

Titre: Power Estimation Framework to Ensure the Reliability of Edge
Title: Devices in Internet-of-Things Applications

Auteur: Mahsa Hadian
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Hadian, M. (2023). Power Estimation Framework to Ensure the Reliability of Edge
Citation: Devices in Internet-of-Things Applications [Thèse de doctorat, Polytechnique
Montréal]. PolyPublie. <https://publications.polymtl.ca/54386/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/54386/>
PolyPublie URL:

**Directeurs de
recherche:** Giovanni Beltrame, & Marios-Eleftherios Fokaefs
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Power Estimation Framework to Ensure the Reliability of Edge Devices in
Internet-of-Things Applications**

MAHSA HADIAN

Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Juillet 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Power Estimation Framework to Ensure the Reliability of Edge Devices in
Internet-of-Things Applications**

présentée par **Mahsa HADIAN**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Foutse KHOMH, président

Giovanni BELTRAME, membre et directeur de recherche

Marios-Eleftherios FOKAEFS, membre et codirecteur de recherche

Heng LI, membre

Manos PAPAGELIS, membre externe

DEDICATION

*Women of Iran,
Heroes of the Year 2022. . .*

ACKNOWLEDGEMENTS

I am extremely grateful to my supervisor, Prof. Marios Fokaefs, for his invaluable advice, continuous support, and patience during my Ph.D. studies. His immense knowledge and plentiful experience have encouraged me throughout my academic research and daily life. I would also like to express my deepest gratitude to my husband. Without his tremendous support, understanding, and encouragement over the past few years, it would have been impossible for me to complete my studies.

RÉSUMÉ

En 2022, le marché mondial de l’informatique de pointe était estimé à 11,24 milliards USD, avec un taux de croissance projeté de 37,9% de 2023 à 2030. Le principe de l’edge computing est que les données doivent être analysées aussi près que possible de leur production. Cela se traduit par un traitement plus rapide et un temps de réponse de l’application plus rapide.

Cependant, les périphériques périphériques et les nœuds périphériques ont généralement des capacités réduites par rapport aux centres de données. La répartition des charges de travail de données et de calcul sur différents niveaux (par exemple, cloud ou périphérique périphérique) peut avoir un impact important sur les performances et l’utilisation des ressources de l’ensemble du système et de son infrastructure. De plus, un manque de fiabilité dû à l’épuisement des batteries ou à l’inaccessibilité de l’alimentation est un autre problème potentiel. En particulier lorsqu’un périphérique périphérique effectue des analyses de données complexes en périphérie, la consommation d’énergie est une ressource particulièrement critique pour les périphériques périphériques. La consommation d’énergie peut être influencée par divers facteurs, tels que le volume de données d’entrée, l’utilisation des ressources et la topologie du réseau informatique de périphérie. L’incertitude de la consommation d’énergie peut entraîner des problèmes de planification, d’allocation des ressources et de fiabilité du système. Afin de maintenir la fiabilité et la fonctionnalité des plates-formes informatiques de pointe, une estimation et une gestion efficaces de la consommation d’énergie, en particulier dans les nœuds à énergie limitée, sont nécessaires.

Compte tenu de cela et pour surmonter ces défis, dans ce travail, (1) nous présentons d’abord un outil d’analyse comparative des performances et de l’énergie pour les plates-formes informatiques de périphérie (PEBEdge) dans les applications de l’Internet des objets (IoT) afin d’évaluer les performances globales et la consommation d’énergie. Le système de référence de PEBEdge est composé de plusieurs niveaux, y compris un niveau de capteur (c’est-à-dire un générateur de charge de travail), un niveau de périphérie (y compris les couches d’application et d’analyse comparative) et un niveau cloud (pour le stockage de données et l’analyse avancée). Cet outil se concentre particulièrement sur les dispositifs périphériques, qui portent les contraintes les plus importantes. PEBEdge nous permet de définir facilement différentes charges de travail qui correspondent à différentes topologies de capteurs et autres déploiements impliquant des appareils périphériques, des nœuds périphériques et le cloud. PEBEdge surveille et collecte l’utilisation des ressources sous forme de métriques de performances système telles que le processeur, la mémoire, les E/S de disque et le réseau, ainsi que

la consommation d'énergie au niveau des périphériques périphériques. PEBEdge permet de configurer et de déployer différentes applications de traitement de données et d'exécuter des expériences pour évaluer les performances et la consommation d'énergie de l'infrastructure pour des charges de travail et des applications données. Il est conçu pour que chaque composant soit implémenté et virtualisé en tant que conteneur, permettant une instanciation, une personnalisation et un remplacement d'exécution faciles avec une implémentation différente utilisant une image différente au moment de l'exécution. Avec les conteneurs Docker, le framework fournit également des capacités d'orchestration robustes pour les expériences à grande échelle et collecte des mesures cohérentes de performance et d'utilisation de l'énergie à travers les déploiements. Le framework PEBEdge nécessite à la fois des composants physiques et logiciels. Étant donné que PEBEdge vise à mesurer la consommation d'énergie sur l'appareil de périphérie, des composants physiques spécifiques sont nécessaires. Pour permettre une plus grande représentativité, flexibilité et extensibilité, tous les autres composants (par exemple, les capteurs) sont virtualisés.

Dans cette thèse, (2) après avoir identifié les paramètres associés au logiciel ayant un impact sur la consommation d'énergie de l'appareil de périphérie, un modèle d'estimation de puissance est construit pour étudier comment différents paramètres (comme les capteurs d'entrée, les numéros de conteneur, les unités de traitement, le CPU, la mémoire, le réseau et le disque) sont corrélées à la consommation d'énergie des appareils périphériques. Dans cette thèse, nous proposons l'utilisation de modèles d'apprentissage automatique pour l'estimation de la consommation d'énergie. Par rapport aux méthodes qui utilisent des modèles analytiques (c'est-à-dire mathématiques ou statistiques) pour estimer la consommation d'énergie (par exemple, WattEdge), les modèles d'apprentissage automatique ont l'avantage d'être indépendants du matériel et généralement applicables. Dans les cas liés aux logiciels, lorsqu'il est nécessaire de se concentrer sur le recyclage des données, les algorithmes d'apprentissage automatique sont une solution supérieure. Parce que les approches d'apprentissage automatique rendent la formation aux données beaucoup plus facile et plus rapide. Inversement, les méthodes analytiques produisent souvent des résultats plus précis, nécessitent moins de ressources informatiques et sont plus interprétables. Différents modèles d'apprentissage automatique ont été utilisés dans les expériences, notamment eXtreme Gradient Boosting, Random Forest, Decision Tree, Linear Regression, AdaBoost Regressor, K-plus proche voisin et des modèles de réseau neuronal, notamment Long Short-Term Memory, Multilayer Perceptron et Dummy Régresseur comme ligne de base de régression. Le processus de formation d'un tel modèle est décrit en détail, y compris comment collecter des données et sélectionner le meilleur modèle parmi une variété d'algorithmes. Les expériences sont menées à l'aide d'un ensemble de données qui comprend 381 232 échantillons collectés à partir de 35 types d'expériences. Sur

la base des mesures de performance, les modèles d'apprentissage automatique semblent être plus performants que les modèles de réseau neuronal dans les résultats présentés. L'arbre de décision et la forêt aléatoire présentent d'excellentes performances parmi les algorithmes d'apprentissage automatique. De plus, la quantité estimée de consommation d'énergie de l'appareil de périphérie nous aide à estimer le temps d'épuisement de la batterie. À l'aide des modèles d'estimation de puissance, nous trouvons le facteur logiciel le plus important qui nous permet de prendre des décisions concernant la modification de la configuration du système lorsque l'appareil de périphérie est au niveau critique de la batterie ; nous déterminons le temps d'épuisement approximatif de la batterie de l'appareil périphérique en fonction des modèles de consommation d'énergie estimés.

Finalement, dans cette recherche, (3) nous offrons aux chercheurs la possibilité d'effectuer des expériences concernant la consommation d'énergie des périphériques de périphérie avec beaucoup moins de temps et sans aucun matériel requis comme un périphérique de périphérie et un wattmètre.

ABSTRACT

As of 2022, the global edge computing market was estimated to be worth USD 11.24 billion, with a projected growth rate of 37.9% from 2023 to 2030. The premise of edge computing is that data should be analyzed as close to its production as possible. This results in faster processing and a quicker application response time.

However, edge devices and edge nodes typically have reduced capabilities compared to data centers. The distribution of data and computation workloads on different tiers (e.g., cloud or edge device) can highly impact the performance and resource utilization of the entire system and its infrastructure. Moreover, a lack of reliability due to the depletion of the batteries or the inaccessibility of power is another potential issue. Especially when an edge device performs complex data analytics on the edge, power consumption is a particularly critical resource for edge devices. Power consumption can be influenced by a variety of factors, such as input data volume, resource utilization, and the topology of the edge computing network. Uncertainty in power consumption can cause issues with planning, resource allocation, and system reliability. In order to maintain the reliability and functionality of edge computing platforms, effective estimation and management of power consumption, especially in energy-constrained nodes, are necessary.

In view of this and to overcome these challenges, in this work, (1) we first present a performance and energy benchmarking tool for edge computing platforms (PEBEdge) in Internet of Things (IoT) applications to benchmark the overall performance and power consumption. PEBEdge’s benchmark system is comprised of multiple tiers, including a sensor tier (i.e., workload generator), an edge tier (including the application and benchmarking layers), and a cloud tier (for data storage and advanced analytics). This tool focuses particularly on edge devices, which carry the most substantial constraints. PEBEdge allows us to easily define different workloads that correspond to different sensor topologies and other deployments involving edge devices, edge nodes, and the cloud. PEBEdge monitors and collects resource utilization as system performance metrics like CPU, memory, disk I/O, and network, as well as power consumption on the edge device tier. PEBEdge allows configuring and deploying different data processing applications and running experiments to evaluate the performance and power consumption of the infrastructure for given workloads and applications. It is designed so that each component is implemented and virtualized as a container, allowing easy instantiation, customization, and runtime replacement with a different implementation using a different image at runtime. With docker containers, the framework also provides robust

orchestration capabilities for large-scale experiments and collects consistent performance and power usage metrics across deployments. The PEBEdge framework requires both physical and software components. Since PEBEdge aims to measure power consumption on the edge device, specific physical components are required. To allow for greater representativeness, flexibility, and extensibility, all other components (e.g., sensors) are virtualized.

In this thesis, (2) after identifying software-associated parameters impacting the edge device’s power consumption, a power estimation model is built to study how different parameters (like input sensors, container numbers, process units, CPU, memory, network, and disk) are co-related with edge devices’ power consumption. In this thesis, we propose the use of machine learning models for the estimation of power consumption. Compared with the methods that use analytical (i.e., mathematical or statistical) models for estimating power consumption (e.g., WattEdge), machine learning models have the advantage of being hardware independent and generally applicable. In software-related cases, when it is necessary to focus on retraining data, machine learning algorithms are a superior solution. Because machine learning approaches make data training much easier and faster. Conversely, analytical methods often produce more accurate results, require fewer computational resources, and are more interpretable. Different machine learning models have been utilized in the experiments, including eXtreme Gradient Boosting, Random Forest, Decision Tree, Linear Regression, AdaBoost Regressor, K-nearest neighbor, and neural network models, including Long Short-Term Memory, Multilayer Perceptron, and Dummy Regressor as a regression baseline. The process for training such a model is described in detail, including how to gather data and select the best model from a variety of algorithms. The experiments are conducted using a dataset that includes 381,232 samples collected from 35 types of experiments. On the basis of the performance metrics, machine learning models appear to perform better than neural network models in the presented results. Decision Tree and Random Forest exhibit excellent performance among the machine learning algorithms. In addition, the estimated amount of edge device power consumption helps us estimate the battery depletion time. Using the power estimation models, we find the most significant software-associated factor that allows us to make decisions about changing system configuration when the edge device is at the critical battery level; we determine the approximate edge device battery depletion time according to the estimated power consumption models.

Eventually, in this research, (3) we provide the researchers with the possibility to perform experiments regarding edge device power consumption with considerably less time and without any required hardware like an edge device and power meter.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	viii
TABLE OF CONTENTS	x
LIST OF TABLES	xiii
LIST OF FIGURES	xv
LIST OF SYMBOLS AND ACRONYMS	xvi
LIST OF APPENDICES	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Context and Motivation	1
1.2 Problem Statement	2
1.3 Research Objectives	3
1.3.1 Objective 1: Creating a Benchmarking Platform and Building a Prototype Infrastructure.	3
1.3.2 Objective 2: Building a Power Estimation Model and Providing Solutions for Power Planning Requirements.	4
1.3.3 Main Objective: Empower decision-makers by Providing Efficient Planning Support, Enabling Rapid Experimentation, and Facilitating the Simulation of "What-if" Scenarios with Diverse Configurations.	5
1.4 Research Contributions: A Pipeline and a Platform (Benchmark and Model) that Support Efficient Planning for IoT Applications on the Edge	5
CHAPTER 2 BACKGROUND AND LITERATURE REVIEW.	7
2.1 Cyber-Physical Systems and IoT Applications	7
2.2 Cloud Computing	9
2.3 Edge Computing	10

2.4	Edge Devices Benchmarking	11
2.5	Performance and EEnergy Consumption Benchmarking	12
2.6	Estimation Models for Edge Computing Platforms	14
CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION		16
3.1	Methodology	16
3.2	Document Structure	17
CHAPTER 4 PEBEdge: A PERFORMANCE AND ENERGY BENCHMARKING TOOL		18
4.1	Introduction	18
4.2	State-of-the-art Approaches	19
4.3	PEBEdge	19
4.4	Architecture	20
4.4.1	Sensor Tier - Workload Generator	20
4.4.2	Edge Tier	22
4.4.3	Cloud Tier:	24
4.4.4	QoL Features	24
4.4.5	Capabilities	25
4.5	Case Study	25
4.5.1	Motivation Example	25
4.5.2	Research Question	26
4.5.3	System Configuration	26
4.5.4	Running Experiments	31
4.5.5	Performing Experiments Considerations	33
4.5.6	Results	33
4.6	Discussion	36
CHAPTER 5 ML-BASED POWER ESTIMATION MODEL		37
5.1	Introduction	37
5.2	State-of-the-art Approaches	37
5.3	ML Algorithms	38
5.3.1	Dataset	39
5.3.2	Measuring Baseline	41
5.3.3	Evaluation Metrics	41
5.3.4	Experimental Results	42
5.3.5	Results Comparison	58

5.3.6	Significance of Error	60
5.4	Suggested Solutions for Improving the Lack of Reliability	61
5.4.1	Identifying the Most Important Parameters Affecting the Power Consumption	61
5.4.2	Potential Software Configuration Adjustment for Managing Power Consumption	65
5.4.3	A Case Study For Battery Depletion Calculation	65
5.5	Discussion	68
CHAPTER 6	LIGHTWEIGHT BENCHMARKING FOR POWER CONSUMPTION	69
6.1	Introduction	69
6.2	Lightweight Benchmarking	69
6.3	Case Study	70
6.3.1	Motivation Example	70
6.3.2	Research question	70
6.3.3	Running Experiment	71
6.4	Discussion	72
CHAPTER 7	DISCUSSION	73
CHAPTER 8	CONCLUSION	78
REFERENCES		80
APPENDICES		92

LIST OF TABLES

Table 4.1	Data generator required configuration values.	27
Table 4.2	MQTT client adapter required configuration value	27
Table 4.3	Spark processor configuration.	29
Table 4.4	Python script processor configuration.	30
Table 4.5	Mosquitto configuration.	31
Table 5.1	Different scenarios and the corresponding data.	40
Table 5.2	Sample dataset for scenario 1.	41
Table 5.3	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the DR algorithm.	44
Table 5.4	DR algorithm metric results on the distinct test dataset.	44
Table 5.5	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the LR algorithm.	45
Table 5.6	LR algorithm metric results on the distinct test dataset.	45
Table 5.7	Defined hyperparameters (including max depth, min samples split, min samples leaf, and max features.) for search space and best-achieved hyperparameters combination for DT.	46
Table 5.8	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the DT algorithm.	47
Table 5.9	DT algorithm metric results on the distinct test dataset.	47
Table 5.10	Defined hyperparameters (including n estimators, max depth, min samples split, min samples leaf, and max features.) for search space and best-achieved hyperparameters combination for RF.	48
Table 5.11	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the RF algorithm.	48
Table 5.12	RF algorithm metric results on the distinct test dataset.	49
Table 5.13	Defined hyperparameters (including n estimators, learning rate, max depth, subsample, and colsample bytree) for search space and best-achieved hyperparameters combination for XGB.	50
Table 5.14	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the XGB algorithm.	50
Table 5.15	XGB algorithm metric results on the distinct test dataset.	51

Table 5.16	Defined hyperparameters (including n estimators, learning rate, and loss.) for search space and best-achieved hyperparameters combination for AR.	51
Table 5.17	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the AR algorithm.	52
Table 5.18	AR algorithm metric results on the distinct test dataset.	52
Table 5.19	Defined hyperparameters (including n neighbors, weights, and "p") for search space and best-achieved hyperparameters combination for KNN.	53
Table 5.20	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the KNN algorithm.	53
Table 5.21	KNN algorithm metric results on the distinct test dataset.	54
Table 5.22	Defined hyperparameters (including neurons, hidden layers, batch size, and epochs) for search space and best-achieved hyperparameters combination for MLP.	54
Table 5.23	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the MLP algorithm.	55
Table 5.24	MLP algorithm metric results on the distinct test dataset.	56
Table 5.25	Defined hyperparameters (including neurons, batch size, and epochs) for search space and best-achieved hyperparameters combination for LSTM.	57
Table 5.26	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the LSTM algorithm.	58
Table 5.27	LSTM algorithm metric results on the distinct test dataset.	58
Table 5.28	Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the XGB, RF, DT, LR, AR, and KNN algorithms.	59
Table 5.29	Feature importance based on the RF algorithm calculation.	62
Table 5.30	Feature importance based on the DT algorithm calculation.	63
Table 5.31	Coefficients based on the LR algorithm calculation.	64

LIST OF FIGURES

Figure 4.1	PEBEdge architecture showing the three tiers and their inter-operate for benchmarking the performance and power consumption of the edge device.	20
Figure 4.2	USB power meter (figure a) and its UI (figure b).	24
Figure 4.3	Results (for one camera sensor) of the experiment using a Python script with the experiment using Spark streaming.	35
Figure 4.4	Results (for one camera sensor) of the experiment using Spark streaming with 1, 2, and 3 workers.	35
Figure 5.1	MLP algorithm progression of MSE for training in 128 neurons, 2 hidden layers, 50 epochs, 500 batch size.	55
Figure 5.2	LSTM algorithm progression of MSE during training in 100 neurons, 50 epochs, and 500 batch size.	57
Figure 5.3	MSE and MAE comparison for the trained models.	60
Figure 6.1	Comparison results of the physical and model-based experiment (Python script).	72
Figure A.1	Results for the ready dataset.	93
Figure A.2	Resource utilization name.	97
Figure A.3	Running monitoring script.	97

LIST OF SYMBOLS AND ACRONYMS

IoT	Internet of Things
PEBEdge	Performance and Energy Benchmarking tool for Edge
ML	Machine Learning
XGB	Extreme Gradient Boosting
RF	Random Forest
DT	Decision Tree
LR	Linear Regression
AR	AdaBoost Regressor
KNN	K-Nearest Neighbor
NN	Neural Network
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
CPS	Cyber-Physical System
HVAC	Heating, Ventilation, and Air-Conditioning
BMS	Building Management System
VM	Virtual Machine
QoE	Quality-of-Experience
QoS	Quality-of-Service
QoL	Quality-of-Life
IoTEF	Internet of Things Edge-cloud Federation
DSPS	Distributed Stream Processing System
WSN	Wireless Sensor Network
IoMT	Internet of Medical Things
JVM	Java Virtual Machine
DVMC	Dynamic Virtual Machine Consolidation
API	Application Programming Interface
RPi	Raspberry Pi
MQTT	Message Queuing Telemetry Transport
M2M	Machine-to-Machine
DAG	Directed Acyclic Graph
MSE	Mean Square Error
MAE	Mean Absolute Error
AMS	Autonomic Management Systems

SLO	Service Level Objective
DR	Dummy Regressor
R2	R-squared Error
RNN	Recurrent Neural Network

LIST OF APPENDICES

Appendix A	A Detailed Description of the Steps Involved in Conducting the Experiment	92
------------	---	----

CHAPTER 1 INTRODUCTION

The writing of this dissertation satisfies a portion of the criteria for the degree of Doctor of Philosophy in computer engineering. Between May 2019 and July 2023, this research project was carried out at Polytechnique Montréal, Quebec, Canada.

1.1 Context and Motivation

Thanks to the growing capabilities of portable devices and the advances in networking and computation, the concept of connected smart devices, better known as the Internet of Things (IoT), has gained considerable traction in recent years. IoT applications have revolutionized everyday life, bringing tremendous value to both individuals and businesses [1]. Insights into IoT proliferation [2] report that the number of connected devices in 2021 was set to hit 46 billion. Also, according to statistics, the total data volume of connected IoT devices will reach 79.4 zettabytes by 2025 [3].

Basically, IoT requires network connectivity and context-aware computation using network resources [4]. Also, with the development of various large-scale real-time applications, real-time processing has become a fundamental and crucial functionality for the IoT [5]. One example of a real-time IoT application is "smart home automation" in smart buildings setup [6], which constitutes a popular example of IoT applications and consists of cameras, security sensors, and localization sensors, among others. These sensors must continuously collect data and give real-time responses to higher-level applications like building management systems (BMS), local controllers, or cloud platforms. A smart residential center must collect various data from sensors, users, or buildings and produce an alert when an unexpected circumstance is identified or automatically switch to power-saving configurations to manage the different costs like energy costs, water costs, maintenance costs, and others. A large amount of sensory data and the respective real-time processing load present considerable challenges for the practical application of large-scale IoT systems.

For a growing number of devices and sensors that are being produced, it is possible for hardware sensors to sense the physical environment and convert it into information. Data consumers can access cloud data after these massive amounts of data are transmitted to the cloud [7]. However, a centralized cloud would create a number of problems, including low throughput, high latency, bandwidth bottlenecks, and concerns about data privacy [8].

Edge computing has been widely embraced in order to address these challenges. Generally,

edge device structure refers to the overall framework, architecture, or arrangement of the edge computing system. It describes how the components, nodes, and layers are organized and interact with each other in the edge computing environment. The concept of edge computing involves a cluster of lower capacity computers, i.e., edge nodes, which are connected with a central and perhaps more powerful computer as the gateway/controller at the edge of the network [9]. Edge devices are connected to edge nodes that process local data, which in turn are potentially connected to a data center for persistent storage and aggregate processing. Increasingly, edge computing platforms are becoming fully or partially independent [10].

The main premise of edge computing is to bring analytics to the edge of the network as close as possible to terminal devices that produce the data to reduce latency and data transfer volume along the network. This makes sense for distributed systems, where some analytics (e.g., local data aggregation, local anomaly detection, and local performance monitoring) are more useful on a local level. Eventually, this decentralization has been shown to achieve a higher quality of data while also improving performance, and reducing power consumption [11, 12]. Thanks to reduced data transfer and the associated network latency, applications on the edge have reduced costs [13]. Overall, edge computing has emerged as a solution to provide trade-offs between large amounts of data storage, high computational capacity, low network latency, and real-time processing [7]. Despite the efficiency provided by local on-device processing and the low latency and privacy protection, the restrictions imposed by the resource-constrained and energy-constrained edge devices and edge nodes may lead edge topologies to suffer from poor scalability and energy-related failures due to insufficient wireless connections or a lack of battery life [7]. The capacity of edge devices and edge nodes is typically limited when compared to data centers. There can be a significant impact on the performance and resource utilization of the entire system and its infrastructure due to the distribution of data and computation workloads among different tiers. Finally, a lack of reliability may arise from the depletion of batteries or the difficulty in obtaining energy.

1.2 Problem Statement

Based on these advantages and challenges, in this context, there are a set of alternatives that decision-makers need to take into consideration. For example, the following are some important trade-offs:

- Low capacity, low latency.
- High capacity, high latency.
- Power capacity challenges, local analytics.

- No power capacity challenges, remote analytics.

According to the mentioned trade-offs, the main thesis of this work is to provide helpful planning strategies in different scenarios for decision-makers in IoT applications. There are challenges associated with the main thesis, as follows:

- Planning necessitates the availability of relevant data, the implementation of "what-if" analysis techniques, and the conduct of fast and efficient experimentation processes. This challenge involves two main aspects: the necessity to explore various settings and topologies to compare alternatives and evaluate behavior at the edge, as well as the requirement to gather data, particularly for building models. This challenge results in some problems for several reasons. Firstly, deploying edge computing can be costly. Secondly, it may not always be feasible or realistic to implement in certain situations. Lastly, the hardware dependency further complicates the matter. Moreover, since this process often requires repetition, the overall cost tends to increase.
- Planning necessitates the implementation of smart estimation techniques. Power consumption is a difficult dimension to capture. It may differ depending on multiple variables, such as the number of hardware resources used, the hardware device specifications, power meters, and other factors. In most cases, existing methods to determine power consumption rely on analytical (i.e., mathematical) models, which are highly dependent on the device being measured. Because the sensitivity, accuracy, and precision of various equipment are likely to differ significantly. Estimating the expected power consumption in various scenarios and for various workloads is necessary for taking some actions in order to improve the lack of reliability.

1.3 Research Objectives

1.3.1 Objective 1: Creating a Benchmarking Platform and Building a Prototype Infrastructure.

Objective 1 consists of creating a benchmarking platform to gather data regarding resource utilization and power consumption in edge devices, along with building a prototype infrastructure. They cover the problems of collecting data, testing a given combination of applications, and allowing for the dynamic extension of the tested environment for increased flexibility.

Chapter 4 presents the detailed steps of building this prototype infrastructure by preparing the components (e.g., VMs, RPi, power meter, and others). This is intended to present

a representative environment as realistically as possible. We use a three-tier architecture comprising the sensors, the edge layer, and a cloud layer. The communication is performed among all layers so that sensors' data are transferred to the edge tier. The edge network communicates with a centralized cloud to save benchmarking data.

1.3.2 Objective 2: Building a Power Estimation Model and Providing Solutions for Power Planning Requirements.

It estimates an approximated value of power consumption according to the input features (resource utilization and others). Existing techniques for calculating power consumption rely on analytical (mathematical) models, which are extremely specific to the device specification (e.g., sensitivity, accuracy, and precision) being measured. For the purpose of reducing complexity, in this work, we use machine learning (ML)-based algorithms for building power consumption models. In contrast to the analytical method, ML algorithms are hardware independent and generally applicable.

Chapter 5 presents the detailed steps involved in collecting the dataset, preprocessing, training, validating, and evaluating the proposed ML models. Dummy Regressor (DR) as a baseline regressor, neural network (NN) models (including Multilayer Perceptron (MLP), and Long Short-Term Memory (LSTM)), along with eXtreme Gradient Boosting (XGB), Random Forest (RF), Decision Tree (DT), Linear Regression (LR), AdaBoost Regressor (AR), K-Nearest Neighbor (KNN) are trained. We use three evaluation metrics in the experiments including mean square error (MSE), mean absolute error (MAE), and R-squared error (R2). The best models based on evaluation metrics are introduced.

In addition, employing trained ML models, in order to improve the lack of reliability imposed by energy-constrained edge devices, we present some solutions. Here, we intend to decrease the energy-related failure risks of edge devices. First, we present the procedure for identifying the most important software-related factor that enables us to manage the power consumption of edge devices. Second, we show some examples of changing the system's software-associated configuration to decrease power consumption in critical situations. And finally, using a case study, we determine the approximate edge device battery depletion time.

1.3.3 Main Objective: Empower decision-makers by Providing Efficient Planning Support, Enabling Rapid Experimentation, and Facilitating the Simulation of "What-if" Scenarios with Diverse Configurations.

The research seeks to achieve simplified configuration and deployment processes to enhance the overall effectiveness of decision-making within Internet of Things (IoT) applications. We provide low-cost and low-time experiment possibilities regarding edge device power consumption by implementing Lightweight Benchmarking. Researchers are able to perform experiments in a much shorter time and without the need for any special hardware, such as an edge device or power meter. Chapter 6 provides a case study to achieve this goal. In a case study, we show how Lightweight Benchmarking helps researchers perform experiments regarding edge device power consumption in a much shorter time and without the need for any special hardware, such as an edge device or power meter.

1.4 Research Contributions: A Pipeline and a Platform (Benchmark and Model) that Support Efficient Planning for IoT Applications on the Edge

- As the contributions of this work, we propose a performance and energy benchmarking tool for edge (PEBEdge) ¹; a benchmarking tool to evaluate and analyze the edge network in real-world scenarios where they are required to run continuously under a fluctuating and increased workload. In this work, a real-time and online monitoring dashboard tracks and measures various resource utilization metrics (CPU consumption, memory usage, network I/O, disk I/O), as well as instantaneous power consumption. PEBEdge can support further research on edge platform benchmarking and advance knowledge and expertise in this domain. By conducting a case study experiment as well as evaluation and comparison analysis, this work provides extensive insight into various implementations for future researchers.
- When it comes to ML, datasets play an important role. There are a few key characteristics of a good ML dataset: it should be large enough to be representative, of high quality, and relevant to the task at hand. An appropriate ML dataset is one that contains variables and features that are appropriately structured, has minimal noise (no irrelevant data), is scalable to large amounts of data, and is easy to analyze. In this work, by preparing the components and running several experiments, we collect a dataset consisting of 381232 samples (train and test) using PEBEdge. The dataset is

¹<https://github.com/polytechnique-ease/EdgeBenchmark>

publicly available ², ³, and it was non-trivial to gather it since we needed some specialized equipment. It is offered to other researchers to support different problems, questions, and contributions.

- We present eight state-of-the-art ML-trained models ⁴ for regression purposes that estimate the level of power consumption of edge devices. Using the best models in order to address power-related reliability issues, we provide some solutions, including identifying the most effective software-related factors of power consumption along with some configuration replacement suggestions. Furthermore, a case study in which an estimation model can be used to estimate battery depletion time on an edge device is presented. In addition, we provide the researchers with the opportunity to conduct experiments on edge device power consumption with substantially less time and without any necessary hardware like an edge device and power meter.

²https://github.com/mahsahadian/Energy_Model_Prediction/blob/main/mahsadatast3_training.csv

³https://github.com/mahsahadian/Energy_Model_Prediction/blob/main/mahsadatast3_test.csv

⁴https://github.com/mahsahadian/Energy_Model_Prediction

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

This section focuses on IoT applications, cloud computing, edge computing, edge devices energy, and performance benchmarking as well as related work regarding predicting the energy consumption and battery life of edge devices. In order to provide a reasonable comparison with these topics, this section provides an overview of the state-of-the-art in these areas.

2.1 Cyber-Physical Systems and IoT Applications

Based on the fast growth of the population density in cities, infrastructure, and services are required to satisfy the city occupants' constraints. A significant increase in digital devices that are becoming smarter and more digital has occurred to this end. Computing and communication capabilities will be set in objects and structures in physical environments [14]. Such systems that connect the cyber-world of computing with the real world are addressed as cyber-physical systems (CPS) ¹. CPSs are heterogeneous distributed systems usually including many sensors and controllers, which are connected to a pool of computing nodes [15]. CPSs intend to recognize changes in the physical environment, study the consequences of changes to their operation, and make smart decisions to reply to the changes by issuing commands to manage physical objects in the system through influencing the physical environment in an autonomous way [16].

IoT opens up effective systems for connecting devices and things efficiently. An IoT-based system can manage numerous issues that appeared as a result of the complexity of recent technologies. In other words, IoT has been developed to improve the way we deal with the environment in cyber-physical systems. The IoT allows physical objects to connect and exchange data with other systems or devices over the Internet or other communication networks. Therefore, devices and humans can interact more reliably and robustly. This led to the significant growth of IoT use in applications like smart buildings and environmental monitoring and safety over the past few years. IoT applications need less human interaction and redesign in the dynamic environments [17].

There has been an exponential increase in the application of the IoT in different areas such as smart healthcare [18], smart homes [19], smart buildings [20–23] and smart city [24, 25]. With the availability of IoT networks in smart buildings, building managers can monitor the occupants and environment in real time. Real-time monitoring and capturing can be used in

¹The phrase "cyber-physical systems" was invented by Helen Gill around 2006 at the National Science Foundation of the United States.

various applications controlled by the BMS [22]. For instance, the smart building systems of the future can improve their energy consumption by controlling and managing the heating, ventilation, and air-conditioning (HVAC) and respond quickly to the potential issues that can put the building off its record to carbon neutrality [26,27].

BMS was first patented in 1996 [28] and is popularly employed to manage HVAC systems. These systems consist of a network of sensors and devices providing managing building environmental conditions and HVAC system performance. BMS collects data from different smart systems and devices distributed in the buildings and integrates it into one interface for building managers and decision-makers. Smart buildings aim to increase user comfort and obtain a more efficient consumption of energy. However, within the last decades, a considerable number of research on BMS has been studied. The efficient energy consumption of buildings and facilities has still not been appropriately considered. Cloud computing with the IoT idea is a new tendency for efficiently managing and processing not only energy-sensing data but performance for smart building [29–31]. One of the main objections in the BMS for such a building is minimizing energy consumption without compromising the user’s comfort and decreasing performance. It is challenging to maintain smart BMS to have an optimal operational and computational state according to different conditions and situations. Lately, some researchers have studied the energy forecasting of smart buildings. Various methodologies have been proposed for the significance of smart building energy forecasting, such as ML and artificial NNs methods, regression methods, and time series [32–34]. A smart building on IoT and cloud-based technology that can perform collaboration and efficient operation with different sensing devices in the building is proposed [35].

Environment monitoring and safety denotes monitoring the weather conditions, water, and other various parameters of the environment in order to ensure the surrounding is protected and safe. The monitoring IoT system consisting of different sensors and other observational things is used to record and deliver timely information about the environment to the data centers. Some of the studies in this area include wireless sensor networks (WSN) for observing environmental monitoring [36,37], wearable computing [38], environment condition monitoring using low-cost ubiquitous sensing system [39,40], public safety in cloud-centric IoTs [41], and planning, construction, and management of sustainable cities [42]. Other examples of areas that use Iot applications are medical system [43–50], industrial operations and processing [51–55].

2.2 Cloud Computing

Cloud computing offers an essential component to most businesses, and it is supposed that cloud computing changes the IT industry since it provides scalability, elasticity, flexibility, reliability, and cost-effectiveness [56]. IoT systems have traditionally discharged cloud providers for computing and permanent storage. Also, cloud computing performs extended use of virtual machines (VM) because it allows workloads to be separated from one another and for the resource usage to be moderately controlled [57]. Cloud computing users process their data in third-party data centers placed distant from the user [58]. Most cloud computing system practices a hypervisor to create a VM. Hypervisor-based virtualization has been used for implementing virtualization and isolation [59]. Containers and VMs are two different ways of achieving resource control.

The principal benefits cover hardware independence, secure user environments, isolation, and increased scalability, together with the number of new properties optimized for different use cases [60]. Both VMs and containers are developed technologies that have benefited from a decade of incremental hardware and software optimizations [60–62]. A container is a software environment where one can install an application component or an application and all the library dependencies, binaries, and a basic configuration expected for the execution [63]. Container presents a particular level of abstraction for the application life-cycle management and probably can deal with many distributed application challenges [64], e.g., performance overhead. Containers are lightweight and introduce lower overhead compared to VMs [57]. Containers cause equal or better performance than VMs in all cases. Both VMs and containers expect tuning to maintain I/O-intensive applications. Several container-based solutions exist, e.g., OpenVZ [65], Rocket [66], LXC [67], and docker [68].

New technologies, like cloud computing and resource virtualization, have enabled the reliable management of computation resources of software systems managing to the demand for self-managing and autonomic systems [69], which brought up the self-adaptive software systems domain. A self-adaptive system senses the fluctuations in the operating conditions and the environment. It adapts and regulates its structure and behavior to meet its goals in the presence of changes.

A MAPE architecture [70], including, Monitoring, Analysis, Planning, and Execution components, provides the design and the implementation of an Autonomic Management System that adjusts the performance of the software system facing changing input workload. Although there has been significant progress in this domain, there are still considerable challenges in architecture and implementing self-adaptive strategies and, finally, limited solutions. Also,

it can ensure the level of automation and robustness demanded from such an autonomic management system.

But as the number of Things grows, these services cannot meet the requirements for IoT devices in real time [71]. These systems require high throughput data intake/bandwidths, low latency response times over wide areas, and physical settings. By decentralizing resource orchestration from data centers to edge networks, fog/edge computing extends the cloud system boundaries. They were set up as hierarchical networks of edge nodes or cloudlets providing deployment of ingestion, processing, and management services.

2.3 Edge Computing

Edge computing is a paradigm that brings the service, computational data, applications, and utilities of cloud computing closer to the end user, allowing faster processing and quick application response time. Edge computing systems provide services closer to them by supplying high bandwidth, ultra-low latency, and real-time access to network information that multiple applications can use. Dong et al., [72] discuss applications of IoT sensors, such as smartphones, wearable devices, and IoT-based thermostats and light control units, for optimizing building energy efficiency, thermal comfort, and indoor air quality. Usually, this is achieved through a functional sensing system that connects the environmental variables (e.g., temperature) integrated into the building's environmental control systems, such as ventilation, heating, and air-conditioning systems. Also, the users of IoT applications look for real-time response; such features can add more load to the networks, resulting in increased latency, storage and high bandwidth costs [73].

Luo et al., [10] investigate the effectiveness of computing resources control at the edge to guarantee low latency and continuous management, consequently improving efficiency and bringing economic benefits. Since the services and functions are closer to users in edge computing, a higher quality-of-experience (QoE) and quality-of-service (QoS) can be achieved for users. Also, edge computing can speed up analysis activities while reducing power usage by eliminating the use of central data centers [74].

Shi and Dustdar [7] demonstrate that IoT applications can transfer some parts of their function procedure logic to the edge of the infrastructure by employing powerful edge devices. Applications are distributed over IoT devices for technical reasons such as communication overhead and latency reduction, enforcing privacy and security, and provenance detection.

Cheng et al., [75] develop GeeLytics, a method that separates sensor data processing at the edge from cloud processing. Based on how quickly decisions need to be made depending on

the data, they can do faster analysis at the edge for certain queries. In addition, they can offload more computationally intensive queries to the cloud for more complex queries. In fact, Geelytics provides flexible and efficient edge analytics for large-scale IoT systems based on dynamic execution by taking into consideration the system properties of heterogeneous edge/cloud nodes and the current workload. Through this, lower latency results and better resource isolation is achieved while exploiting the minimal edge-to-cloud bandwidth usage. But, they do not consider power consumption or battery life when offloading analytics to the edge.

Javed et al., [76] propose IoT edge-cloud federation (IoTEF), a fault-tolerant and dynamic architecture for IoT applications. It uses cloud technologies including docker, Kubernetes, and Apache Kafka, and deploys them for edge computing. Their architecture consists of four layers: Application Isolation, Data Transport, Distributed OS, and Unified Federated Management. As a result of this architecture, latency can be minimized, network bandwidth can be reduced, and both node and network failures can be handled. However, they do not discuss the energy consumption of the edge computing platform.

Ramprasad et al., [77] propose an approach for pre-processing and streamlining BMS data to a cloud-based database for a larger scale and more complex analysis. The new database architecture and supporting infrastructure to help the streaming of BMS data are presented in a way that support efficient, low-latency, dynamic, and cost-efficient control of the building sensor network based on Big Data concepts and novel virtualization technologies. To this end, they create an architecture comprising three sections using edge and cloud framework for IoT application. Our case study experiment is based on this research on the subject application (IoT sensors in smart buildings).

Silva et al. [78] discuss state-of-the-art edge processing tools that enable data analytics and highlight the lack of benchmarks for edge processing frameworks. They point out the challenges faced by stream processing pipelines in terms of low-latency processing demands and underlying infrastructure needs and focus on the importance of edge resources for IoT data processing to meet the low-latency requirements.

2.4 Edge Devices Benchmarking

Kruger and Hanke [79] introduce micro and macro-benchmarking methods for gateway devices in IoT applications. They use *lmbench* for producing micro-benchmarking data and CoAP for generating macro-benchmarking information. They conclude that their micro and macro-benchmarking studies show that power consumption and flexibility are more signifi-

cant factors to be taken into consideration when picking WSN gateway devices for application in IoT areas. Nevertheless, they focus on latency and do not measure some resource usage like storage or network and do not present power consumption results.

A methodology for performing IoT benchmarks which includes application, workloads, and data generation, named RIoT Bench, is proposed by Shukla et al. [80]. Using RIoT Bench, they assess distributed stream processing systems (DSPS) in streaming IoT applications. However, some resource utilization metrics, such as storage and network, are not measured. In addition, no data is collected regarding power consumption.

Das et al. [81] implement a well-defined benchmark methodology of the edge processing framework, named EdgeBench. they use compute time, time-in-flight, End-to-end latency, Payload size, and CPU and memory utilization as the performance metrics and utilize these metrics for End to End Latency, Bandwidth Utilization, Local Resource Utilization and Infrastructure Cost evaluation. As a case study, using EdgeBench, they compare two edge computing platforms, Amazon AWS Greengrass and Microsoft Azure IoT Edge. Their outcomes demonstrate that the performance of Greengrass and Azure Edge are similar. The only difference is the fact that Azure Edge displays more increased end-to-end latency because of the batch-based processing method. However, they don't take power consumption into account when benchmarking.

Limaye and Adegbiya [82], introduce a benchmark set, HERMIT, for the Internet of medical things (IoMT) that is useful for healthcare systems. This benchmark suite is a step toward enabling research into effective processors for new IoMT applications.

2.5 Performance and EEnergy Consumption Benchmarking

Aslanpour et al. [83] survey the literature to recognize common performance metrics and present a taxonomy of the various metrics to evaluate the performance of cloud and edge computing and outlined open challenges for future directions. When computation occurs in IoT devices area, the first layer is called mist computing which normally comprises sensors and micro-controllers. They indicate that performance metrics depend mainly on the computing model/layer in which the IoT application is orchestrated. For instance, availability and evaluating the response time are major concerns in both cloud and edge, while the availability of IoT devices to handle the offloaded tasks is critical in the mist model. Moreover, certain performance metrics are common in all layers but have particular requirements, such as resource utilization which can be measured in terms of containers, VMs, or a hybrid of both depending on the corresponding computing layer.

Carpio et al. [84] benchmark the performance of modular edge cloud computing systems in terms of scalability, resource utilization, and latency by evaluating three different scenarios: cloud-only, edge-only, and mixed edge-cloud. The measurements showed that edge-cloud runs about 10 times faster than only-cloud until a certain number of concurrent processes where the system can no longer scale, and only-cloud performs better.

Defining strategies for reducing energy costs in computers requires monitoring processor power. Currently, processors contain a large number of counters that can be used to monitor system events. Also, in the edge nodes, factors such as CPU utilization, idle stage, memory and storage resources, network connectivity bandwidth, and communication protocols have been identified to impact the performance and energy consumption [85–95].

Rashti et al. [96] introduce a movable and adjustable platform for power monitoring, named WattProf. The WattProf system includes a programmable monitoring board and sensors for separate system elements. Also, it contains a host runtime environment and programming interface. They show the functionality of WattProf using two benchmarks, namely STREAM and DGEMM. They demonstrate how to perform fine-grained application power monitoring tasks over a number of system components, with multi-KHz sampling resolution.

Energy consumption and system performance of data centers change considerably with different system resource configuration and allocation strategies, workload, and types of running tasks in the cloud [97]. Also, research efforts focused on profiling and analyzing energy consumption in cloud environments. In existing research outcomes, profiling and analysis are carried out using energy benchmarks actively or monitoring the energy profile of a single system component, such as CPU, memory, and disk, at runtime. A framework for energy management and development of an energy-aware system has been proposed based on the availability of energy models for each hardware component [98]. Joulemeter is a power meter for VMs [99] which uses software components to capture the resource usage of VMs and converts it to the energy consumed due to the power model of each hardware resource.

Jung et al. [100] proposed Mistral, a holistic optimization system that balances power consumption and application performance costs based on adaptation actions and decision-making in a single unified framework. They developed an analytical evaluation model based on the variable and fixed costs to apply traditional economic methods. This model presented that business strategies can be derived for both cloud providers and cloud consumers. Based on the Mistral, cloud systems' energy efficiency can be analyzed and evaluated according to economic frameworks.

Also, a cloud cost model has been proposed by Mach and Schikuta [101] to calculate the energy consumption based on the number of Java virtual machine (JVM) instances on each

server. Nevertheless, it is not easy to measure the actual numbers of JVM because of the dynamic nature of the JVM life cycle.

Besides, Lee and Zomaya [102] proposed two energy-aware task consolidation mechanisms of the cloud to reduce energy consumption in cloud environments. They believe that there is an increasing linear relation between CPU utilization and energy consumption. Also, they indicate each task to the minimum energy consumption resource with no degradation in performance. The consequences of experiments show promising energy savings.

Hui et al. studied energy-aware VM consolidation and VM migration for green cloud computing in [103]. They recommended a new framework of dynamic virtual machine consolidation (DVMC) for green cloud computing. They proposed a new VM placement policy, namely Space Aware Best Fit Decreasing, a new migration VM selection policy, and utilization-based migration VM Selection. A DVMC framework has been developed to reduce energy consumption.

2.6 Estimation Models for Edge Computing Platforms

Nabavi et al. [104] use three different ML methods to forecast residential and commercial energy needs in Iran. Multiple LR, logarithmic multiple LR, and nonlinear autoregressive NNs with exogenous input are among these methods. A number of factors are taken into account when developing these models, including the share of renewable energy sources in the final energy consumption, the gross domestic production, the population, the price of natural gas, and the price of electricity. However, the used features are not in accordance with technical effective parameters.

According to Priyadarshini et al., [105], ML models are employed to analyze energy consumption in smart homes. The approach relies on ML techniques, such as DT, RF, XGB, and KNN, in order to predict the power consumption of multiple datasets. Additionally, they propose an Ensemble Model based on the DT-RF-XGB algorithm for analyzing consumption and comparing it with the baseline. Based on their study, they found that the DT-RF-XGB-based Ensemble Model outperformed all other baseline algorithms with an R2 of approximately 0.99. However, they do not focus on edge devices in smart homes. In this regard, the main difference is in the dataset's features and output. They use Kaggle's "smart-home-dataset-with-weather-information dataset," which includes 32 inputs of different natures (e.g., humidity, lighting power, microwave, wind speed, and garage door energy consumption). The Kaggle dataset includes the instantaneous power consumption by each device. Finally, they predict total energy consumption. In this work, the dataset includes

the level of utilization of computation resources.

Alden et al. [106] examine the capability of LSTM models to forecast power consumption in smart homes. Compared with edge computing platforms, because of the increased variability and uncertainty in user consumption behavior, individual residence power traces are more erratic and unpredictable. However, the input parameter measurements (devices that consume energy) are straightforward. In their research, HVAC, lighting, kitchen appliances, and heat pumps are some examples of LSTM model input features.

Bourhnane et al. [107] Artificial NNs along with Genetic Algorithms for prediction and scheduling in smart buildings. Their proposed models have the advantage of being trained and validated using real-world data collected from a PV installation and SB electrical appliances. Their plan, however, does not include a study of the energy consumption of network devices.

Connectivity is a critical concern between the edge nodes and devices. Orsini et al. [90] propose a method of forecasting bandwidth for mobile devices. They show that in an environment in which mobile device connectivity is constantly changing, the XGB model is the most appropriate model for predicting its future. Afterward, they indicate which sensor data is most pertinent to this task. In spite of the fact that XGB was able to deliver a high level of forecast accuracy, its training phase might need to be offloaded to a more powerful cloud system. Their solution relies on a federated approach, which enables the provision of less powerful devices with pre-trained models from the same edge cloud or from a larger number of users, forming a distributed intelligence.

Rodrigues et al., [108], using deep NNs, presents "SyNERGY" an energy measurement and prediction framework on embedded platforms. For the purpose of building a regression model, they use the supervised learning method in ML. By first executing a set of models in the training set, their prediction methodology focuses on modeling the energy at a per-layer level. It results in the target platform's execution behavior being captured down to the layer level.

CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

3.1 Methodology

This section describes the approach and methodology used throughout the research for this dissertation. A holistic energy measurement approach in edge computing called WattEdge [109] framework, open-sourced on GitHub ¹ is considered state-of-the-art research in the fields of benchmarking, power consumption estimation models, and managing the power consumption of highly resource-limited edge nodes. The following are the main differences between our work approach and the WattEdge approach:

In energy and performance benchmarking framework:

WattEdge is concerned with benchmarking hardware-related parameters that influence the consumption of power. For example, communication mechanisms such as WiFi, Bluetooth, ZigBee, USB, HDMI, VNC, and so on, as well as battery type (e.g., Lithium-ion) and charge level, are regarded as key considerations for edge node benchmarking. Because the scope of our task is software engineering, we concentrate on software-related aspects such as processing units and workforce numbers.

All PEBEdge components, including sensors, are virtualized to improve representativeness, flexibility, and extensibility. By design, each component is designed and virtualized as a container, making it simple to instantiate, alter, and replace with a different implementation using a different image during runtime. The WattEdge architecture does not support these capabilities.

In building power consumption model

In building power consumption models, WattEdge employs analytical modeling, which is very reliant on the device being measured. Furthermore, analytical methods frequently assume linear relationships, which may limit their ability to correctly capture data patterns. In addition, analytical approaches may have limitations when it comes to successfully generalizing unknown data. In this work, we develop power consumption models using ML-based techniques. ML techniques are hardware-independent and widely applicable. All components in the PEBEdge architecture are virtualized and emulated in distinct containers using a software-based approach with numerous inputs (e.g., device data and different apps). Because there is a requirement to focus on retraining data in this scenario, ML algorithms are a

¹<https://github.com/aslanpour/wattedge>

superior solution. Because ML approaches make data training much easier and faster. Analytical methods, on the other hand, are frequently interpretable, utilize fewer computational resources, and produce more accurate findings.

3.2 Document Structure

This document, with the title “power estimation framework to ensure the reliability of edge devices in the IoT Applications”, is a dissertation submitted to Polytechnique Montréal in partial completion of the requirements for the degree of Philosophy Doctor in computer engineering. It follows the dissertation layouts permitted by the school. Specifically, Chapter 1 provides an introduction by stating the context and motivation, the problem statement, and the research objectives and contributions. Chapter 2 introduces the related work on the deployment of IoT applications using edge devices, benchmarking energy and performance of the edge computing platform, and related work regarding estimation models of edge devices. Chapter 3 presents the methodology and structure used in the research work. Chapter 4 proposes the PEBEdge architecture, a framework for benchmarking edge applications. Along with a description of the implementation of the case study with our concrete application and set up for edge device benchmarking. Chapter 5 presents ML models, including setup and measurement, dataset preparation and data analysis, building ML models, and comparing performance results. Then, by employing the ML models, we propose some solutions, including identifying the most important software-related factor that enables us to manage the power consumption, changing the system software-associated configuration for decreasing power consumption in critical situations, Along with a case study for calculating the edge device battery depletion time in order to compensate for the lack of reliability imposed by the energy-constrained edge devices. Chapter 6 shows how we provide the researcher with the low-cost and low-time possibility to perform experiments regarding edge device power consumption. Chapter 7 presents our discussion, and finally, Chapter 8 summarizes the thesis and provides our work limitations with some suggestions for future research.

CHAPTER 4 PEBEdge: A PERFORMANCE AND ENERGY BENCHMARKING TOOL

4.1 Introduction

In computer science, a benchmark is a tool and a methodology used to assess and compare systems or components based on particular attributes like performance, reliability, or security [110]. Benchmarking, using the appropriate tools, can be used to efficiently and economically evaluate a variety of applications and topologies under various workloads [111]. In the context of edge computing, the distribution of data and computation workloads on different tiers (e.g., cloud or edge device) can affect the performance, resource utilization, and power consumption of the entire system and its infrastructure. Importantly, it is necessary for energy-constrained edge devices that cause edge topologies to suffer from energy-related failures due to insufficient battery life [7]. Therefore, it is essential to be able to assess this influence before, during, and after a system’s deployment—in other words, whenever crucial decisions are being made about its design and implementation. To this purpose, it is necessary to benchmark the various performance and energy characteristics along the edge network in order to aid practitioners and decision-makers in quickly and affordably analyzing a wide range of scenarios, ultimately improving the quality of service in production. This section presents PEBEdge as an end-to-end performance and energy benchmarking tool for edge computing platforms in IoT applications to assess and monitor resource utilization and power consumption. PEBEdge is designed to be flexible and extensible to accommodate various application architectures. PEBEdge is also employed for load, stress, and capacity testing, with a particular emphasis on low-capacity edge servers and edge devices. We show how PEBEdge may be utilized in a study that contrasts the various edge topologies for an IoT application. Various metrics are employed in performance benchmarking, including response time, and resource utilization (e.g., CPU, memory), each of which characterizes a distinct aspect of the system performance. For power consumption benchmarking, the external power analyzer measures the power consumption of the entire edge device. Using external power measurement tools allows for the creation of precise accuracy limitations for the power measurement tools [110].

4.2 State-of-the-art Approaches

WattEdge [109] framework, is considered state-of-the-art research in the field of power consumption and resource utilization benchmarking. In order to manage power consumption, WattEdge focuses on benchmarking hardware-related factors that impact power consumption. For instance, connectivity means such as WiFi, Bluetooth, ZigBee, USB, HDMI, VNC, etc are considered important factors for edge node benchmarking, as well as battery type (e.g., Lithium-ion) and charge level. Since our work scope is software engineering, we focus on software-related factors such as processing units and workers' numbers. The optimum models, nevertheless, integrate both strategies (hardware and software). After that, a decision to alter the software or hardware configuration might be made. Some configurations are more affordable than others. For instance, replacing WiFi with Bluetooth is not an option when there is a considerable distance. PEBEdge is representative of a real IoT application multi-tier architecture including sensors, edge application servers, and cloud data storage parts. Layers are parallel and do not affect each other. All of the PEBEdge components, such as the sensors, are virtualized to allow for more representativeness, flexibility, and extensibility. Each component is developed and virtualized as a container by design, making it simple to instantiate, customize, and replace at runtime with a different implementation using a different image. The platform also offers strong orchestration capabilities for large-scale experiments using docker containers and gathers consistent performance and energy usage measurements across deployments. These features are not available in WattEdge architecture. PEBEdge and WattEdge frameworks both require physical (e.g., external power meter) and software components.

4.3 PEBEdge

PEBEdge is a general framework that enables the benchmarking of applications on the edge for a variety of architectures and topologies. PEBEdge follows a multi-tier architectural style with microservices, in which both physical and virtual components are combined. Microservices are preferred to allow for enhanced extensibility, interoperability, and scalability for PEBEdge, while multiple tiers are used to define the various elements of the benchmark system and to show the representativeness of its architecture, including the sensor tier (i.e., workload generator), the edge tier (including application and benchmarking layer), and the cloud tier (for data storage and more demanding analytics). Every component is implemented as a container, which allows for easy instantiation, customizability and runtime replacement by a different implementation using a different image.

Docker containers also allow the framework to provide robust orchestration capabilities to large-scale experiments and consistently collect performance and energy usage metrics across the entire deployment. For instance, using docker-compose, to scale or run multiple container services within a single machine, each running in isolation, while measuring the performance and the power consumption. It is required to have both physical and software components in the PEBEdge framework. One of the goals of PEBEdge is to monitor power consumption on the edge device, which implies that certain components need to be physical. Otherwise, we virtualize all other components that do not require energy monitoring (e.g., sensors) to allow for higher flexibility and extensibility.

4.4 Architecture

Figure 4.1 presents the main components of the proposed framework and how they inter-operate to deliver an end-to-end solution for evaluating IoT network and application performance. The framework consists of multiple tiers: the sensor tier for workload generation, the edge tier consisting of the benchmarking layer and the application layer, and the cloud tier for large storage and processing.

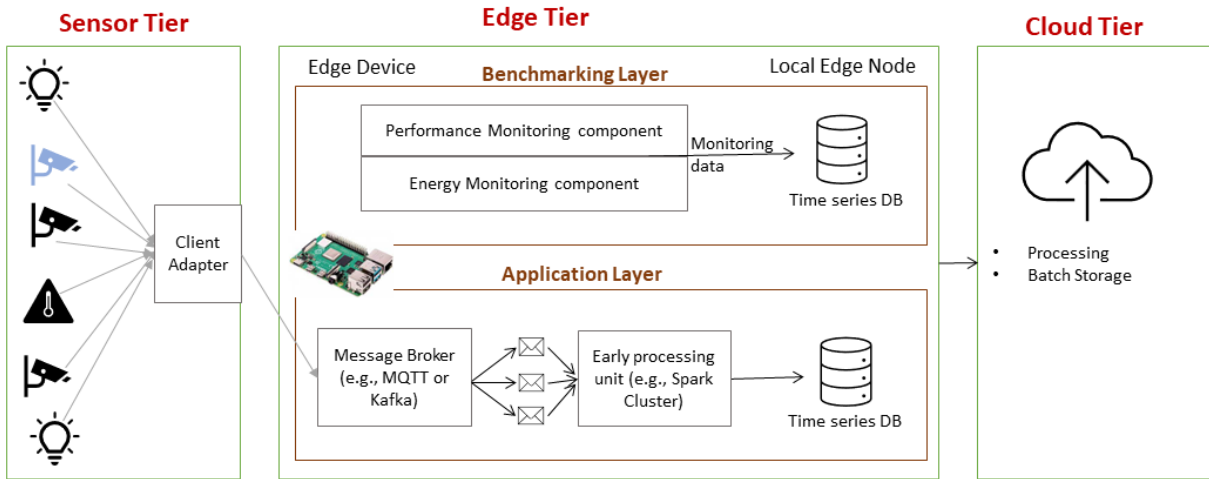


Figure 4.1 PEBEdge architecture showing the three tiers and their inter-operate for benchmarking the performance and power consumption of the edge device.

4.4.1 Sensor Tier - Workload Generator

The sensor tier is responsible for the generation of data from different sensors. The workload generator uses virtualized sensors to generate the data (e.g., pictures, text, or numerical). The

virtualized sensor can be regarded as a logical relation/ mapping of multiple physical sensors of the same or different types (e.g., camera, light, or temperature sensors). It generates the data within a given range of values and emits rates that are implemented using Python scripts. The virtualized sensors can be replaced with physical sensors. The use of virtualized sensors allows different sensors to be instantiated, removed, or replaced. The sensors are assumed to generate data continuously and are configurable. The sensor tier has its own docker file and docker-compose file. To build the image, first, it is required to build docker-compose, then start the image as docker service using the docker-compose up ¹. Listing 4.1 shows the template of building a docker file into the docker image. This template provides the flexibility for researchers to customize and configure their own virtual sensor by substituting the parameters. This exemplifies the extensibility feature of the PEBEdge architecture. The ability to modify the parameters in the template showcases the adaptability and versatility of the PEBEdge framework, allowing researchers to tailor the virtual sensor configuration according to their specific requirements.

```

1 FROM python:3.6-slim
2
3 RUN mkdir camera
4 WORKDIR camera
5 RUN mkdir imagesout
6
7 RUN apt-get update && apt-get install -y build-essential \
8     cmake \
9     wget \
10    git \
11    libgtk2.0-dev \
12    && apt-get -y clean all \
13    && rm -rf /var/lib/apt/lists/*
14 COPY /requirements.txt /camera/requirements.txt
15 RUN pip install -r requirements.txt
16
17 COPY / /camera
18
19
20 ENTRYPOINT ["python3", "/camera/main.py"]

```

Listing 4.1 The template of building docker file into the docker image

A client adapter is responsible for collecting data from the workload generator and adapting and sending it according to the message broker. The client adapter provides a sim-

¹<https://github.com/polytechnique-ease/EdgeBenchmark/blob/main/Sensor%20Side/DataGenerator>

ple application programming interface (API) to receive data from the containers-sensors as post requests. Subsequently, the client adapter forwards the data to a message broker, like Mosquitto or Kafka ². The client adapter can be modified to connect to different message-oriented middle-ware ³. Users, also, can create his/her own adapter. To this end, after running the "YourAdapter.py", the following parts must be edited. In the run function before initiating the handler, user can put any code to be run only once before receiving requests. Another part is initiating variables and passing them to the post function. Listing 4.2 shows the variables "client" and "b" as examples. These variables should be passed to the function "partial" after the variable "YourAdapter"(which is the name of the class). Then "class constructor" must be used as Listing 4.3.

```

1 def run(server_class=HTTPServer, handler_class=YourAdapter, port=8088):
2     client = "xx"
3     b = "bb"
4     handler = partial(YourAdapter, client, b)
5     server_address = ('', port)
6     httpd = server_class(server_address, handler)
7     print('Server running at localhost:8088...')
8     httpd.serve_forever()

```

Listing 4.2 Editing "run" function for creating users own adapter.

```

1 def __init__(self, client, b, request, client_address, server):
2     self.request = request
3     self.client_address = client_address
4     self.server = server
5     self.client = client
6     self.b = b
7     print(client)

```

Listing 4.3 Class constructor for creating users own adapter.

4.4.2 Edge Tier

The edge tier consists of the edge device and the local edge node. Basically, an edge device is any piece of hardware that can manage data flow at the boundary between two networks. An edge device provides a gateway into enterprise or service provider core networks. The edge device can be a lightweight device Raspberry Pi (RPi) ⁴ where a message broker like message queuing telemetry transport (MQTT) ⁵, data processing unit (e.g., Spark stream-

²<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Sensor%20Side/ClientAdapter/MQTTAdapter>

³<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Sensor%20Side/ClientAdapter/YourAdapter>

⁴<https://www.raspberrypi.org/>

⁵<https://mqtt.org/>

ing ⁶) and monitoring unit are deployed on. The sensors can pair with the edge devices via WiFi, Bluetooth, or through the local area network. Bringing the data to a device that is computationally capable and has enough memory can minimize data loss and facilitate the rest of the process. From a functional perspective, in the PEBEdge architecture, the edge tier consists of two layers: the application layer and the benchmarking layer.

Application Layer This layer includes the application that will be benchmarked. Any application is supported as long as it is deployed on containers and on the edge device to allow for monitoring. The types of applications that PEBEdge benchmarks typically fall into the IoT category. At least three components are required in the application layer: a message broker, an early processing unit, and a time series database. The application layer receives the generated data transmitted from the sensor tier.

A client adapter forwards the data to a message broker, like Mosquitto or Kafka. Then it is sent to the early processing unit for cleaning, filtering, processing, and finally to the time series database for initial storage. The message broker and processing unit are located on the edge device and the database is located on the local edge node. PEBEdge expects that the application be deployed in docker containers. Otherwise, monitoring cannot function ⁷.

Benchmarking Layer Basically, the benchmarking layer consists of energy and performance monitoring components and the database for monitoring data. Energy and performance monitoring components are located on the edge device and the database is located on the local edge node. For the benchmarking layer, it is required to set up docker for monitoring purposes ⁸. Running monitoring applications in the benchmarking layer can affect the performance and power consumption of the edge device. Therefore, it can be considered a negligible potential challenge in the benchmarking layer.

In PEBEdge architecture, an external physical tool called the USB Power Meter is needed for monitoring the power consumption in the edge device. Figure 4.2 shows the USB Power Meter and its user interface in our experiment. The USB power meter is attached directly to the edge device and exports the required values (voltage (Volt), current (Ampere), and power consumption(Watt)).

⁶<https://Spark.apache.org/>

⁷<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Edge%20Side/Processing>

⁸<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Edge%20Side/monitoring>



Figure 4.2 USB power meter (figure a) and its UI (figure b).

4.4.3 Cloud Tier:

This tier is responsible for large storage and computing resources (such as complex computation and advanced analytics) for IoT services. Stored processed data from the application layer database in the edge tier are transmitted to the cloud tier for further computational purposes and final storage. Any type of database such as InfluxDB or MongoDB can be used as cloud storage to allow for more scalable, high flexibility, and simplicity and provide high-speed performance, among others.

4.4.4 QoL Features

PEBEdge has several quality-of-life (QoL) features for developers, practitioners, and users. On top of the functional requirements, the framework ensures the following quality properties:

- **Extensibility:** PEBEdge is built in such a way that adding new workloads, monitoring tools, or support for new edge computing or cloud computing networks is simple. Many of the components in PEBEdge can be replaced by any other sensor, edge device, or cloud storage. Also, it is possible to apply PEBEdge to any application as long as the docker interface is available.
- **Usability:** PEBEdge requires a basic understanding of distributed technology (e.g., cloud). Most components of the benchmark, including installation, monitoring, and workload generation, were automated using container technology. Users can easily build their test scenarios and view the results in an easy-to-understand report.
- **Maintainability:** PEBEdge is developed to be utilized by ongoing projects and to al-

low for continuous improvement. The PEBEdge⁹ code is open-source and bug reporting and resolution, as well as the addition of new features, follow community-standard procedures.

4.4.5 Capabilities

The PEBEdge is intended for endurance and capacity testing to ensure architecture stability and its capability of handling loads under extreme conditions. An endurance test evaluates how the application performs over a long period of time with a normal workload. Under endurance testing, we test the system’s performance under certain load conditions over a long period of time. Endurance testing aims to check for system problems such as memory leaks.

Through endurance testing, we analyze the capacity and limitations of the edge device to work under a continuous workload (not necessarily stressful), and the defined metrics are measured. We consecutively launched experiments without shutting down the RPi to make sure that it can handle requests over a long period. To determine how many sensors (or generally load data) the application can handle before either performance or stability becomes unacceptable, the capacity test is performed. This is a way to avoid potential problems in the future and to have better visibility into events. In this study, the number of sensors is the limitation for each set of experiments.

4.5 Case Study

We now discuss how the framework allows users to plan scenarios, design use cases, create and run experiments, and evaluate the proposed framework using the benchmark tool. We present a variety of experiments to show how different parameters can be set up for a particular deployment of an IoT application. To this end, we carried out a case study using the PEBEdge to demonstrate how testing and benchmarking experiments can be conducted for applications on the edge device.

4.5.1 Motivation Example

The primary motivation example for this section revolves around comparing the deployment of Spark streaming and Python scripts on an RPi and examining the resulting outputs. The second motivation example focuses on investigating the configuration procedure involving

⁹<https://github.com/polytechnique-ease/EdgeBenchmark>

the modification of container numbers (via adjusting the number of workers) and its impact on the output (power consumption).

4.5.2 Research Question

The first research question is how easy and fast it is to set up and change an experiment (e.g., swing between Spark and the Python script or change the worker numbers) and which configurations should be changed during the transition process. The second research question is how easy it is to analyze the results. In fact, we try to know what the effects of changing software-associated configurations are on the results (edge device power consumption).

4.5.3 System Configuration

Sensor Tier Configuration

We, firstly, need a VM, named VM1, to act as the data producer and host workload generator which consists of docker images for the various sensors. Sensors generate lots of data every second. The implementation of the virtual sensors is done using Python 3 and they are deployed as docker containers in the sensor tier. When simulating a sensor, it is important to know the size, frequency, and type of data being sent. In the current version of our case study, the containers simulate cameras that send frames of a video of varying sizes between 141KB and 148KB every 1/5/10 seconds. The client adapter is also deployed on VM1. Here, the client adapter subscribes to an MQTT broker and sends data using publish/subscribe protocol. Following is the Github path ¹⁰ and details for the codes to be deployed on the sensor tier.

Client Adapter: Code is responsible for collecting data from the data generator and adapting and sending it according to the message broker.

- MQTT Adapter: In this folder, the adapter subscribes to an MQTT broker and sends data using publish/subscribe protocol.
- YourAdapter: The user can create his/her own adapter, the code in the folder will facilitate the task.

Data Generator: Code is responsible for generating data simulating cameras with docker containers.

¹⁰<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Sensor%20Side>

Configuration Steps: First, the configuration file (Sensor Side / DataGenerator / data-generator - variables.env) needs to be edited to route the data from the data generator to the client adapter. Table 4.1 presents the data generator required configuration values:

Table 4.1 Data generator required configuration values.

Variable	Value
CLIENT_ADAPTER_IP	The IP of the Adapter which is using to send data to the edge device, the IP should be the same as the Sensor Side Machine.
CLIENT_ADAPTER_PORT	The PORT on what the Adapter which is used to send data is listening.

Next, if the MQTT Adapter is used, it is required to edit the configuration file (Sensor Side / ClientAdapter / MQTTAdapter / sensor - variables.env) to route the data from the client adapter to the edge side. Table 4.2 presents the MQTT client adapter required configuration values. Otherwise, for creating another adapter, folder (Sensor Side / ClientAdapter / YourAdapter) is defined for the new adapter and the file (Sensor Side / ClientAdapter / YourAdapter / sensor - variables.env) is defined for env variables to be used inside the code.

Table 4.2 MQTT client adapter required configuration value

Variable	Value
MQTT_SERVER_IP	The message broker IP on the edge device, this should be the same IP of the edge device (e.g., tcp://1.1.1.1).
MQTT_SERVER_PORT	The message broker PORT on the edge device.
MQTT_SERVER_TOPIC	The topic that the MQTT Adapter is publishing data to.

Edge Tier Configuration

Secondly, we use an RPi with 4GB (RAM) as the edge device and a second VM, named VM2, acting as a local edge node. Processed and monitoring data are transferred to the local time-series database (e.g., InfluxDB ¹¹) on the local edge node to be stored. Following is the Github path ¹² and details for the codes to be deployed on the edge tier.

¹¹<https://www.influxdata.com/>

¹²<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Edge%20Side>, <https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/DB%20Side>

Edge Device/Processing: Code is responsible for processing and filtering the data sent from sensors.

- Spark: Processing using Spark.
- Simple Script: Processing using a Python script.
- Your Example: Processing using own code.

Edge Device/Monitoring: Code is responsible for monitoring the edge using docker stats.

Local Edge Node: The code is responsible for saving data in the local edge node.

- InfluxDB: Code for a MongoDB Database.
- MongoDB: Code for an InfluxDB Database.
- Yourexample: A docker-compose.yml file in which there is a time series database used for saving benchmarking data.

Configuration Steps First, it is required to edit the configuration file to route data. If Spark is used as the processor, the Spark configuration file(Edge Side/Processing/simple script/Spark-Edge-variables.env) is used to edit the conf. file to route data from the Spark to the database. Table 4.3 presents the Spark Processor configuration in the experiment.

Table 4.3 Spark Processor configuration.

Variable	Value
InfluxDB_PROTOCOL	The protocol used to connect to the InfluxDB database
InfluxDB_IP	The IP of the InfluxDB database
InfluxDB_PORT	The PORT of the InfluxDB database
InfluxDB_TOKEN	The TOKEN used to authenticate to the InfluxDB database
InfluxDB_ORG	The name of InfluxDB organization
InfluxDB_BUCKET	The name of InfluxDB Bucket
InfluxDB_WRITE_TIMEOUT	The timeout of writing in the InfluxDB
MQTT_SERVER_IP	The message broker IP on the edge device
MQTT_SERVER_PORT	The message broker PORT on the edge device
MQTT_TOPIC	The topic that script is subscribing to
MASTER_IP	The IP of the Spark master
MASTER_PORT	The port on which the Spark master is listening
Spark_EXECUTOR_MEMORY	Amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g., 512m, 2g).
Spark_DRIVER_MEMORY	Amount of memory to use for the driver process, i.e. where SparkContext is initialized, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g., 512m, 2g).
Spark_MEMORY_FRACTION	Fraction of (heap space - 300MB) used for execution and storage. The lower this is, the more frequently spills and cached data eviction occurs. The purpose of this configuration is to set aside memory for internal metadata, user data structures, and imprecise size estimation in the case of sparse, unusually large records.
Spark_MEMORY_STORAGEFRACTION	Amount of storage memory immune to eviction, expressed as a fraction of the size of the region set aside by Spark.memory.fraction. The higher this is, the less working memory may be available to execution and tasks may spill to disk more often.

In our experiment for an RPi with 4GB (RAM), following are the values for Spark memory when running a Spark cluster with 1 worker:

Spark_EXECUTOR_MEMORY="1g"

Spark_DRIVER_MEMORY="2g"

Spark_MEMORY_FRACTION="0.9"

Spark_MEMORY_STORAGEFRACTION="0.2"

If a Python script is used as the processor, a configuration file(Edge Side / Processing / simple script / simple - script- edge - variables.env) is used to edit the configuration file to route data from the Python script to the database. Table 4.4 presents the Python script Processor configuration in the experiment.

Table 4.4 Python script Processor configuration.

Variable	Value
MongoDB_DATABASE_NAME	The name of the MongoDB database
MongoDB_COLLECTION_NAME	The name of the MongoDB collection
MongoDB_DATABASE_IP	The IP of the MongoDB database (this should be the Local edge node IP)
MongoDB_DATABASE_PORT	The PORT of the MongoDB database
MongoDB_DATABASE_USERNAME	The username used to authenticate to the MongoDB database
MongoDB_DATABASE_PASSWORD	The password used to authenticate to the MongoDB database
MQTT_SERVER_IP	The message broker IP on the edge device, should be the same IP of the edge device(e.g., 1.1.1.1)
MQTT_SERVER_PORT	The message broker PORT on the edge device
MQTT_TOPIC	The topic that script is subscribing to

Also, It is needed to configure mosquitto.conf file according to the Table 4.5

Table 4.5 Mosquitto configuration.

Variable	Value
Listener	1883
Allow_anonymous	true
Max_queued_messages	0
Max_connections	-1
Log_type	all
Max_inflight_messages	0

Cloud Tier

Finally, the data is sent to a more stable and secure location like a cloud server for further processing and final storage. Thus, a cloud account is required. We use Digital Research Alliance of Canada ¹³. The experiments run scenarios on the RPi to measure the resource utilization metrics, such as CPU, memory, disk, and network, as a performance metric, and instantaneous power consumption. While the experiments and scenarios are running, the RPi is monitored and the defined metrics are collected on the local edge node. The CPU percentage, memory, disk, and network bandwidth are monitored every second by calling docker stats. Moreover, response time is calculated for each message sent and stored in the local edge node. Response time is defined as the period between the sensor’s data emission time and the moment the data is stored at the local edge node. Also, we record USB Power Meter output data to measure active and total power consumption per millisecond.

4.5.4 Running Experiments

Running Experiment Using Spark Streaming

This experiment uses the MQTT ¹⁴ message protocol, and Spark streaming as the processing unit. MQTT follows a publish-subscribe model to publish and subscribe to a message based on topic (i.e., the IoT sensors used as the publisher and Spark Streaming as the subscriber). MQTT is a lightweight communication protocol for machine-to-machine (M2M) standardized under ISO since 2016 (ISO/IEC 20922:2016) ¹⁵.

As soon as data is published by the sensor in the MQTT broker, the Spark cluster (the consumer) will retrieve the data and begin processing and storing it. To do this, a job is submitted using “Spark-submit”, making the Spark Driver launched, which executes the

¹³<https://ccdb.computecanada.ca>

¹⁴<https://www.emqx.com/>

¹⁵<https://www.iso.org/standard/69466.html>

`main()` method of the code. The Driver contacts the Master and requests resources to launch the Executors. The Master launches the Executors on behalf of the Driver to establish a direct connection with the Driver. The Driver determines the total number of tasks by continuously checking the Lineage. Also, the Driver creates the logical and physical plan. Once the physical plan is generated, Spark allocates the tasks to the Executors. The task runs on an Executor and each task, upon completion, returns the result to the Driver. Finally, Spark releases all the resources from the Master.

Spark Streaming is a micro-batch architecture, implying that the jobs are triggered on the Spark Engine periodically. Spark will pull messages from Mosquitto, process them using the directed acyclic graph (DAG) we defined, and save the results to a database. The proposed framework identifies four main objectives for the running job on Spark:

- **Mapping:** converting each string record into a JSON object.
- **Filtering:** Images that are larger or smaller than the specified interval will be filtered.
- **Information Extraction:** Each image is processed to get information about it (i.e., the size or the extension of the image).
- **Storage:** Connect to the time-series database and save the record with several pieces of information (time of receiving data from the sensors, time of receiving the data in Spark, time after the data has been processed in Spark and etc).

Spark consumes a large amount of memory. So, we changed four parameters including `Spark.executor.memory`, `Spark.driver.memory`, `Spark.memory.fraction`, and `Spark.memory.storage` fraction to make it work. Additionally, the timeout needed to be increased. Therefore, we increased the write timeout in the database. Also, on the sensors side, `COMPOSE_HTTP_TIMEOUT` must be set.

Changing worker numbers Spark workers play a crucial role in the execution of tasks within the Spark framework. Adding more workers enhances parallelization and increases memory requirements. With additional workers, Spark can efficiently process large volumes of data by dividing the workload and performing computations in parallel. This capability is particularly valuable when dealing with big data scenarios where timely processing is crucial. The duration of the experiment represents the time taken to ensure the complete storage of all sensor-generated records in the database. To modify the number of workers in the Spark setup, just need the following code to be executed on the RPi:

Docker-compose up scale worker=1, 2, or 3

Running Experiment Using Python Script

In this part, we replace the Spark streaming with a simple Python script. This experiment uses Python script for the processing section and MongoDB database instead of InfluxDB ¹⁶. In the configuration section, it is needed to just change the variables in the Table 4.4: In the processing part, the only change is running "docker build -t yourrepo/Pythonprocessor" instead of "docker build -t yourrepo/Spark-job". All other configurations and settings are identical. It is clear how easy and fast it is to change Spark with the Python script.

4.5.5 Performing Experiments Considerations

For running the experiments and achieving the results, the first step is to plan different scenarios as realistically as possible. A scenario includes details such as the number of cameras, type, size, and length of the message to be sent based on the edge device limitation beforehand. At first, one camera sends data to the RPi, the RPi is monitored during the receiving and processing, and measurements are collected. In our scenarios, 100% of the records were successfully saved in the database compared to records sent from the Sensors.

To achieve sufficiently accurate measurements, robust preliminary actions are taken such as:

- RPi is also shut down after each experiment as a cooling down process.
- RPi is not re-positioned to avoid environmental conditions (e.g., different temperatures).
- Removing containers, turning off the RPi, and a 30-minute cooling process is considered between each test.
- The operating system of RPi is not updated and upgraded for each test with minimal installations to run the experiments under the same condition.

4.5.6 Results

Based on the outlined procedure, it becomes evident that initiating the experiment and modifying software-related configurations require minimal parameter adjustments (e.g., defining IP addresses for the deployed components), while the remaining connections are established automatically. This simplicity of setup ensures a streamlined experience for researchers. Furthermore, this streamlined setup offers notable advantages in terms of experiment execution

¹⁶<https://github.com/polytechnique-ease/EdgeBenchmark/tree/main/Edge%20Side/Processing/simple%20script>

speed and result analysis ease. The efficiency of the process enables researchers to swiftly conduct experiments and obtain the necessary insights from the results. Consequently, this aspect directly addresses the first research question, examining the simplicity, speed, extensibility, and ease of result analysis facilitated by the presented procedure.

Concerning the second research question, the investigation focuses on examining the impact of two specific software configurations, namely the choice of the processing unit and the number of workers, on power consumption. Figure 4.3 presents a comparative analysis of the experiment's outcomes, specifically for a single camera, employing a Python script as the processing unit and Spark streaming as the processing unit (with a single worker). Figure 4.3 demonstrates that Spark streaming exhibits lower power consumption compared to the Python script. This finding suggests that in scenarios where power resources are critical, transitioning from a Python processing unit to Spark streaming can lead to reduced power consumption. The results highlight the potential energy efficiency advantages offered by utilizing Spark streaming as the processing unit, contributing to more sustainable and power-conscious computing practices. However, Spark consumes a large amount of memory, compared with the Python script processor.

Similarly, Figure 4.4 presents a comparison of power consumption on an edge device (specifically, RPi) when employing Spark streaming as the processing unit under various conditions, including different numbers of workers. The configurations considered consist of one worker, two workers, and three workers. Observing Figure 4.4, it becomes evident that as the number of parallel workers increases, the power consumption of the edge device also rises. This finding suggests that, in critical power-constrained scenarios, mitigating power consumption on edge devices can be achieved by reducing the number of workers. However, it is essential to note that employing fewer workers diminishes Spark's capacity for processing large datasets and executing computations in parallel. These results highlight the trade-off between power consumption and computational capabilities when configuring the number of workers in Spark streaming.

Detailed configuration and results for the case study can be found in Github ¹⁷.

¹⁷<https://github.com/polytechnique-ease/EdgeBenchmark>

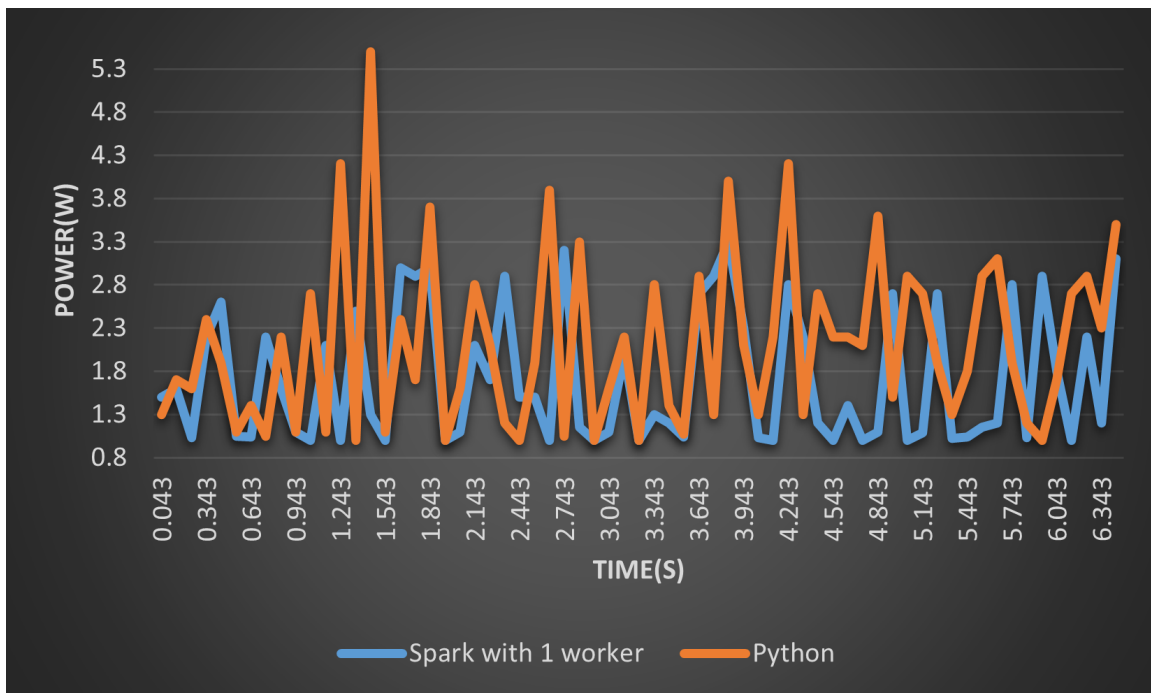


Figure 4.3 Results (for one camera sensor) of the experiment using a Python script with the experiment using Spark streaming.

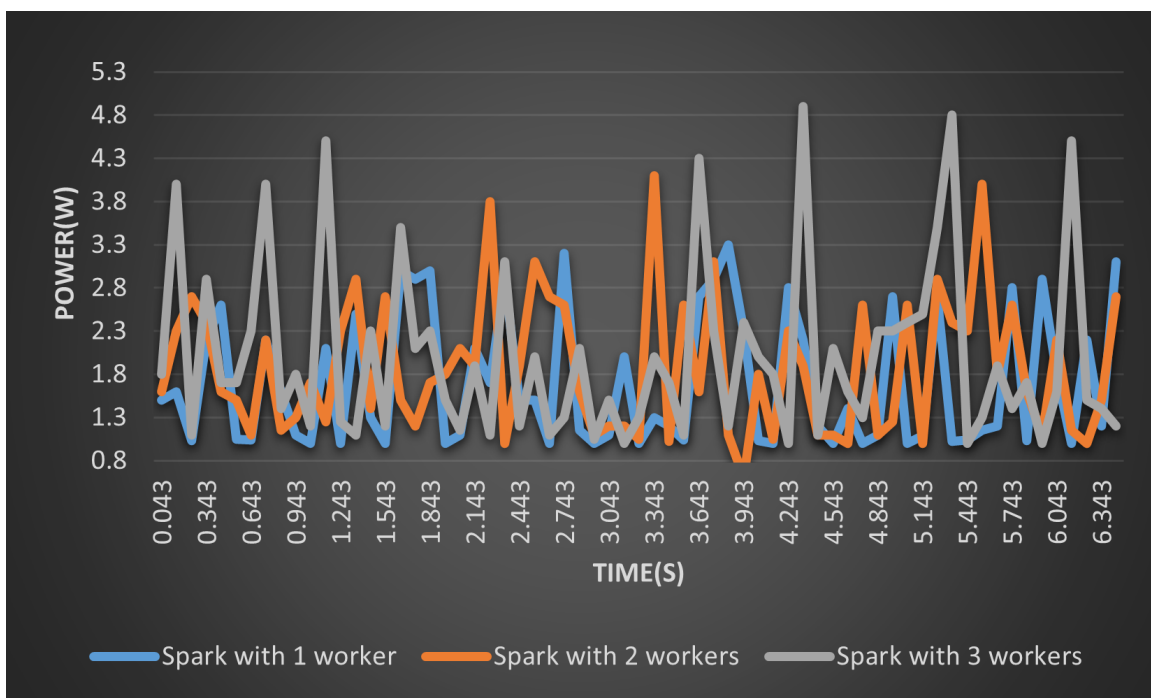


Figure 4.4 Results (for one camera sensor) of the experiment using Spark streaming with 1, 2, and 3 workers.

4.6 Discussion

The original purpose for running the benchmark was to compare Python and Spark as processor units. Through running experiments, we showed how PEBEdge facilitates benchmarking experiments. This presentation demonstrated how straightforward it is to set up and configure our experiment and how to generate the workload needed for it. Our presentation demonstrated how easy and fast it is to replace Spark streaming with Python scripts for benchmarking applications. The replacement only requires a few configuration changes and docker deployment. We showed, using PEBEdge, how fast it is to run experiments and how easy it is to get the data to analyze the results of the experiments. It is also clear how extensible PEBEdge is. Through this architecture, IoT developers can easily extend and inter-operate their IoT applications while monitoring performance and energy usage. For instance, the developer can easily plug in a physical energy monitoring device, replace the physical energy monitoring device with a virtualized monitoring software, and remove the energy component from the tool depending on the use case. Similarly, the software components can be replaced with the counterpart (e.g., Mosquito vs Kafka as a messaging blocker, choice of processing unit). Another advantage of this benchmark tool is that the reliability and dynamic maintenance of the IoT applications are guaranteed. This is made possible by reducing the network workload by enabling the computation at the edge of the network close to the data generating source (sensors), and it allows physical and virtual mobility. We also note however that the limitation of PEBEdge when the benchmark is applied for the hybrid environment for the scenario where both physical and virtual components are used.

CHAPTER 5 ML-BASED POWER ESTIMATION MODEL

5.1 Introduction

Power estimation models help in determining the power requirements of a system or device. This information is crucial for resource planning, such as designing power supply systems, allocating power budgets, and estimating battery life. In this work, a power estimation model is developed to investigate the relationships between several characteristics (such as input sensors, container numbers, process units, CPU, memory, network, and disk) and the power consumption of edge devices. The estimated power usage of edge devices aids in enabling us to evaluate when the battery will run out.

To this end, using the PEBEdge framework we set up an environment to perform some experiments in order to collect a data set that contains different parameters like CPU and memory and their co-related instantaneous power consumption.

The first step is to create a VM that will serve as the data producer and host workload generator for the sensors. The experiments we conduct simulate the operation of cameras and temperature sensors. Python 3 is used to implement the virtual sensors, which are deployed as docker containers. The Client Adapter is also deployed on the VM. Secondly, we utilize an RPi with 4GB of RAM as an edge device, while a second VM serves as a local edge node. Data that has been processed and monitored is transferred to a local time-series database on the edge node where it will be stored. Finally, a cloud server is then used to store the data so that it can be processed and stored in a more secure and stable location. Consequently, it is necessary to create a cloud account. It is through the Digital Research Alliance of Canada that we conduct our research ¹.

5.2 State-of-the-art Approaches

WattEdge [109] framework, is considered state-of-the-art research in the field of building power estimation models for edge devices. In order to build the power consumption model, WattEdge uses analytical modeling. Analytical methods are a set of techniques and methodologies for analyzing and solving problems by investigating the underlying components, and principles of a system or dataset. To extract insights, make estimations, or draw conclusions, these methods often entail the application of mathematical and statistical techniques, as well

¹<https://ccdb.computecanada.ca>

as logical reasoning. Basically, analytical methods are considerably dependent on the device being measured. Because, sensitivity, accuracy, and precision are likely to differ significantly amongst various devices. The specific capabilities and restrictions of the measurement device must be taken into consideration while designing analytical methods. Additionally, analytical methods often presuppose linear relationships, which may restrict their capacity to precisely represent data patterns. Also, when it comes to handling noisy or incomplete data or generalizing well to unknown data, analytical methodologies may have limits. If the analytical method's fundamental assumptions are violated, the model's performance may suffer.

Compared with analytical modeling, in this work, we use ML-based algorithms for building power consumption models. In contrast to the analytical method, ML algorithms are hardware independent and generally applicable. Given enough training examples and proper regularization techniques, ML algorithms can frequently handle noisy or incomplete data well and generalize effectively to new, unknown data. Also, ML algorithms are capable of capturing and understanding complicated relationships, allowing them to effectively handle nonlinear patterns. As mentioned in Chapter 4, in the PEBEdge architecture, all components are virtualized and simulated in different containers with a software-based approach that has several inputs (e.g., device data and different applications). In this case, there is a requirement for concentrating on retraining data, so ML algorithms are a better option. Since data can be trained considerably easier and faster by ML methods. However, analytical methods are often interpretable, require fewer computational resources, and have greater accuracy in their results [112,113].

5.3 ML Algorithms

This section describes the detailed steps involved in collecting the dataset, preprocessing, training, validating, and evaluating the proposed ML models. DR as a baseline regressor, NN models (including MLP, and LSTM), along with XGB, RF, DT, LR, AR, and KNN are trained, and the best models based on evaluation metrics are introduced.

The proposed models are used to estimate the instantaneous power consumption for edge devices on a computing structure for IoT applications. The data we use as input for this model include camera and temperature sensor numbers, processing unit type in the edge device, container numbers, and resource utilization metrics including CPU, memory, disk input, disk output, network transmission, and network reception. In these models, the output is the instantaneous power consumption of the edge device.

5.3.1 Dataset

Our experiments are designed to run 35 scenarios on the RPi to obtain the corresponding running time, camera and temperature sensor numbers, processing unit type in edge device, container numbers, and resource utilization metrics including CPU, memory, disk input, disk output, network transmission and network reception as well as instantaneous power consumption for each sample. Table 5.1 presents different scenarios and the corresponding data for them. In order to have an inclusive dataset, the scenarios must include various conditions. We try to design different scenarios in order to cover all possible combinations in accordance with our hardware computation capacity and experiment time limitations. For example, since the camera is sending images, we try for a maximum of 50 sensors, while we try for a maximum of 500 temperature sensors. In this regard, combinations of processing units (Spark and Python), number of workers (1, 2, 3), camera sensors (1, 10, and 50), and temperature sensors (10, 100, and 500) are considered to follow the design representativeness. There is a sample dataset for scenario 1 shown in Table 5.2 demonstrating the exact features and target (instantaneous power consumption). This dataset contains the following parameters:

1. **Number:** Number of data samples.
2. **Running time:** The timestamp for experiments (second).
3. **Camera:** The number of camera sensors.
4. **Temperature:** The number of Temperature sensors.
5. **Processing unit:** The processing unit in the edge device(RPi). For the Spark processor, a value of 1 is specified, while for the Python script processor, a value of 0 is specified.
6. **Container:** The number of containers.
7. **CPU:** The percentage of CPU usage.
8. **Memory:** The percentage of memory usage
9. **Disk input/output:** Disk input/output refers to the operation of reading or writing to or from a physical disk (defined in KB/s). Simply put, it refers to the speed at which data is transferred between the hard disk drive and the RAM.

Table 5.1 Different scenarios and the corresponding data.

Scenario number	Scenario definition	Quantity
1	Spark/1cam/1w	3,755
2	Spark/1cam/2w	3,935
3	Spark/1cam/3w	3,796
4	Spark/1cam/10temp/1w	5,267
5	Spark/1cam/10temp/2w	5,347
6	Spark/1cam/10temp/3w	5,146
7	Spark/10cam/1w	3,777
8	Spark/10cam/2w	4,057
9	Spark/10cam/3w	4,011
10	Spark/10cam/100temp/1w	7,358
11	Spark/10cam/100temp/2w	8,383
12	Spark/10cam/100temp/3w	7,747
13	Spark/10temp/1w	5,155
14	Spark/10temp/2w	5,221
15	Spark/10temp/3w	5,295
16	Spark/50cam/1w	4,993
17	Spark/50cam/2w	4,824
18	Spark/50cam/3w	4,700
19	Spark/50cam/500temp/1w	33,793
20	Spark/50cam/500temp/2w	34,495
21	Spark/50cam/500temp/3w	32,859
22	Spark/100temp/1w	7,587
23	Spark/100temp/2w	7,755
24	Spark/100temp/3w	7,977
25	Spark/500temp/1w	32,336
26	Spark/500temp/2w	30,276
27	Spark/500temp/3w	30,995
28	py/1cam	3,585
29	py/1cam/10temp	5,271
30	py/10cam	3,859
31	py/10cam/100temp	8,037
32	py/10temp	5,295
33	py/50cam	5,441
34	py/100Temp	7,245
35	py/500Temp	31,659
-	Total	381,232

10. **Network transmission/reception:** Network transmission/reception displays the amount of transmitted and received data (defined in KB/s).
11. **Instantaneous power consumption:** As a target of the dataset, shows the power (Watt) that RPi is consuming at any one moment in time.

Table 5.2 Sample dataset for scenario 1.

No.	Time(s)	Cam.	Temp.	Process unit	Container	CPU (percent.)	Memory (percent.)	Disk input (Kbps)	Disk output (Kbps)	Network trans. (Kbps)	Network rec. (Kbps)	Power consumption (W)
1	0.09	1	0	1	4	76.63	9	0	8,736,768	8,531,673	380,279	5.011
2	2.19	1	0	1	4	5.25	7	0	1,355,776	2,033,559	8,579,970	2.110
3	4.19	1	0	1	4	0.66	3	0	1,355,776	28,105	9,519	1.050
4	6.19	1	0	1	4	1.21	0	0	40,960	1,214,705	977,993	1.102
5	8.19	1	0	1	4	32.54	10	0	8,736,768	9,898,152	412,252	2.323

5.3.2 Measuring Baseline

The USB power meter utilized in this study is the AVHzY, CT-3 model ². This power meter boasts a 4.5-digit display. A "4.5-digit display" refers to the number of digits that the meter can show on its display screen. Each digit can display a numerical value from 0 to 9. The "4.5" indicates that there are four full digits, and the display can also show a half-digit, which typically displays only the numbers 0 or 1. Also, the Power meter provides precise measurements with a maximum gain error of 0.1% for both voltage and current readings. The "maximum gain error of 0.1%" refers to the highest possible discrepancy between the measured value and the actual value for both voltage and current readings. To ensure consistency and accuracy in the presentation of numerical results, a significant figure of 4 will be employed. This level of precision allows for a reliable analysis and interpretation of the recorded data, enabling a thorough examination of the experimental findings.

In addition, when the RPi is idle and not performing resource-intensive tasks, the current consumption is relatively low. It can vary depending on the specific model. Based on our baseline measurement, RPi consumes between 200-300 mA ³. According to this value, we are provided with the scale of the actual power consumption.

5.3.3 Evaluation Metrics

Three evaluation metrics are utilized in the experimentation. We used mean square error (MSE), mean absolute error (MAE) [114], and R-squared Error (R^2) [115], as evaluation metrics.

MSE is defined as the mean of the square of the difference between the actual and anticipated

²<https://store.avhzy.com>

³<https://pip.raspberrypi.com/categories/505-computers>

values. MSE is calculated as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad (5.1)$$

Where:

n represents the number of data points.

Y_i is the actual observed value for the i -th data point.

$\hat{Y}_i$ is the predicted or estimated value for the i -th data point.

MAE is a measure of error for matched observations expressing the same phenomena. MAE is calculated as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i| \quad (5.2)$$

Where:

n represents the number of data points.

Y_i is the actual observed value for the i -th data point.

$\hat{Y}_i$ is the predicted or estimated value for the i -th data point.

$|\cdot|$ denotes the absolute value of the enclosed quantity.

How closely the data resembles the regression-fitted line is determined using R^2 . It provides the regression models' quality of fit. The coefficient of determination (R-squared or R^2) is calculated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2} \quad (5.3)$$

Where:

n represents the number of data points.

Y_i is the actual observed value for the i -th data point.

$\hat{Y}_i$ is the predicted or estimated value for the i -th data point.

$\bar{Y}$ is the mean of the observed values.

5.3.4 Experimental Results

The first step in preprocessing is to normalize the training dataset. Here, using MinMaxScaler in the Sklearn library, we normalize the training dataset. where min and max equal the features range.

For performing the experiments, regarding the data set division, we consider two main ap-

proaches.

In the first approach, we use *RepeatedKfold* for 5-fold cross-validation and two times repeat on the training data (10 results per each metric and model). It achieves detailed values and also the mean and standard deviation of the metrics (MAE, MSE, R2). In addition, we evaluate our trained models on the test data separately.

In the second approach, we split the dataset into distinct test and train parts. We, therefore, consider experiment numbers 26, 27 and 35 (total 92,930 samples) in Table 5.1 as test dataset and other experiments (total 288,302 samples) as train dataset. According to this division, 25% of the entire dataset belongs to the test section and 75% to the train section.

Since the value of the model output is a continuous variable. Therefore, we need regression models. Various ML regression models have been used in our experiments, such as DR [116] XGB [117], RF [118], DT [119], LR [120], AR [121], KNN [122], and NN models, including LSTM [123], MLP [124].

Python 3.8 [125] was used to train the ML models, using the Scikit-Learn library [126]. In addition, in Python 3.8, NN models were trained with the Keras high-level NN library [127] using TensorFlow [128].

Dummy Regressor (DR)

DR is a simple baseline model used for regression tasks. It serves as a reference point for evaluating the performance of more complex regression models. DR makes predictions based on simple rules and does not take the input features into account. It can be used to assess whether a regression model performs better than a naive approach. DR provides basic metrics such as mean, median, or constant values as predictions, allowing comparisons against other models. While not suitable for accurate predictions, DR helps establish a baseline for model evaluation and serves as a starting point for assessing the effectiveness of more sophisticated regression algorithms [116].

Results Using Cross-validation Approach

In the first manner, we employ *RepeatedKfold* to do 5-fold cross-validation and two iterations on the training set of data (10 results for each metric and model). Since this is 5-fold cross-validation and 2 repeats, totally there are 10 values for each metric. The standard deviation and mean parameters are calculated from these 10 values for each algorithm. Table 5.3 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the DR algorithm. To assess the efficacy of our met-

ric results, we compute the ratio of MSE and MAE (absolute value) to the average power consumption (output) within the dataset.

Table 5.3 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the DR algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	2.760E-03	1.920	1.437E-03	0
MAE	5.071	1.920	2.641	5.090E-03
R2	14.20	-	-	9.990E-01

Results Using Distinct Experiments as Training and Test Dataset

In addition, in order to make sure that the models work for unseen scenarios, we divide the dataset into distinct train and test portions to evaluate the ML models. The metric results achieved by the DR model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.4.

Table 5.4 DR algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
DR	6.000E-04	2.190	2.33

Linear Regression (LR)

LR is a fundamental statistical modeling technique used to understand and quantify the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and aims to find the best-fit line that minimizes the sum of squared residuals. LR estimates the coefficients for each independent variable, representing the slope of the line, and an intercept term that determines the position of the line. It provides valuable insights into the direction and strength of the relationship, allowing for predictions and inference. LR is widely used in various fields for tasks such as trend analysis, forecasting, and determining the impact of variables on an outcome.

Results Using Cross-validation Approach

Given that the experimental methodology employed in this study involves a 5-fold cross-validation procedure with 2 repeats, it results in a total of 10 metric values for each evaluation

measure. From these 10 values, the standard deviation and mean parameters are computed for each algorithm. This statistical analysis allows for a comprehensive assessment of the performance of the algorithms under investigation. Table 5.5 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the LR algorithm.

Table 5.5 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the LR algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	1.820E-04	1.920	9.475E-05	0
MAE	1.099	1.920	5.723E-01	2.660E-03
R2	1.255E-03	-	-	3.321E-04

Results Using Distinct Experiments as Training and Test Dataset

In addition, in order to make sure that the models work for unseen scenarios, we divide the dataset into distinct train and test portions to evaluate the ML models. The metric results achieved by the LR model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.6.

Table 5.6 LR algorithm metric results on the distinct test dataset.

ML algorithm	MSE (W)	MAE(W)	R2
LR	1.820E-04	1.105	4.611E-04

Decision Tree (DT)

DT is a simple yet powerful supervised learning algorithm that uses a tree-like structure to make decisions or predictions based on input features. It recursively partitions the data based on the feature values, creating a tree where each internal node represents a decision based on a specific feature, and each leaf node represents a predicted outcome or class label. The decision-making process follows a set of rules defined by the tree structure, making it interpretable and easy to understand. DTs are effective for both classification and regression tasks, and they can handle both categorical and numerical features. They are particularly useful for capturing non-linear relationships and interactions between features, making them a popular choice in various domains [119].

Hyper-Parameters Tuning

In the case of the DT experiment, using *RandomizedSearchCV* function from *sklearn* library, we tune model hyperparameters including max depth, min samples split, min samples leaf, and max features. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. We define the hyperparameter search space as shown in Table 5.7. The best-achieved combination of hyperparameters is also presented. As it is shown in Table 5.7 the best hyperparameter combination is as follows: 'min samples split': 2, 'min samples leaf': 1, 'max features': 'log2', 'max depth': 5.

Table 5.7 Defined hyperparameters (including max depth, min samples split, min samples leaf, and max features.) for search space and best-achieved hyperparameters combination for DT.

Hyper parameter	Combination 1	Combination 2	Combination 3	Combination 4	Best combination
Max depth	None	5	10	15	5
Min samples split	2	5	10	-	2
Min samples leaf	1	2	4	-	1
Max features	None	sqrt	log2	-	log2

Results Using Cross-validation Approach

The experimental design implemented in this study involves a 5-fold cross-validation methodology with 2 repeats, resulting in a total of 10 metric values for each evaluation criterion. By analyzing these 10 values, the standard deviation and mean parameters are computed for each algorithm. This statistical analysis enables a comprehensive assessment of algorithmic performance, providing valuable insights into their effectiveness and consistency across the evaluation metrics. Using the Cross-validation approach and best-achieved hyper-parameter combination, Table 5.8 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the DT algorithm.

Table 5.8 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the DT algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	1.810E-04	1.920	9.427E-05	0
MAE	1.099	1.920	5.723E-01	2.670E-03
R2	1.088E-03	-	-	4.260E-04

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by the DT model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.9.

Table 5.9 DT algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
DT	1.810E-04	1.100	9.580E-04

Random Forest (RF)

RF is an ensemble learning method that combines multiple DTs to create a powerful predictive model. Each tree in the RF is trained on a random subset of the data, and the final prediction is determined by aggregating the predictions of all the individual trees. RF introduces randomness in two ways: by randomly selecting subsets of features for each tree and by utilizing bootstrap sampling to create diverse training sets. This randomness helps to reduce overfitting and improve the model's generalization ability. Additionally, RF provides estimates of feature importance, allowing for effective feature selection. It is widely used for classification and regression tasks due to its ability to handle high-dimensional data, capture complex interactions, and provide robust predictions [118].

Hyper-Parameters Tuning

In the case of the RF experiment, using *RandomizedSearchCV* function from *sklearnlibrary*, we tune model hyperparameters including n estimators, max depth, min samples split, min samples leaf, and max features. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. We define the hyperparameter search space as shown in Table 5.10. As it is presented in Table 5.10 the best hyperparameter combination is as follows: 'n estimators': 100, 'min samples

split': 2, 'min samples leaf': 2, 'max features': 'log2', 'max depth': 5

Table 5.10 Defined hyperparameters (including n estimators, max depth, min samples split, min samples leaf, and max features.) for search space and best-achieved hyperparameters combination for RF.

Hyper parameter	Combination 1	Combination 2	Combination 3	Combination 4	Best combination
N estimators	10	30	50	100	100
Max depth	None	5	10	15	5
Min samples split	2	5	10	-	2
Min samples leaf	1	2	4	-	2
Max features	auto	sqrt	log2	-	log2

Results Using Cross-validation Approach

The experimental setup in this study incorporates a 5-fold cross-validation technique with 2 repetitions, resulting in a total of 10 metric values for each evaluated parameter. From these collected values, the mean and standard deviation parameters are computed for each algorithm. Using the Cross-validation approach and best-achieved hyper-parameter combination, Table 5.11 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the RF algorithm.

Table 5.11 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the RF algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	1.800E-04	1.920	9.375E-05	0
MAE	1.099	1.920	5.723E-01	2.670E-03
R2	1.771E-03	-	-	3.146E-04

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by RF model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.12.

Table 5.12 RF algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
RF	1.800E-04	1.101	4.854E-04

eXtreme Gradient Boosting (XGB)

XGB is an optimized implementation of the gradient-boosting ML algorithm, known for its exceptional predictive power and efficiency. XGB builds a strong ensemble model by iteratively combining weak prediction models, typically DTs, to minimize the overall prediction error. It introduces several key enhancements, such as a novel regularization technique called "gradient-based regularization", which helps prevent overfitting, and a more efficient parallel tree construction algorithm. XGB also incorporates a customized loss function and employs an optimized approximation for finding the best splits during tree building. These optimizations make XGB highly effective in handling large-scale datasets and achieving state-of-the-art performance in various ML tasks, including regression, classification, and ranking problems [117].

Hyper-Parameters Tuning

In the case of the XGB experiment, using *RandomizedSearchCV* function from *sklearnlibrary*, we tune model hyperparameters including n estimators, learning rate, max depth, subsample, and colsample bytree. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. We define the hyperparameter search space as shown in Table 5.13. The best-achieved combination of hyperparameters as presented in Table 5.13 is: 'subsample': 0.8, 'n estimators': 200, 'max depth': 3, 'learning rate': 1.0, 'colsample bytree': 0.8.

Table 5.13 Defined hyperparameters (including n estimators, learning rate, max depth, subsample, and colsample bytree) for search space and best-achieved hyperparameters combination for XGB.

Hyper parameter	Combination 1	Combination 2	Combination 3	Combination 4	Best combination
N estimators	50	100	200	-	200
Learning rate	0.001	0.01	0.1	1.0	1
max depth	3	6	9	-	3
subsample	0.6	0.8	1	-	0.8
colsample bytree	0.6	0.8	1	-	0.8

Results Using Cross-validation Approach

Since this is 5-fold cross-validation and 2 repeats, totally there are 10 values for each metric. The standard deviation and mean parameters are calculated from these 10 values for each algorithm. Using the Cross-validation approach and best-achieved hyper-parameter combination, Table 5.14 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the XGB algorithm.

Table 5.14 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the XGB algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	1.911E-04	1.920	9.953E-05	0
MAE	1.109	1.920	5.776E-01	2.610E-03
R2	7.674E-03	-	-	2.307E-04

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by the XGB model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.15.

Table 5.15 XGB algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
XGB	1.919E-04	1.098	5.660E-04

AdaBoost Regressor (AR)

AR is a boosting algorithm that combines multiple weak regression models to create a strong predictive model. It works by iteratively training weak regressors on the data, assigning higher weights to instances that were previously mispredicted, and adjusting the weights to focus on the harder-to-predict cases. The final prediction is a weighted combination of the weak regressors, where each regressor's contribution is determined by its performance. AR is particularly effective when dealing with complex and non-linear relationships between variables, as it adapts to the data and places more emphasis on challenging instances. It is widely used for tasks such as predicting continuous numerical values and can provide accurate and robust predictions even in the presence of noisy or inconsistent data [121].

Hyper-Parameters Tuning

In the case of the AR experiment, using *RandomizedSearchCV* function from *sklearn* library, we tune model hyperparameters including n estimators, learning rate, and loss. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. We define the hyperparameter search space as shown in Table 5.16. The best-achieved combination of hyperparameters is 'n estimators': 200, 'loss': 'exponential', 'learning rate': 0.001 which is presented in Table 5.16

Table 5.16 Defined hyperparameters (including n estimators, learning rate, and loss.) for search space and best-achieved hyperparameters combination for AR.

Hyper parameter	Combination 1	Combination 2	Combination 3	Combination 4	Best combination
N estimators	50	100	200	-	200
Learning rate	0.001	0.01	0.1	1.0	0.001
Loss	Linear	Square	Exponential	-	Exponential

Results Using Cross-validation Approach

Using the Cross-validation approach and the best-achieved hyper-parameter combination, Table 5.17 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the AR algorithm.

Table 5.17 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the AR algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	1.900E-04	1.920	9.895E-05	0
MAE	1.100	1.920	5.729E-01	2.640E-03
R2	1.248E-03	-	-	3.678E-04

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by AR model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.18.

Table 5.18 AR algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
AR	1.900E-04	1.100	2.525E-04

K-nearest neighbor (KNN)

KNN is a simple yet powerful algorithm used for both classification and regression tasks. It operates by finding the K closest training instances to a given test instance in the feature space and making predictions based on the majority class (for classification) or the average value (for regression) of those K neighbors. KNN assumes that similar instances tend to have similar outputs and uses the concept of proximity to determine predictions. The value of K, representing the number of neighbors considered, is a crucial parameter that affects the algorithm's performance. KNN is a non-parametric method and does not make any assumptions about the underlying data distribution. It is easy to understand, versatile and can handle complex relationships, making it a popular choice for various ML tasks KNN [122].

Hyper-Parameters Tuning

In the case of the KNN experiment, using *RandomizedSearchCV* function from *sklearn* library, we tune model hyperparameters including n neighbors, weights, and "p". The parameter "p"

refers to the "p-norm" used to measure the distance between data points. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. We define the hyperparameter search space as shown in Table 5.19. The best-achieved combination of hyperparameters as presented in Table 5.19 is 'weights': 'uniform', 'p': 2, 'n neighbors': 11.

Table 5.19 Defined hyperparameters (including n neighbors, weights, and "p") for search space and best-achieved hyperparameters combination for KNN.

Hyper parameter	Combination 1	Combination 2	Combination 3	Combination 4	Combination 5	Best combination
N neighbors	3	5	7	9	11	11
Weights	Uniform	Distance	-	-	-	Uniform
"p"	1	2	-	-	-	2

Results Using Cross-validation Approach

The standard deviation and mean parameters are calculated from these 10 values for each algorithm. Using the Cross-validation approach and best-achieved hyper-parameter combination, Table 5.20 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the KNN algorithm.

Table 5.20 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the KNN algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	2.100E-04	1.920	1.093E-04	0
MAE	1.137	1.920	5.921E-01	2.53E-03
R2	8.828E-02			1.156E-03

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by the KNN model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.21.

Table 5.21 KNN algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
KNN	1.900E-04	1.140	7.837E-02

Multilayer Perceptron (MLP)

MLP is a type of artificial NN that consists of multiple layers of interconnected nodes. It uses a feed-forward architecture, where information flows from the input layer through the hidden layers to the output layer. Each node applies a non-linear activation function to the weighted sum of its inputs, allowing the MLP to model complex relationships and handle non-linear data. MLPs are commonly used for tasks such as regression and classification, and they have found applications in various domains including image recognition and natural language processing [124].

Hyper-Parameters Tuning

In the case of the MLP experiment, using *RandomizedSearchCV* function, we tune model hyperparameters including neurons, hidden layers, batch size, and epochs. Randomized search finds the best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. In our model, hidden layers with *relu* activation function are followed by an output layer with *linear* activation function. Also, Adam loss function Optimizer is used to update the weights and minimize the MSE as the loss function. We define the hyperparameter search space as shown in Table 5.22. The best-achieved combination of hyperparameters is also presented in Table 5.22. As shown, the best hyperparameter combination is: 'neurons': 128, 'hidden layers': 2, 'epochs': 50, 'batch size': 500

Table 5.22 Defined hyperparameters (including neurons, hidden layers, batch size, and epochs) for search space and best-achieved hyperparameters combination for MLP.

Hyper parameter	Combination 1	Combination 2	Combination 3	Best combination
Neurons	64	128	256	128
Hidden layers	1	2	3	2
Batch size	100	500	1000	500
Epochs	50	100	200	50

Figure 5.1 shows the MLP algorithm progression of MSE during training in 128 neurons, 2 hidden layers, 50 epochs, and 500 batch size.

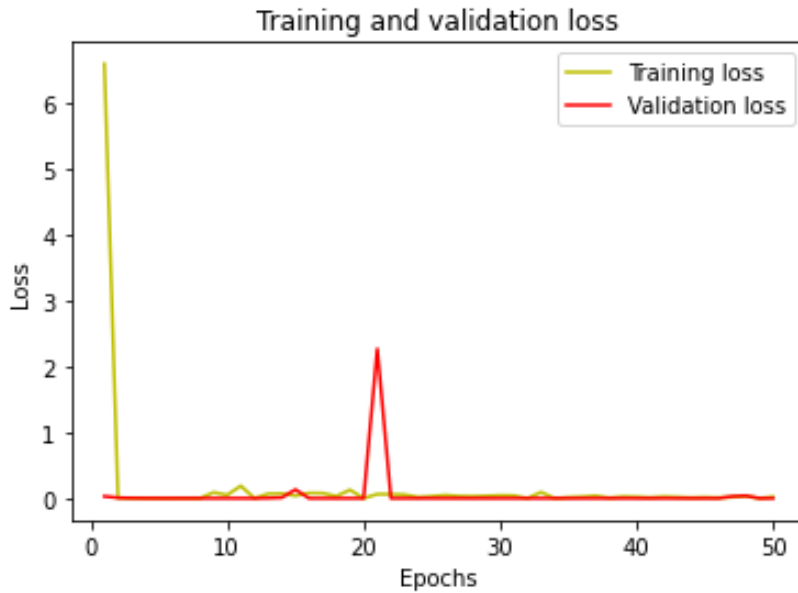


Figure 5.1 MLP algorithm progression of MSE for training in 128 neurons, 2 hidden layers, 50 epochs, 500 batch size.

Results Using Cross-validation Approach

Since this is 5-fold cross-validation and 2 repeats, totally there are 10 values for each metric. The standard deviation and mean parameters are calculated from these 10 values for each algorithm. Using the Cross-validation approach and best-achieved hyper-parameter combination, Table 5.23 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the MLP algorithm.

Table 5.23 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the MLP algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	2.000E-03	1.920	1.041E-03	2.740E-03
MAE	2.740	1.920	1.427	1.669
R2	9.988	-	-	14.98

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by the MLP model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.24.

Table 5.24 MLP algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
MLP	2.200E-03	3.385	58.89

Long Short-Term Memory (LSTM)

LSTM is a type of recurrent neural network (RNN) architecture that addresses the limitations of traditional RNNs in capturing long-range dependencies. It is designed to effectively model and learn from sequences of data by incorporating memory cells and gating mechanisms. LSTM networks are equipped with special memory cells that can retain information over long periods of time, allowing them to capture and remember important patterns or contexts from past inputs. The gating mechanisms, consisting of input, forget, and output gates, regulate the flow of information within the network, enabling effective learning and information retention. LSTM networks have been widely utilized in tasks involving sequential data such as speech recognition, natural language processing, and time series prediction [123].

Hyper-Parameters Tuning

In the case of LSTM experiments, using *RandomizedSearchCV* function from *sklearn* library, we tune model hyperparameters including neurons, batch size, and epochs. Randomized search finds the-best combination of hyperparameters within the given search space. Then, the best model is used to make predictions. In our model, the LSTM layer is followed by an output layer with *linear* activation function. Also, the Adam loss function Optimizer is used to update the weights and minimize the MSE as the loss function. We define the hyperparameter search space as shown in Table 5.25. The best-achieved combination of hyperparameters is also presented in Table 5.25. As shown, the best hyperparameter combination is: 'neurons': 100, 'epochs': 50, 'batch size': 500.

Table 5.25 Defined hyperparameters (including neurons, batch size, and epochs) for search space and best-achieved hyperparameters combination for LSTM.

Hyper parameter	Combination 1	Combination 2	Combination 3	Best combination
Neurons	50	100	200	100
Batch size	100	500	1000	500
Epochs	50	100	200	50

Figure 5.2 shows the LSTM algorithm progression of MSE during training in 100 neurons, 50 epochs, and 500 batch size.

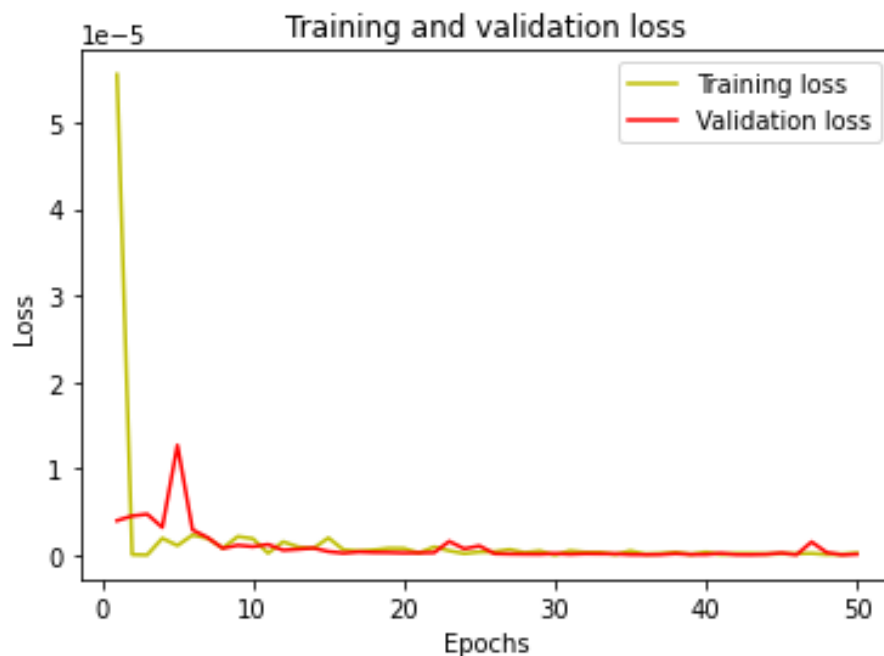


Figure 5.2 LSTM algorithm progression of MSE during training in 100 neurons, 50 epochs, and 500 batch size.

Results Using Cross-validation Approach

Using the Cross-validation approach and best-achieved hyper-parameter combination, since this is 5-fold cross-validation and 2 repeats, totally there are 10 values for each metric. The standard deviation and mean parameters are calculated from these 10 values for each algorithm. Table 5.26 presents the result of the cross-validation experiment that shows the mean and standard deviation for MSE, MAE, and R2 in the LSTM algorithm.

Table 5.26 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the LSTM algorithm.

Metric	Mean (W)	Average actual output (W)	Relative error (W)	Standard deviation (W)
MSE	3.000E-04	1.920	1.562E-04	9.000E-05
MAE	1.482	1.920	7.718E-01	2.662E-01
R2	6.415E-01	-	-	5.158E-01

Results Using Distinct Experiments as Training and Test Dataset

The metric results achieved by the LSTM model on the test dataset (three large experiments with about 25% of whole data) are presented in Table 5.27.

Table 5.27 LSTM algorithm metric results on the distinct test dataset.

ML algorithm	MSE(W)	MAE(W)	R2
LSTM	2.900E-04	1.478	6.208E-01

5.3.5 Results Comparison

Table 5.28 provides a comprehensive comparison of the obtained results, employing DR as the baseline regressor, along with LR, DT, RF, XGB, AR, KNN, and NN (including MLP and LSTM). Based on the information presented in the table, several conclusions can be drawn as follows.

- The results clearly demonstrate that the trained models significantly outperform the baseline regressor. The metrics for MSE and MAE indicate substantially smaller values in comparison to the baseline model.
- Trained models are performing better at minimizing the squared errors (MSE) rather than the absolute errors (MAE) or capturing the overall variance (R2) of the data. This can be a result of the fact that if the relationship between the features and the target variable in the dataset is nonlinear, it can lead to a discrepancy between the metrics. MSE, being sensitive to squared errors, may perform better when the model captures quadratic or higher-order relationships. On the other hand, MAE and R2 may not be as good at capturing the nonlinearities and may provide lower values. Also, when heteroscedasticity, which refers to the unequal variance of errors across the range of

feature variables, is present, the errors may have different variances at different levels of the predictor variables. This can impact the performance of metrics like MAE and R2.

- In Figure 5.3, we present a comparison of the MSE and MAE for the different models. The findings reveal that the ML models outperform the NN models. Furthermore, upon closer examination of the results, it becomes evident that LR, DT, RF, XGB, and AR exhibit outstanding performance and demonstrate considerable similarity in their outcomes. Consequently, LR, DT, and RF models are considered to be the most favorable models in this study.

Table 5.28 Result of cross-validation experiments including mean and standard deviation for MSE, MAE, and R2 in the XGB, RF, DT, LR, AR, and KNN algorithms.

ML algorithm	MSE mean(W)	MAE mean(W)	R2 Mean
Dummy Regressor (DR)	2.760E-03	5.071	14.20
Linear Regression (LR)	1.820E-04	1.099	1.255E-03
Decision Tree (DT)	1.810E-04	1.099	1.088E-03
Random Forest (RF)	1.800E-04	1.099	1.771E-03
eXtreme Gradient Boosting (XGB)	1.911E-04	1.109	7.674E-03
AdaBoost Regressor (AR)	1.900E-04	1.100	1.248E-03
K-nearest neighbor (KNN)	2.100E-04	1.137	8.828E-02
Multilayer Perceptron (MLP)	2.000E-03	2.740	9.988
Long Short-Term Memory (LSTM)	3.000E-04	1.482	6.415E-01

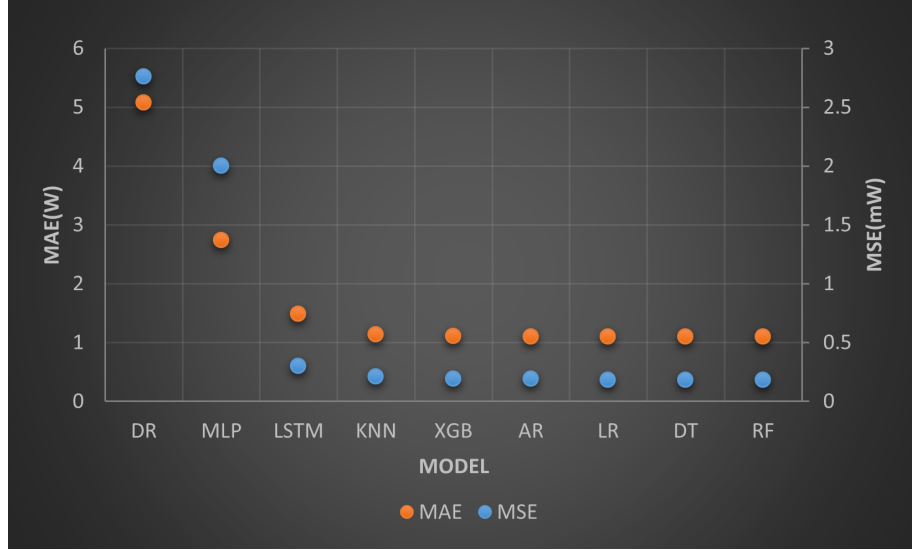


Figure 5.3 MSE and MAE comparison for the trained models.

5.3.6 Significance of Error

As shown in Table 5.28, the MSE for RF is 1.8E-04 watts. Regarding the significance of the error, for a battery with a capacity of 4000 mAh, we compare the battery depletion time difference between the model's actual output power (1.92 watts) with and without considering the error:

$$P(\text{output}) = 1.92 \text{ Watt}$$

$$P(\text{output considering error}) = 1.92018 \text{ Watt}$$

The voltage of the edge terminal is 5 volts. Therefore, the average currents (in Ampere) for the edge device are calculated as follows:

$$I = \frac{P}{V} \quad (5.4)$$

$$I(\text{output}) = \frac{1.92}{5} = 0.384$$

$$I(\text{output considering error}) = \frac{1.92018}{5} = 0.384036$$

Based on the battery capacity formula, the equation is given by:

$$C(\text{mAh}) = I(\text{mA}) \times t(\text{h}) \quad (5.5)$$

$$t = \frac{C}{I} \quad (5.6)$$

$$t(\text{without considering error}) = \frac{4000}{384} = 10 \text{ hours and } 25 \text{ minutes}$$

$$t(\text{considering error}) = \frac{4000}{384.036} = 10 \text{ hours and } 24 \text{ minutes and } 55 \text{ seconds}$$

As evident from the data, the comparison of battery depletion time with and without considering the error reveals a minimal difference of only 5 seconds. Consequently, the significance of the error in our models is deemed negligible.

5.4 Suggested Solutions for Improving the Lack of Reliability

Using the best-achieved models, the purpose of this section is to provide some solutions to improve the lack of reliability imposed by energy-constrained edge devices as follows.

1. Identifying the most significant factors that impact edge device power consumption.
2. Changing the system software-associated configuration for decreasing power consumption in critical situations.
3. Determining the approximate edge device battery depletion time using power estimation models

5.4.1 Identifying the Most Important Parameters Affecting the Power Consumption

Unlike the analytical method developed by WattEdge, we use ML algorithms for feature-importance targets. The concept of feature importance in ML refers to identifying the most important features or variables that contribute the most to the outcome of the model. As a result, feature importance indicates which variables have the greatest impact on the model's predictions.

Table 5.28 demonstrates that among the ML algorithms, LR, DT, and RF perform well and

have considerable similarities in their findings.

Feature Importance in RF Algorithm

In the RF algorithm, using the `feature_importances` attribute, it is possible to rank the dataset feature based on their importance. Table 5.29 shows the feature importance based on the RF algorithm calculation. Results show that CPU percentage has the most value of the feature importance in this calculation, network reception and transmission place second and third, respectively.

Table 5.29 Feature importance based on the RF algorithm calculation.

Feature Rank	Feature Name	Feature Importance
1	CPU	0.4120
2	Network Reception	0.2580
3	Network Transmission	0.2530
4	Disk Input	0.0170
5	Camera Numbers	0.0130
6	Temperature Numbers	0.0130
7	Memory	0.0110
8	Container Numbers	0.0110
9	Disk Output	0.0060
10	Processing Unit	0.0012

Feature Importance in DT Algorithm

In the DT algorithm, using the `feature_importances` attribute, it is possible to sort the dataset feature based on their importance. Table 5.30 shows the feature importance based on the DT algorithm calculation. Results show that CPU percentage has the most value of the feature importance in this calculation. network reception and transmission place second and third, respectively.

Table 5.30 Feature importance based on the DT algorithm calculation.

Feature Rank	Feature Name	Feature Importance
1	CPU	0.415
2	Network Reception	0.2590
3	Network Transmission	0.2370
4	Camera Numbers	0.0240
5	Disk input	0.0170
6	Temperature Numbers	0.0120
7	Memory	0.0100
8	Container Numbers	0.0090
9	Disk Output	0.0080
10	Processing Unit	0.0050

Feature Importance in LR Algorithm

In the case of the LR model, generally, coefficients are found to be considered as the feature importance. Larger coefficients present a stronger effect on the target variable. Table 5.31 shows the feature importance based on the LR algorithm calculation. Results show that disk input has the most value of the feature importance in this calculation. The CPU and memory place second and third, respectively. A negative coefficient in LR indicates that the independent variable and the dependent variable have an inverse relationship. In other words, when the independent variable rises, the dependent variable falls.

Table 5.31 Coefficients based on the LR algorithm calculation.

Feature Rank	Feature Name	coefficient
1	Disk Input	8.085e-06
2	CPU	5.895e-06
3	Memory	6.019e-07
4	Network transmission	5.977e-07
5	Container Numbers	4.712e-07
6	Camera Numbers	7.744e-08
7	Temperature Numbers	1.147e-08
8	Network reception	2.627e-09
9	Disk output	-8.482e-07
10	Processing unit	-6.178e-06

Feature Importance Results and Comparison with WattEdge

Our results demonstrate the importance of different factors in the power consumption of power-constrained edge devices. Based on the DT and LR results tables, there is a strong similarity between the order of the results. CPU, Network reception, and Network Transmission have the most significant impact on power consumption. The RPi is intended to outperform standard micro-controllers in terms of performance. It takes advantage of a more powerful CPU, allowing it to run a full operating system such as Linux. The greater performance necessitates the use of additional power. Hence, CPU power consumption is more compared with other parameters. Also, the RPi must use its network interface to send and receive data packets whenever it is engaged in network communication, whether it is sending or receiving data. The more data the RPi must process and transport, the higher the network bandwidth must be. When workload generators deliver images, they use a lot more network bandwidth and have a significant influence on power consumption at various data transmission and reception rates. previous studies [109, 129] confirm our findings.

5.4.2 Potential Software Configuration Adjustment for Managing Power Consumption

WattEdge focuses on hardware-related aspects that influence power consumption. For instance, in critical situations in terms of battery level, in order to manage power consumption, they suggest changing the hardware configuration like replacement in connectivity means such as WiFi, Bluetooth, ZigBee, USB, HDMI, VNC, and Ethernet or battery type (e.g., Lithium-ion) and charge level.

Because the scope of our task is software engineering, we focus on software-related aspects such as processing units and worker numbers. For example, in the crucial condition, based on Figure 4.3, we suggest switching from the Python script processing unit to the Spark streaming in order to decrease power consumption. Based on the other software-associated parameters like container numbers or possible computation resources (e.g., CPU, memory, storage), as long as system performance is in the satisfactory range, we can swap to different options. However, the best models include both approaches (hardware and software). Following that, a decision to change the software or hardware configuration may be made. Some combinations are less expensive than others. When there is a significant distance, for example, replacing WiFi with Bluetooth is not an option. In addition, It is important to dynamically redeploy software between edge devices and nodes to redistribute power consumption. In edge computing systems, where computing tasks are distributed across multiple devices and nodes, power consumption can become imbalanced. Some devices or nodes may consume more power than others, leading to inefficiencies or potential overload. To address this issue, dynamically redeploying software-related parameters, and shifting them from heavily loaded devices to underutilized ones, can help distribute the power consumption more evenly. By redistributing the workload, the goal is to optimize power usage and ensure efficient utilization of resources within the edge computing network.

5.4.3 A Case Study For Battery Depletion Calculation

This section presents a case study in which an estimation model is used to anticipate battery depletion on an edge device that performs the task of data analysis. In other words, we now discuss how the power estimation model allows users to plan scenarios, design use cases, and create and run experiments to estimate battery lifetime. We present a variety of experiments to study the results.

Motivation Example

The motivational example for this section is to calculate battery depletion time in different situations. It is intended to illustrate how our power estimation model facilitates the elimination of battery failures in edge computing platforms. Also, we investigate how different loads or different topologies affect battery consumption and depletion.

As a result, the battery depletion time can be determined. For instance, if we obtain that the battery depletion time is n hours. Thus, the edge node could be addressed in less time, such as $n-1$ hours.

Running Experiments

Firstly, we must take the following steps for the experiments on the intended edge devices to provide the input data:

1. The values for edge device resource utilization, including CPU, memory, disk input/output, and network transmission/reception.
2. The type of processing unit and the number of containers.
3. The load volume (camera and temperature sensors)

Secondly, based on the ML models performance metrics results, we use RF model ⁴ in Github, as the best model, and feed that with each set of input data(1,2,3) separately.

According to the power estimation model output, the average power consumption in watts for edge devices 1,2,3 are 4.501, 6.300, and 6.002, respectively.

$$P1 = 4.501 \text{ Watt}$$

$$P2 = 6.300 \text{ Watt}$$

$$P3 = 6.002 \text{ Watt}$$

Finally, using these results, it is possible to calculate the battery depletion time by the following steps:

1. Calculating Average Current:

The electrical power consumption (P) in a circuit can be calculated using the formula:

$$P = V \times I \tag{5.7}$$

⁴https://github.com/mahsahadian/Energy_Model_Predition

Where:

P is the power in watts (W),

V is the voltage in volts (V), and

I is the current in amperes (A).

The voltage of the edge terminal is 5 volts. Therefore, the average currents (in Ampere) for edge device 1,2,3 are calculated as follows:

$$I = \frac{P}{V} \quad (5.8)$$

$$\begin{aligned} I1(A) &= \frac{4.501}{5} = 0.9002 \\ I2(A) &= \frac{6.300}{5} = 1.260 \\ I3(A) &= \frac{6.002}{5} = 1.200 \end{aligned}$$

2. Calculating Battery Efficient Capacity:

Based on the battery capacity formula, the equation is given by:

$$C = I \times t \quad (5.9)$$

Where:

C is the battery capacity in milliampere-hours (mAh).

I is the current in amperes (mA) that the battery can provide.

t is the time in hours (h) for which the battery can sustain that current.

First, we consider a small RPi battery ⁵that has a capacity of 4000 mAh with a maximum output of 5V and 3A (maximum). Then, following the lithium battery discharge efficiency principle, a coefficient of 0.95 is applied to the nominal battery capacity. Therefore, the actual battery capacity is 3800 mAh after multiplying 4000 by 0.95.

⁵<https://www.amazon.com/VGE-Battery-Raspberry-4000mAh-Adhesive/dp/B09BNRKQD8>

$$\begin{aligned}
C(\text{mAh}) &= 4000 \times 0.95 \\
&= 3800
\end{aligned}$$

3. Calculating RPi Battery Depletion Time:

The actual battery capacity is 3800 mAh. Therefore, the battery depletion time for edge devices 1,2,3 are calculated as follows:

$$t = \frac{C}{I} \quad (5.10)$$

$$\begin{aligned}
t1 &= \frac{3800}{900.2} = 4 \text{ hours and } 15 \text{ minutes} \\
t2 &= \frac{3800}{1260} = 3 \text{ hours and } 1 \text{ minute} \\
t3 &= \frac{3800}{1200} = 3 \text{ hours and } 10 \text{ minutes}
\end{aligned}$$

In another case, if we use just one battery to parallel these edge devices, the average current (in Ampere) would be 3.360. Consequently, the depletion time would be 1 hour and 10 minutes.

5.5 Discussion

In our experiments, data sets from 35 different types of experiments were used to conduct the experiments, resulting in 381232 samples being collected. Using cross-validation, we ran numerous experiments, including MLP, LSTM, XGB, RF, DT, LR, AR, and KNN. Also, applying a separate test dataset, all experiments were repeated. It is possible to run each experiment in less than an hour. But, it would take a lot more time and effort to run these experiments on the physical infrastructure, without the model. We found that the ML models we use in our experiment are capable of more generalization compared to the analytical (i.e., mathematical or statistical) models from the perspective of an edge device type, which is the main advantage of the method.

CHAPTER 6 LIGHTWEIGHT BENCHMARKING FOR POWER CONSUMPTION

6.1 Introduction

This section is dedicated to elucidating the methodology employed in facilitating researchers to conduct experiments pertaining to edge device power consumption. Our approach significantly reduces the time required for experimentation while eliminating the necessity for specialized hardware, such as an edge device or power meter.

6.2 Lightweight Benchmarking

The PEBEdge framework exhibits a specific limitation wherein it relies on physical infrastructure, namely the requirement of a power-measuring device and an RPi for operation. However, an alternative solution arises when employing a power consumption prediction model, as it enables the complete replacement of the hardware infrastructure. By employing the proposed model as a simulator in our research, the user can eliminate the need for RPi, power meters, or any other costly equipment. The model enables experimentation solely with the utilization of a laptop while considering the underlying PEBEdge architecture. This approach offers a cost-effective and convenient alternative, simplifying the experimental setup and removing the requirement for specialized hardware.

In this work, we created PEPEdge, gathered all the data, and trained the models. By utilizing such a model, the need for physical measurement devices and an RPi is obviated. To illustrate this point, let us consider a hypothetical scenario wherein a new user aims to simulate and investigate an experiment focused on analyzing the power consumption of an edge device under various conditions. This example serves to demonstrate the potential application of our approach to facilitating the study and analysis of edge device power consumption in diverse scenarios.

In this experimental setup, the user specifies the input parameters, including the type of application, the number of cameras, and the number of temperature sensors involved. These devices transmit information at arbitrary intervals, such as sending a camera image every minute and a temperature signal every second. The flexibility of the experiment allows for varying data transmission speeds, accommodating a range of real-world scenarios. For the processing unit, two options are available: Spark streaming or Python code. The user can choose either of these options based on their specific requirements. Additionally, the RPi can

be configured with different numbers of workers, including 1, 2, or 3. This inherent flexibility offers researchers the opportunity to conduct systematic experimentation and optimize the processing setup according to their specific objectives. Regarding the computation resource utilization (e.g., CPU, memory, storage, and network bandwidth), the maximum available resource amounts for the edge device (e.g., total CPU cores, total memory, storage, and network bandwidth) are considered. With the availability of these specified inputs, users gain the capability to conduct a comprehensive simulated benchmark experiment utilizing solely the trained model. This streamlined approach enables the generation of power consumption outputs as the primary outcome of the experiment. The trained model serves as a powerful tool for simulating and predicting power consumption, facilitating in-depth analysis and insights without the necessity of running resource-intensive experiments. Furthermore, it should be noted that since all parameters are simulated, researchers have the opportunity to replicate the experiments solely by utilizing the provided models.

6.3 Case Study

Utilizing the trained models, we replicate the physical experiment conducted in Chapter 4, which constitutes the case study component of this research. By leveraging the insights and predictions generated by the trained models, we recreate the experimental conditions and parameters to validate and compare the obtained results. This iterative process of integrating the trained models with the physical experiment allows for a comprehensive evaluation of the models' performance and their applicability in real-world scenarios. Instead of going through the workload, application, processing unit, and so on, we just replace them with the trained model.

6.3.1 Motivation Example

The motivation example for this section stems from the intention to compare and contrast the configurations and outcomes of a real experiment, as previously conducted in the case study of Chapter 4, with experiments using ML models. This comparative analysis aims to investigate the advantages and disadvantages associated with the utilization of trained models as opposed to the physical approach.

6.3.2 Research question

The primary research question focuses on the experimental setup using trained models. It seeks to understand the process employed to configure and conduct experiments utilizing

trained models.

The second research question revolves around assessing the level of error observed between the results obtained from trained models and the physical benchmark. This research inquiry aims to quantify and evaluate the extent of disparities or deviations between the outcomes derived from trained models and the established physical benchmark.

6.3.3 Running Experiment

To initiate experiments utilizing trained models, the following inputs must be determined. These include:

- Specification of the application type, including the number of cameras and temperature sensors involved. In the experiment, a single camera is utilized as the sensor.
- Selection of the processing unit, offering two options: Spark streaming or Python code. In the trained model experiments, we use both Python and Spark processing units.
- Determination of the worker numbers, specifically applicable when utilizing Spark streaming. Here we use one worker.
- Specification of the edge device's resources, including the CPU percentage, memory percentage, storage capacity, and network bandwidth. We repeat the value for resource utilization in the case study of Chapter 4.

It is noteworthy that the proposed approach eliminates the need for any hardware and complicated configurations. For example, parameters such as IP settings and other related configurations are not required.

Following the analysis of performance metrics results obtained from the ML models (see Table 5.28), the RF model emerges as the top-performing model. Consequently, the RF model is selected as the preferred model for further analysis. Subsequently, the RF model is fed with the input data for evaluation. Figure 6.1 presents a comparison of the achieved power consumption results between the physical experiment and the RF model, utilizing Python as the chosen processing unit. This comparison allows for an assessment of the consistency and accuracy of power consumption predictions derived from the RF model.

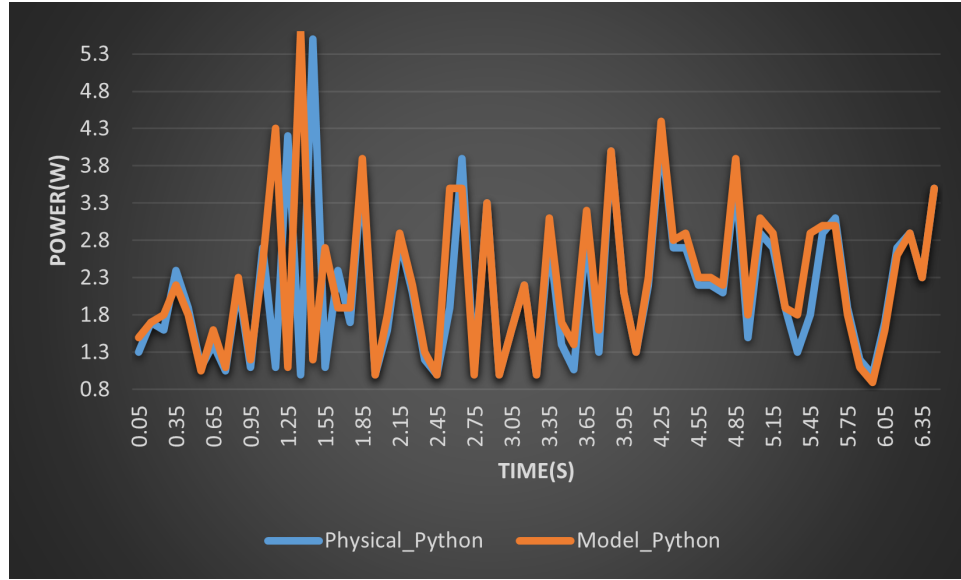


Figure 6.1 Comparison results of the physical and model-based experiment (Python script).

By examining the power consumption results from both the physical experiment and the RF model under different processing units, an understanding of the model's performance and its alignment with real-world scenarios can be obtained. The findings derived from Figure 6.1 demonstrate a remarkable similarity between the results obtained from the physical experiment and the model-based experiment and the average values are approximately the same with each other. Notably, conducting the experiment solely based on the model-based approach, without the need for any hardware, yields outcomes that closely align with the results from the physical experiment. Moreover, the model-based experiment exhibits a significantly reduced time requirement compared to the physical experiment.

6.4 Discussion

As a result, since all the parameters are simulated in this work, the user is able to repeat the experiments only using the provided models. Also, compared with the several hours expected for obtaining the results from the real experiment setting and device configuration, using the provided models, the user can repeat and run the simulated experiments and achieve the outputs in various conditions in a few minutes. In addition, in terms of the required hardware for performing the experiment, It is clear that the achievement of this work enables the user low-cost experiments.

CHAPTER 7 DISCUSSION

When opposed to data centers, edge devices, and edge nodes often have fewer capabilities. These restrictions are brought on by the lesser storage and computation capacity, as well as the lower power autonomy, of these devices. While doing complicated data analytics at the edge, edge devices are more susceptible to power usage. The depletion of batteries or the challenge of acquiring energy may result in a loss of reliability. We need to be able to forecast needed power consumption in various scenarios and for various workloads in order to prevent an unplanned disruption in the edge computing network.

To this end, a benchmarking edge platform and prototype structure, called PEBEdge, is required. PEBEdge is a benchmarking tool to assess and examine the edge network in real situations where it must operate constantly under a changing and growing burden. In this work, a real-time and online monitoring dashboard displays immediate power consumption in addition to tracking and measuring several resource utilization indicators (CPU consumption, memory usage, network I/O, and disk I/O). Benchmarking edge platforms research can be facilitated by PEBEdge, which can also enhance knowledge and proficiency in the field. This work offers considerable insight into various implementations for future researchers by conducting a case study experiment as well as evaluation and comparison analysis. It seeks to tackle the issues with data collection and testing a specific set of applications. I provided a step-by-step explanation of how to construct this infrastructure by getting the components ready in Chapter 4. (e.g., VMs, RPi, power meter, and others). It uses a three-tier design with a cloud layer, an edge layer, and a sensors layer. All layers communicate with one another in order to deliver sensor data to the edge tier. To store benchmarking data in the main data source, the edge network connects with a centralized cloud. A case study was created to demonstrate how PEBEdge can be employed by developers for evaluating power consumption in edge devices. The study showcased the ease and speed of experiment setup and modification, such as transitioning between Spark and Python scripts or adjusting the number of workers. Important configurations during the transition process were highlighted, emphasizing PEBEdge's extensibility.

ML-based power estimation models for edge computing platforms are provided in Chapter 5. We used PEBEdge to gather a replication package with 381,232 samples (train and test). The data is openly accessible and made available to other researchers so they can support various issues, inquiries, and contributions. Following that, ML models for power estimation were provided in order to anticipate the amount of power consumed by edge devices. According

to the input features (resource utilization and others), the suggested model calculates an estimation of the value of power consumption. I described in detail the model's training, validation, and assessment steps. The best model in terms of the performance metric is chosen after training on baseline regressors (DR), NN models (including MLP and LSTM) and ML algorithms (including XGB, RF, DT, LR, AR, and KNN). To address reliability concerns, the study suggested identifying key parameters influencing power consumption and exploring potential software configuration adjustments. An example showcasing the prediction of the edge device's battery depletion time was also provided.

In Chapter 6, we presented a lightweight benchmarking approach for power consumption, demonstrating the capability of trained models to offer researchers efficient and cost-effective experiments for edge platform benchmarking.

Through the integration of both contributions, namely the benchmark framework and estimation mode, a planning tool can be developed to effectively assist decision-making processes for planners, architects, and experimenters.

In the end, although we cannot claim that the presented models will be accurate and appropriate in all case studies, by following the proposed pipeline one can retrain a new model for a new set of data, gathered by the benchmark.

Automatic System Improvement

Another topic that can be discussed is the fact that software systems usually encounter highly volatile environments, and their incoming traffic can lead to high fluctuations for reasons that cannot be predicted. Cloud computing provides solutions by offering high-quality services, and flexible and elastic resources, like VMs and containers, which can be quickly and efficiently scaled up or down to meet the ongoing workload demands. Also, it suggests the necessity for effective and modern management systems, which will accomplish the intentions of low cost automatically and consistently as well as high-quality services. One way to manage software systems is to automate this process. Several methods have been proposed and used to implement autonomic management systems (AMS), such as threshold-based, control-based, and model-based approaches.

An adaptive system is a system able to function correctly, within parameters fixed by the service level objective (SLO), without human intervention [130]. Different AMS generally should follow the MAPE-K reference architecture [69]. The MAPE framework starts with the monitoring module, which is responsible for collecting metrics and measurements to cover the managed system's state. These metrics will be transferred to the analysis module, which will recognize every off-measurement which has possibly problematic cases. After the planning module evaluates the condition, the best possible and efficient actions should be proposed to

address the issue if there is a problem. Then passed on to the next step, which is the execution module. Since the execution module can access the software system and the infrastructure, it can apply and execute the corrective actions. Different AMS implementations and adaptation methods may focus on different MAPE-K architecture modules, usually on the analysis and planning. The monitoring and execution modules are third-party tools available by the cloud or the software provider [131].

PEBEdge can continuously monitor and collect measurements about response time (as performance) and power consumption. Therefore, PEBEdge can be used in the AMS in the following ways:

- It can be used to generate data (without the real infrastructure) and train the ML model.
- Then it can be deployed as a monitoring system (for both energy and performance) for the real system.
- The ML model can be used as the analysis and planning modules.
- If an energy or performance bottleneck is detected a migration of the analytical task may be invoked temporarily or permanently to the edge node.
- The fact that the entire infrastructure is containerized can enable this adaptive action to be quick and without much impact to guarantee the reliability of the entire network.

Research Limitation:

There are some limitations regarding our work as follows:

- The utilization of actual IoT smart sensors, as opposed to virtual counterparts, introduces a challenge that can potentially impact the research findings due to the actual sensors' inherent limitations. These limitations could include factors such as sensor accuracy, variability in real-world conditions, maintenance requirements, potential data gaps, and the need for calibration, among others. This limitation is duly acknowledged within the scope of this work. Furthermore, it is important to highlight the limitations of PEBEdge in the context of a hybrid environment, wherein both physical and virtual components are integrated. These limitations become particularly relevant when applying the benchmark in such hybrid scenarios.

Regarding this limitation, extending sensor testing to real-world deployments for validation of the experiment results is helpful. Future work can test PEBEdge in diverse

hybrid scenarios for benchmark reliability. Also, it is possible to consider adding metrics for hybrid system assessment. For data handling issues in real-world scenarios, it is necessary to develop algorithms for handling data gaps and imputation. All in all, we need to investigate methods to model and control variability in real-world conditions.

- We included resource utilization in the building power estimation model that may bias the results and be considered as a research limitation.

Regarding this limitation, performing model refinement is helpful. To this end, it is necessary to evaluate the impact of resource utilization on bias and model performance. We need to identify which specific resources contribute to bias. Using the presented feature selection techniques in this work, we identify the most relevant and non-biased resources for power estimation. Then, refining the power estimation model by removing or adjusting the biased features while maintaining accuracy is performed.

- Systematic error is a type of error that occurs consistently and predictably in the same direction in repeated measurements or experiments. It is a deviation from the true value caused by flaws or limitations in the experimental design, measurement instruments, or data collection process. Systematic errors affect the accuracy of the results and can lead to incorrect conclusions. In the same way, we might have systematic errors in our repetitive experimentation.

Regarding this limitation, it is necessary to calibrate measurement instruments regularly to ensure their accuracy and minimize systematic errors. Also, we can employ different measurement techniques or instruments to cross-validate results and identify potential systematic errors. Then Using mathematical models, we estimate how systematic errors accumulate and affect the overall outcomes.

- Testing the reliability and robustness of the benchmark framework can be considered another limitation of this work that can be mitigated in future work.

Regarding this limitation, there are several ways that it can be performed, as follows:

Identify Variables: It is possible to determine the factors that might influence the power consumption measurements. These variables could include changes in the RPi's workload, ambient temperature, power supply fluctuations, or different RPi models.

Replication: We can repeat the experiment multiple times under the same conditions. Aim for a sufficient number of repetitions (e.g., 10 or more) to achieve a statistically significant sample.

Randomize: It is possible to randomize the order of measurements during each repetition to minimize any potential bias introduced by the measurement sequence.

Statistical analysis: We can perform statistical analysis on the collected data to assess the variability and consistency of the measurements. Common statistical methods include calculating means, standard deviations, and performing reliability analysis (e.g., Cronbach's alpha).

CHAPTER 8 CONCLUSION

In this study, a benchmark tool called PEBEdge is proposed to help the IoT developer test and evaluate IoT devices' resource utilization, and power consumption. PEBEdge follows a multi-tier architecture consisting of sensors, edge, and cloud tiers and a microservice architecture combining both physical and virtual components. Specifically, we measured resource utilization, including CPU usage, memory, disk I/O, network bandwidth, and power consumption. A real-world scenario was created demonstrating how the PEBEdge can be used by the developers to evaluate the power consumption of the edge devices. We demonstrated the ease and speed with which experiments can be set up and modified, such as transitioning between Spark and a Python script or adjusting the number of workers. We also highlighted the configurations that should be modified during the transition process. Additionally, we showcased the high level of extensibility of PEBEdge. Furthermore, from the achieved results, it is concluded that, in comparison with Python scripts, Spark streaming is faster, more efficient, and consumes less power. However, Spark consumes a large amount of memory and is fairly heavy for running on an RPi. Also, As the number of parallel workers increases, the power consumption of the edge device also increases. This discovery implies that, in situations where power is limited, reducing the number of workers can help reduce power consumption on edge devices.

Then, this study used the PEBEdge in conjunction with ML models in order to estimate power consumption. Datasets from 35 different types of experiments were used to collect the samples, resulting in 381,232 samples. Based on regression metrics, we evaluated and compared the performance of the proposed ML models. In terms of performance metrics, it was found that the ML models outperformed the NN models. The results of DT and RF show significant similarities and offer excellent performance among the ML algorithms. However, the RF algorithm outperforms overall. We present solutions for improving the lack of reliability using trained models. Identifying the most important parameters affecting power consumption and potential software configuration adjustments for managing power consumption are parts of the suggested solutions. Moreover, an example of obtaining the time at which the edge device's battery will run out was presented. In the end, We present lightweight benchmarking for power consumption that shows the ability of our trained models to provide researchers with low-time and low-cost experiments in edge platform benchmarking.

In terms of future research directions, the present study focused on a single-edge device, which may not provide a comprehensive or representative analysis. Therefore, we propose

extending the investigation to include multiple edge devices that handle larger volumes of simultaneous data. Additionally, exploring different models and configurations in future studies would enhance the breadth and depth of the research findings.

Furthermore, the platform utilized in this study holds the potential for improving performance and optimizing power consumption within resource-constrained IoT environments. A promising avenue for future work involves the design, implementation, and evaluation of a self-adaptive algorithm that achieves a balance between performance and energy efficiency based on dynamic input. This self-adaptive mechanism can autonomously adjust its behavior at runtime, responding intelligently to system variations and adopting the most suitable configuration. Given the output generated by our model, the self-adaptive algorithm strives to adapt the model parameters to accomplish predefined objectives, taking into account the changes in input variables. By dynamically responding to fluctuations in the system, this approach facilitates effective goal achievement and offers a promising direction for future research endeavors.

REFERENCES

- [1] A. Sestino, M. I. Prete, L. Piper, and G. Guido, “Internet of things and big data as enablers for business digitalization strategies,” *Technovation*, p. 102173, 2020.
- [2] N. G. How many iot devices are there in 2022? [Online]. Available: <https://techjury.net/blog/how-many-iot-devices-are-there/>
- [3] Data volume of internet of things (iot) connections worldwide in 2019 and 2025. [Online]. Available: <https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/>
- [4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [5] F. Chen, Z. Zhao, G. Min, W. Gao, J. Chen, H. Duan, and P. Yang, “Speed control of mobile chargers serving wireless rechargeable networks,” *Future Generation Computer Systems*, vol. 80, pp. 242–249, 2018.
- [6] M. Razmara, G. R. Bharati, M. Shahbakhti, S. Paudyal, and R. D. Robinett, “Bilevel optimization framework for smart building-to-grid systems,” *IEEE transactions on Smart Grid*, vol. 9, no. 2, pp. 582–593, 2016.
- [7] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016.
- [8] M. Qin, L. Chen, N. Zhao, Y. Chen, F. R. Yu, and G. Wei, “Power-constrained edge computing with maximum processing capacity for iot networks,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4330–4343, 2018.
- [9] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.
- [10] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, “Resource scheduling in edge computing: A survey,” *IEEE Communications Surveys & Tutorials*, 2021.

- [11] E. Fotopoulou, A. Zafeiropoulos, F. Terroso-Sáenz, U. Şimşek, A. González-Vidal, G. Tsiolis, P. Gouvas, P. Liapis, A. Fensel, and A. Skarmeta, “Providing personalized energy management and awareness services for energy efficiency in smart buildings,” *Sensors*, vol. 17, no. 9, p. 2054, 2017.
- [12] S. Zhou and L. Zhang, “Smart home electricity demand forecasting system based on edge computing,” in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2018, pp. 164–167.
- [13] A. M. Ghosh and K. Grolinger, “Edge-cloud computing for internet of things data analytics: Embedding intelligence in the edge with deep learning,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 2191–2200, 2020.
- [14] Y. Liu, Y. Peng, B. Wang, S. Yao, and Z. Liu, “Review on cyber-physical systems,” *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 1, pp. 27–40, 2017.
- [15] A. Burg, A. Chattopadhyay, and K.-Y. Lam, “Wireless communication and security issues for cyber-physical systems and the internet-of-things,” *Proceedings of the IEEE*, vol. 106, no. 1, pp. 38–60, 2017.
- [16] J. Shi, J. Wan, H. Yan, and H. Suo, “A survey of cyber-physical systems,” in *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2011, pp. 1–6.
- [17] P. K. Donta, S. N. Srirama, T. Amgoth, and C. S. R. Annavarapu, “Survey on recent advances in iot application layer protocols and machine learning scope for research directions,” *Digital Communications and Networks*, 2021.
- [18] P. Pace, G. Aloï, R. Gravina, G. Caliciuri, G. Fortino, and A. Liotta, “An edge-based architecture to support efficient applications for healthcare industry 4.0,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 481–489, 2018.
- [19] T.-Y. Yang, C.-S. Yang, and T.-W. Sung, “An intelligent energy management scheme with monitoring and scheduling approach for iot applications in smart home,” in *2015 Third International Conference on Robot, Vision and Signal Processing (RVSP)*. IEEE, 2015, pp. 216–219.
- [20] C.-S. Shih, J.-J. Chou, N. Reijers, and T.-W. Kuo, “Designing cps/iot applications for smart buildings and cities,” *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 3–12, 2016.

- [21] D. Minoli, K. Sohraby, and B. Occhiogrosso, "Iot considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, 2017.
- [22] K. Akkaya, I. Guvenc, R. Aygun, N. Pala, and A. Kadri, "Iot-based occupancy monitoring techniques for energy-efficient smart buildings," in *2015 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2015, pp. 58–63.
- [23] D. Sembroiz, D. Careglio, S. Ricciardi, and U. Fiore, "Planning and operational energy optimization solutions for smart buildings," *Information Sciences*, vol. 476, pp. 439–452, 2019.
- [24] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart cities: A survey on data management, security, and enabling technologies," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2456–2501, 2017.
- [25] H. Arasteh, V. Hosseinneshad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "Iot-based smart cities: A survey," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. IEEE, 2016, pp. 1–6.
- [26] T. A. Nguyen and M. Aiello, "Energy intelligent buildings based on user activity: A survey," *Energy and buildings*, vol. 56, pp. 244–257, 2013.
- [27] A. De Paola, M. Ortolani, G. Lo Re, G. Anastasi, and S. K. Das, "Intelligent management systems for energy efficiency in buildings: A survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, pp. 1–38, 2014.
- [28] E. J. Knibbe, "Building management system," Oct. 15 1996, uS Patent 5,565,855.
- [29] B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma, "Cloud computing for internet of things & sensing based applications," in *2012 Sixth International Conference on Sensing Technology (ICST)*. IEEE, 2012, pp. 374–380.
- [30] Y. Zuo, F. Tao, and A. Y. Nee, "An internet of things and cloud-based approach for energy consumption evaluation and analysis for a product," *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 4-5, pp. 337–348, 2018.

- [31] C. Wei and Y. Li, "Design of energy consumption monitoring and energy-saving management system of intelligent building based on the internet of things," in *2011 International Conference on Electronics, Communications and Control (ICECC)*, 2011, pp. 3650–3652.
- [32] T. Efendigil, S. Önüt, and C. Kahraman, "A decision support system for demand forecasting with artificial neural networks and neuro-fuzzy models: A comparative analysis," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6697–6707, 2009.
- [33] J. Runge and R. Zmeureanu, "Forecasting energy use in buildings using artificial neural networks: A review," *Energies*, vol. 12, no. 17, p. 3254, 2019.
- [34] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption," *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 902–924, 2017.
- [35] J. Yu, M. Kim, H.-C. Bang, S.-H. Bae, and S.-J. Kim, "Iot as a applications: cloud-based building management systems for the internet of things," *Multimedia Tools and Applications*, vol. 75, no. 22, pp. 14 583–14 596, 2016.
- [36] L. M. Oliveira and J. J. Rodrigues, "Wireless sensor networks: A survey on environmental monitoring." *J. Commun.*, vol. 6, no. 2, pp. 143–151, 2011.
- [37] M. T. Lazarescu, "Design of a wsn platform for long-term environmental monitoring for iot applications," *IEEE Journal on emerging and selected topics in circuits and systems*, vol. 3, no. 1, pp. 45–54, 2013.
- [38] M. Swan, "Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0," *Journal of Sensor and Actuator networks*, vol. 1, no. 3, pp. 217–253, 2012.
- [39] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the implementation of iot for environmental condition monitoring in homes," *IEEE sensors journal*, vol. 13, no. 10, pp. 3846–3853, 2013.
- [40] S. Fang, L. Da Xu, Y. Zhu, J. Ahati, H. Pei, J. Yan, and Z. Liu, "An integrated system for regional environmental monitoring and management based on internet of things," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1596–1605, 2014.
- [41] B. Kantarci and H. T. Mouftah, "Trustworthy sensing for public safety in cloud-centric internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 360–368, 2014.

- [42] J. Zhao, X. Zheng, R. Dong, and G. Shao, "The planning, construction, and management toward sustainable cities in china needs the environmental internet of things," *International Journal of Sustainable Development & World Ecology*, vol. 20, no. 3, pp. 195–198, 2013.
- [43] N. Bui and M. Zorzi, "Health care applications: a solution based on the internet of things," in *Proceedings of the 4th international symposium on applied sciences in biomedical and communication technologies*, 2011, pp. 1–5.
- [44] R. S. Istepanian, S. Hu, N. Y. Philip, and A. Sungoor, "The potential of internet of m-health things "m-iot" for non-invasive glucose level sensing," in *2011 annual international conference of the IEEE engineering in medicine and biology society*. IEEE, 2011, pp. 5264–5266.
- [45] C. Doukas and I. Maglogiannis, "Bringing iot and cloud computing towards pervasive healthcare," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2012, pp. 922–926.
- [46] W.-T. Sung and Y.-C. Chiang, "Improved particle swarm optimization algorithm for android medical care iot using modified parameters," *Journal of medical systems*, vol. 36, no. 6, pp. 3755–3763, 2012.
- [47] S. Amendola, R. Lodato, S. Manzari, C. Occhiuzzi, and G. Marrocco, "Rfid technology for iot-based personal healthcare in smart spaces," *IEEE Internet of things journal*, vol. 1, no. 2, pp. 144–152, 2014.
- [48] Y. J. Fan, Y. H. Yin, L. Da Xu, Y. Zeng, and F. Wu, "Iot-based smart rehabilitation system," *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1568–1577, 2014.
- [49] B. Xu, L. Da Xu, H. Cai, C. Xie, J. Hu, and F. Bu, "Ubiquitous data accessing method in iot-based information system for emergency medical services," *IEEE Transactions on Industrial informatics*, vol. 10, no. 2, pp. 1578–1586, 2014.
- [50] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. Da Xu, S. Kao-Walter, Q. Chen, and L.-R. Zheng, "A health-iot platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box," *IEEE transactions on industrial informatics*, vol. 10, no. 4, pp. 2180–2191, 2014.
- [51] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *2012*

- sixth international conference on innovative mobile and internet services in ubiquitous computing*. IEEE, 2012, pp. 751–756.
- [52] L. Dürkop, H. Trsek, J. Jasperneite, and L. Wisniewski, “Towards autoconfiguration of industrial automation systems: A case study using profinet io,” in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE, 2012, pp. 1–8.
 - [53] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, “On optimal scheduling in duty-cycled industrial iot applications using ieee802. 15.4 e tsc,” *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, 2013.
 - [54] Z. Bi, L. Da Xu, and C. Wang, “Internet of things for enterprise systems of modern manufacturing,” *IEEE Transactions on industrial informatics*, vol. 10, no. 2, pp. 1537–1546, 2014.
 - [55] Q. Chi, H. Yan, C. Zhang, Z. Pang, and L. Da Xu, “A reconfigurable smart sensor interface for industrial wsn in iot environment,” *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1417–1425, 2014.
 - [56] M. A Vouk, “Cloud computing—issues, research and implementations,” *Journal of computing and information technology*, vol. 16, no. 4, pp. 235–246, 2008.
 - [57] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.
 - [58] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
 - [59] M. Chae, H. Lee, and K. Lee, “A performance comparison of linux containers and virtual machines using docker and kvm,” *Cluster Computing*, vol. 22, no. 1, pp. 1765–1775, 2019.
 - [60] R. Morabito, J. Kjällman, and M. Komu, “Hypervisors vs. lightweight virtualization: A performance comparison,” in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.
 - [61] Z. Kozhimbayev and R. O. Sinnott, “A performance comparison of container-based technologies for the cloud,” *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

- [62] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 171–172.
- [63] E. Casalicchio, “A study on performance measures for auto-scaling cpu-intensive containerized applications,” *Cluster Computing*, vol. 22, no. 3, pp. 995–1006, 2019.
- [64] —, “Autonomic orchestration of containers: Problem definition and research challenges.” in *VALUETOOLS*, 2016.
- [65] “Openvz linux containers.” [Online]. Available: <https://openvz.org/>
- [66] “Coreos is building a container runtime rocket.” [Online]. Available: <http://coreos.com/blog/rocket/>
- [67] “Linux containers.” [Online]. Available: <http://linuxcontainers.org>
- [68] “Docker container.” [Online]. Available: <https://www.docker.com/resources/what-container/>
- [69] K. J. Åström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.
- [70] M. Fokaefs, C. Barna, and M. Litoiu, “Economics-driven resource scalability on the cloud,” in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2016, pp. 129–139.
- [71] A. M. Abdulazeez, S. R. Zeebaree, and M. A. Sadeeq, “Design and implementation of electronic student affairs system,” *Academic Journal of Nawroz University*, vol. 7, no. 3, pp. 66–73, 2018.
- [72] B. Dong, V. Prakash, F. Feng, and Z. O’Neill, “A review of smart building sensing system for better indoor environment control,” *Energy and Buildings*, vol. 199, pp. 29–46, 2019.
- [73] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, “Iot-based big data storage systems in cloud computing: perspectives and challenges,” *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2016.
- [74] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Clonecloud: elastic execution between mobile device and cloud,” in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

- [75] B. Cheng, A. Papageorgiou, F. Cirillo, and E. Kovacs, “Geelytics: Geo-distributed edge analytics for large scale iot systems based on dynamic topology,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 565–570.
- [76] A. Javed, J. Robert, K. Heljanko, and K. Främling, “Iotef: A federated edge-cloud architecture for fault-tolerant iot applications,” *Journal of Grid Computing*, vol. 18, no. 1, pp. 57–80, 2020.
- [77] B. Ramprasad, J. McArthur, M. Fokaefs, C. Barna, M. Damm, and M. Litoiu, “Leveraging existing sensor networks as iot devices for smart buildings,” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, 2018, pp. 452–457.
- [78] P. Silva, A. Costan, and G. Antoniu, “Towards a methodology for benchmarking edge processing frameworks,” in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 904–907.
- [79] C. P. Kruger and G. P. Hancke, “Benchmarking internet of things devices,” in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2014, pp. 611–616.
- [80] A. Shukla, S. Chaturvedi, and Y. Simmhan, “Riotbench: A real-time iot benchmark for distributed stream processing platforms,” *arXiv preprint arXiv:1701.08530*, 2017.
- [81] A. Das, S. Patterson, and M. Wittie, “Edgebench: Benchmarking edge computing platforms,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 175–180.
- [82] A. Limaye and T. Adegbiya, “Hermit: A benchmark suite for the internet of medical things,” *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4212–4222, 2018.
- [83] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, “Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research,” *Internet of Things*, vol. 12, p. 100273, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660520301062>
- [84] F. Carpio, M. Delgado, and A. Jukan, “Engineering and experimentally benchmarking a container-based edge computing system,” in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.
- [85] J. M. Paniego, L. Libutti, M. P. Puig, F. Chichizola, L. De Giusti, M. Naiouf, and A. De Giusti, “Unified power modeling design for various raspberry pi generations

- analyzing different statistical methods,” in *Argentine Congress of Computer Science*. Springer, 2019, pp. 53–65.
- [86] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.
 - [87] M. Asaad, F. Ahmad, M. S. Alam, and Y. Rafat, “Iot enabled monitoring of an optimized electric vehicle’s battery system,” *Mobile Networks and Applications*, vol. 23, no. 4, pp. 994–1005, 2018.
 - [88] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, “Evaluating the effectiveness of model-based power characterization,” in *USENIX Annual Technical Conf*, vol. 20, 2011.
 - [89] F. Kaup, S. Hacker, E. Mentzendorff, C. Meurisch, and D. Hausheer, “Energy models for nfv and service provisioning on fog nodes,” in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–7.
 - [90] G. Orsini, W. Posdorfer, and W. Lamersdorf, “Saving bandwidth and energy of mobile and iot devices with link predictions,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 8, pp. 8229–8240, 2021.
 - [91] F. Rieger and C. Bockisch, “Survey of approaches for assessing software energy consumption,” in *Proceedings of the 2nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems*, 2017, pp. 19–24.
 - [92] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, “The power of arm64 in public clouds,” in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 459–468.
 - [93] L. Ardito and M. Torchiano, “Creating and evaluating a software power model for linux single board computers,” in *Proceedings of the 6th International Workshop on Green and Sustainable Software*, 2018, pp. 1–8.
 - [94] S. Sagkriotis, C. Anagnostopoulos, and D. P. Pazaros, “Energy usage profiling for virtualized single board computer clusters,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019, pp. 1–6.
 - [95] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, “Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research,” *Internet of Things*, p. 100273, 2020.

- [96] M. Rashti, G. Sabin, D. Vansickle, and B. Norris, “Wattprof: A flexible platform for fine-grained hpc power profiling,” in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 698–705.
- [97] Z. Zhang and S. Fu, “Characterizing power and energy usage in cloud computing systems,” in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE, 2011, pp. 146–153.
- [98] J. Stoess, C. Lang, and F. Bellosa, “Energy management for hypervisor-based virtual machines,” in *USENIX annual technical conference*, 2007, pp. 1–14.
- [99] R. Neugebauer and D. McAuley, “Energy is just another resource: Energy accounting and energy pricing in the nemesis os,” in *Proceedings Eighth Workshop on Hot Topics in Operating Systems*. IEEE, 2001, pp. 67–72.
- [100] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, “Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures,” in *2010 IEEE 30th International Conference on Distributed Computing Systems*. IEEE, 2010, pp. 62–73.
- [101] W. Mach and E. Schikuta, “A consumer-provider cloud cost model considering variable cost,” in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2011, pp. 628–635.
- [102] Y. C. Lee and A. Y. Zomaya, “Energy efficient utilization of resources in cloud computing systems,” *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.
- [103] H. Wang and H. Tianfield, “Energy-aware dynamic virtual machine consolidation for cloud datacenters,” *IEEE Access*, vol. 6, pp. 15 259–15 273, 2018.
- [104] S. A. Nabavi, A. Aslani, M. A. Zaidan, M. Zandi, S. Mohammadi, and N. Hossein Motlagh, “Machine learning modeling for energy consumption of residential and commercial sectors,” *Energies*, vol. 13, no. 19, p. 5171, 2020.
- [105] I. Priyadarshini, S. Sahu, R. Kumar, and D. Taniar, “A machine-learning ensemble model for predicting energy consumption in smart homes,” *Internet of Things*, vol. 20, p. 100636, 2022.
- [106] R. E. Alden, H. Gong, C. Ababei, and D. M. Ionel, “Lstm forecasts for smart home electricity usage,” in *2020 9th International conference on renewable energy research and application (ICRERA)*. IEEE, 2020, pp. 434–438.

- [107] S. Bourhmane, M. R. Abid, R. Lghoul, K. Zine-Dine, N. Elkamoun, and D. Benhadou, “Machine learning for energy consumption prediction and scheduling in smart buildings,” *SN Applied Sciences*, vol. 2, pp. 1–10, 2020.
- [108] C. F. Rodrigues, G. Riley, and M. Luján, “Synergy: An energy measurement and prediction framework for convolutional neural networks on jetson tx1,” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2018, pp. 375–382.
- [109] M. S. Aslanpour, A. N. Toosi, R. Gaire, and M. A. Cheema, “Wattedge: A holistic approach for empirical energy measurements in edge computing,” in *International Conference on Service-Oriented Computing*. Springer, 2021, pp. 531–547.
- [110] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems benchmarking: for scientists and engineers*. Springer, 2020, vol. 1.
- [111] —, *Systems Benchmarking*, 1st ed. Springer International Publishing, 2020.
- [112] N. Ahmed, A. L. Barczak, M. A. Rashid, and T. Susnjak, “Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models,” *Journal of Big Data*, vol. 9, no. 1, p. 67, 2022.
- [113] D. Didona, F. Quaglia, P. Romano, and E. Torre, “Enhancing performance prediction robustness by combining analytical modeling and machine learning,” in *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, 2015, pp. 145–156.
- [114] T. Chai and R. R. Draxler, “Root mean square error (rmse) or mean absolute error (mae),” *Geoscientific model development discussions*, vol. 7, no. 1, pp. 1525–1534, 2014.
- [115] A. Ash and M. Shwartz, “R2: a useful measure of model performance when predicting a dichotomous outcome,” *Statistics in medicine*, vol. 18, no. 4, pp. 375–384, 1999.
- [116] M. A. Hardy, *Regression with dummy variables*. Sage, 1993, vol. 93.
- [117] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou *et al.*, “Xgboost: extreme gradient boosting,” *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.
- [118] M. Pal, “Random forest classifier for remote sensing classification,” *International journal of remote sensing*, vol. 26, no. 1, pp. 217–222, 2005.

- [119] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, “An introduction to decision tree modeling,” *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.
- [120] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.
- [121] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [122] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [123] A. Graves and A. Graves, “Long short-term memory,” *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.
- [124] H. Taud and J. Mas, “Multilayer perceptron (mlp),” *Geomatic approaches for modeling land change scenarios*, pp. 451–455, 2018.
- [125] C. Sheridan, *The Python language reference manual*. Lulu Press, Inc, 2016.
- [126] O. Kramer and O. Kramer, “Scikit-learn,” *Machine learning for evolution strategies*, pp. 45–53, 2016.
- [127] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.
- [128] M. Abadi, “Tensorflow: learning functions at scale,” in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, pp. 1–1.
- [129] G. Kecskemeti, W. Hajji, and F. P. Tso, “Modelling low power compute clusters for cloud simulation,” in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2017, pp. 39–45.
- [130] A. Computing *et al.*, “An architectural blueprint for autonomic computing,” *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.
- [131] M. Fokaefs, Y. Rouf, C. Barna, and M. Litoiu, “Evaluating adaptation methods for cloud applications: An empirical study,” in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 632–639.

APPENDIX A A DETAILED DESCRIPTION OF THE STEPS INVOLVED IN CONDUCTING THE EXPERIMENT

As discussed before, to run the experiments 2 VMs (VM1 and VM2), 1 Edge device (RPi), a power meter, and a computer to record the energy of the edge device are required.

Prerequisite

Steps 1, 2, and 3 need to be executed on VM1, VM2, and the edge device.

Step 1 : Install docker

Step 2 : Install docker-compose

Step 3 : Clone the repository

Step 4 : Access the right folders.

1. Sensor Tier: cd Sensor Side/
2. Edge Device: cd edge Side/
3. Local edge Node: cd DB Side/

Step 5 : Install the power meter software on the computer.

Following the manual and installing the software from the provided link¹:

Performing the task

1. Inside the DB side folder/local edge node

A. OPTION 1: Spark (processing) + InfluxDB(SAVING DATA)

- a. cd influxDB/
- b. docker-compose up
- c. Opening the browser and open localhost:8086.
- d. Filling the form and save the Organization name and the bucket name; choose to explore from the left panel.

The ready dataset would be as Figure A.1.

B. OPTION 2 : Python SCRIPT (FILTERING) + MONGODB (SAVING DATA)

- a. cd mongodb/

¹https://drive.google.com/drive/u/0/folders/1mMV2qPtrhsiYuaeDA7boKEAcR4J_iGno

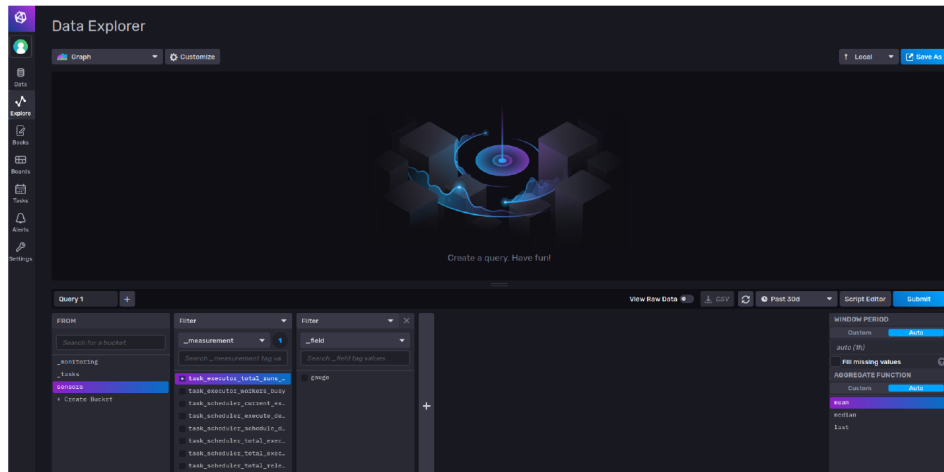


Figure A.1 Results for the ready dataset.

b. Changing the following values the docker-compose file

MONGO_INITDB_ROOT_USERNAME: root

MONGO_INITDB_ROOT_USERNAME: example

c. docker-compose up

d. Using the following link to install MongoDB compass to manage the DB.

<https://www.mongodb.com/docs/compass/current/install/>

e. Creating a new connection

e. Creating a new database and saving the database name and collection name.

Database is finalized after this step.

C. OPTION 3: User own application

a. cd youreexample/

b. The following service for the benchmarking data as Listing A.1 is required in docker-compose.

```

1  influxdb_MSR:
2    image: influxdb:1.8.3
3    restart: always
4    ports:
5      - "8087:8086"
6    volumes:
7      - ./influxdb_MSR:/var/lib/influxdb

```

Listing A.1 Required service for the benchmarking data.

- c. Adding the database to the docker-compose file as a service.
- d. docker-compose up

2. Inside the edge Side folder/edge Device

A. OPTION 1: Spark (processing) + InfluxDB(SAVING DATA)

- a. cd edge/ Side/Processing/Spark
- b. cd Spark-master/
- c. docker build -t yourrepo/Spark-master. Her, repo name is mahsahadian so that is the name that is used in docker-compose.
- d. cd ../Spark-worker/
- e. docker build -t yourrepo/Spark-worker
- f. cd ../Spark-job/maven-Spark-processing/
- g. cd ..
- h. docker build -t yourrepo/Spark-job
- i. For the configuration, Spark-edge-variables.env should be edited.

Variables for the INFLUXDB configuration:

```
INFLUXDB_PROTOCOL
INFLUXDB_IP
INFLUXDB_PORT
INFLUXDB_TOKEN
INFLUXDB_ORG
INFLUXDB_BUCKET
INFLUXDB_WRITE_TIMEOUT
```

Variables for the MQTT configuration:

```
MQTT_SERVER_IP
MQTT_SERVER_PORT
MQTT_TOPIC
```

Variables for the SPARK configuration:

```
MASTER_IP
MASTER_PORT
```

```
SPARK__EXECUTOR__MEMORY
SPARK__DRIVER__MEMORY
SPARK__MEMORY__FRACTION
SPARK__MEMORY__STORAGEFRACTION
```

It is required to change the InfluxDB_IP with the edge device's IP and the port as configured in DB side docker-compose. All the other variables need to be changed according to the Database Configuration (organization name, Bucket name and etc.). The MQTT SERVER IP is the same as the edge device's IP. the MQTT SERVER PORT is configured in the mosquitto.conf file. "sensors" is used as a topic. The master IP is the same as the edge device's IP. Listener is the same MQTT SERVER PORT in Spark-edge-variables.env.

j. cd ..

k. docker-compose up. For running Spark with more than 1 worker, it is required to use docker-compose up --scale Sparkworker=number

B. OPTION 2: Python SCRIPT (FILTERING) + MONGODB (SAVING DATA)

a. cd edge/ Side/Processing/script

b. For the configuration, simple-script-edge-variables.env should be edited.

Variables for the MONGODB configuration:

```
MONGODB__DATABASE__NAME
MONGODB__COLLECTION__NAME
MONGODB__DATABASE__IP
MONGODB__DATABASE__PORT
MONGODB__DATABASE__USERNAME
MONGODB__DATABASE__PASSWORD
```

MONGODB configuration values can be achieved from Database Side. The username and the password are the same from the docker-compose on the DB side. The listener is the same MQTT SERVER PORT in simple-script-edge-variables.env.

c. docker build -t yourrepo/Pythonprocessor.

yourrepo/Pythonprocessor should be the same name in the docker-compose file.

c. docker-compose up

C. OPTION 3: User own application

a. `cd edge/ Side/Processing/YourExample`

b. The application needs to be running in docker containers so it is only needed to run user's own application using docker-compose or only with docker run. mosquitto can be used as a message broker which is already in the docker-compose file. Also, it is possible to change it with another message broker.

3. Inside the Sensor Side folder

There are 2 main components:

- Data generator: generating data as images and sending them as post requests to the Client Adapter.
- Client Adapter: accepting requests from the data generator, and sending the data to the edge side according to the message broker.

A. Client Adapter:

a. Using MQTTAdapter:

1. For the configuration, `/ClientAdapter/MQTTAdapter/sensor-variables.env` should be edited.

MQTT SERVER IP is the same as the edge device's IP. MQTT SERVER PORT is configured in the `mosquitto.conf` file.

2. `cd MQTTAdapter/`
 3. `docker-compose build`
 4. `docker-compose up`
 5. Starting `mqttadapter_app_1 ... done`
- Attaching to `mqttadapter_app_1`

b. Using user own Adapter:

1. `cd YourAdapter/`
2. `open YourAdapter.py`

In the run function, before initiating the handler, it is possible to put any code to be run only once before receiving requests.

B. Data generator:

1. It is needed to choose the measurement name in the `videogenerator.py`.

`Jsondata["measurment_name"] = "t_1w_111_1_10s"`

2. After each time changing the measurement name, it is required to run docker-compose build.

3. In this step, it is possible to run any number of emulated sensors. for example for 50 sensors:

COMPOSE_HTTP_TIMEOUT=2000 docker-compose up --scale app=50

However, before launching the sensors it is required to go back to the edge device and launch the MONITORING TOOL as Figure A.2 and Figure A.3.

```

    'net_tx': float(self.get_network_throughput(container_stats)['tx'])

    json_body = [
        {
            "measurement": "a_85_1wor",
            "tags": {
                "container_name": name,
                "short_id": cont.short_id
            },
            "fields": {
                'short_id': cont.short_id,
                'cpu_percent': float(self.get_cpu_percent(container_stats)),
                #'cpu_power': float(self.client.powerapi.formula.find_one({"target": cont.name}),
                'memory': float(self.get_memory(container_stats)['memory']),
                'memory_limit': float(self.get_memory(container_stats)['memory_limit']),
                'memory_percent': float(self.get_memory(container_stats)['memory_percent']),
                'memory_utilization': float(self.get_memory(container_stats)['memory_utilization']),
                'disk_i': float(self.get_disk_io(container_stats)['disk_i']),
                'disk_o': float(self.get_disk_io(container_stats)['disk_o']),
                'net_rx': float(self.get_network_throughput(container_stats)['rx']),
                'net_tx': float(self.get_network_throughput(container_stats)['tx'])
            }
        }
    ]

```

Figure A.2 Resource utilization name.

```

pi@raspberrypi:~/Documents/NewRepo/EdgeBenchmark/Edge Side/monitoring $ python monitoring.py
Monitoring V2
Using: Docker remote API
{'date': '1657320615.3', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec

Monitoring V2
Using: Docker remote API
{'date': '1657320616.38', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec

Monitoring V2
Using: Docker remote API
{'date': '1657320617.39', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec

```

Figure A.3 Running monitoring script.

Also, for the power consumption, first, the power meter is plugged to the RPi and connected to the computer. Then, the software on the computer near the edge device is opened to view and launch capturing power consumption data.