# Power Estimation Models for Edge Computing Devices

Michalis Kasioulis, Moysis Symeonides, George Pallis, and Marios D. Dikaiakos

Department of Computer Science, University of Cyprus, Nicosia, Cyprus
{mkasio01, msymeo03, pallis, mdd}@ucy.ac.cy

**Abstract.** The increasing demand for energy-efficient solutions in IoT devices and edge computing calls for novel methodologies to generate accurate power models for diverse devices, enabling sustainable growth and optimized performance. This paper presents a methodology for creating power models for edge devices and their embedded components. The proposed methodology collects power and resource utilization measurements from the edge device and generates both additive and regression models. The methodology is evaluated on a Raspberry Pi 4 device using a smart plug for power monitoring and various benchmarking tools for CPU and network sub-components. The evaluation shows that the generated models achieve low error, demonstrating the effectiveness of the proposed approach. Our methodology can be applied to any edge device, providing insights into the most efficient power consumption model. The heterogeneity of edge devices poses a challenge to creating a global power model, and our approach provides a solution for developing device-specific power models. Our results indicate that the generated models for Raspberry Pi 4 scored a maximum of 8% MAPE.

**Keywords:** Power Consumption · Power Modeling · IoT · Edge Computing · Edge Benchmarking

## 1 Introduction

The Internet of Things (IoT) is an ever-growing paradigm that enables a vast number of interconnected devices to communicate and share data. According to [4], the number of IoT and edge devices is expected to reach 6.5 billion devices by 2030, or a three-fold increase since 2020. According to Cisco Annual Internet Report [1], by 2023 50% of all networked communication will account for machine-to-machine communications. Therefore, the energy footprint of those devices needs to be studied, and the estimation of their power consumption is crucial for designing energy-efficient systems and optimizing their operation. Energy-efficient design and operation of edge devices and systems can significantly reduce the need for frequent battery replacement or maintenance.

One of the main challenges in estimating power consumption is the heterogeneity of edge devices [6]. A global model for power estimation is difficult to create, and each device together with its attached peripherals must be individually characterized. As edge computing becomes more prevalent, the energy

consumption forecasting of edge devices is more crucial. For estimating the cumulative energy footprint of large-scale geo-distributed edge deployments, we need a tailored power model for every edge device. However, most studies on power profiling of IT equipment focus on high-end devices [3]. More work needs to be imposed on edge devices considering the vast amount and the fast pace of increase that calls for efficient ways to estimate their power and carbon footprint.

In this study, we present a methodology for building power consumption models for edge devices, that takes into account the impact of their different hardware sub-components. Our approach involves: (i) the execution of containerized stressing processes dedicated for each hardware sub-component of the edge device as defined by the user, e.g., CPU and Network I/Os, (ii) an end-to-end monitoring sub-system that collects resource utilization and power consumption measurements from the respective device, (iii) a machine learning pipeline, integrated to our monitoring sub-system, capable of training various machine learning models, including linear regression models, random forests, gradient boosting, etc, and (vi) evaluation of the created models by executing real-world Machine Learning (ML) benchmarks and considering well-known evaluation metrics like Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE). It should be noted that the users can select to create additive models composed of individual models, each of which is dedicated to a specific hardware sub-component.

The main contributions of the current work are listed below.

1. A device-agnostic methodology that involves an end-to-end pipeline for generating power consumption models for edge devices and can provide insights about the most efficient power consumption models. On top of that, we provide an out-of-the-box procedure for evaluating the most accurate power models with real-world workloads using statistical error indicators.
2. A complete power consumption model for a well-known edge device, namely Raspberry Pi 4, evaluating the applicability and accuracy of our methodology with the results indicating a maximum MAPE of 8% and the models being available to the research community, enabling broader adoption.

The rest of the paper is structured as follows: Section 2 provides details about the related work. Section 3 describes our methodology. Section 4 provides technical details about the creation of the power model for an edge device (RPi 4), and a real-world scenario evaluation. Finally, Section 5 concludes the paper.

## 2 Related Work

The process of power consumption modeling is a complex procedure that requires the introduction of various stressors on top of edge devices while at the same time, capturing resource utilization and power consumption measurements of the edge device acting as the system under test (SUT). On top of that, researchers need to build various statistical and ML models based on the captured metrics. Considering the heterogeneity of edge environments, the latter procedure is time-consuming and requires numerous manual configurations during deployment and

ML training. Targeting both power consumption modeling and benchmarking of edge devices, we present the state-of-the-art.

**Edge benchmarking:** This is the process of evaluating the performance of an edge device under different workloads and configurations. In the context of edge benchmarking, Kang et al. [9] made use of a Coral Dev Board that is equipped with a Tensor Processing Unit (TPU) and a Jetson Nano that is equipped with a Graphical Processing Unit (GPU) in order to run a set of AI applications over them and evaluate their performance based on their inference speed, accuracy, and power consumption. Similarly, Bekaroo et al. [5] performed a comparative analysis of the power consumption of Raspberry Pi under different utilization levels. In this study, the power consumption of Raspberry Pi was compared against other workstations and results indicated that the power consumption of Raspberry Pi was significantly lower than that of a desktop and a laptop and higher than that of a smartphone and a tablet. Although the aforementioned studies provide some key insights into the power consumption behavior of edge devices, *no further modeling of power consumption for those devices was performed.*

**Power consumption modeling:** A power model is a mathematical representation of how much power a device or system consumes under various operating conditions. It takes as input a set of parameters related to the device load, and outputs an estimate of the device's power consumption. Focusing on single-board edge devices, Paniego et al. [11] created a power estimation model for Raspberry Pi 3. The authors made use of a set of performance counters that reflect the resource utilization of CPU and memory. Even if the evaluation results indicated an average percentage error less than 5%, the authors did not consider the power that is consumed by the network sub-components. On the contrary, Kaup et al. [10] created power estimation models for various edge devices considering both processing and network sub-components. They highlighted that the power consumption of a single-board edge device can be described by multiple models dedicated to the underlying hardware sub-components. Unfortunately, the latter efforts *do not offer a general way for the creation of power consumption models, leaving it up to the users to perform the device's stressing, monitor the sub-components utilization, and tune the parameters of the models.*

## 3 Methodology

The generation of power consumption models for edge devices requires the installation and deployment of multiple software tools, their configuration, a set of repeatable actions for stressing specific underlying components, the selection of the proper ML or mathematical models and their parameters, the evaluation of the generated models, and, generally, a lot of manual and time-consuming steps. The main goal of this work is to abstract the generation of power consumption models of diverse edge devices. To achieve the latter, we propose a set of well-defined steps in our methodology workflow as shown in Fig. 1.

First, we created a monitoring and storage solution that captures both resource utilization metrics from the edge device under test and power consumption metrics as provided by the power reporting tools. We achieved the latter by providing abstract interfaces for user-defined monitoring probes and using

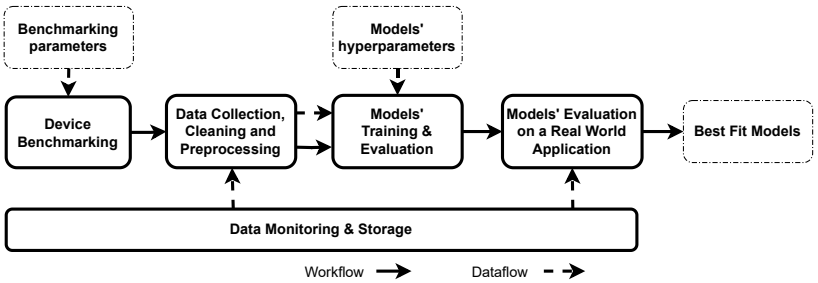


**Fig. 1.** Workflow for Power Models Generation & Evaluation

containerization as our deployment strategy. To obtain a representative dataset, our pipeline executes a set of repeatable benchmark workloads that utilize different components of the device, such as the CPU and network. In order for our pipeline to execute the set of stressors in any compute node, the device benchmarking sub-module executes them in a containerized environment. The module executes repeatable stressing workloads under various configurable utilization levels to ensure the statistical significance of the measurements.

Before the training of the models, our workflow pre-processes the collected data in order to "clean" them. The pre-processing step includes the outliers detection and removal. When the dataset is marked ready, the workflow propagates it to the Model Generation module. This module is responsible for the generation of the selected models and auto-tunes their parameters based on the user's configurations. The output of the Models Generation module is a set of ML and mathematical models capable of predicting the power consumption of an edge device based on its resource utilization metrics.

Finally, in this evaluation step, the workflow deploys real-world containerized workloads on the examined edge device and monitors its resource utilization. The best-fit models are selected based on the accuracy reported when applied to the collected results from the workload execution period.

### 3.1 Implementation Aspects

**Data Monitoring & Storage.** The deployment overview is illustrated in Fig.2 where the running services are presented next to each device involved in the deployment stack. A monitoring agent is deployed on the edge device under test. Specifically, we use the containerized version of Netdata, which is a lightweight monitoring agent reported to have less than 1% CPU computational footprint, a few hundred of MiB RAM requirements, and minimal disk usage [2]. When the agent is deployed, it continuously collects the underlying resource utilization and other monitoring metrics like CPU temperature by directly making use of the host's OS mechanisms reporting real-time metrics (cgroups, pseudofiles, etc.). As a utilization metric for the CPU board, we chose to use the percentage of the total CPU utilization based on the full capacity. For the network boards, we use the kilobits sent or received per second (kb/s). For the accurate collection of the resource utilization metrics, we rely on the underlying monitoring system which monitors CPU utilization by measuring the percentage of time

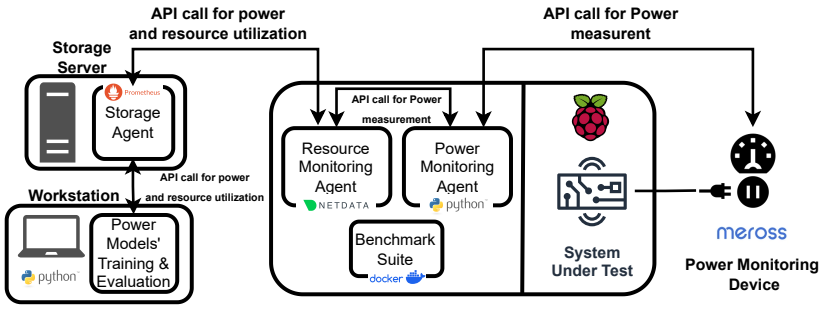


**Fig. 2.** Deployment Overview

the CPU has actually been executing machine code. The monitoring system can also monitor the CPU frequency at runtime which can be taken into account when creating power models for machines whose CPU architecture supports the Dynamic Voltage and Frequency Scaling (DVFS) mechanism.

Moreover, we created a background process with extensible interfaces that acts as a probe for the power consumption metering external system. It should be noted that the edge device under test and the power meter should have a common communication channel, e.g. both to be connected to the same WiFi network. In our prototype, we integrated Meross [1] Smart WiFi plug to our system under test, and we extended the respective connector interface to perform periodically a remote call to the smart plug API retrieving real-time power measurements and disseminating them to the monitoring agent. The periodicity of monitoring is configurable with its default value being 1 second, providing more fine-grained data to the next step. The results are then transmitted to a remote monitoring storage server via HTTP requests. The monitoring storage utilizes a time series database [2] allowing fast time range queries and is placed on a remote virtual machine (VM) that provides ample processing and storage capacity.

**Device Benchmarking.** To test the separate components independently, we encapsulate various benchmarks each of which is tailored to a specific component. For the CPU, we use the stress Linux command[3], which allows us to stress the CPU at various utilization levels. The stress command has an optional parameter that runs all the stressors sequentially for a specified period of time as defined by the user, with each stressor being executed by a separate CPU thread. Therefore, we ran the stressors sequentially using 1...N CPU threads, where N is the maximum number of threads that the specified CPU can run in parallel. Following this approach, allowed us to capture different CPU utilization states at different time scales. It is worth mentioning that during the CPU stressing, the rest of the device components are going through low utilization with minimal power overhead that does not affect the overall performance of the device.

For the network interfaces, the iperf tool [4] was integrated to transmit packages between the edge device and another device acting either as a client or

[1] `http://bit.ly/3LvNO76` [2] `http://prometheus.io`
[3] `http://linux.die.net/man/1/stress` [4] `http://iperf.fr/`

server, that is connected to the same local network via either wired or wireless communication channels. For the purposes of this test, in one case the device under test acted as a client, sending network packets at a specified rate while measuring outward traffic. In the second case, the device acted as a server receiving packets at a specified rate while monitoring inward traffic. For the automation of the stressing procedure, the stressor gradually increases the bandwidth speed and runs each iteration for a total of 15 minutes in the case of WiFi which has limited bandwidth availability and 10 minutes in the case of Ethernet by default.

**Data Collection, Cleaning & Preprocessing.** Our method measures power consumption by first determining the idle power consumption of the system when all components are idle and monitoring services are active. The device collects power measurements from the power monitoring device via the WiFi channel and communication with the storage agent is done via the Ethernet. Once idle power consumption is obtained, we collect CPU benchmarking results using relevant timestamps. For network power estimation, we execute network benchmarks and gather network and CPU utilization statistics, as well as the overall power consumption of the device. To estimate power consumption specific to network components, we subtract idle power consumption and CPU-related power consumption using the generated CPU power model. Due to asynchronous data collection, a minimal number of outliers are observed. By default, outliers are defined as points more than 3 standard deviations from the mean and are removed prior to the training procedure for both training and evaluation data.

**Models' Training & Evaluation.** In the models' generation step, the users can specify the type of models they want to create, including additive models consisting of separate models for each of the individual device's sub-components, multiple regression models that take as input the set of input variables representing resource utilization metrics in this case, or ensemble models like Random Forests, along with their parameters for training and tuning. Users can also select specific monitoring metrics from the device's sub-components to be included in the modeling procedure. Since our methodology is agnostic to ML and mathematical models, users can make use of different libraries and model types including among others Random Forest, Linear Regression, or even Deep Learning (DL) methods. With the models, possible parameters, and input defined, the next step is the training of the selected models. Once the training procedure is complete, the generated models are returned to the user along with the respective performance indicators, such as MSE and MAPE. Before evaluating the models that were found to be more accurate, users can fine-tune the parameters of the respective models with the use of any tools dedicated to hyperparameters' auto-tuning. An example parameter is the number of estimators used in the construction of a random forest. This flexibility allows users to generate power models tailored to edge devices, adjusting the models' parameters and optimizing their accuracy.

Except for the simple or more advanced ML approaches, the Model Generation module allows users to create an additive power model. Eq. 1 defines an additive power model for CPU and network sub-components where $P_{CPU}(u)$

represents the CPU's power consumption when it is in use at a certain utilization rate $u$ in %, $P_{eth,dn}(r)$, $P_{eth,up}(r)$, $P_{wifi,up}(r)$, $P_{wifi,dn}(r)$ represent the power consumption of Ethernet and WiFi download and upload links when in use at a certain rate $r$ in kb/s, and $P_{idle}$ the idle state power consumption in Watts.

$$P_{system} = P_{CPU}(u)+P_{eth,dn}(r)+P_{eth,up}(r)+P_{wifi,up}(r)+P_{wifi,dn}(r)+P_{idle} \quad (1)$$

**Models' Evaluation on a Real World Application.** Our methodology supports the evaluation of models using containerized workloads, offering flexibility, and abstracting the underlying device's heterogeneity, without resource overhead [8]. By default, our approach utilizes two popular ML inference workloads [7]. The first is the Image Classification and Detection (ICD) workload [12], part of the MLPerf benchmark suite, which includes real-world ML applications for inference. Users can customize workload parameters such as image count, inference model, and ML backend. Our example configuration in section 4 employs 1000 images for inference with the resnet50 model [5]. The ML backend, like TensorFlow, provides tools and APIs for ML model building and training. To ensure compatibility with ARM-based edge devices, we made the necessary adjustments to the Dockerfile of the selected workload. Our second workload involves a lightweight Python server [6] exposing an API for a simple yet realistic ML inference service. A workload generator disseminates images over the network for inference. The edge device under test runs the service, while the image workload generator acts as a client, loading images into memory and sending them for inference over the network. The devices are interconnected via Ethernet and WiFi networks. The user applies the most accurate power models from the model generation step to the collected data during workload execution, comparing estimated and actual power consumption. The output includes error metrics and plots demonstrating model performance on independent workloads.

## 4 Real World Use Case

### 4.1 Experimental Setup

The experimental setup is comprised of a Raspberry Pi 4 that serves as the System Under Test (SUT) and is powered by the Meross Smart Plug. The device is equipped with a quad-core ARM Cortex-A72 CPU, 4GB of RAM, and a 32GB SD card for storage. Raspberry Pi 4 is considered an edge device that can be used by a variety of edge applications including ML applications. The SUT is equipped with a Gigabit Ethernet interface with a bandwidth of 1 Gbps and an integrated dual channel (2.4 GHz and 5 GHz) wireless antenna. The device's idle power consumption was found to be 3.10W.

### 4.2 Generated Models

For the purposes of this study, we created i) an additive model that is composed of CPU and network models and two regression models, ii) a random forest

[5] `http://bit.ly/45Dzbr0` [6] `http://flask.palletsprojects.com`

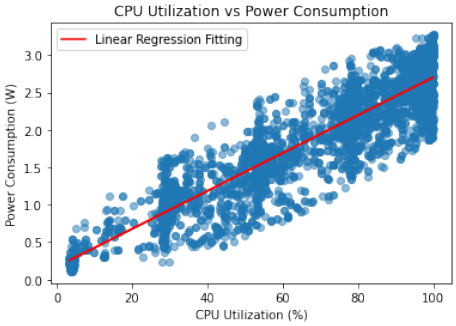


**Fig. 3.** Distribution of CPU utilization and power consumption with fitted line

with two hyperparameters (number of estimators and max depth) for which we applied hyperparameter tuning using the Ray framework [7], and iii) a linear regression model. Both regression models take as input the set of parameters and as output the estimated power consumption of the device. The training of the two regression models was performed using scikit-learn[8], by applying cross-validation, splitting input dataset at 80% training and 20% test data. For the additive models, we fitted polynomials with multiple degrees, using the numpy polyfit function [9] and we selected the ones with the lowest MSE and MAPE.

**CPU:** Fig. 3 shows the distribution of CPU utilization and power consumption of the SUT. The set of points is distributed across all the CPU utilization values. The red line represents the regression line that was fitted to the data to represent the linear relation of the CPU utilization relevant to the respective power consumption. It was found that the linear regression model had an MSE of 0.098 and a MAPE of 17% when applied to the collected data. It can also be observed that the maximum power drawn by the CPU component is just above 3 Watts. The linear regression model for CPU utilization and Power Consumption in Watts ($P_{CPU}(u)$) is represented in the equation Eq. 2 below, where parameter $u$ represents the CPU utilization (%).

$$P_{CPU}(u) = 0.025u + 0.17 \text{ W} \tag{2}$$

**Network:** In Fig. 4, we provide the distribution of Ethernet and WiFi utilization rates relevant to the respective power consumption along with a red line that represents the polynomial regression model fitted to the data in each case. The observed gaps on all the diagrams are due to the scaling we perform for sampling for rates above 100Mbps in order to obtain an even distribution of sample rates. In the case of WiFi we were able to achieve a rate of a maximum of 25Mbps for download and 50Mbps for upload links.

In the case of Ethernet, it was found that the best fit was a third-degree model with an MSE of 0.0035 and 0.004, which correspond to 2.95% and 4.61% MAPE for download and upload links respectively. For low values of network utilization, the estimated power overhead is negative due to the error that exists in the estimation of CPU power overhead that overestimates the power consumption in

[7] `https://ray.io/` [8] `https://scikit-learn.org/` [9] `https://bit.ly/3HE06ZK`

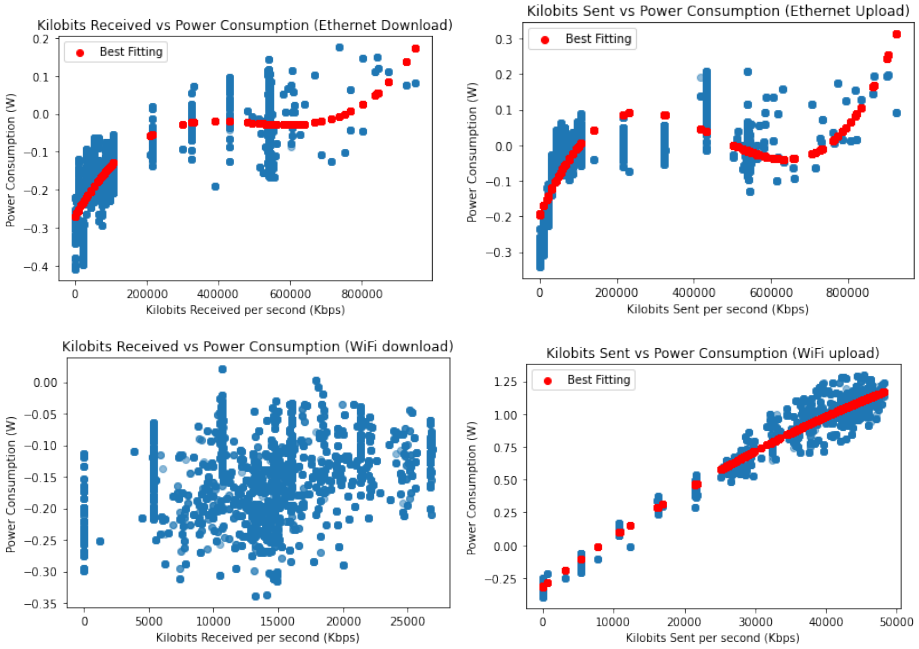


**Fig. 4.** Distribution of Network Resource Utilization and Power Consumption

some cases where CPU utilization is low. Overall, we conclude that the Ethernet power overhead is minimal even in the case that the link is undergoing full utilization in which case the increase of power consumption is relatively low. Another thing that is important to note here, is that packet loss is observed during the transmission of packets through the Ethernet channel, especially at high rates where the loss can reach up to 3.5% of the total packages. This can add some noise to the data. On average the maximum allowed stable bandwidth we could achieve was at 600Mbps. From the top plots in Fig. 4, we can observe that in the case of upload, there is a deviation downwards at 500Mbps while in the case of upload, there is a flattening up to the same rate before the power consumption is increased again. One intuitively understands that the system follows an optimization strategy when sending and receiving data over the link.

In the case of WiFi, due to the high amount of packet loss that exists on the network link, we have not been able to identify a proper distribution in the case of the receiver which was acting as the iperf server so we decided to omit it from our additive model since the power overhead was minimal and close to idle value. Also, the available bandwidth was significantly lower than in the case of Ethernet. This was mainly caused by the weakness of the onboard WiFi antenna to perform well at higher speeds. The usage of an external WiFi module would possibly fix this limitation although it would cost an extra overhead for powering it on. One thing that is noticeable in the case of WiFi is the power consumption that is imposed in the case of heavy load during upload that accounts for a non-

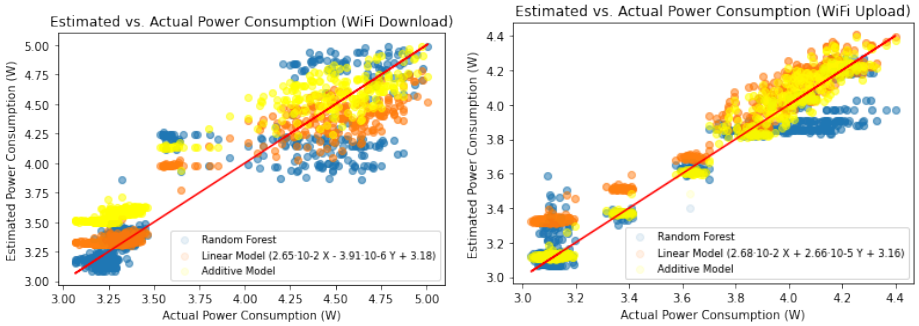


**Fig. 5.** Actual vs estimated power consumption for inference over WiFi

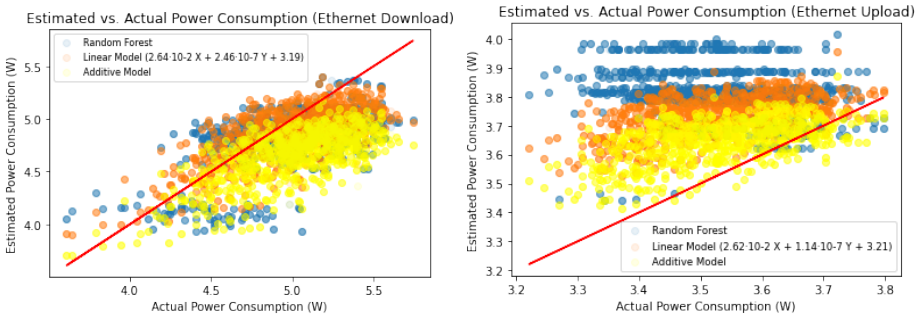


**Fig. 6.** Actual vs estimated power consumption for inference over Ethernet

negligible 1.5W marginal power. The best fit for WiFi upload is a second-degree polynomial with an MSE of 0.0054 and a MAPE of 0.015%.

The generated polynomial models for network interfaces are provided below where $r$ represents the network utilization rate at kilobits sent or received per second and functions $P_{eth,dn}$, $P_{eth,up}$, $P_{wifi,up}$ the respective power consumption in Watts for Ethernet download, Ethernet upload and WiFi upload links.

$$P_{eth,dn}(r) = 2.39 \cdot 10^{-18} \cdot r^3 - 3.51 \cdot 10^{-12} \cdot r^2 + 1.65 \cdot 10^{-6} \cdot r - 1.7 \cdot 10^{-1} \text{ W} \quad (3)$$

$$P_{eth,up}(r) = 5.04 \cdot 10^{-18} \cdot r^3 - 6.82 \cdot 10^{-12} \cdot r^2 + 2.55 \cdot 10^{-6} \cdot r - 1.96 \cdot 10^{-1} \text{ W} \quad (4)$$

$$P_{wifi,up}(r) = -2.03 \cdot 10^{-10} \cdot r^2 + 4.06 \cdot 10^{-5} \cdot r - 3.16 \cdot 10^{-1} \text{ W} \quad (5)$$

### 4.3 Evaluation of the results

**Inference over WiFi:** Fig. 5 presents the experimental results after applying our models to the set of data collected during the execution of the inference workload over WiFi. The linear regression equation is included in the legends of the graphs where X represents CPU utilization (%) and Y the network utilization rate (Kb/s). The number of data points retrieved from the storage agent for both the sender and the receiver was totaled at 5400 in order to ensure a good sample size while avoiding missing important information. The red line represents the perfect prediction where the estimated value equals the actual value. In the case of the sender, the additive model performed better than the rest with an

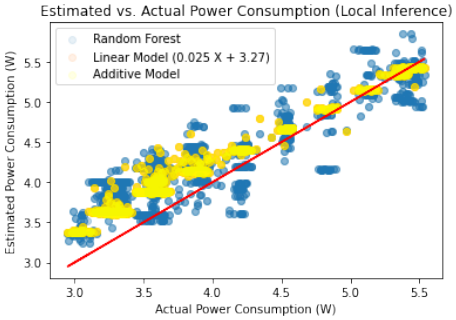


**Fig. 7.** Actual vs estimated power consumption for inference on the device

MSE of 0.0035 and a MAPE of 1.15%, while in the case of the receiver, the linear regression model outscored the rest with an MSE of 0.0288 and MAPE of 3.44%. In terms of the estimated energy, the additive model reported a total of 36469.02 J while the actual reported energy was 36310.92 J and the linear regression model reported a total of 38538.44 J compared to the total actual consumption of 38244.78 J. This is due to the high error that is observed for the additive model for low power consumption values. Also, for high power consumption values, the Random Forest cannot perform as expected. This might be due to the linear relationship between CPU utilization and power consumption which can make it difficult to be accurately captured by the Random Forest. To this end, CPU utilization is the main contributor to the system's power consumption which is better captured by the additive and linear regression models.

**Inference over Ethernet:** The relevant results for the execution of the inference workload over Ethernet are presented in Fig. 6. The same number of samples as in the case of WiFi was collected, for the same reason. In the case of download over Ethernet, all the models tend to underestimate the predicted value while in the case of upload, they tend to overestimate it in most of the cases. The reported results indicated that the additive model was by far the most efficient for estimating the power for images loaded and sent over Ethernet with a recorded MSE of 0.0227 and a MAPE of 3.65% while on the other side for the receiver performing the inference, the Linear Regression model recorded the lowest error with an MSE of 0.0739 and 4.33% MAPE. The energy consumption values of the aforementioned models were estimated to be 36744.10 J and 48712.32 J in the cases of the sender and the receiver while the actual reported energy consumption was found to be 35552.26 J and 49887.45 J respectively.

**Inference on the device:** For local inference, the experiment was repeated at different CPU utilization levels by limiting the CPU utilization of the container in each iteration at a scale of 10%. As we lower the CPU utilization, the execution time increases and that is why the majority of data points lie in the area between 3 and 4 Watts. The total number of points collected in this case was 10800, due to the multiple iterations performed. The additive and linear regression models return the same results since there is no network overhead. Evaluation results for this scenario are illustrated in Fig. 7. One thing to point out is the fact that the linear model performs really well at high values where utilization is

high while slightly overestimating the values close to idle resource utilization. In this case, the MSE and MAPE were 0.1003 and 8.06%, respectively and the generated models estimated a total of 87254.52 J while the total reported energy consumption based on the power meter readings was 81363.12 J.

## 5 Conclusion

This work presented a methodology for generating and evaluating power consumption models for edge devices. Our methodology can generate both additive and regression models, and we highlighted its effectiveness by creating accurate power consumption models for a Raspberry Pi 4, and by performing ML inference from a real-world application. Our methodology is useful for creating and evaluating power models for a wide range of edge devices, enabling better energy efficiency and sustainability. Future work includes the creation of an automated tool adopting the pipeline of the proposed methodology, and making use of it for more devices and attached components, like cellular communication. Moreover, we plan to conduct an evaluation of the power consumption of edge applications by integrating power consumption models with evaluation tools like emulators.

## References

1. Cisco annual internet report (2018–2023). Tech. rep., Cisco (2018)
2. How to optimize the netdata agent's performance. https://learn.netdata.cloud/guides/configure/performance (2023)
3. Alan, I., Arslan, E., Kosar, T.: Energy-performance trade-offs in data transfer tuning at the end-systems. Sustainable Computing: Informatics And Systems (2014)
4. Alsop, T.: Number of edge enabled internet of things (iot) devices worldwide from 2020 to 2030, by market. Tech. rep. (12 2022), `www.statista.com/statistics/1259878/edge-enabled-iot-device-market-worldwide/`
5. Bekaroo, G., Santokhee, A.: Power consumption of the raspberry pi: A comparative analysis. In: IEEE International Conference On Emerging Technologies And Innovative Business Practices For The Transformation Of Societies (EmergiTech). pp. 361–366 (2016)
6. Delicato, F., Pires, P., Batista, T., Delicato, F., Pires, P., Batista, T.: The resource management challenge in iot. Resource Management For Internet Of Things (2017)
7. Earney, S.: What is edge computer vision, and how does it work? http://xailient.com/blog/what-is-edge-computer-vision-and-how-does-it-work/
8. Georgiou, J., Symeonides, M., Kasioulis, M., Trihinas, D., Pallis, G., Dikaiakos, M.D.: Benchpilot: Repeatable & reproducible benchmarking for edge micro-dcs. In: IEEE ISCC (2022)
9. Kang, P., Jo, J.: Benchmarking modern edge devices for ai applications. IEICE TRANSACTIONS On Information And Systems **104**, 394–403 (2021)
10. Kaup, F., Hacker, S., Mentzendorff, E., Meurisch, C., Hausheer, D.: The progress of the energy-efficiency of single-board computers. Tech. rep., Netsys (2018)
11. Paniego, J., Libutti, L., Puig, M., Chichizola, F., De Giusti, L., Naiouf, M., De Giusti, A.: Unified power modeling design for various raspberry pi generations analyzing different statistical methods. In: Computer Science–CACIC 2019: 25th Argentine Congress Of Computer Science, CACIC 2019. pp. 53–65 (2020)
12. Reddi, V., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C., Anderson, B., Breughe, M., Charlebois, M., Chou, W.: Mlperf inference benchmark. In: ACM/IEEE ISCA (2020)