries to examine the binary and extract information about the machine-level instructions in the program and their relationship with high-level structures in the code, for example, loops. They chose floating point operations, memory accesses number of bytes moved per memory operation, and size of data structures in loops as key features to build a multi-variate regression model for energy consumption. Similar to previous the energy models, they select features that perform well across all benchmarks applications. Hence, these models are useful when comparing different systems at an architectural level to understand changes in the architecture that lead to better energy-use across a range of applications.

Walker et al. (2015) [77]. The authors propose building a power model to create an intelligent run-time management software that can use the information from the power model to apply energy-saving techniques for the applications executing on a mobile CPU. They explore two types of models, one based on PMCs and one based on CPU utilization as features to the model. The main characteristics to build a power estimation model for a run-time management system was chosen on the properties of light-weight, accuracy and responsiveness. They measure the power of the individual CPU as opposed to the whole board using the Agilent N6705 Power analyser. They also measure voltage and current at 10 ms intervals. The PMC information and power data was collected for a variety of benchmark applications including MiBench, video playback, and other workloads to exercise certain architectural features.

Since only four PMCs can be monitored simultaneously for the ARM Cortex-A8 on the BeagleBoardxM, multiple experiments were run to capture every PMC. The PMC data was then correlated with power individually to select candidate features in the final power estimation model. The first feature was the one with the highest correlation factor and was used for building the base model. Other PMCs were then selected with the base model as reference. The main goal was to make the PMC-based model applicable to a variety of workloads hence features were selected based on how well it improved the accuracy across all workloads. The final estimation model was then built with the candidate features across multiple DVFS settings. Since one of the characteristics of the model was responsiveness, the power model was built to predict instantaneous power as opposed to average power of the application. Since PMC information is difficult to obtain on embedded platforms, the authors explore the use of CPU-utilization as a candidate feature for the power model. The power model was built for two types of mobile CPU cores (ARM Cortex A7 and Cortex A15) on an Odroid-XU+E board. The board is equipped with voltage and current sensors that were sampled at 50 ms. They use regression analysis to correlate the average CPU utilization with the power at multiple DVFS settings. They use the MiBench embedded benchmark suite to exercise both processors. The also experiment with varying number of cores.

Goel et al. (2016) [28]. The authors present a methodology to derive static and dynamic power values of individual cores and uncore components. In contrast to previous approaches [29], they use analytical equations together with empirical observations from curated micro-benchmarks to estimate the effective capacitance. Correlating capacitance with PMCs was already done in a previous study from 2012 [70], as we mention in Section 4.1.

They present separate models for the core, uncore, dynamic and static power. They validate their methodology with a set of benchmarks. The methodology uses core and uncore voltage, package temperature, and PMCs to create models that can estimate power consumption for sequential and parallel applications across all system frequencies. Finally, the authors also perform sensitivity analysis of energy consumption at several voltage and frequency levels. They study how energy efficiency relates to DVFS and to the memory intensity. They conclude that memory bound must be considered when choosing an optimal frequency to obtain the most energy efficient setup.

Mazouz et al. (2017) [51]. The authors present a methodology to derive energy models of the hardware components of multi-core processors. They calculate the total energy of the CPU as the sum of the static and dynamic energy. The static energy is calculated by first calculating the static power of the core (ALU, FPU, L1 cache, L2 cache, etc.) and uncore (L3 cache) components. The dynamic energy is calculated as the sum of the energy estimated for each hardware component. The hardware components are the following: FE (front-end, number of instructions issued to the back-end), INT (integer instructions related energy consumption), LD (memory loads L1 cache), ST (memory store L1 cache), FP (floating point instructions related energy consumption), L2 cache, L3 cache. To estimate the energy of each component, the authors run a set of micro-benchmarks to isolate the power of each component. They apply linear regression to obtain the weights for every component. The energy measurements are obtained using PMC information and the interface RAPL. They claim that the methodology is flexible to use any other source of power measurement as input. The interesting part of their model is that they not only provide accurate energy values per-component, but also for each instruction type, giving a nice overview of the energy cost per specific type of operation. They do this by running a set of micro-benchmarks with specific instructions and isolating their power and energy values. This allows for code optimization in terms of energy efficiency. They validate their models by using a different set of benchmarks that run different set of operations.

Table 4
Description of energy measurement and estimation tools.

Tool	Research paper	Language interface	Operating systems	User-friendly	Maturity	Research/commercial	Uses
ARM Streamline	[60]	CLI and GUI	W, L	Yes	2010	Commercial	Power, PMU
Powmon	[78]	CLI	M, W, L	Not tested	2017	Research	Power, PMU
Intel Power Gadget	[31]	CLI and GUI	M, W, L	Yes	2012	Commercial	Energy, Power, DRAM
McPAT	[50]	CLI	L	Yes	2009	Research	Power, architecture
PAPI	[75]	CLI	L	Not tested	2010	Research	Power

Walker et al. (2017) [78]. The authors build a run-time power model for ARM's Cortex A7 and A15 for the purpose of run-time power management and design space exploration for the application. They perform a thorough investigation of PMC event selection using statistical techniques such as hierarchical clustering and R^2 analysis to capture the relationship between power and the PMC data. They also measure collinearity of the PMCs with each other to avoid duplicating information that is used for building the power models.

They use the optimal PMC events from the PMC event selection phase and use correlation techniques to relate these events to the clock frequency and CPU Voltage. They model the power at the cluster-level which includes multiple cores and their caches. They also built models for varying number of cores and different core-affinities. Their benchmarks include 60 workloads curated from different sources such as MiBench, LMBench and others, to exercise the CPU, memory and the I/O.

5. Power and performance monitoring tools

This section provides an overview of available tools that can facilitate building power models. (See Table 4.) The tools are characterized on the following criteria:

- Language interface of the tool: How can the tool be accessed and used? Has the developer of the tool provided a set of Command-Line Interface (CLI) options or a Graphical User Interface (GUI)?
- Research Paper: The research paper where the tool is presented.
- Operating systems supported: Is there support for different types of operating systems. MacOS (M), Linux (L) and Windows (W).
- User-friendly: This will depend on the language interface provided and the availability of documentation of the tool.
- Maturity: The duration since the establishment of the tool and the technical support for the tool.
- Research or commercial: Is the tool the result of a research endeavor or a commercial product?

ARM Streamline Performance Analyser can be used to monitor the power profile and performance counter for mobile CPUs based on the ARM architecture. The tool provides both graphical and command line interfaces to obtain real power values on the target device but has to be interfaced with the necessary power measuring equipment such as an ARM energy probe or the power sensors on-board. Moreover, the tool does not provide energy estimation models based on the data it collects. A recent effort in this direction was made by SyNERGY [61] that leveraged the tool for building energy estimation models.[3] Finally, the reported overhead of using ARM Streamline with gator daemon (the latter runs on the target device) is within 0.5 to 3% error.

Intel Power Gadget uses the Intel RAPL interface [31] to provide power and energy estimations of the core and uncore of the processor, together with the DRAM. It has both a GUI and a command line tool, and they provide an API to extract information from sections of code. The API is limited to C/C++ code. The command line tool can be used to obtain real time energy values during the execution of a specific script or command. The script can contain runs of any programming language. However, the energy values correspond to the energy of the whole processor, not only of the energy responsible for that particular application. Thus, our recommendation is to make energy estimations while no other application is running in the background, and comparing several executions under the same setups. Reported errors claim that RAPL gives results within 2.3% of actual measurements for the DRAM; and that RAPL slightly underestimates the power for some workloads [19].

McPAT can be used together with Sniper [13] to simulate the execution of a C application. McPAT outputs power and energy consumption values separately for the different components: FP unit, L2 cache, etc. C code can be instrumented to obtain power measurements at the functional level. McPAT gives more fine granular information compared to Intel Power Gadget, giving energy estimations at the application, hardware, and functional level. However, only tasks that require a low number of instructions can be executed, since it introduces a significant overhead. For instance, machine learning algorithmic runs can be executed as long as the datasets are small ($\approx$100k instances, 50 attributes). McPAT reported errors range between 10 and 20 percent depending on the processor.

Powmon is an experimental software that can be used to obtain power and performance counter information on mobile CPUs.[4] PAPI is an interface that is widely used in the community and provides an API to access performance counter information and also the specific RAPL interface registers to estimate energy and power consumption.

6. Energy estimation in machine learning

Estimation of energy consumption can be useful for machine learning experts for several reasons, as covered in Sections 3 and 4.5. In this section, we present an overview of the current research in machine learning regarding energy and power estimation with emphasis on the progress made in deep learning.

Machine learning models such as deep neural networks are characterized by parameters or weights that are used to transform input data into features. These models consist of two distinct phases of computation: the training phase and the inference phase. In the training phase, the deep neural network is designed, the number of layers, the size and type of each layer are selected, and weight parameters are learned. During the inference, the fixed weight parameters are tested on example input data to extract features from the input data. Current approaches for training these models mainly rely on desktop or server systems (including high-end GPUs, CPUs or FPGAs) to support the training of deeper models on large data sets. Meanwhile, the inference phase is typically performed on low-end embedded systems, for example, smart phones, wearables and others. However, a large

[3] An initial version of the framework is available here: https://github.com/Crefeda/SyNERGY.

[4] Please refer to: http://www.powmon.ecs.soton.ac.uk/powermodeling/.

Table 5
Areas where energy estimation has been applied in deep learning.

Paper	Training	Inference	CPU	GPU
[34]		×		×
[79]		×		
[61]		×	×	
[12]		×		×
[15,62,71],	×			×

body of research has emerged to optimize the energy-efficiency of these machine learning models and are driven by early energy modeling approaches applied in machine learning field [79].

One of the first models to estimate the energy of a neural network model was to count the number of multiply-accumulate (MAC) to model the number of floating-point operations for the CPU or the GPU and the number of weights to model the number of main memory accesses, of a pre-trained model, as proxies for energy [34]. Since the weights have to be loaded from DRAM that have high relative energy costs compared to a MAC operation, a large number of optimization efforts such as pruning [34], compression [33] and compact models [42], focused on reducing the number of weights or parameters of the neural network models.

Using the number of weights was deemed too simplistic and further attempts were made to improve the energy modeling approaches by incorporating an energy cost for different types data in different levels in the memory hierarchy. The authors in [79] built an energy estimation model by counting the number of times each data value is reused across the memory hierarchy using an optimization procedure. There are four-levels of memory hierarchy considered: DRAM, global buffer, array and register file, and the energy-cost for each level is extracted from a 65 nm process. This energy estimation model is used to map an arbitrary shaped neural network model onto their application-specific accelerators such as Eyeriss. While the authors provide an energy estimation model to map the neural network model to their specific architecture design, details of their optimization procedure are not provided [14]. This methodology was later extended to prune weights during the training phase that account for higher energy-consumption [79].

Modeling the power at the application-level using performance counters for general-purpose processors have also begun to emerge. SyNERGY [61] presents a methodology to produce application-level energy measurements using models built from performance counters. The application tested is a simple inference based on convolutional neural networks on a single core of an ARM A57 CPU. They model the energy consumption of the convolutional layers at the system-level that incorporates the power used by the processor, memory and other peripherals. The performance counters chosen for the study are the number of SIMD instructions executed and the number of main memory accesses. Since the convolutions are commonly restructured into matrix–matrix multiplications the assumption is that the two most important performance counter are the ones that contribute to computation and data movement. The instantaneous power is measured at the SoC level with the interactive Linux governor for power management and no change to the DVFS settings. These instantaneous power values are then converted to corresponding energy values. The authors initially built a simple multi-variable linear regression model to correlate the performance counter information with corresponding energy profile of the application. Finally, the authors extend the model to predict the number of SIMD instruction and bus accesses to the applications MAC count. This was done to relate the application-level features directly to the energy profile.

NeuralPower [12] applies a regression based approach to model the power consumption and the run-time for a desktop GPU. These models were built for the three main layers of a convolutional neural network, i.e., convolutional, pooling and fully-connected layers. They use real power and timing values to build predictive models for average power and runtime. The power values are obtained on an NVidia GTX1070 GPU using *nvidia-smi*. The authors use application-level features, such as, kernel size, number of layers and others as input for the power estimation model. They exclude the impact of voltage and frequency scaling by keeping the GPU in a fixed state. The output of the power predictive model and the runtime model are combined to give an estimate of energy-use per layer. Each of the per-layer estimates are subsequently added to get an estimate of the power consumption of the overall convolutional neural network. However, the authors do not provide the final prediction model to the reader and have a limited number of ConvNet models in the study.

DeLight [62] models the energy consumption of a simple feed-forward neural network during its training phase. The authors model the energy-use in terms of basic arithmetic operations and communication of shared weights in a distributed training setting.[5] Two types of operation are modeled; multiply-add, which is a function of the number of connections between neurons of two adjacent layers, and the activation function, which is a scalar multiplication. The communication energy is a function of the number of shared weights. The coefficients for modeling these operations are obtained by running a micro-benchmarks on the CUDA cores of the Nvidia TK1 embedded platform.

Table 5 summarizes the areas in which energy estimation has been applied to deep learning. Most studies use energy estimation techniques during the inference phase and apply post-processing techniques to reduce the energy consumption of the neural network. While the general rule of thumb followed is to keep the count of the number of parameters as low to reduce the overhead in performance, very few studies have emerged to explicitly incorporate energy-use *a priori* during the training phase [62]. Approaches to automate the process of designing energy-aware neural network models use techniques such as reinforcement learning [74], Bayesian optimization [71] and genetic algorithms [15]. Furthermore, deep learning models are generally trained on desktop GPUs and energy estimation models covering GPUs will become necessary.

7. Use cases

This section demonstrates two uses cases that estimate the energy at the application-level based on the classification in Section 3 and the technique based on performance counter information described in Section 4, for estimating the energy consumption in data mining and machine learning scenarios.

7.1. Data stream mining

Data stream mining algorithms build machine learning models online, as the data arrives, by reading the data only once. One of the main principles is that the data should not be stored, just the

[5] Sharing the weight parameter space among different cores.

Table 6
Energy and accuracy of the VFDT and the HAT in concept drift and non-concept drift datasets with 5 million instances.

			Energy (J)		
Alg	Dataset	Acc (%)	Total	Processor	DRAM
HAT	RandomRBF (0.001)	**65.75**	578.03	558.34	19.69
HAT	RandomTree	**97.75**	758.80	728.32	30.49
VFDT	RandomRBF (0.001)	56.45	**516.92**	496.43	20.49
VFDT	RandomTree	97.40	**363.30**	348.60	14.70

Table 7
Measured versus predicted energy consumption of the Convolutional layers of three example ConvNets on Cortex-A57.

ConvNet	Measured (mJ)	Predicted (mJ)	Relative Accuracy (%)
Inception-V3	10945.73	8073.55	73.70
MobileNet	2453.19	1546.80	63.05
DenseNet	8777.78	6148.16	70.00

necessary statistics of the data. A typical scenario is a sensor or an IoT (Internet of Things) network producing a potentially infinite stream of data and the data stream mining algorithm building the model in real time. The state-of-the-art algorithms in this area are able to handle changes in the input data distribution, known as concept drift [41].

The goal of this use case is to compare the energy consumption of two stream mining algorithms in two scenarios with and without concept drift. The first algorithm, the Very Fast Decision Tree (VFDT) [21], is able to build and update a decision tree in real time, but is not able to handle concept drift. The Hoeffding Adaptive Tree (HAT) [6] extends the VFDT to handle concept drift by changing the structure of the model to better approximate the new data. This use case investigates the extra energy cost needed to handle concept drift.

We have chosen to estimate the energy consumption using Intel Power Gadget, described in Section 5. This tool provides real-time estimations with almost no overhead. The datasets have been synthetically generated with MOA [7] (Massive Online Analysis) and the algorithms were run in the same platform. We use prequential evaluation, where instances are tested and then trained, giving online accuracy measurements. The chosen datasets are the random tree (no concept drift), and random RBF (Radial Basis Function, with concept drift, 0.001 speed change), both with 5 million instances.[6]

The results of running both algorithms 10 times and averaging the results are shown in Table 6. We can observe that HAT obtains higher accuracy for both datasets. For the concept drift dataset the HAT algorithm obtains 16% higher accuracy than the VFDT, at the cost of 1.12x more joules (11.8% higher energy consumption). On the other hand, for the random tree dataset, both algorithms obtain similar levels of accuracy (less than 0.5% difference) but the VFDT consumes significantly less energy compared to HAT. This is expected, since the extra computations performed by HAT to handle concept drift are not needed in this dataset.

We can conclude that to obtain higher accurate models for concept drift scenarios comes at a cost of energy consumption. However, by doing an initial analysis of the energy consumed by the algorithm, the researcher or machine learning expert can make the adequate choices depending on the set of constraints. For instance, in a scenario with embedded devices where the battery is the main constraint, the researcher might choose to sacrifice that 16% of accuracy to extend the battery's life. Another option is to give the different alternatives to the user, providing them with the necessary information regarding energy efficiency and accuracy.

7.2. Convolutional neural network inference

An inference phase of a convolutional neural network (or ConvNet model) consists of passing an image through a pre-trained ConvNet. It undergoes a series of transformations of the input data into features that can be used to build image classifiers, object detectors and other computer vision applications. These transformations are arranged into layers such as Convolutional, pooling, normalization, fully-connected layers and others. Typically, these inferences are performed on mobile and embedded systems that have limited battery-life [47].

Our goals are (i) to use an existing methodology to acquire per-layer energy of an example ConvNet, (ii) apply a regression-based approach to performance counter information to predict convolutional layer energy, (iii) validate with real energy measurements. We use the methodology proposed in SyNERGY [61], to find the functional-level or per-layer sections of the application by inserting code annotations in the Caffe C++ code. The code annotation library is provided by the ARM Streamline tool which is a pre-requisite to allow such fine-grained instrumentation of the code. The power is captured using the power sensor on the Jetson TX1 that provides system-level power. Similarly, performance counter information is captured using the ARM Streamline tool.

We use the regression model in [61], with coefficients for the number of SIMD instructions and number of bus accesses to be $x_1 = 3.34E-05$ and $x_2 = 3.18E-06$ respectively. We test this model on three example pre-trained ConvNet models, Inception-v3 [72], MobileNet [39] and DenseNet [40], as given in Table 7. We show both the predicted energy and actual measured energy for a single-threaded execution of ConvNet on an ARM Cortex-A57 from the Jetson TX1 board. We obtain an average relative accuracy[7] of 68.91% across the three test ConvNets. The predicted and measured energy consumption for the convolutional layers of MobileNet is lowest and they account for 95% of the computation within a deep convolutional neural network [47]. Therefore, given a choice between the three ConvNets that MobileNet is the most energy-efficient ConvNet choice under similar execution environments.

8. Conclusions

Machine learning algorithms consume significant amounts of energy. However, the lack of evaluations based on energy consumption of these algorithms can be attributed to the lack of appropriate tools to measure and build power models in existing machine learning suites, and because estimating energy consumption is a challenging task.

[6] More information about the datasets is available here: https://moa.cms.waikato.ac.nz/details/classification/streams/.

[7] The absolute difference between actual and estimated energy over the actual energy.

This paper addresses that challenge by presenting a review of the key approaches to estimate energy consumption from the computer architecture field, mapped to machine learning applications. We also describe the state-of-the-art methods to estimate energy consumption in particular for data mining and convolutional neural networks. Our synthesis of the surveyed papers provides the necessary guidelines to expose energy consumption methods to machine learning audiences interested in incorporating energy as metric in the design of machine learning systems. To demonstrate the usefulness of the synthesis, we present two use cases, which show, from the data mining and neural networks perspectives, how to apply the different estimation approaches. We show that the benefits of further research in energy estimations can help machine learning researchers gain significant insights when building machine learning systems.

Our survey also reveals the current state of energy estimations in machine learning. In particular, there are several works emerging to enable energy evaluations in machine learning either through energy prediction modeling as seen in NeuralPower [12] or by direct integrating power monitoring tools to existing machine learning suites as seen in SyNERGY [61]. However, current modeling approaches face another challenge; which is the rapid changes in neural network designs, implementations and hardware. Moreover, there is a fragmentation of the machine learning software ecosystem with multiple software suites and no common benchmarking suites. These modeling approaches will have to be adaptable to these changes and also be comprehensive. This is because most works target a few computational intensive layers with majority of the work confined to just convolutional neural networks.

Another area in which energy estimation is lacking are GPUs that are extensively used by machine learning researchers to train machine learning models. Few works such as NeuralPower [12] build desktop GPU-based energy estimation models while others target mobile CPUs and GPUs [61]. Future work will require further research to integrate the literature on GPU-based estimation methods applied for machine learning scenarios. Finally, there is a recent upsurge to build application specific hardware for machine learning [45]. To keep pace with both advancements in neural network designs and advances in the hardware community will require energy estimation models for these new emerging architectures either gathered on real systems or through neural network hardware simulators such as ScaleSIM [63]. As future work, we aim to investigate energy estimation approaches for GPUs and emerging application-specific hardware.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jpdc.2019.07.007.

Acknowledgments

Eva García-Martín and Håkan Grahn work under the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation (grant: 20140032) in Sweden. Crefeda Faviola Rodrigues and Graham Riley are funded under the European FP7-INFRASTRUCTURES-2012-1 call (grant: 312979) and part-funded by ARM Ltd., UK under a Ph.D. Studentship Agreement.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, Software available from tensorflow.org, URL http://tensorflow.org/.

[2] R.W. Ahmad, A. Gani, S.H.A. Hamid, F. Xia, M. Shiraz, A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues, J. Netw. Comput. Appl. 58 (2015) 42–59.

[3] C. Auth, A. Cappellani, J.-S. Chun, A. Dalis, A. Davis, T. Ghani, G. Glass, T. Glassman, M. Harper, M. Hattendorf, et al., 45 nm high-k+ metal gate strain-enhanced transistors, in: VLSI Technology, 2008 Symposium on, IEEE, 2008, pp. 128–129.

[4] F. Bellosa, A. Weissel, M. Waitz, S. Kellner, Event-driven energy accounting for dynamic thermal management, in: Proceedings of the Workshop on Compilers and Operating Systems for Low Power, COLP'03, vol. 22, 2003.

[5] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade, Decomposable and responsive power models for multicore processors using performance counters, in: Proceedings of the 24th ACM International Conference on Supercomputing, ACM, 2010, pp. 147–158.

[6] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: Int'L Symposium on Intelligent Data Analysis, Springer, 2009, pp. 249–260.

[7] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604, URL http://portal.acm.org/citation.cfm?id=1859903.

[8] R.A. Bridges, N. Imam, T.M. Mintz, Understanding GPU power: A survey of profiling, modeling, and simulation methods, ACM Comput. Surv. 49 (3) (2016) 41.

[9] D. Brooks, P. Bose, V. Srinivasan, M.K. Gschwind, P.G. Emma, M.G. Rosenfield, New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors, IBM J. Res. Dev. 47 (5.6) (2003) 653–670.

[10] D. Brooks, V. Tiwari, M. Martonosi, Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, Vol. 28, ACM, 2000.

[11] D. Burger, T.M. Austin, The simplescalar tool set, version 2.0, ACM SIGARCH Comput. Archit. News 25 (3) (1997) 13–25.

[12] E. Cai, D.-C. Juan, D. Stamoulis, D. Marculescu, NeuralPower: Predict and deploy energy-efficient convolutional neural networks, in: Asian Conference on Machine Learning, 2017, pp. 622–637.

[13] T.E. Carlson, W. Heirman, S. Eyerman, I. Hur, L. Eeckhout, An evaluation of high-level mechanistic core models, ACM Trans. Archit. Code Optim. (ISSN: 1544-3566) (2014) http://dx.doi.org/10.1145/2629677.

[14] Y.-H. Chen, T. Krishna, J.S. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, IEEE J. Solid-State Circuits 52 (1) (2017) 127–138.

[15] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, et al., ChamNet: Towards efficient network design through platform-aware model adaptation, arXiv preprint arXiv:1812.08934, 2018.

[16] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter, The scalable heterogeneous computing (SHOC) benchmark suite, in: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010, pp. 63–74.

[17] H. David, E. Gorbatov, U.R. Hanebutte, R. Khanna, C. Le, Rapl: memory power estimation and capping, in: Low-Power Electronics and Design, ISLPED, 2010 ACM/IEEE International Symposium on, IEEE, 2010, pp. 189–194.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

[19] S. Desrochers, C. Paradis, V.M. Weaver, A validation of dram rapl power measurements, in: Proceedings of the Second International Symposium on Memory Systems, ACM, 2016, pp. 455–470.

[20] E.E. Devices, Watts up PRO, 2009.

[21] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proc. 6th SIGKDD Int'L Conf. on Knowledge Discovery and Data Mining, 2000, pp. 71–80.

[22] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan, Full-system power analysis and modeling for server environments, in: International Symposium on Computer Architecture, IEEE, 2006.

[23] S. Eranian, Perfmon2: a flexible performance monitoring interface for Linux, in: Proc. of the 2006 Ottawa Linux Symposium, 2006, pp. 269–288.

[24] E. Garcia-Martin, N. Lavesson, H. Grahn, Identification of energy hotspots: A case study of the very fast decision tree, in: International Conference on Green, Pervasive, and Cloud Computing, Springer, 2017, pp. 267–281.

[25] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, V. Boeva, How to measure energy consumption in machine learning algorithms, in: C. Alzate, A. Monreale, H. Assem, A. Bifet, T.S. Buda, B. Caglayan, B. Drury, E. García-Martín, R. Gavaldà, I. Koprinska, S. Kramer, N. Lavesson, M. Madden, I. Molloy, M.-I. Nicolae, M. Sinn (Eds.), ECML PKDD 2018 Workshops, Springer International Publishing, Cham, 2019, pp. 243–255.

[26] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, A.C. Berg, Low-power image recognition challenge, in: Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, IEEE, 2017, pp. 99–104.

[27] C. Gilberto, M. Margaret, Power prediction for intel xscale processors using performance monitoring unit events power prediction for intel xscale processors using performance monitoring unit events, in: ISLPED, vol. 5, 2005, pp. 8–10.

[28] B. Goel, S.A. McKee, A Methodology for modeling dynamic and static power consumption for multicore processors, in: IPDPS, 2016, pp. 273–282.

[29] B. Goel, S.A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, M. Cesati, Portable, scalable, per-core power estimation for intelligent resource management, in: Green Computing Conference, 2010 International, IEEE, 2010, pp. 135–146.

[30] B. Goel, S.A. McKee, M. Själander, Techniques to measure, model, and manage power, in: Advances in Computers, Vol. 87, Elsevier, 2012, pp. 7–54.

[31] P. Guide, Intel® 64 and ia-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2 (14.9), Intel, 2011, pp. 14–31–14–40.

[32] M. Hähnel, B. Döbel, M. Völp, H. Härtig, Measuring energy consumption for short code paths using RAPL, ACM SIGMETRICS Perform. Eval. Rev. 40 (3) (2012) 13–17.

[33] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149, 2015.

[34] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[35] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2011.

[36] J.L. Henning, SPEC CPU2006 benchmark descriptions, ACM SIGARCH Comput. Archit. News 34 (4) (2006) 1–17.

[37] M.D. Hill, J.R. Larus, A.R. Lebeck, M. Talluri, D.A. Wood, Wisconsin architectural research tool set, ACM SIGARCH Comput. Archit. News 21 (4) (1993) 8–10.

[38] M.A. Hoque, M. Siekkinen, K.N. Khan, Y. Xiao, S. Tarkoma, Modeling, profiling, and debugging the energy consumption of mobile devices, ACM Comput. Surv. 48 (3) (2016) 39.

[39] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861, 2017.

[40] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely Connected Convolutional Networks.

[41] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 97–106.

[42] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50× fewer parameters and <0.5 MB model size, arXiv preprint arXiv:1602.07360, 2016.

[43] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093, 2014.

[44] 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, ACM, 2001, pp. 135–140.

[45] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, ACM, 2017, pp. 1–12.

[46] S. Kaxiras, M. Martonosi, Computer architecture techniques for power-efficiency, Synth. Lect. Comput. Archit. 3 (1) (2008) 1–207.

[47] N.D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, F. Kawsar, An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices, in: Proceedings of the 2015 International Workshop on Internet of Things Towards Applications, ACM, 2015, pp. 7–12.

[48] M.A. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W.A. Ward, R. Campbell, L. Carrington, Characterizing the performance-energy tradeoff of small ARM cores in HPC computation, in: European Conference on Parallel Processing, Springer, 2014, pp. 124–137.

[49] B.C. Lee, D.M. Brooks, Accurate and efficient regression modeling for microarchitectural performance and power prediction, in: ACM SIGOPS Operating Systems Review, Vol. 40, ACM, 2006, pp. 185–194.

[50] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on, IEEE, 2009, pp. 469–480.

[51] 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, ACM, 2017, pp. 15–26.

[52] H. Mehta, R.M. Owens, M.J. Irwin, Energy characterization based on clustering, in: Proceedings of the 33rd Annual Design Automation Conference, ACM, 1996, pp. 702–707.

[53] M. Moudgill, J.-D. Wellman, J.H. Moreno, Environment for PowerPC microarchitecture exploration, IEEE Micro 19 (3) (1999) 15–25.

[54] K. Nose, T. Sakurai, Analysis and future trend of short-circuit power, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 19 (9) (2000) 1023–1030.

[55] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, R. Sakellariou, A survey of power and energy predictive models in HPC systems and applications, ACM Comput. Surv. 50 (3) (2017) 37.

[56] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic Differentiation in PyTorch, 2017.

[57] K. Rajamani, H. Hanson, J.C. Rubio, S. Ghiasi, F.L. Rawson, Online Power and Performance Estimation for Dynamic Power Management, RC-24007, Tech. Rep, IBM, 2006.

[58] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, F. Rawson, Application-aware power management, in: Workload Characterization, 2006 IEEE International Symposium on, IEEE, 2006, pp. 39–48.

[59] S.K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, A.C. Kestelman, System-level power estimation tool for embedded processor based platforms, in: Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, ACM, 2014, p. 5.

[60] C.F. Rodrigues, G. Riley, M. Luján, Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1, in: Workload Characterization (IISWC), 2017 IEEE International Symposium on, IEEE, 2017, pp. 114–115.

[61] C.F. Rodrigues, G. Riley, M. Luján, Synergy: An energy measurement and prediction framework for convolutional neural networks on Jetson TX1, in: International Conference on Parallel and Distributed Processing Techniques and Applications, CSREA Press, 2018, pp. 375–382.

[62] B.D. Rouhani, A. Mirhoseini, F. Koushanfar, Delight: Adding energy dimension to deep neural networks, in: Proceedings of the 2016 International Symposium on Low Power Electronics and Design, ACM, 2016, pp. 112–117.

[63] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, T. Krishna, Scale-sim: Systolic CNN accelerator, arXiv preprint arXiv:1811.02883, 2018.

[64] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, An instruction-level energy model for embedded VLIW architectures, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 21 (9) (2002) 998–1010.

[65] Semiconductor Industries Association, Model for Assessment of CMOS Technologies and Roadmaps (MASTAR), 2005.

[66] Y.S. Shao, D. Brooks, Energy characterization and instruction-level energy model of Intel's Xeon Phi processor, in: Proceedings of the 2013 International Symposium on Low Power Electronics and Design, IEEE Press, 2013, pp. 389–394.

[67] W.-T. Shiue, C. Chakrabarti, Memory exploration for low power, embedded systems, in: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, ACM, 1999, pp. 140–145.

[68] K. Singh, M. Bhadauria, S.A. McKee, Real time power estimation and thread scheduling via performance counters, ACM SIGARCH Comput. Archit. News 37 (2) (2009) 46–55.

[69] C. Spearman, The proof and measurement of association between two things, Am. J. Psychol. 15 (1) (1904) 72–101.

[70] V. Spiliopoulos, A. Sembrant, S. Kaxiras, Power-sleuth: A tool for investigating your program's power behavior, in: Modeling, Analysis & Simulation of Computer and Telecommunication Systems, MASCOTS, 2012 IEEE 20th International Symposium on, IEEE, 2012, pp. 241–250.

[71] D. Stamoulis, E. Cai, D.-C. Juan, D. Marculescu, Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks, in: 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2018, pp. 19–24.
[72] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[73] C. Talarico, J.W. Rozenblit, V. Malhotra, A. Stritter, A new framework for power estimation of embedded systems, Computer (2) (2005) 71–78.
[74] M. Tan, B. Chen, R. Pang, V. Vasudevan, Q.V. Le, Mnasnet: Platform-aware neural architecture search for mobile, arXiv preprint arXiv:1807.11626, 2018.
[75] D. Terpstra, H. Jagode, H. You, J. Dongarra, Collecting performance data with PAPI-C, in: Tools for High Performance Computing 2009, Springer, 2010, pp. 157–173.
[76] V. Tiwari, S. Malik, A. Wolfe, M.T.-C. Lee, Instruction level power analysis and optimization of software, in: Technologies for Wireless Computing, Springer, 1996, pp. 139–154.
[77] M.J. Walker, A.K. Das, G.V. Merrett, B. Hashimi, Run-Time Power Estimation for Mobile and Embedded Asymmetric Multi-Core CPUs, 2015.
[78] M.J. Walker, S. Diestelhorst, A. Hansson, A.K. Das, S. Yang, B.M. Al-Hashimi, G.V. Merrett, Accurate and stable run-time power modeling for mobile and embedded cpus, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (1) (2017) 106–119.
[79] T.-J. Yang, Y.-H. Chen, V. Sze, Designing energy-efficient convolutional neural networks using energy-aware pruning, in: Computer Vision and Pattern Recognition, CVPR, 2017 IEEE Conference on, IEEE, 2017, pp. 6071–6079.
[80] W. Ye, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, The design and use of simplepower: a cycle-accurate energy estimation tool, in: Proceedings of the 37th Annual Design Automation Conference, ACM, 2000, pp. 340–345.
[81] Y. Zhang, R.Y. Chen, W. Ye, M.J. Irwin, System level interconnect power modeling, in: ASIC Conference 1998. Proceedings. Eleventh Annual IEEE International, IEEE, 1998, pp. 289–293.

Eva Garcia-Martin is a Ph.D. student in Machine Learning at Blekinge Institute of Technology, in Sweden. She is working under the project Scalable resource-efficient systems for big data analytics funded by the Knowledge Foundation, advised by Niklas Lavesson and Håkan Grahn. The main focus of her thesis is on making machine learning algorithms more energy efficient. In particular, she has studied the energy consumption patterns of streaming algorithms, and then proposed new algorithm extensions that reduce their energy consumption. Personal website: https://egarciamartin.github.io/.

Crefeda Faviola Rodrigues is a Ph.D. student in Advanced Processor Technology (APT) group at The University of Manchester and she is supervised by Mr. Graham Riley and Dr. Mikel Lujan. Her research is part funded by ARM and IS-ENES2 Project. Her research topic is "Efficient execution of Convolutional Neural Networks on low power heterogeneous systems". The main focus of her thesis is to enable energy efficiency in deep learning algorithms such as Convolutional Neural Networks or ConvNets on embedded platforms like the Jetson TX1 and Snapdragon 820. Personal website: https://personalpages.manchester.ac.uk/staff/crefeda.rodrigues/.

Graham Riley is a Lecturer in the School of Computer Science at the University of Manchester and hold a part-time position in the Scientific Computing Department (SCD) at STFC, Daresbury. His research is application-driven and much of his research has been undertaken in collaboration with computational scientists in application areas such as Earth System Modeling (including the U.K. Met Office) and, previously, computational chemistry and biology. His aim is to apply his experience in high performance computing and software engineering for (principally) scientific computing to new application domains. He is also interested in techniques and tools to support flexible coupled modeling in scientific computing and in performance modeling techniques for large-scale heterogeneous HPC systems, where energy efficiency is increasingly key. Personal website: http://www.manchester.ac.uk/research/graham.riley/.

Håkan Grahn is professor of computer engineering since 2007. He received a M.Sc. degree in Computer Science and Engineering in 1990 and a Ph.D. degree in Computer Engineering in 1995, both from Lund University. His main interests are computer architecture, multicore systems, GPU computing, parallel programming, image processing, and machine learning/data mining. He has published more than 100 papers on these subjects. During 1999–2002 he was head of department for the Dept. of software engineering and computer science, and during 2011–2013, he was Dean of research at Blekinge Institute of Technology. Currently he is project leader for BigData@BTH – "Scalable resource-efficient systems for big data analytics", a research profile funded by the Knowledge foundation during 2014–2020. Personal website: https://www.bth.se/eng/staff/hakan-grahn-hgr/.