Automated Power Modeling of Computing Devices: Implementation and Use Case for Raspberry Pis

Houssam Kanso, Adel Noureddine, Ernesto Exposito

▶ To cite this version:

Houssam Kanso, Adel Noureddine, Ernesto Exposito. Automated Power Modeling of Computing Devices: Implementation and Use Case for Raspberry Pis. Sustainable Computing : Informatics and Systems, 2023, 37, 10.1016/j.suscom.2022.100837 . hal-03912723

HAL Id: hal-03912723
https://hal.science/hal-03912723

Submitted on 9 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Automated Power Modeling of Computing Devices: Implementation and Use Case for Raspberry Pis

Houssam Kanso[a,*], Adel Noureddine[a] and Ernesto Exposito[a]

[a] *Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France*

ARTICLE INFO

Keywords:
Power Consumption
Performance
Measurement
Empirical Experimentation
Automated Software Architecture

ABSTRACT

Monitoring the power consumption of smart and connected devices is a challenging task with heterogeneous devices and a variety of hardware and software configurations. In order to accurately monitor these devices and follow the speedy changes in their configuration, a new approach is needed. In this paper, we present an automated architecture and approach to empirically generate power models for a large set of devices. Our approach allows conducting automated benchmarks to collect power data and metrics, generating or updating accurate power models, and allowing software tools to query and retrieve the most accurate and up-to-date power model of a specific device configuration. We also present a proof-of-concept implementation for modeling the power consumption of Raspberry Pi devices. Finally, we conduct a comprehensive experiment, modeling the entire current lineup of Raspberry Pi devices with error margins as low as 0.3%, and then we discuss the impact of multiple device configurations on power consumption.

1. Introduction and Challenges

With the explosion of smart and connected devices, there is a need to monitor and optimize the power consumption of these devices. Around 2008, it was estimated that the world had more connected devices than people, and by 2030 it is expected that 500 billion devices will be connected to the Internet [12]. The impact of all these devices on ICT power consumption and carbon footprint is undeniably rising. Recent estimations expect that ICT will account for as much as 14% of the total worldwide carbon footprint [3].

These connected devices have different CPU architecture than current and legacy computing devices (*i.e.*, ARM/RISC compared to x86/CISC architecture), and have a huge variety of hardware and software configuration and components. External power meters can provide accurate power measurements for specific workloads and environments (such as in [1]). However, these meters are costly financially, scale badly for a large park of devices, have a time-consuming setup, and require physical access to each device. Therefore, it is important to provide software-based power models for these devices. However, without embedded power sensors or constructors' power models and API, it is challenging to provide accurate power models for this variety of device configurations. In addition, monitoring the power consumption at run-time (in addition to other performance metrics) helps software developers to detect misbehaving software, or specific power drains due to a particular hardware configuration.

Current power estimation techniques are either based on mathematical formulas (such as in [22]), or on a static data set used to generate an empirical model (such as in [24]). In addition, such models target a single device release with no way to estimate other revisions or variations of the device without manually conducting the experiments again.

Our main motivation is to provide an automated approach to model the power consumption of various devices, while allowing the models to be updated, extended, improved, and shared. We argue that such an approach leads to democratizing power comprehension of hardware and software in different devices and environments.

Providing an accurate and automated approach to solve these questions is challenging, and in particular:

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

In this paper, we present an automated approach and architecture to empirically generate power models for, potentially, unlimited devices and configurations. Our approach provides always up-to-date and accurate power models with low error rates. The approach follows a crowd-sourcing architecture where benchmarking components can run on any device, generate empirical data, and lastly, our power model generator component will generate an accurate power model

*Corresponding author
Email addresses: houssam.kanso@univ-pau.fr (H. Kanso); adel.noureddine@univ-pau.fr (A. Noureddine); ernesto.exposito@univ-pau.fr (E. Exposito)
ORCID(s): 0000-0002-4811-191X (H. Kanso); 0000-0002-8585-574X (A. Noureddine)

for the specific device, or improve the model if a previous one already exists. Monitoring software can connect to our architecture to query and retrieve the most accurate and up-to-date power model for their devices. The data collection and model generation phases are relatively fast as they only take a few minutes. Once the model is generated, power consumption can be estimated with negligible overhead in real-time. With time, the accuracy of the model can be improved as more data are collected and fed to generate more accurate regression models.

The main novelties of this work can be summarized as follows: (i) energy estimation models are always up-to-date due to the continuous data benchmarking as new models are automatically generated when necessary, (ii) model generation is done in an automated manner from the data collection to the testing (with human intervention needed only to start the benchmarking process, and (iii) our proposed approach is a collaborative one where benchmarks can also be crowd-sourced, and energy estimation models are shared across users and devices.

We provide a proof-of-concept implementation for automated power modeling of the entire current set of Raspberry Pi devices. A comprehensive experiment validates our approach, implementation, and power models. We generate various and accurate power models, using two regression models, with very low error rates as low as 0.3%. Our models are vastly more accurate than existing models with as much as 10 times lower error rates. We also discuss and analyze the impact of multiple hardware and operating system configurations, and discuss a use-case scenario in remote power monitoring.

Our paper is organized as follows. Section 2 discusses related work. In Section 3 we present our architecture for automated power modeling. In Section 4, we detail the proof-of-concept implementation of our architecture for modeling the power consumption of Raspberry Pi devices. In Section 5, we describe our experimental setup, present our generated power models, and discuss the results and validity of our approach. Finally, we conclude and layout future directions and perspectives in Section 6.

2. Related Work

Several approaches and tools have been proposed to measure or estimate the power consumption of computing devices. Some focus on hardware meters while others use software-based approaches. In this section, we review the related approaches to monitoring and estimating power in computers and IoT devices, and discuss the relevant work around power and regression benchmarks.

In [2], the authors studied the energy impact of users' operations in Raspberry Pi compared to other computing devices. Measurements were carried out using hardware meters due to the lack of accurate software approaches. In [26], a method for energy estimation of Zolertia RE-Mote devices was proposed. It combines offline profiling with online energy estimation, using both a software-based mode and a hardware-based one. The former uses theoretical energy for each operating state, taken from the datasheet, and captures the time spent in each state, and provide energy estimations with an error margin of 53% The latter uses an integrated circuit to measure accurately the power consumption of each state in real-time. A hybrid (hardware and software) power measurement platform for wireless IoT devices called EMPIOT was proposed in [9]. It mainly targets the issue of the power consumption measurement for peripherals. It studied the impact of various design parameters on precision and overhead. It was evaluated by running sleep, encryption, and communication workloads on five different computing devices. SMARTWATTS [13] is a power monitoring platform that increases the accuracy of its CPU and DRAM power models by using an online calibration technique for containers. The authors argue that this approach can be implemented on various machines because it does not need any training phase or pre-configuration. Since 2014, many systems use the Running Average Power Limit (RAPL) feature for power consumption measurements. This feature became available on most Intel CPUs. Yet many devices having ARM or AMD processors do not support RAPL.

In [10], a micro-benchmark-based modeling approach for heterogeneous processors was proposed, in which the authors state that statistical modeling has a significant initial cost during the model training. In [6], the authors propose a technique to generate CPU power models without having a profound knowledge of the CPU architecture. It automatically detects the hardware performance counters correlated to the power consumption and generates the power model from the selected features. Their approach supports changing the learning approach according to the need and software-defined power metering. In [5], a power prediction by applying linear regression to on-chip performance monitoring counters, in particular for the number of micro-operations that are fetched, completed, and retired in each monitoring cycle. In [27], a piece-wise linear function was proposed to estimate the power consumption of an AMD processor. The result of this approach showed a better fit to the collected data when compared with linear and exponential functions. The micro-benchmark data was collected by stressing four AMD performance counters. In [21], the authors reviewed direct measurement and estimation models. Most of the reviewed models apply linear regression techniques on hardware performance counters. They showed that previous studies were limited regarding the considered workloads and the impact of the complexity for each model. In [15], the authors proposed an approach based on statistical power modeling by applying regression analysis on high-level activity metrics. For example, they collected the time spent in each state and the occurrence of certain events and focused mainly on the interaction between the processor and the memory. In [8], the authors surveyed the literature approaches used for energy consumption modeling and prediction for data centers. They noticed a linear relationship between power consumption and CPU utilization.

PowerPi [16] and EMM [17] proposed power models for a particular Raspberry Pi version (RPi 2B and RPi 3B+, respectively). In both approaches, they empirically created a data set correlating power consumption from a power meter to CPU utilization. A linear regression analysis was then applied to generate a power model. However, two main limitations of their approach: first, the model targets only one particular device version and cannot be used on other models due to the distinct power consumption of each RPi model [23]. And second, improving the accuracy of the regression requires manually conducting the experiments again with additional data. In [25], the authors proposed five different power modeling techniques by correlating software metrics with physical power measurements using Mantis [11]. The latter generates a power model using a one-time model fitting technique by collecting software metrics and correlating them to the measured power. It collects metrics from the main system components, namely CPU, memory, and disk. In the paper [25], the authors found that the CPU is the major energy consumer in a computer, but the relation between power and utilization is not always linear.

In [4], the authors proposed a deep neural network (DNN) that predicts the remaining battery of IoT devices. This approach is based on pre-processing the data (eliminate missing values, convert them to numerical format, and normalize data). Then, a Moth Flame Optimization is used to select the optimal features that are input for a DNN model. It provided advantages regarding the feature selection and the battery life estimation accuracy. However, the approach's overload was not calculated and tends to be high due to the significant processing needed. In [28], the authors have gone beyond only predicting battery life using machine learning models to propose a DNN alongside a blockchain. The use of blockchain, as a secure and trustworthy prediction storage, improved the authenticity of the prediction from a security point of view. The DNN predicted the remaining battery with an average accuracy of 90%.

The related work is summarized in Table 1. Although some results appear consistent with prior research, the existing research has multiple limitations: i) Proposed energy estimation models are quickly out-of-date due to software or kernel updates, or new revisions. Such updates are becoming more frequent in modern software, therefore reducing the efficiency of a model generated from a particular software version or hardware revision, ii) The process of model generation is not fully automated and requires human intervention for different tasks (running the benchmark, collecting data, generating, and validating the energy model), and iii) Estimation models are not easily and automatically shared in an effective way between different devices and users. Users need to manually acquire the appropriate energy model for their devices.

To the best of our knowledge, none of the power estimation model generation approaches is based on a continuous improvement method or proposed a technique to automate model generation on a large scale.

Table 1
Summary of related work.

Paper	Suitability	Error (%)	Description
[2]	Any	N/A	Hardware solution
[26]	IoT	4-18	Hardware/Software
[9]	IoT	3.5	Hardware/Software
[13]	Containers	N/A	RAPL
[6]	CPU	1.5	Performance counters
[27]	CPU	3-7	Performance counters
[25]	CPU	10	Performance counters
[10]	AMD APU	3-7	Regression
[5]	CPU	2.6	Regression
[15]	ARM SoC	5	Regression
[4, 28]	IoT	5.17	DNN
Most comparable related work			
[16]	RPi 2B	14.56	Regression
[17]	RPi 3B+	40.76	Regression

3. Automated Power Modeling Architecture

In this section, we present our automated power modeling architecture where we describe the architecture of both our client and server components, along with the automatic generation of power models.

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

Concretely, our architecture, in Figure 1, is composed of three distinct but complementary components: a data collection and benchmarking client, a machine learning and power modeling server, and a power estimator client. Simply put, our benchmarking client will collect run-time software, hardware, and power metrics, which are then sent to the power modeling server. The latter will then generate empirical power models based on the current and previously collected metrics, using machine learning techniques. Finally, the power estimator client will query the server for the most up-to-date and accurate model for the device it's running on, and use that model to estimate the power consumption of the device.

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

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

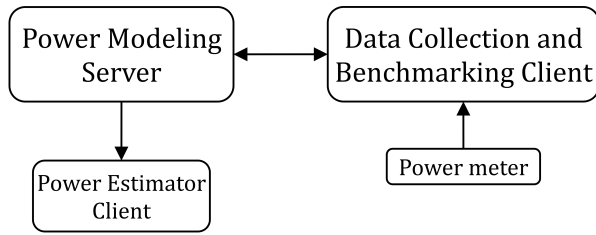


Figure 1: A general overview of our Automated Power Modeling Architecture

3.1. Data Collection and Benchmarking Client Architecture

Figure 2 presents the architecture of our data collection and benchmarking client. Its main role is to collect software, hardware, and power metrics of run-time and real-world workloads. The collected data will then be used as a training and validation data set for the machine learning algorithms in our server.

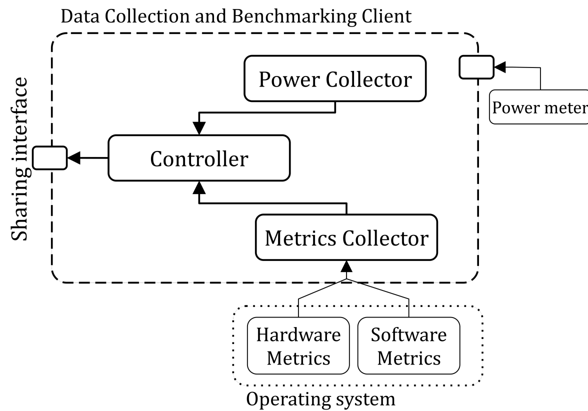


Figure 2: Benchmarking Client Architecture

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

The main components of our architecture are the following:

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

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

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

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

3.2. Power Modeling Server Architecture

Figure 3 presents our power modeling server architecture. Its main role is to receive generate accurate and always up-to-date power models using the data collected from the benchmarking client. The server acts as a centralized entity towards multiple benchmarking clients, receiving data from multiple similar or different devices.

For example, benchmarking clients can collect data from workloads running on multiple instances of a same-type device. This data collection can also spread across time, and the server will regenerate a new updated power model each time a new data set of metrics is received from a benchmarking client.

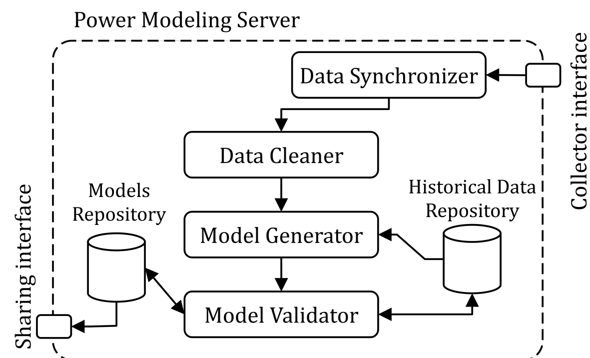


Figure 3: Power Modeling Server Architecture

The main components of our architecture are the following:

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

Data Synchronizer : this component processes the received data (which might be in multiple formats or files), synchronize timestamps between the metric and power data, verifies and synchronizes clock diversion

between the timestamp of the computing device and the one from the physical power sensors or meters.

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

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

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

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

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

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

4. Implementation for Raspberry Pi Power Models

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

4.1. Data Collection and Benchmarking Client

Our implementation of the benchmarking client consists of two main components: a workload-generating benchmark and a data collector. Our architecture supports multiple types of data collections and can generate power models for multiple hardware components. However, in our proof-of-concept implementation, we focus on generating an accurate power model for the ARM processor of Raspberry Pi devices by collecting CPU metrics.

The workload generated by the benchmark consists of applying variable loads on the device's processor. We decide to apply a stress load on the CPU covering the entirety of the load range, i.e., we stress the CPU from 0% all up to 100%, with an incremental step of 5%. The load is applied for 60 seconds for each percentage step. We also saved the workload timestamp and store everything in a CSV file.

The data collector component is a program collecting CPU metrics. In particular, we collect CPU cycles from the Linux *proc* interface (`/proc/stat`). Then we calculate the CPU utilization (ranging from 0 to 1) and save this data to another CSV file.

We calculate the CPU utilization by calculating the ratio of the busy cycles (CPU cycles in user and kernel mode) with the total cycles which includes idle ones:

$$u[t] = \frac{c_{busy}[t] - c_{busy}[t-1]}{c_{total}[t] - c_{total}[t-1]} \quad (1)$$

where:

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

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

4.2. Power Modeling Server

We build our power modeling server following a decision algorithm presented in Figure 4.

We first collect the three generated CSV files from the client (in our implementation, sharing the CSV file from a common storage location). We then normalize and clean the data:

- Remove irrelevant data points (the ones from before, after, and separating the workloads),
- Synchronize timestamps of the three CSV files,
- Synchronize the clock diversion between the Raspberry Pi timestamp and the power meter,
- Aggregate the three CSV files into one containing the label power, the CPU utilization, and the timestamp,
- Remove the inconsistency in the data that results in anomalies (peaks and troughs present at the beginning and end of workloads).

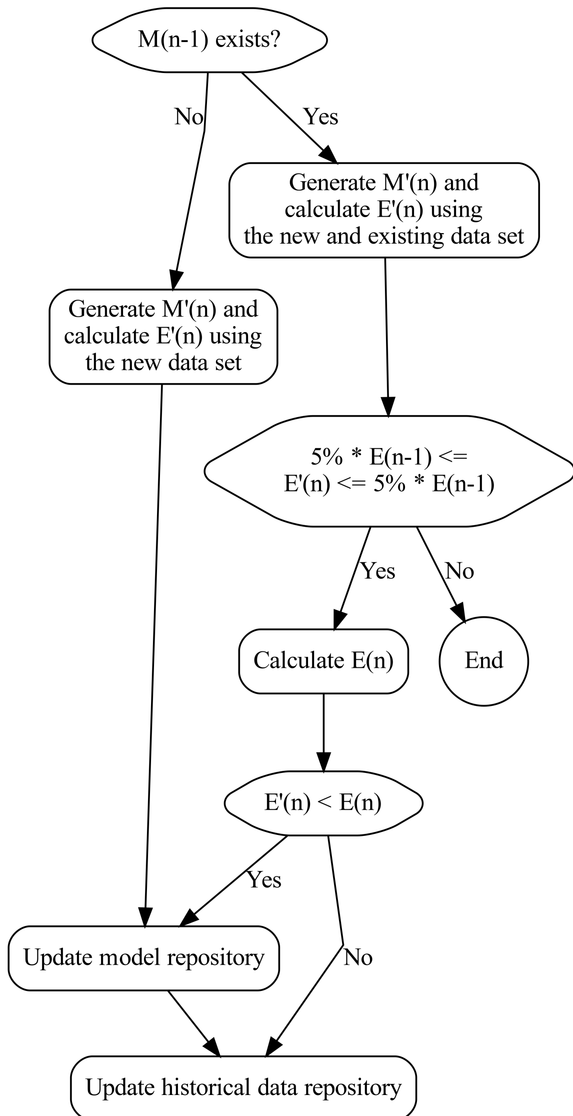


Figure 4: Our Implementation Approach for Power Modeling

We then proceed with the generation and validation of our power model. First, we check if an estimation model already exists on the server for the specific Raspberry Pi device and version. If no model exists, then we proceed to generate a new power model ($M'(n)$) using the collected data, calculate its average error $E'(n)$. Furthermore, we save the new power model to the model repository and the collected data to the historical data repository.

However, if a current power model exists ($M(n-1)$ along with its average error $E(n-1)$), then we proceed with the following process:

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

At the end of this process, the server will contain additional data points which will, over time, improve the accuracy of our empirical power model generation.

In our implementation, we use linear and polynomial regression algorithms to generate power models, as a correlation was observed between CPU utilization and power consumption in Raspberry Pi devices.

In the next section, we detail the empirical experimentation to validate our implementation and power models.

5. Empirical Validation and Discussions

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

Table 2
The variety of Raspberry Pis used during experiments

Model	Rev.	OS	CPU Architecture	Cores	Released
Zero W	1.1	32	armv6l	1	2017
1B	2	32	armv6l	1	2012
1B+	1.2	32	armv6l	1	2014
2B	1.1	32	armv7l	4	2015
3B	1.2	32	armv7l	4	2016
3B+	1.3	32	armv7l	4	2018
4B	1.1	32/64	armv7l/aarch64	4	2019
4B	1.2	32/64	armv7l/aarch64	4	2019

5.1. Experimental Setup

Our experimental setup consists of 8 Raspberry Pi devices from different generations and revisions as detailed in Table 2, dating back from 2012 until the latest current model. We run our workload on both 32 bits (armv7l) and 64 bits (aarch64) operating systems for model 4B (for each of the 2 revisions we used).

We used the same SD card to boot Raspberry OS on all devices, switching operating systems and ARM architecture accordingly. We also automated the benchmark experimentation by adding a boot script to `/etc/rc.local`.

To collect power consumption, we use the PowerSpy2 power meter [1]. PowerSpy2 is a Bluetooth power meter used for advanced and accurate analysis. To reduce interference in the experiments, we use a separate computer to connect to the meter and collect power metrics.

We run all our experiments on Raspberry Pi OS (version based on Debian 9 stretch), with Linux kernel 4.14. Our components and tools are written in Python and run with version 2.8, and in C compiled with GCC 6.3.0. For Raspberry Pi 4B, we the supported version of the OS based on Debian 10 buster with Linux Kernel 5.4, and GCC 8.3.0.

To further reduce interference on the accuracy of our experiments, and as we aim to generate a CPU power model, we disconnected all external peripherals during the workload (including the monitor through HDMI, keyboard, and mouse through USB ports and the network through the Ethernet interface). We also disabled from the operating system all network interface cards (i.e., WiFi and Bluetooth). We also limited the running applications to the minimum as to only monitor the power impact of the workload. Furthermore, we made sure that every device was cooled down before running the experiments, as overheating can have an impact on power consumption. Specifically, each Raspberry Pi was disconnected from its power supply until the device was cooled down.

5.2. Benchmark Data Collection

To collect our CPU metrics, we wrote a minimal C program that read the `/proc/stat` file every second and calculated the CPU utilization. The latter was then saved into a CSV file.

[1] https://www.alciom.com/en/our-trades/products/powerspy2/

As described in our implementation in Section 4.1, we stressed the CPU from 0% to 100% with a 5% increment.

We initially used the same stress command (or stress-ng) used in the literature [17] to specify a percentage CPU load. However, we noticed that the CPU load was inconsistent, with the actual CPU load altering between 0% or 100% in various time duration, rather than consistently stabilizing at the asked percentage load. Instead, we used a Python script, CPULoadGenerator [2], which consistently stressed the CPU at the asked percentage with a small degree of variation. Figure 5 outlines the differences in CPU load consistency between the two tools.

For each experiment, we stressed the CPU for 60 seconds for each CPU load step and is followed by a 10-second pause. A 60-second pause precedes each experiment in order to reduce the impact of our script on the results. In total, each experimental benchmark runs on average for about 25 minutes (24 min and 20 sec).

Each benchmark generates a total of 1460 data points. After the cleaning phase, we end up with more than a thousand data points. These are then used to generate our empirical power models. For the purpose of our experiments, we run our benchmark a few times for the Raspberry Pi 3B+ and Raspberry Pi 4B rev 1.2 (64 bits) and ended up with around 5400 data points for the former, and around 2000 data points for the latter. In total, RPi Zero has 666 data points, 1B has 1034, 1B+ has 954, 2B has 1137, 3B has 1105, 3B+ has 5383, 4B 1.1 (32 bits) has 1089 and its 64 bits version has 998, 4B 1.2 (32 bits) has 1017 and its 64 bits has 2014 data points. The difference in the number of data points for each device type is due to our additional experiments (in particular for the 3B+ and 4B, for instance, to compare 32 bits vs 64 bits, or the validation of the power model generator). Additionally, our cleaning script strips the waiting time between each experiment run of the stress benchmark. As the stress command is not precisely perfect in its timing, the duration of each test might vary by a few seconds, hence the additional data points.

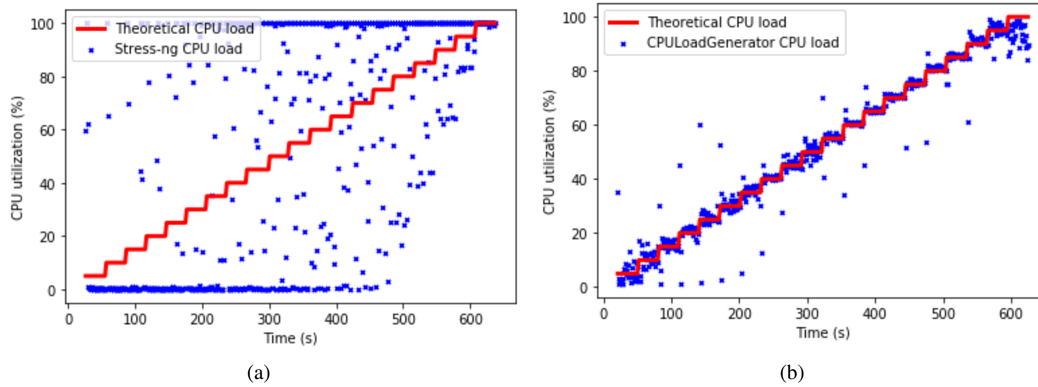
5.3. Regression Power Models

For our experiment, we used two regression algorithms in our server to generate the power models: linear and polynomial regression. We choose these two regression algorithms because of the visible correlation between power and CPU utilization. However, other machine learning techniques and algorithms can be applied to generate different regression models.

Our power modeling server is implemented in Python scripts, automating the data cleaning, synchronization, power modeling, and storing data into the repositories. We, then, generated power models for all different Raspberry Pi models.

Table 3 outlines the generated power models using linear regression algorithms, and Table 4 outlines the models generated with polynomial regression models.

[2] https://github.com/GaetanoCarlucci/CPULoadGenerator



(a) (b)

Figure 5: CPU theoretical load compared to the load generated by (a) the stress-ng command and (b) the CPULoadGenerator

Table 3
Generated power models using linear regression algorithms

Raspberry Pi	Estimation Model
RPi Zero W Rev 1.1	$P = 0.4733 \times U + 0.9201$
RPi 1B Rev 2	$P = 0.1424 \times U + 2.9117$
RPi 1B+ Rev 1.2	$P = 0.1220 \times U + 1.3143$
RPi 2B Rev 1.1	$P = 1.1488 \times U + 1.2903$
RPi 3B Rev 1.2	$P = 3.4774 \times U + 1.0782$
RPi 3B+ Rev 1.3	$P = 3.2983 \times U + 2.0022$
RPi 4B Rev 1.1	$P = 3.7121 \times U + 2.2058$
RPi 4B Rev 1.1 (64 bits)	$P = 4.4958 \times U + 2.3073$
RPi 4B Rev 1.2	$P = 3.4842 \times U + 2.2434$
RPi 4B Rev 1.2 (64 bits)	$P = 4.5344 \times U + 2.2857$

5.4. Validation of Power Models

To validate the accuracy of our power models, we compare the power consumption calculated by our models to the power consumption measured from the power meter. This allows us to calculate the absolute difference between these two values for every data point in the collected metrics. We then calculate an average error using all measurements from all devices with different regression models.

Figure 7 presents the measured correlation between CPU load (in percentage), and the power consumption (in watts) for our two regression models (linear and polynomial) and the actual measurements from the power meter. Except for Raspberry Pi 1B and 1B+, our empirical benchmarks show a better fit for the polynomial regression.

This translates into a lower average error for the polynomial model as compared to the linear model for all experiments and Raspberry Pi devices, as seen in Figure 6. The average error for the linear models varies from as low as 0.34% for RPi 1B+, to 7.81% for RPi 3B. In contrast, the highest average error for the polynomial model is 3.83% to the RPi 3B+.

5.5. Impact of Raspberry Pi Revisions

The Raspberry Pi Foundation often revises its current offering of devices, with modifications to various hardware components. As our implementation focuses on generating power models for the CPU, we suspect that revisions on the USB port or other minor modifications will only have a small impact on the average error of our models.

We proceed to run our benchmarking client and generate power models for the Raspberry Pi 4B revisions 1.1 and 1.2, as seen in the previous Tables 3 and 4. For this particular device, the differences between revisions 1.1 and 1.2 are minimal and related to the USB-C connector as some electronic components were added and reallocated to fix a fault regarding the connector.

Table 5 shows minor differences in the average error between the rev 1.1 and 1.2 for the same OS architecture (32 or 64 bits). This difference is not negligible for accurate power measurements as an increase of up to 56% was observed for using the power model of another revision. However, the average error compared to the power meter is still low in both power models when switching revisions (*i.e.*, around 3% to 4% for rev 1.1). Therefore, we recommend generating and using power models for specific revisions, while still allowing estimator clients to use another revision power model if one isn't provided for the specific revision.

5.6. Impact of 32 and 64 bits Raspberry Pi Versions

Newer Raspberry Pi devices have a 64 bit supported ARM architecture, where users can run either a 32 or a 64 bits operating system. As a stable 64 bits version of Raspberry OS hadn't been released during our experiments, we use its latest beta version (arm64-2020-08-24). Recent experiments had shown that a 64 bits OS on a Raspberry Pi 4 provides a much higher performance compared to a 32 bits OS, up to doubling the performances in benchmarks [7]. Therefore, we suspect that our power models generated in a 32 bits OS would not provide a similar accuracy on a 64 bits OS.

For both Raspberry Pi 4B revisions, we generate power models running our benchmarks on a 64 bits OS, as seen in the previous Tables 3 and 4. As we suspected, our benchmarks running on a 64 bits OS have, on average, higher power consumption than the same device running the same benchmarks on a 32 bit OS.

Table 4
Generated power models using polynomial regression algorithms

Raspberry Pi	y-intercept	Degree 1	2	3	4	5	6	7	8	9
RPi Zero W Rev 1.1	0.85	7.21	-135.52	1254.81	-6329.45	18502.37	-32098.03	32554.68	-17824.35	4069.18
RPi 1B Rev 2	2.826	3.54	-43.59	282.49	-1074.12	2537.68	-3761.78	3391.05	-1692.84	357.80
RPi 1B+ Rev 1.2	1.251	1.86	-18.11	101.53	-346.39	749.56	-1028.80	863.88	-403.27	79.93
RPi 2B Rev 1.1	1.36	5.14	-103.3	1027.17	-5323.64	15592.04	-26675.60	26412.96	-14023.47	3089.79
RPi 3B Rev 1.2	1.52	10.05	-234.19	2516.32	-13733.56	41739.92	-73342.79	74062.65	-39909.43	8894.11
RPi 3B+ Rev 1.3	2.48	2.93	-150.40	2278.69	-15008.56	51537.32	-98756.89	106478.93	-60432.91	14053.68
RPi 4B Rev 1.1	2.57	2.79	-58.95	838.88	-5371.43	18168.84	-34369.58	36585.68	-20501.31	4708.33
RPi 4B Rev 1.1 (64 bits)	3.41	-11.83	137.31	-775.89	2563.40	-4783.02	4974.96	-2691.92	590.36	
RPi 4B Rev 1.2	2.59	12.34	-248.01	2379.83	-11962.42	34444.27	-58455.27	57698.69	-30618.56	6752.27
RPi 4B Rev 1.1 (64 bits)	3.41	-3.07	47.75	-271.97	879.97	-1437.47	1133.33	-345.13		

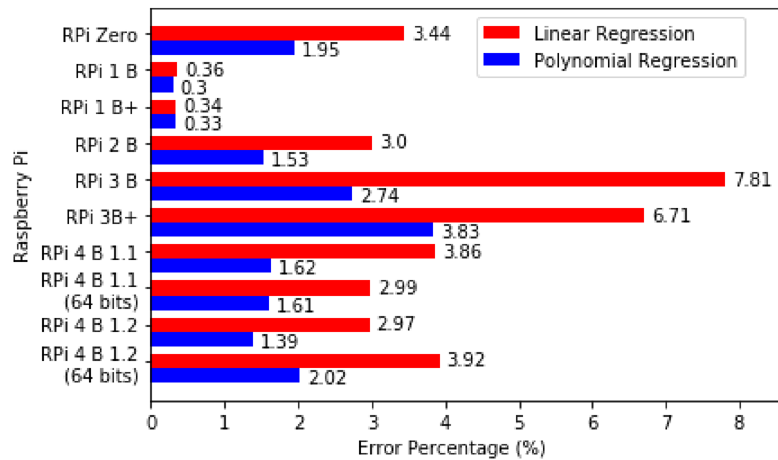


Figure 6: Average Error percentage for linear and polynomial regression algorithms per Raspberry Pi

Table 5 presents the average error of each generated power model when applied to the benchmark data of every experiment. We observe, consistently, that a different architecture highly impacts the accuracy of the generated power models, up to more than 5 times. For instance, RPi 4B rev 1.2 32 bits power models are nearly 5 times less accurate when used on the same revision but with a 64 bits OS. These results confirm our hypothesis and the higher performance of a 64 bits OS on supported devices as seen in the literature [7].

Consequently, we recommend using power models generated specifically for the device's architecture.

5.7. Impact of Connected Peripherals

Raspberry Pi devices are designed to be easily connected to external peripherals via a variety of physical interfaces. To assess the impact of connected peripherals on the validity of our energy models, we conduct the same benchmark experience in two different scenarios on a Raspberry Pi 4B, revision 1.2. In the first scenario, we disconnect all peripherals and follow the procedure mentioned in Section 5.1. In the second scenario, we launch the benchmark after connecting the Raspberry Pi to a screen using the mini HDMI interface, a USB wired keyboard, and a wireless mouse, and we activate WiFi.

We generate two power models, one for each scenario, and calculate its average error. As seen in Table 6, both CPU

Table 5
Comparison of the average error of the linear models for 32 bits and 64 bits OS, for both revisions of Raspberry Pi 4B

RPi / Power model	RPi 4 B 1.1 32 bits	RPi 4 B 1.1 64 bits	RPi 4 B 1.2 32 bits	RPi 4 B 1.2 64 bits
RPi 4 B 1.1 32 bits	**3.86**	12.47	4.64	12.47
RPi 4 B 1.1 64 bits	10.68	**2.99**	12.22	2.97
RPi 4 B 1.2 32 bits	3.70	14.65	**2.97**	14.63
RPi 4 B 1.2 64 bits	10.60	3.97	12.09	**3.92**

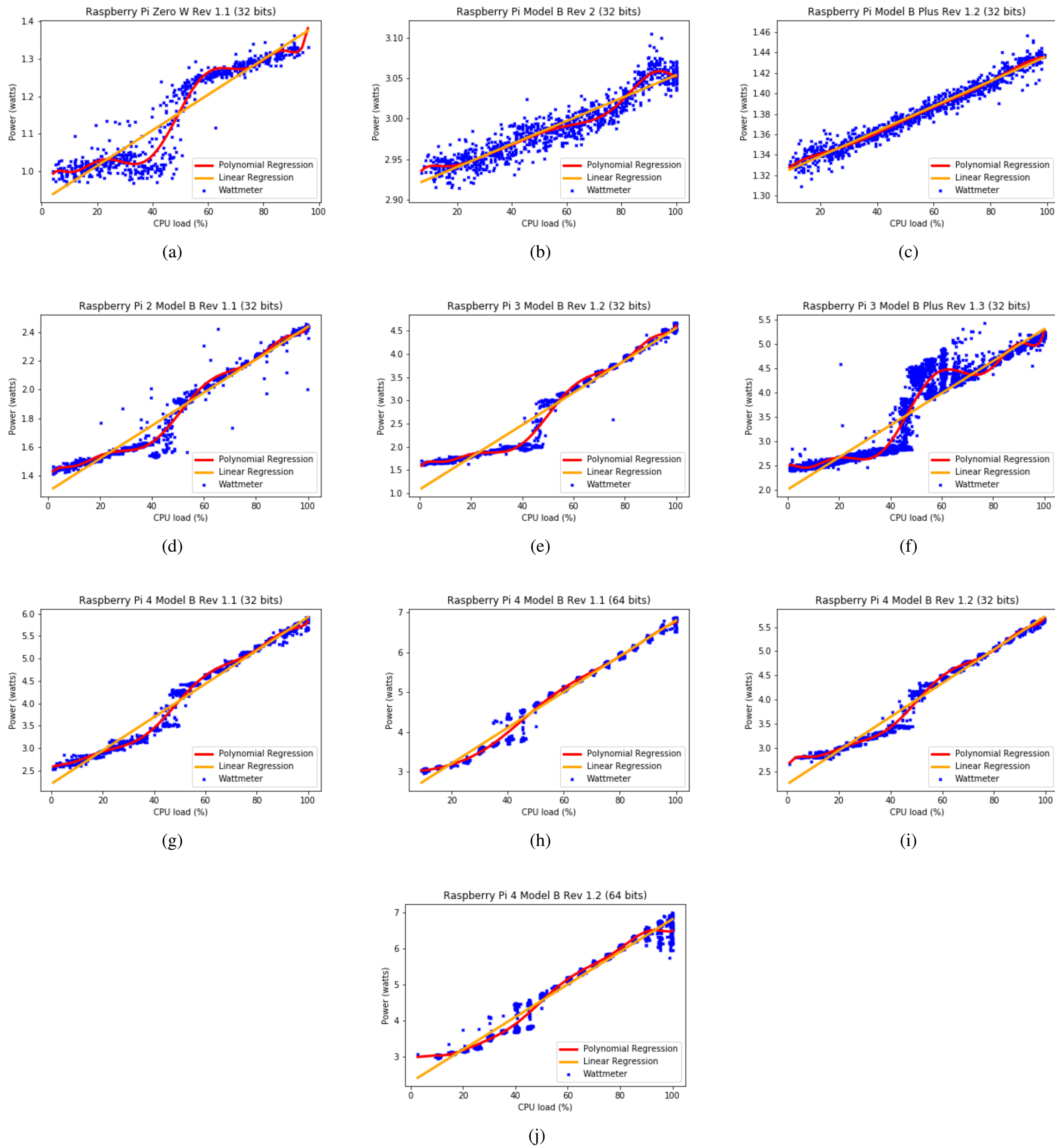


Figure 7: Linear vs polynomial regression power estimation models of the (a) RPi Zero W (b) RPi 1B (c) RPi 1B+ (d) RPi 2B (e) RPi 3B (f) RPi 3B+ (g) RPi 4B 1.1 (h) RPi 4B 1.1 (64 bits) (i) RPi 4B 1.2 (j) RPi 4B 1.2 (64 bits)

linear and polynomial models have a lower error when disconnecting the peripherals. However, the models generated with the peripherals are still within an acceptable margin below 8%. This proves that our approach can generate CPU power models with an acceptable accuracy even with interference from connected peripherals.

Table 6 also shows the average error when using the power model of one scenario onto the data of the other scenario, *i.e.*, using the power model generated with the peripherals on benchmarking data generated without the peripherals, and vice versa. The average error shows a lower accuracy of the model, but still within a range below 8%. This means our CPU power models generated in an ideal benchmark setup (with peripherals disconnected), are still accurate enough to, not only estimate the power of the CPU, but to estimate the power of the Raspberry Pi device (as the CPU is shown to be the most power-consuming component).

Raspberry Pi devices are often used in a headless server setup (such as to control industrial machines, control heating or lightning in a smart home or city, a web or NAS server,

Table 6
Comparison of the average error of the linear and polynomial models with and without peripherals on a Raspberry Pi 4B, rev. 1.2

Power Model	Data	
	with	without
Linear with	**4.19%**	7.83%
Linear without	7.34%	**2.92%**
Polynomial with	**3.73%**	7.72%
Polynomial without	6.91%	**1.64%**

etc.). In these situations, our approach generates accurate CPU power models without interference from peripherals.

5.8. Comparison to the State of the Art Models

To assess the validity of our models and our automated approach in generating up-to-date models, we compare our generated power models to the ones provided by the state of the art. In particular, we tested and compared our models to two models: PowerPi [16] and EMM [17].

PowerPi provided linear power models for the Raspberry Pi 2B and used a custom tool to stress the CPU using an infinite loop doing two additions of two integer variables. `cpulimit` was used to limit the CPU utilization which was stressed with a 10% step.

EMM provided linear power models for the Raspberry Pi 3B+ and used the `stress-ng` command to stress the CPU for a particular load. As we explained in Section 5.2, we found that using this command generates an inconsistent CPU load. In comparison, we use a more random CPU load generator (compared to PowerPi), and a more consistent CPU load tool (compared to EMM). We also stressed the CPU with a 5% step, collecting more data points at more CPU load percentages. Therefore, we have a more comprehensive and complete data set to generate more accurate power models. Our architecture also follows a crowd-sourced approach allowing adding additional benchmarks and data to constantly improve the generate power models.

PowerPi announces an average error of 1.2%, while EMM announces an average error of 1.25% with a maximum of 3%. However, in our experiments, we found that both models have a much higher error rate: 14.56% for PowerPi and 40.76% for EMM (cf. Figure 8). In contrast, our linear models provide an error rate of 3% for RPi 2B, and 6.71% for RPi 3B+, and even much lower error rates for our polynomial models (1.53% and 3.83%, respectively).

In addition, our approach is based on a dynamic model generation that deals with dynamic workloads. It is validated on a large variety of Raspberry Pi devices in comparison with state-of-the-art methods (which are tested on a single device). It also allows the collection of a large data set due to its decentralized data collection technique. Another advantage of our approach is its flexibility: new models can be added or modified as the system evolves. As such, it can integrate into complex and moving environments (such as when new devices are often introduced or updated).

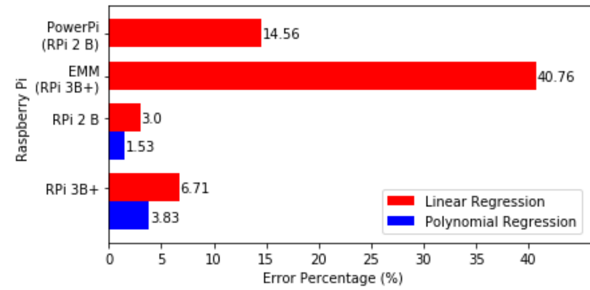


Figure 8: Average Error percentage for linear and polynomial regression algorithms for our approach compared to the literature

5.9. Overhead of Regression Models

Most of our generated polynomial models have a degree of 9, requiring calculations up to the power of 9. In our analysis, we found that polynomial models with a higher degree than 9 have negligible accuracy improvements but with a higher calculation complexity. As these models have a much higher accuracy than the linear ones, we compared the overhead of running both models in our implementation of the power estimator client.

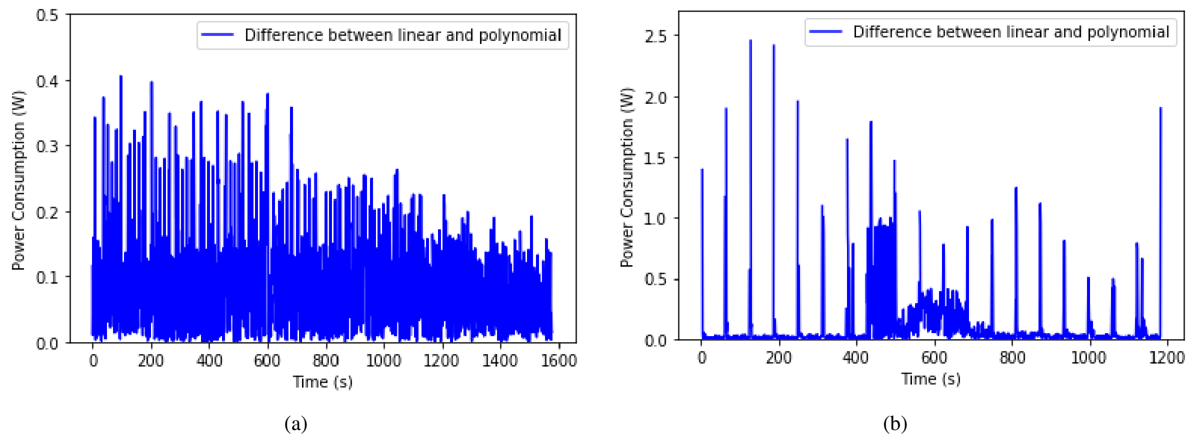
Our client is a minimal C program reading CPU cycles, calculating CPU utilization, and applying the power models. The client monitors power consumption at run-time and provides a power value every second. We compare the power overhead of running the client with both power models, and also in comparison to the base power consumption without our client. We conduct our experiment on two Raspberry Pi models: RPi Zero W for a low-power device, and the 3B+ for a more recent higher-power one.

Figure 9 outlines the absolute differences in Watts between both our implementations: linear and polynomial models. For both devices, although we observe some rare data points of higher diversion, the overall difference is quite low with an absolute average difference of 0.084 watts (corresponds to a relative difference of 1.32%) for the RPi Zero W, and only 0.109 watts (corresponds to a relative difference of 0.72%) for the RPi 3B+. These numbers show a negligible overhead for using our polynomial models over the linear ones, even on low-power devices. We, therefore, recommend using the polynomial power models even for run-time power monitoring.

5.10. Validation of the Power Model Generator

The core idea of our power modeling generator, described in Section 4.2, is to allow third-party benchmarking clients to send new benchmark metrics to further improve the accuracy of the generated power models. In this section, we validate our approach with a breakdown of a step-by-step experiment of the linear power model in a Raspberry Pi 3B+ device.

We run our experiment 10 times, emulating 7 benchmarks sending data incrementally one after the other for one



(a) (b)

Figure 9: Power overhead of the (a) RPi Zero W and the (b) RPi 3B+

Table 7
Breakdown of our Power Model Approach (linear model on RPi 3B+, bold for the selected model)

Step	M'(n)	E'(n)	E(n)	M(Server)
S1	$P = 3.357 \times U + 2.024$	**7.64%**	-	$S1$
S2	$P = 3.271 \times U + 2.032$	**6.61%**	7.07%	$S2$
S3	$P = 3.272 \times U + 2.011$	**6.13%**	6.23%	$S3$
S4	$P = 3.285 \times U + 2.014$	6.58%	**6.51%**	$S3$
S5	$P = 3.296 \times U + 2.004$	6.72%	**6.66%**	$S3$
S6	$P = 3.294 \times U + 1.997$	6.45%	**6.43%**	$S3$
S7	$P = 3.292 \times U + 1.992$	**6.25%**	6.27%	$S7$

particular device (Raspberry Pi 3B+). Our server implementation then runs our model generator algorithms and keeps the best accurate power model in every step.

The result of this breakdown is outlined in Table 7 for the linear model, and Table 8 for the polynomial model. Each row of the table represents a new server iteration (receiving new data, data normalization and cleaning, validation, power model generation, and comparison, etc.). $M'(n)$ indicates the newly generated power model in the server, along with its error rate $E'(n)$. $E(n)$ is the error rate of the currently saved power model using all the data. And $M(Server)$ is the power model that is saved after the current iteration.

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

We argue that the more benchmark data we have, the more our architecture and approach will provide empirical power models with better accuracy.

5.11. Use Case of Remote Power Monitoring

A use case illustrating the advantages of our approach is remote power monitoring of a park of deployed Raspberry Pi devices. Examples of such use cases vary from monitoring environmental metrics [20, 19], smart management [14], or health [18]. In particular, smart devices, such as Raspberry Pis, send collected metrics and their status (including CPU statistics) to a central monitoring service.

An example of the latter is Zabbix [3], an open-source server used for real-time monitoring of a large number of clients. In each Raspberry Pi client, a Zabbix agent is installed to allow remote monitoring and management. It can send CPU utilization and many other metrics for the device in real-time. We developed a prototype plugin for Zabbix to integrate our power models and architecture into its web interface. Our plugin updates the power models of the monitored devices by connecting to our power modeling server. It also tweaks the Zabbix web interface to calculate the power consumption of monitored devices, in real-time, and displays them along with the CPU utilization, as seen in Figure 10.

With our approach and power models, remote management tools can efficiently, accurately and with no overhead on the monitored devices, monitor the power consumption in real-time. It also allows these tools to always have updated and accurate power models, and to support new device power models easily by just calling our sharing interface.

5.12. Threats to Validity

Our approach and experimentation suffer from the following threats to validity:

- Our implementation is limited to one type of single-board computers, i.e., Raspberry Pi devices. We run

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

Table 8
Breakdown of our Power Model Approach (polynomial model on RPi 3B+, bold for the selected model)

Step	P[0]	Degree of M'(n)									E'(n)	E(n)	M(Server)
		1	2	3	4	5	6	7	8	9			
S1	2.34	11.95	-334.15	3997.0	-23579.71	76091.11	-140313.95	147466.42	-82193.81	18858.15	**3.68**	-	$S1$
S2	2.45	5.33	-186.67	2534.97	-15983.94	53687.34	-101595.06	108702.03	-61390.2	14229.05	**3.73**	4.15	$S2$
S3	2.49	2.66	-129.57	1976.33	-13076.81	45047.92	-86526.32	93476.51	-53148.84	12380.92	**3.63**	3.79	$S3$
S4	2.48	3.07	-146.99	2215.22	-14580.93	50066.17	-95944.1	103459.96	-58730.58	13660.99	3.8	**3.71**	$S3$
S5	2.49	2.96	-151.39	2289.23	-15064.96	51709.94	-99070.29	106810.86	-60621.69	14098.12	3.82	**3.73**	$S3$
S6	2.51	1.69	-123.33	2010.51	-13596.98	47304.09	-91325.7	98938.95	-56341.33	13134.87	3.78	**3.71**	$S3$
S7	2.52	0.74	-102.99	1810.06	-12545.46	44155.31	-85798.75	93326.76	-53291.99	12449.07	3.74	**3.69**	$S3$

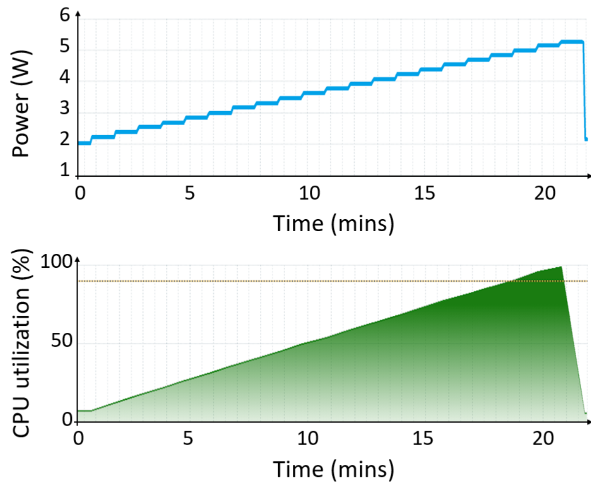


Figure 10: Power consumption of a Raspberry Pi displayed on a Zabbix server web interface

our experiments on a wide variety of RPi device models from each generation, but some devices and revision models were not modeled.

- Our implementation only uses one metric to correlate to the power consumption, i.e., CPU utilization which is calculated from the measured CPU cycles. Even though our results show a strong correlation, and allow us to generate very accurate power models (that can have an average error as low as 0.3%), we did not investigate additional metrics or other hardware components (such as the WiFi or Bluetooth).

- Although we made sure no interference happened to our experiments and data collection, we could not formally discard that no external factors impacted the results. In particular, the experiments spanned over 4 months in different weather conditions (winter and spring) and therefore different room temperatures. Additionally, the power overhead of the benchmarking client was not subtracted from the collected data because we wished to emulate real usage of the client (where it is difficult to automatically deduce a variable overhead. This deduction process might require additional data collection, such as the CPU cycles of the client process, and therefore adds an additional overhead itself).

6. Conclusion and Future Directions

In this paper, we presented an architecture to automate the generation of always up-to-date and accurate power models for a variety of devices. Our approach allows crowd-sourced benchmarking of devices, the collection of various metrics, and the generation of specific power models based on the collected data. A sharing interface allows power estimator clients to query and retrieve the most accurate power model available for the client's device.

We implemented a proof-of-concept client and server to automate the generation of power models for Raspberry Pi devices. We also conducted a comprehensive experiment validating our approach, algorithms, and power models. The latter provides high accuracy with error rates as low as 0.33% and up to 7.81% for linear models, and 0.3% up to 3.83% for polynomial models. Furthermore, we analyzed and discussed the impact of device revisions, CPU and OS architectures, and the overhead of both generated power model types. Finally, we validated our approach in the power model generator and provided an example of a use-case scenario of our models in remote power monitoring.

In the future, we plan to study and model the power consumption of additional hardware components of Raspberry Pi devices, such as the network (WiFi and Ethernet), Bluetooth, HDMI, and connected USB devices. We also plan in expanding our implementation to cover additional single-board devices, such as BeagleBoard, and mobile and embedded devices. In addition, we would like to study the impact of additional software and hardware metrics on power consumption, as this might help in improving the accuracy of the generated power models. Furthermore, we plan to study and implement additional machine learning algorithms and automate the selection of the most efficient algorithm. We also plan to investigate reinforced learning approaches and their integration into our architecture.

Acknowledgment

The project leading to this publication has received funding from Excellence Initiative of Université de Pau et des Pays de l'Adour - I-Site E2S UPPA, a French "Investissements d'Avenir" programme.

References

[1] Astudillo-Salinas, F., Barrera-Salamea, D., Vazquez-Rodas, A., Solano-Quinde, L., 2016. Minimizing the power consumption in Raspberry Pi to use as a remote WSN gateway. 2016 8th IEEE Latin-American Conference on Communications, LATINCOM 2016 doi:10.1109/LATINCOM.2016.7811590.

[2] Bekaroo, G., Santokhee, A., 2016. Power consumption of the Raspberry Pi: A comparative analysis. 2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies, EmergiTech 2016 , 361–366doi:10.1109/EmergiTech.2016.7737367.

[3] Belkhir, L., Elmeligi, A., 2018. Assessing ICT global emissions footprint: Trends to 2040 & recommendations. Journal of Cleaner Production 177, 448–463. doi:10.1016/j.jclepro.2017.12.239.

[4] Bhattacharya, S., Kumar Reddy Maddikunta, P., Meenakshisundaram, I., Reddy Gadekallu, T., Sharma, S., Alkahtani, M., Haider Abidi, M., 2021. Deep Neural Networks Based Approach for Battery Life Prediction. Computers, Materials & Continua 69, 2599–2615. doi:10.32604/cmc.2021.016229.

[5] Bircher, W.L., Valluri, M., Law, J., John, L.K., 2005. Runtime identification of microprocessor energy saving opportunities, in: Proceedings of the 2005 international symposium on Low power electronics and design - ISLPED '05, ACM Press, New York, New York, USA. p. 275. doi:10.1145/1077603.1077668.

[6] Colmant, M., Rouvoy, R., Kurpicz, M., Sobe, A., Felber, P., Seinturier, L., 2018. The Next 700 CPU Power Models. Journal of Systems and Software 144, 382–396. doi:10.1016/j.jss.2018.07.001.

[7] Croce, M., 2020. Why you should run a 64 bit os on your raspberry pi4.

[8] Dayarathna, M., Wen, Y., Fan, R., 2016. Data center energy consumption modeling: A survey. IEEE Communications Surveys and Tutorials 18, 732–794. doi:10.1109/COMST.2015.2481183.

[9] Dezfouli, B., Amirtharaj, I., Li, C.C.C., 2018. EMPIOT: An energy measurement platform for wireless IoT devices. Journal of Network and Computer Applications 121, 135–148. doi:10.1016/j.jnca.2018.07.016, arXiv:1804.04794.

[10] Diop, T., Jerger, N.E., Anderson, J., 2014. Power modeling for heterogeneous processors. ACM International Conference Proceeding Series , 90–98doi:10.1145/2576779.2576790.

[11] Economou, D., Rivoire, S., Kozyrakis, C., Ranganathan, P., 2006. Full-System Power Analysis and Modeling for Server Environments. Workshop on Modeling, Benchmarking and Simulation (MoBS) , 807–812.

[12] Evans, D., 2011. The Internet of Things: How The Next Evolution of the Internet is Changing Everything. CISCO white paper , 1–78arXiv:arXiv:1011.1669v3.

[13] Fieni, G., Rouvoy, R., Seinturier, L., 2020. SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), IEEE, guillaume2020. pp. 479–488. doi:10.1109/CCGrid49817.2020.00-45, arXiv:2001.02505.

[14] Ishak, S.N., Malik, N.N.N.A., Latiff, N.M.A., Ghazali, N.E., Baharudin, M.A., 2017. Smart home garden irrigation system using raspberry pi, in: 2017 IEEE 13th Malaysia International Conference on Communications (MICC), pp. 101–106. doi:10.1109/MICC.2017.8311741.

[15] Jose, N.Y., Geza, L., 2013. Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip. Microprocessors and Microsystems 37, 319–332.

[16] Kaup, F., Gottschling, P., Hausheer, D., 2014. PowerPi: Measuring and modeling the power consumption of the Raspberry Pi. Proceedings - Conference on Local Computer Networks, LCN , 236–243doi:10.1109/LCN.2014.6925777.

[17] Kesrouani, K., Kanso, H., Noureddine, A., 2020. A Preliminary Study of the Energy Impact of Software in Raspberry Pi devices, in: 29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, Bayonne, France. pp. 231–234. doi:10.1109/WETICE49692.2020.00052.

[18] Kumar, R., Rajasekaran, M.P., 2016. An iot based patient monitoring system using raspberry pi, in: 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16), pp. 1–4. doi:10.1109/ICCTIDE.2016.7725378.

[19] Kumar, S., Jasuja, A., 2017. Air quality monitoring system based on iot using raspberry pi, in: 2017 International Conference on Computing, Communication and Automation (ICCCA), pp. 1341–1346. doi:10.1109/CCAA.2017.8230005.

[20] Lewis, A., Campbell, M., Stavroulakis, P., 2016. Performance evaluation of a cheap, open source, digital environmental monitor based on the raspberry pi. Measurement 87, 228–235. doi:https://doi.org/10.1016/j.measurement.2016.03.023.

[21] Möbius, C., Dargie, W., Schill, A., 2014. Power consumption estimation models for processors, virtual machines, and servers. IEEE Transactions on Parallel and Distributed Systems 25, 1600–1614. doi:10.1109/TPDS.2013.183.

[22] Noureddine, A., Rouvoy, R., Seinturier, L., 2015. Monitoring energy hotspots in software. Automated Software Engineering 22, 291–332. doi:10.1007/s10515-014-0171-1.

[23] Raspberry Pi Foundation, 2020. Power Supply - Raspberry Pi Documentation. https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md.

[24] Reddy, B.K., Walker, M.J., Balsamo, D., Diestelhorst, S., Al-Hashimi, B.M., Merrett, G.V., 2017. Empirical cpu power modelling and estimation in the gem5 simulator, in: 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), IEEE. pp. 1–8.

[25] Rivoire, S., Ranganathan, P., Kozyrakis, C., 2008. A comparison of high-level full-system power models, in: Proceedings of the 2008 Conference on Power Aware Computing and Systems, USENIX Association, USA. p. 3.

[26] Sabovic, A., Delgado, C., Bauwens, J., De Poorter, E., Famaey, J., 2020. Accurate Online Energy Consumption Estimation of IoT Devices Using Energest. Lecture Notes in Networks and Systems 97, 363–373. doi:10.1007/978-3-030-33506-9_32.

[27] Singh, K., Bhadauria, M., McKee, S.A., 2009. Real time power estimation and thread scheduling via performance counters. ACM SIGARCH Computer Architecture News 37, 46–55. doi:10.1145/1577129.1577137.

[28] Somayaji, S.R.K., Alazab, M., MK, M., Bucchiarone, A., Chowdhary, C.L., Gadekallu, T.R., 2020. A Framework for Prediction and Storage of Battery Life in IoT Devices using DNN and Blockchain, in: 2020 IEEE Globecom Workshops (GC Wkshps, IEEE. pp. 1–6. doi:10.1109/GCWkshps50303.2020.9367413, arXiv:2011.01473.