Towards Energy-Aware TinyML on Battery-Less IoT Devices

Adnan Sabovic[a], Michiel Aernouts[a], Dragan Subotic[a], Jaron Fontaine[b], Eli De Poorter[b], Jeroen Famaey[a]

[a] *University of Antwerp - imec, IDLab*
Sint-Pietersvliet 7, 2000 Antwerp, Belgium
[b] *IDLab, Ghent University - imec*
Technologiepark-Zwijnaarde 126, 9052 Ghent

Abstract

With the advent of Tiny Machine Learning (tinyML), it is increasingly feasible to deploy optimized ML models on constrained battery-less Internet of Things (IoT) devices with minimal energy availability. Due to the unpredictable and dynamic harvesting environment, successfully running tinyML on battery-less devices is still challenging. In this paper, we present the energy-aware deployment and management of tinyML algorithms and application tasks on battery-less IoT devices. We study the trade-offs between different inference strategies, analyzing under which circumstances it is better to make the decision locally or send the data to the Cloud where the heavy-weight ML model is deployed, respecting energy, accuracy, and time constraints. To decide which of these two options is more optimal and can satisfy all constraints, we define an energy-aware tinyML optimization algorithm. Our approach is evaluated based on real experiments with a prototype for battery-less person detection, which considers two different environments: (i) a controllable setup with artificial light, and (ii) a dynamic harvesting environment based on natural light. Our results show that the local inference strategy performs best in terms of execution speed when a controllable harvesting environment is considered. It can execute 3 times as frequently as remote inference at a harvesting current of 2mA and using a capacitor of 1.5F. In a realistic harvesting scenario with natural light and making use of the energy-aware optimization algorithm, the device will favor remote inference under high light conditions due to the better accuracy of the Cloud-based model. Otherwise, it switches to local inference.

Email addresses: `adnan.sabovic@uantwerpen.be` (Adnan Sabovic), `michiel.aernouts@uantwerpen.be` (Michiel Aernouts), `dragan.subotic@uantwerpen.be` (Dragan Subotic), `jaron.fontaine@ugent.be` (Jaron Fontaine), `eli.depoorter@ugent.be` (Eli De Poorter), `jeroen.famaey@uantwerpen.be` (Jeroen Famaey)

1. Introduction

Over the last decade, the Internet of Things (IoT) concept has grown steadily, representing one of the critical roles of the Internet of the future [1]. It allows billions of tiny devices to connect and communicate with each other while performing different tasks such as collecting, processing, and transmitting data with the aim to simplify and improve daily life [2][3]. The improvements in sensors, low-power communication technologies, and processor efficiency, as well as their low price and easy maintenance, allow such devices to innovate in various fields, from home automation [4] and industrial monitoring [5] to sports activity monitoring [6] and predictive healthcare [7].

Nowadays, most IoT devices rely on batteries, which can provide stable power, but at the same time are bulky, short-lived, and dangerous if not carefully protected. These batteries can contain toxic chemical components that are harmful to the environment, which makes their maintenance, disposal, and replacement expensive and definitely ecologically unacceptable. As the number of IoT devices is growing at an amazing rate, it is clear that the use of batteries must be reconsidered if we want to make a sustainable IoT vision come true [8].

Tiny battery-less IoT devices that entirely depend on harvested environmental energy are a promising solution to alleviate the IoT's battery problem. These devices collect energy from different environmental and renewable sources (e.g., solar, vibration, thermal) and store it in small capacitors that act as the main energy storage [2]. These capacitors are more resistant to capacity degradation compared to batteries, which prolongs their lifetime to potentially decades. Also, they are less sensitive to extreme temperatures and can support operations in a wide temperature range. On the other side, batteries operate poorly in cold temperatures as their efficiency drops and show similar behavior when the high-temperature scenarios are considered due to overheating. Finally, battery-less devices are easy to recycle and practically maintenance-free, which makes them environmentally friendly and suitable for applications in hard-to-reach locations and large-scale deployments [2] [3].

Machine learning (ML) is successfully employed in many fields and applications (e.g., object detection, image classification, and audio recognition)[9], where it is used for data analysis, making systems intelligent in terms of decision-making. These ML algorithms are mostly based on neural networks, achieving high accuracy, but at the same time requiring large computational power and memory resources [10]. As battery-less IoT devices are resource-constrained with very limited power supply, and they are usually equipped with limited computing and storage capabilities [11], deploying ML on battery-less IoT devices is highly challenging. Currently, most of these devices are only used to collect and send data to the Cloud, where the ML-based data processing and decision-making algorithms are executed.

In order to tackle these challenges, there is a new concept called Tiny Machine Learning (tinyML), with the aim of designing, developing, and running optimized ML models on ultra-low-power IoT devices with minimal energy consumption [12]. There are a lot of benefits and advantages that come with this technology [12] [9]. By integrating ML models within tiny battery-less IoT devices, each of these devices becomes able to process data and make decisions locally, without a need to transmit the collected data to the Cloud. Using this approach, network load and latency are reduced, and the security and privacy of data are increased, making the local inference approach more beneficial.

Despite all the advantages and improvements, there are still many challenges when it comes to deploying tinyML models on battery-less IoT devices. As these devices operate in a dynamic and unpredictable energy environment, power failures can occur at any moment, resulting in intermittent on-off behavior of the device. Based on that, it is crucial to find a way of enabling successful task cycles by handling such behavior and reducing the possibility of power failures. This problem can be solved with task-based scheduling models [13] [14], where the application is divided into connected tasks that perform atomic functions. However, traditional task-based scheduling algorithms do not explicitly consider the available and required energy. In this paper, we study the energy-aware deployment and management of tinyML algorithms on battery-less IoT devices. The task scheduling strategy is built based on our previous work on energy-aware task scheduling [2] aiming to extend energy-aware resource management and execution to ML tasks. The proposed task scheduler is able to intelligently schedule application tasks based on their priorities and dependencies, taking into account the harvested and available energy, energy consumed by the task as well as the required amount of energy that needs to be collected for its successful execution. By considering energy awareness, our approach can avoid power failures during tasks and maintain forward progress. We developed, trained, and deployed a Convolutional Neural Network (CNN) on the Cloud, after which we employed state-of-the-art network pruning techniques to convert Cloud-based NN models into smaller tinyML models that can be executed on constrained battery-less IoT devices.

The main contributions in this paper are: (i) a system architecture to enable energy-aware tinyML inferencing tasks on battery-less IoT devices, with support for task-offloading; (ii) a formal mathematical model to calculate the optimal decision in terms of tinyML task execution as a function of energy harvesting and storage; (iii) a hardware-software prototype of the proposed solution that enables person detection on battery-less IoT devices; (iv) realistic evaluation and validation of our approach based on the device prototype.

The remainder of this paper is structured as follows. Section 2 reviews the state of the art on battery-less computing and scheduling, and tinyML model deployment on battery-less IoT devices. In Section 3, the proposed system architecture is described together with the energy-aware optimization algorithm and mathematical model. Section 4 describes how the ML model for person detection is developed, trained, and integrated on battery-less IoT devices and the Cloud, including the description of the used prototype. Section 5 presents an accurate device profiling methodology to determine the current consumption and execution time of different application tasks and device states, which is used as input to the scheduling algorithm. Section 6 shows the evaluation and validation results, together with the discussion. Finally, our conclusions are provided in Section 7.

2. Related Work

With the emergence of battery-less IoT devices, it becomes possible to execute and run different computer programs on small embedded systems without the need for a dedicated battery as a power source [15]. The interest in using these devices in various fields, from wireless sensor networks (WSN) to different IoT applications, constantly grows. However, they are still mainly used for simple sensing applications, where the device needs to collect some data and transmit it to the Cloud or a more powerful machine for further processing. The integration of tinyML models and algorithms would make these devices capable of executing more intelligent tasks, which pushes the limit of battery-less computing and processing. There are still some challenges that this concept faces. First, it is required to enable these devices to successfully operate despite the unpredictable energy-harvesting environment. Second, it is important to find a way to convert a huge ML model into a lightweight version that is more suitable for running on a battery-less IoT device without losing too much accuracy.

2.1. Battery-less task scheduling

Traditional computing models and static sequential applications cannot handle the intermittent on/off behavior of battery-less IoT devices, as they assume the device has an uninterrupted execution of tasks and rely on volatile memory to maintain application progress. The task-based models, which split the program into atomic connected tasks and store the necessary data in non-volatile memory, show better performance in preserving forward progress, making them more suitable for battery-less devices.

In [16] and [13], two low-overhead programming models for intermittent computing on energy-harvesting devices have been presented. By saving the results of each task in non-volatile memory forward progress and data consistency can be ensured. They are based on static task flows that can result in the risk of task starvation as the scheduler will not advance any other task if the previous one cannot be completed. To overcome this issue, Yildrim et al. [14] proposed InK, a dynamic scheduler based on priorities and event triggers that is able to adapt and react to changes in available energy. InK always tries to execute the task with the highest priority, without considering energy availability, which can result in task failure if the available energy depletes during the execution. To address this, we presented an energy-aware task scheduler in our previous work [2]. It is able to schedule tasks in an energy-aware fashion, taking into account energy harvesting input, stored energy, and energy consumed by tasks, as well as their priorities.

There are also some other energy-aware task scheduling approaches that have been presented. In [17], AsTAR, an energy-aware task scheduler that rapidly identifies optimum scheduling rates, supports heterogeneity and addresses environmental dynamism ensuring extremely low-performance overhead in terms of memory, execution time, and energy was presented. Karimi et al. [18] presented an energy-aware scheduling framework to execute real-time periodic tasks with atomic sensing operations, considering the Radio Frequency (RF) energy harvesting environment. Their periodic energy model may not cover all types of energy-harvesting sources, which means that the framework cannot be applied in all cases. In both cases, they focused on using supercapacitors, which are not necessarily the most optimal solution for battery-less devices [19]. They can reduce performance as they need more time to charge, decreasing the total number of executed application cycles. Finally, Delgado et al. [3] presented a theoretical analysis of an energy-aware task scheduling algorithm, while we focused more on creating a scheduling framework that enables the deployment of energy-aware scheduling algorithms on real battery-less IoT devices.

The energy-aware task scheduling algorithm presented in [2] is generic and can be used with different technologies and applications, and integrated on different IoT platforms. Based on that, in this work, we consider some of its main features: (i) required voltage thresholds are calculated for each application task, (ii) task dependencies are developed in the form of the parent-child relationship, (iii) task constraints such as repeat task and data availability are included, (iv) task priorities are included where the task with the highest priority will always be selected first if all set constraints are satisfied. In contrast to all mentioned task schedulers, in this work, we have designed more intelligent solutions that enable constrained IoT devices to make decisions locally instead of depending on the more powerful edge and Cloud devices. In addition to traditional application tasks, we include tinyML algorithms and tasks deployed on battery-less IoT devices. This is the first work, where the scheduling of tinyML tasks in an energy-aware manner is shown. Taking into account the energy-aware optimization algorithm, the scheduler is able to make the most optimal solution between different inference strategies, respecting the accuracy, energy, and time constraints.

2.2. TinyML on battery-less IoT devices

TinyML is a technology that allows the implementation of different resource-constrained ML models and intelligent tasks locally on edge IoT devices, such as battery-less low-power IoT devices. This offers devices the ability to process and analyze data at the extreme edge and opens the possibility of employing such devices for novel IoT applications, such as object or face detection, that will replace traditional ones. This section describes prior art that focused on the deployment of tinyML models on small embedded devices [15] [20].

Benninger et al. [21] proposed EdgeEye, a stand-alone edge computing device, capable of data-centric processing and performing machine learning inference on images captured with an ultra-low power camera. They presented the integration of a tinyML model for people counting, where the final results of the inference are transmitted using LoRaWAN. Prasanna et al. [22] presented the implementation of gesture and speech recognition applications on the Arduino Nano 33 BLE sense device. Both models were trained and deployed from the Edge Impulse framework. Saffari and Tan et al. [23] presented a human occupancy detection system that uses battery-free cameras and a DL model implemented on a Raspberry Pi 4 Model B. Their camera harvests energy from ambient light and transmits data to the receiver, the board on which detection algorithm is implemented, using backscatter communication. All three mentioned papers are focused on implementing tinyML models on battery-powered devices, while we consider a more resource-constrained battery-less IoT device that uses a capacitor to store energy collected from its environment.

Giordano et al. [24] presented a battery-free smart camera for continuous image processing that combines a tinyML algorithm for face identification, a power management module with an energy harvester, buck converter, and a capacitor, an energy harvesting circuit that can host a thermal and solar energy harvester, and LoRa module for sending only the end result of local inference. The device waits until the capacitor is fully charged in order to perform an inference cycle. To avoid inference starting when the capacitor is not fully charged, they implemented a cold start mechanism, which defines a threshold above which charging becomes more efficient and the system should operate. Their proposed algorithm was trained to identify five different persons on the image, where they also studied the trade-off between consumed energy and accuracy obtained based on the size of the captured image. The same author proposed a similar system in [25]. They presented a use-case where a battery-less sensor node performed a neural network-based facial recognition at the edge on a CNN accelerator. Once the external trigger is activated, the device wakes up and takes a picture via the integrated camera, which is later used as an input to the neural network. When the local inference task is executed, the final result is sent to a gateway using LoRa. A small battery-less computer vision platform has been presented by Jokic et al. [10]. They used an ultra-low power image sensor and an ML system-on-chip to recognize faces on images, achieving self-sustainable operations by using solar energy harvesting with a small on-board solar cell. If the face is identified, the display on the device is updated, and the code is shown for the next 60 seconds. Their setup was only tested indoors.

All three mentioned solutions only considered the on-device intelligence, where inference is performed locally and without the possibility of sending captured data to the Cloud, where a decision can be made remotely as well. In the first two approaches, long-range communication technology was used just for sending the final results of inference. In the last paper, the results were automatically shown on the display of the board.

In contrast, our work allows the possibility of making decisions on both parts, battery-less IoT devices, and the Cloud. We study the trade-off between the energy consumption and the accuracy of inference results, analyzing under which circumstances it is better to continue and make the decision locally or to wait until the required amount of energy is collected for sending the data to the Cloud where the higher accuracy model is deployed. We estimate the possible latency that can occur in both cases, respecting the defined task deadlines. Also, in our work, we consider an energy-aware task scheduler, which allows us to intelligently schedule all defined tasks based on the available energy, avoiding power failures and maintaining forward progress.

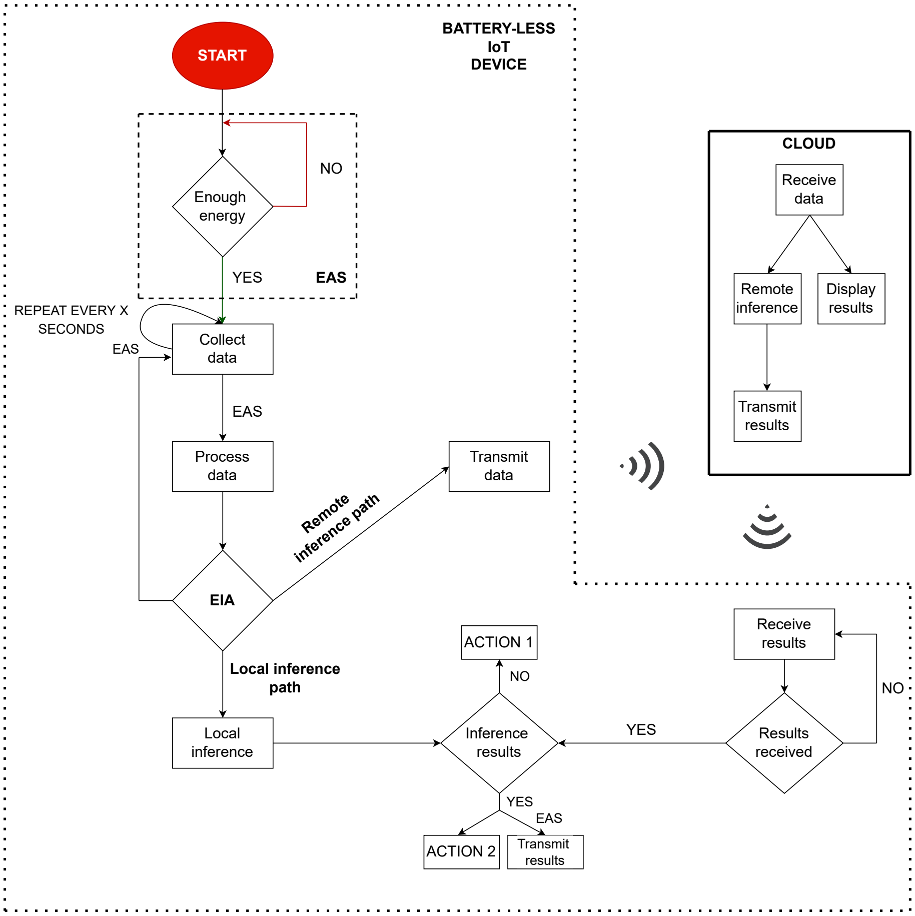


Figure 1: System architecture demonstrating the deployment of an energy-aware IoT application on the battery-less IoT device and decision-making process that can run locally or in the Cloud depending on task schedules and energy availability. Both parts of the system, the battery-less IoT device and the Cloud, can be connected via different wireless technologies.

3. System Architecture

In this section, the proposed system architecture that can be used for the deployment of energy-aware IoT applications on battery-less IoT devices is presented. It consists of two main parts: (i) the battery-less IoT device and (ii) the Cloud, which can be connected via different wireless technologies, such as Bluetooth Low Energy (BLE) or LoRaWAN, as shown in Figure 1. The energy-aware aspect of our approach is supported by integrating the energy-aware task scheduler (EAS) on the battery-less IoT device. Based on the available energy and required voltage thresholds that the device needs to reach, the scheduler decides when to execute each application task. Based on the Energy-aware Inference Algorithm (EIA), the device is able to decide whether to perform inference locally or to send the captured data to the Cloud for remote inference.

3.1. Energy-Aware Optimization Algorithm

The energy awareness of our approach is achieved by extending the energy-aware task scheduler presented in our previous work [2]. It uses a task profile to decide when the capacitor voltage is high enough to execute it, completely avoiding power failures and maintaining forward progress. The energy-aware IoT application is divided into atomic tasks characterized by an execution order implemented in the form of a parent/child relationship. In this way, each successor task in the flow will be connected with its predecessor, considering the output from the previous task as its input. Each parent task can have one or more dependent child tasks, which are selected only if their constraints (e.g., repeat task, data availability, comparison on output, etc.) are satisfied. As the proposed scheduling approach is based on priorities as well, the task with the highest priority will always be selected first if there are multiple candidate tasks that satisfy deadlines and other defined constraints.

In this work, we study the trade-off between energy consumption and accuracy, analyzing under which circumstances it is better to perform inference locally, at the cost of some accuracy, or to send the data to the Cloud in order to perform more accurate remote inference. If the local inference is selected, the accuracy loss comes from the fact that the model running on the device is a low-power version that has been pruned. Besides the energy consumption, another important parameter for our studies is the total time $\boldsymbol{\tau}$ the device needs to perform the full inference path, starting with a local inference or data transmission to the Cloud, and ending when results from the selected ML model are confirmed on the IoT device by performing the defined action (cf. Figure 1). The actuation part is defined as ACTION 1 and ACTION 2 in Figure 1, and can be implemented in the form of turning on an LED once the inference results are available on the device or other actions such as turning on the buzzer if a positive decision is made. Based on that, in our work, we also consider a trade-off in terms of latency, taking into account the deadline t_D before which results must be confirmed on the device. To decide which of these two options is preferable and can satisfy both, energy and timing constraints, at the same time, we define an energy-aware inference algorithm, shown in Algorithm 1. Table 1 lists the set of parameters used in our energy-aware optimization algorithm.

After the device completes the data processing task T_{pr}, the EAS will need to select the next task among three possible ones (Lines 1-2). The collect data task T_{col} is its own child task, so it will be already in the task list T. The two other tasks, local inference T_{loc} and data transmission T_{data}, are child tasks of the processing data task. One of these two tasks will be added to the list and executed, but only if its energy and time constraints can be satisfied.

The total time ($\tau_{loc}(V_0, I_h)$) for the local inference path includes time values of different states of the device (Line 3): t_{li} is the time needed to perform local inference, t_{act} is the execution time of the defined action (ACTION 1 and ACTION 2 in Figure 1), and t_1 that represents the time needed to reach the required voltage threshold $Vreq_{loc}$ for the full inference path, starting with local

Table 1: Set of parameters of the energy-aware optimization algorithm

Parameter	Definition
$T = \{T_{col}, T_{pr}, T_{loc}, T_{data}\}$	Set of tasks to be executed
$\tau = \{\tau_{loc}, \tau_{data}\}$	Set of times to execute inference path
$t = \{t_1, t_2, ..., t_n\}$	Set of times to reach V_{req}
$V_{req} = \{Vreq_{data}, Vreq_{loc}\}$	Set of required voltage thresholds
V_0	Initial voltage
V_c	Measured capacitor voltage
I_h	Harvesting current
t_D	Task deadline
t_v	Voltage check interval
t_{li}	Time to execute local inference
t_{act}	Time to execute defined action
t_{tx}	Time to send data to the Cloud
t_{cl}	Execution time in the Cloud
t_{rx}	Time to receive results from the Cloud

Algorithm 1: Energy-aware optimization algorithm

```
if $T_{pr}$ is completed then
    $T_{col}, T_{loc}, T_{data}$;
    $\tau_{loc}(V_0, I_h) = t_1 + t_{li} + t_{act}$;
    $\tau_{data}(V_0, I_h) = t_2 + t_{tx} + t_{cl} + t_{rx} + t_{act}$;
    if $\tau_{loc}$ and $\tau_{data} \leq t_D$ then
        $T_{data} \leftarrow$ highest priority task;
        $\mathbf{T} \leftarrow \mathbf{T} \cup \{T_{data}\}$;
        while $V_c \leq Vreq_{data}$ and $\tau_{data} \leq t_D$ do
            sleep $t_v$ seconds;
        end
        if $V_c \geq Vreq_{data}$ and $\tau_{data} \leq t_D$ then
            execute $T_{data}$;
        end
    end
    else if $\tau_{loc} \leq t_D$ and $\tau_{data} > t_D$ then
        $T_{loc} \leftarrow$ highest priority task;
        $\mathbf{T} \leftarrow \mathbf{T} \cup \{T_{loc}\}$;
        while $V_c \leq Vreq_{loc}$ and $\tau_{loc} \leq t_D$ do
            sleep $t_v$ seconds;
        end
        if $V_c \geq Vreq_{loc}$ and $\tau_{loc} \leq t_D$ then
            execute $T_{loc}$;
        end
    end
    else if $\tau_{loc}$ and $\tau_{data} \geq t_D$ then
        remove data;
        $T_{col} \leftarrow$ highest priority task;
    end
end
```

inference and ending when the appropriate action is performed. The first two time values can be measured before the application starts, using one of the available power profiling tools [26] [27]. The time value t_1 can be calculated using the mathematical model and equations presented in our previous work [28]:

$$t_1(V_0, Vreq_{loc}, I_s) = -\rho(I_s) C \ln(\frac{Vreq_{loc} - I_h \rho(I_s)}{V_0 - I_h \rho(I_s)}) \tag{1}$$

where V_t is replaced with $Vreq_{loc}$, V_0 is the current voltage of the capacitor, and I_h is the estimated harvesting current. The chosen capacitor size is represented as C, while the load resistance is modeled as a function $\rho(I_s)$. V_0 is the measured starting voltage from which the local inference path starts. I_h is the harvesting current that can be estimated in three ways [2]: (i) worst case estimation ($I_h = 0$) where the harvesting current can simply be estimated as 0, resulting in the longest possible waiting time until the threshold is reached, (ii) perfect prediction ($I_h =$ known) where the harvesting current is constant and defined before the experiment starts, which makes the calculation much easier, and (iii) predicted estimation, where the harvesting current is predicted using a time-series prediction algorithm. While the worst-case prediction increases the required voltage threshold of tasks and thus the time spent charging the capacitor, it results in less power failures, which can occur if the predictor over-estimates the harvesting current.

On the other hand, the total time ($\tau_{data}(V_0, I_h)$) for the remote inference path includes (Line 4): the time t_2 needed to reach the required voltage threshold $Vreq_{data}$ for this specific inference path, the execution time of data transmission (t_{tx}), the total execution time in the Cloud t_{cl}, the time the device needs to receive remote inference results t_{rx}, and the action execution time t_{act}. Similar to the previous case, t_{tx}, t_{rx}, and t_{act} can be measured before the application starts using power profiling tools. It must be noted that t_{tx} and t_{rx} already include the cost of possible network latency, considering the following 4 types of delays: transmission delay, propagation delay, queuing delay, and processing delay. The execution time in the Cloud (t_{cl}) can be estimated by measuring the inference time of the neural network algorithm running in the Cloud in advance. Finally, t_2 can be calculated based on Equation 2:

$$t_2(V_0, Vreq_{data}, I_s) = -\rho(I_s) C \ln(\frac{Vreq_{data} - I_h \rho(I_s)}{V_0 - I_h \rho(I_s)}) \tag{2}$$

Based on the current capacitor voltage V_0 and harvesting current I_h, the EIA is able to calculate the time needed for the execution of both inference paths and based on the obtained value select the optimal solution, respecting the defined deadline t_D. In case both solutions can be executed before the deadline expires, the task with the highest priority will be selected and added to the task list for execution (Lines 5-7). As the heavy-weight ML model deployed in the Cloud can provide more accurate results, the remote inference path is given the highest priority. In order to execute the full inference path in an energy-aware manner, the required voltage threshold V_{req} must be calculated, considering the mathematical model and equations presented in our previous work [28]:

$$V_{req} = \frac{V_{min} - I_h \rho(I_s)(1 - e^{(\frac{-t_s}{\rho(I_s)C})})}{e^{(\frac{-t_s}{\rho(I_s)C})}} \tag{3}$$

where V_t is replaced with V_{min}, the minimum operating voltage below which the device is not able to stay on, and will shut down. In this case, the average current consumption I_s and execution time t_s are measured for the full inference path, which is considered as a one unit, as shown in Section 5. Starting at V_{req} ($Vreq_{loc}$ and $Vreq_{data}$), a safer execution of all tasks considered in one inference path can be ensured.

The device will measure the capacitor voltage V_c at predefined intervals (i.e., every t_v seconds) to check if the required voltage threshold V_{req} of the selected path is reached. As long as the capacitor voltage is below this value, the device is in a deep sleep state to conserve energy (Lines 8-9). Once the capacitor voltage is equal to or higher than the required threshold, and the selected path can still be completed within the deadline, it will be executed. In this way, more accurate results can be provided in case enough time and energy are available to perform remote inference.

If the remote inference solution cannot be executed within the deadline, the EAS will proceed with local inference. In this case, the local inference task will be selected as the next highest priority task in the flow and added to the task list (Lines 15-17). Similar to the first case, the local inference path will only be executed if the required amount of energy is collected and execution can be completed within the deadline. Finally, if none of the proposed solutions can be completed within the defined deadline, the scheduler will remove the processed data, and select the collect data task to be repeated again. In this way, data freshness is ensured as the device will not work with expired data.

3.2. ML and TinyML Workflow

The main idea of our approach is to run two ML models, the low-power tinyML model that can be deployed on the constrained battery-less IoT device, and the heavy-weight ML model deployed in a Cloud data center. To prepare both versions of the ML model, several phases are required, as shown in Figure 2. The first phase is to build the ML model that will be capable of providing the desired intelligence in our system architecture. Then, the next step is to train the model on a computer or server using pre-collected training data. For this purpose, typical ML frameworks such as TensorFlow, Edge Impulse, or Pytorch can be used [29].

Once this step is done, the obtained model will be used in two ways. First, the heavy-weight pre-trained ML model will be deployed on a Cloud server, as it requires a lot of computational and memory resources. In this way, the remote decision can be made by running the inference task in the Cloud. Second, the obtained model will be optimized and converted into a tinyML model that has inherited all the properties of the original one, but with reduced computational

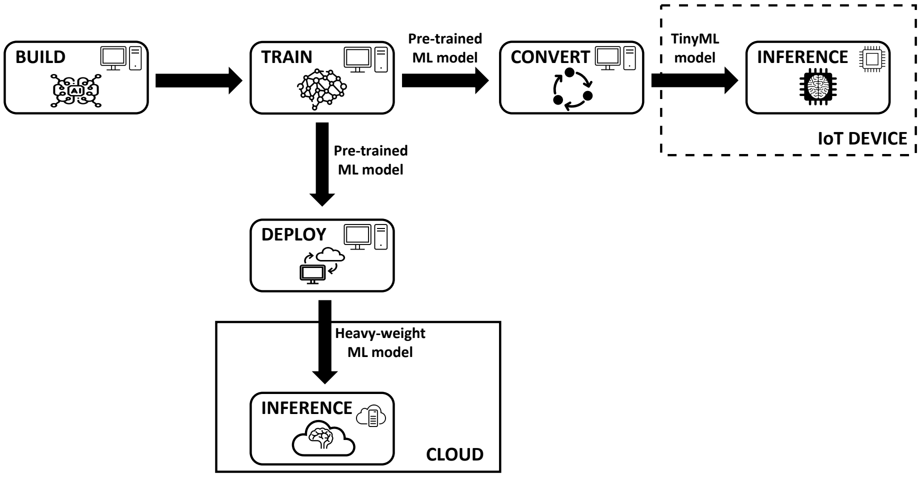


Figure 2: A neural network is trained based on prior datasets. The trained neural network is deployed on the Cloud device but is too complex to run on constrained IoT devices. Through pruning and quantization, a second tinyML network is created that can be run locally on the IoT device, albeit at lower accuracy.

and memory overhead, and thus lower accuracy, so it can be executed on constrained IoT devices [29]. Once the model is deployed on the device, it becomes able to perform local inference tasks based on the data it collects.

3.2.1. ML model optimization

Once the training process is done and the desired accuracy is achieved, the heavy-weight ML model must be optimized before its deployment on the constrained IoT device (cf. Figure 2). We use a combination of several techniques to convert the ML model into a form that can be run on an embedded device:

1. **Knowledge Distillation (KD)** - this is the technique of transferring knowledge from a large model (teacher) or set of models to a small well-optimized model (student) [30]. In this way, a small model will try to collect all the necessary knowledge in order to solve the same task as a larger model. The main goal of this technique is to transfer knowledge with the minimum loss function, where the target is a distribution of class probabilities provided by the softmax function used on the teacher side [12]. This probability distribution assigns the highest probability to one particular output, with all other probability classes close to 0. As this result cannot provide much information, the distribution must be spread across the class label, providing more targets that indicate which classes the teacher found more similar to the predicted one [31]. The final student model will be trained based on these values in order to achieve similar accuracy as its teacher.

2. **Pruning** - pruning technique is used to convert a dense neural network to

a more sparse network, reducing the size of the network with limited loss of accuracy [12]. It is an iterative process that uses a training model and systematically removes weights that are below a defined threshold over a different number of epochs [32]. If the pruned network cannot provide the same accuracy as the dense network from which it is created, the retraining of the remaining weights will be performed. During the retraining process, the pruned neurons without input and output connections will be removed as well, reducing the total size of the network. De Leon et al. [33] presented the depth pruning technique with an auxiliary network that acts as a new head of the pruned model. Their technique is easily interpretable, requiring no special hardware support during inference. Using their approach, the pruned model shares weights with the based one, reducing the final model size, memory overhead, and accuracy loss, which saves energy in constrained IoT devices and provide more accurate on-device inference results.

3. **Quantization** - after finishing the distillation and pruning part, the next step is to quantize the model, which will reduce the model size even more. Quantization is the technique that reduces the numerical precision of a network by converting higher precision values to lower ones [30]. The weights and activations of the ML model are usually represented as 32-bit or 64-bit floating-point values. By mapping these values down to 8-bit integers, the precision of the ML algorithm is reduced to fit the MCU architecture, enabling faster computation, lower power consumption, less memory overhead, and deployment on different embedded platforms [32] [12].

4. **Deployment** - the final step is to convert the optimized ML model into a form readable on the embedded device. An interpreter, such as TensorFlow Lite [34], is used to convert the ML model (i.e., the script written in Python) into a file written in any language, typically C or C++, that is understandable to the MCU. Finally, the tinyML model is deployed and compiled on the embedded device, where it is used for local inference.

3.3. On-device local inference

The energy-aware IoT application that contains different tasks is implemented on the device. The first task in the flow is collecting data, which can be performed using a camera module or different types of sensors. The captured data can be images, videos, or different sensing data such as temperature, humidity, or pressure that is later used as input to the tinyML model implemented on the device. This task is a periodic task and can be repeated every X seconds if enough energy is available. Once the data is captured, it will be processed and edited to be suitable as input for the ML model.

The required voltage thresholds for each application task must be calculated. Based on the obtained values, the scheduler determines if enough energy is collected for task execution. Using the optimization algorithm (Section 3.1), the optimal solution between local inference and sending data to the Cloud will be selected and executed. In case the decision is made locally, once the results are ready, the device will confirm them by performing the defined action and optionally sending the resulting output to the Cloud.

3.4. Cloud-based remote inference

Once the training phase is done (cf. Figure 2), the pre-trained heavy-weight ML model will be deployed on a Cloud server. It will be running and waiting for input data sent from the battery-less IoT device. If data is available, it will be edited if necessary and passed to the model, where the remote decision will be made. The final result will be sent back to the battery-less IoT device, based on which it will perform the appropriate action. In this way, both parts of the system, the battery-less IoT device, and the Cloud are connected and capable of making decisions.

4. Prototype Implementation: Person Detection

As a proof of concept for our algorithms, this section describes our prototype of an energy-aware IoT application implemented on a battery-less IoT device, including all defined tasks and the order of their execution. A brief overview of the used devices, divided into different subsystems, along with the deployment on the Cloud is given. In the end, we explain how the ML model for person detection is deployed on the battery-less IoT device and the Cloud.

4.1. Energy-aware IoT application

For evaluation purposes, we developed an energy-aware IoT application that is able to decide whether or not a person is detected in a captured image. The energy-aware application was deployed on the battery-less IoT device based on the proposed system architecture (cf., Figure 1). It is composed of seven main tasks. The first task in the flow is to capture the image using the connected camera module. This is a periodic task that can be repeated every X seconds if enough energy is available. Once the capturing task is done, the image is decoded, converted into grayscale, and processed. Using the optimization algorithm, the device will decide which type of inference is preferable at that moment in time. If the decision is made locally, the result will be confirmed by performing the defined action, which in our case is briefly turning on the appropriate LED (i.e., green LED if the person is detected and red LED if it is not), and optionally sent to the Cloud. Otherwise, the captured image will be sent to the Cloud for remote inference. Once the final result is available, it will be sent back to the battery-less IoT device where it will be confirmed by briefly turning on the appropriate LED.

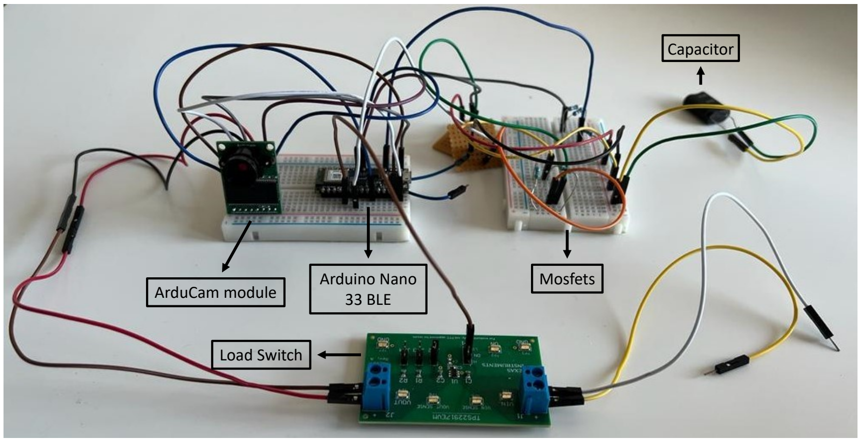


Figure 3: Microcontroller subsystem consisting of two main parts, the microcontroller unit on which the energy-aware IoT application and tinyML algorithms are running, and the ArduCam camera module for capturing and processing images.

4.2. Microcontroller subsystem

The microcontroller subsystem consists of two main parts: (i) the microcontroller unit (MCU), and (ii) the camera module connected to the MCU, as shown in Figure 3. The Arduino Nano 33 BLE [35] was chosen, due to its support to execute tinyML models. It is a miniature-sized 3.3V compatible board, based on the Nordic nRF52840 MCU [36] and running on ARM Mbed OS. The board has 1MB CPU Flash and 256kB RAM memory, featuring a more powerful processor compared to its predecessors, a 32-bit ARM Cortex-M4 CPU running at 64 MHz. These technical specifications of the board, allow us to deploy and run different types of neural networks, which convert the board into a more intelligent unit capable of making decisions locally. Also, the board supports serial and parallel interfaces such as serial peripheral interface (SPI) and inter-integrated circuit (I2C) bus that are used for configuration and communication with different sensing units such as the smart camera. From the communication perspective, the Arduino Nano 33 BLE supports BLE connectivity enabled through the Arduino BLE library [37].

In order to be able to capture and process the image, we consider the Arducam Mini 2 Megapixels (MP) Plus camera module [38]. It is a high-definition SPI camera that integrates a 2MP CMOS image sensor OV2640, providing miniature size, an easy-to-use hardware interface and an open-source code library [39]. The camera is connected to the Arduino Nano 33 BLE through different SPI and I2C pins, which enable the sensor configuration and are used for camera commands and data streams. There are also VCC and GND pins used for powering the camera module. Once the camera task is executed, the current continues to flow through these pins, which drastically increases the sleep current consumption of the Arduino board (above 108mA). To solve this issue, we added an additional Load Switch Evaluation Board (TPS22919EVM) [40] that provides the current to the camera module only when the defined GPIO pin is triggered. This resulted in around 100 times less current consumption during the sleep state (around 1.14mA), which is more suitable for battery-less IoT devices. Finally, the output of the camera is a JPEG 160x120 image that is decoded as a sequence of Minimum Coded Units, which are 16x8 blocks of pixels, and converted into grayscale.

To enable our Arduino Nano 33 BLE board to measure the voltage on the capacitor and compare it to the calculated voltage threshold of each application task, an additional voltage divider was added. Based on the obtained voltage value, the energy-aware task scheduler will know if the task is ready to be executed or the device needs to sleep more in order to collect enough energy. The current consumption will increase, as the voltage measurement circuit contains additional resistors. To reduce this, we used MOSFETS that act as circuit switches. In this way, the harvested energy can be used better and the battery-less IoT device can be modified to act in an energy-aware fashion.

4.3. Intelligent Power Management Unit

There are different types of Power Management Units (PMUs) available in the market. PMUs have multiple roles in the circuit, from charging the capacitor, the main energy storage of battery-less IoT devices, to regulating the output voltage to the load and extracting maximum power from the energy harvester (e.g., solar cell) [19]. As in our implementation, we consider a real energy harvesting environment with solar panels that harvest ambient light energy, there is a need to manage this incoming energy by using the appropriate PMU.

The AEM10941 [41] is an integrated energy management circuit designed by e-peas [42], for solar and thermal harvesters, that extracts DC power from up to 7-cell solar panels to simultaneously store energy in a rechargeable element (e.g., capacitor), after which it will be converted to a stable voltage to operate an MCU and peripherals. This solution can supply the system with two independent regulated voltages, the low-voltage output (LVOUT) which generates 1.2V or 1.8V, and the high-voltage output (HVOUT) which generates from 1.8V to 4.1V. The board starts harvesting energy at 380mV with an input power of only 3μA.

The power management is performed using a single inductor boost/buck regulator, with the aim to charge as much as possible energy from the solar panel and store it in the capacitor. The capacitor charges only when its voltage is lower than a specific value due to the implemented RG trigger circuit. When the capacitor voltage reaches the turn-on threshold (V_{turnon}) the battery-less IoT device is turned on and when its capacitor voltage drops below the turn-off threshold ($V_{turnoff}$) the device is turned off [2].

There are three main pins that have to be used on the e-peas evaluation board: (i) the BATT pin, which is the connection to the capacitor, (ii) the SRC pin, which is the connection to the harvested energy source, and (iii) the

HVOUT/LVOUT pin that provides the output voltage to the connected device, which is in our case the Arduino board.

Depending on the capacitor voltage, the board can be logically divided into four different modes [2]:

i) Voltage below $V_{turnoff}$ (Discharged), where the PMU only charges the capacitor without providing supply to the MCU (LVOUT and HVOUT are deactivated).

ii) Voltage between $V_{turnoff}$ and V_{turnon}, where there are two possibilities. First, when the capacitor charges from $V_{turnoff}$ (Discharged), then the PMU only charges the capacitor without providing supply, and the second one, when the capacitor already reached V_{turnon} (Ready-Charged), then the PMU provides the output voltage supply and charges the capacitor.

iii) Voltage between V_{turnon} and the maximum allowed (V_{max}) (Charged), where in the availability of harvesting current, the PMU charges the capacitor up to V_{max} and continues supplying the output voltage.

iv) Voltage above V_{max} (Overcharged), where the capacitor charging will be deactivated and the output voltage supplying is continued.

4.4. Wireless communication subsystem

To establish the communication between the battery-less IoT device and the IoT gateway, the wireless communication subsystem that supports a short-range data transfer via BLE is considered. There are different reasons why this technology is chosen for the communication part. We consider our prototype implementation as an indoor battery-less solution where energy efficiency is one of the key parameters and communication range is less of an issue. Also, the selected wireless communication subsystem should support the transfer of different amounts of data, from the simple message to the captured image. Taking all of these into account, the only technology that supports the transfer of sufficiently large amounts of data (i.e., captured image), enables short-range communication without consuming too much energy, and is suitable for battery-less IoT devices, is BLE. BLE is a wireless personal area network (WPAN) technology designed by the Bluetooth Special Interest Group [43], which compared to Classic Bluetooth is intended to provide considerably reduced power consumption and cost while covering a similar communication range. The main idea is to have two types of devices: central devices that act as clients waiting to receive data, and peripheral devices that are servers responsible for providing collected data from different sensors. In our case, both the IoT device and the IoT gateway/Cloud, can send and receive different types of data. The Arduino device turns on BLE only if it needs to send or receive data. Once BLE is turned on, the device starts advertising to let other devices, the IoT gateway in our case, know that it exists and is ready to connect. These advertising packets contain a list of services the device provides, based on which the connection can be established.

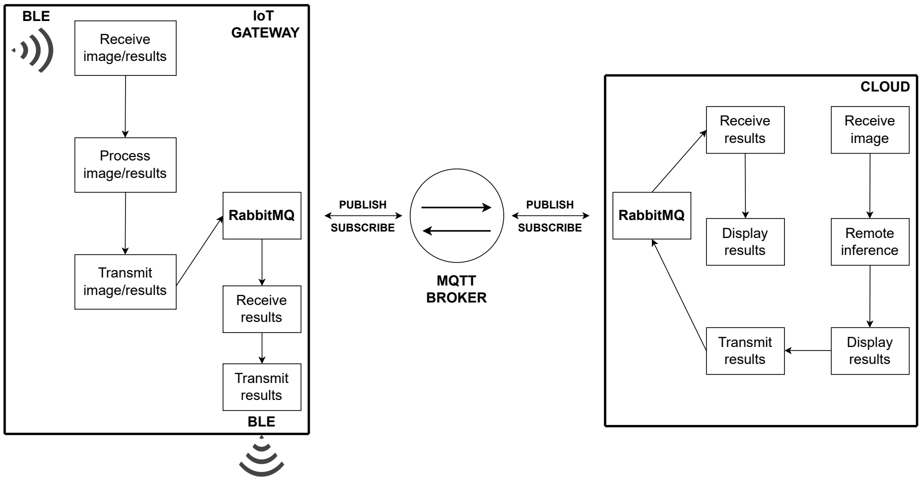


Figure 4: IoT Gateway/Cloud implementation demonstrating the communication and decision-making process for the image recognition application, running inference remotely in the Cloud.

Once the connection is established, data can be sent or received. In our case, the device can send two types of data, the captured image and the end result of the inference. In the first case, the device will send the image and wait for the response from the gateway. Once it receives data from the gateway, it will disconnect and turn off BLE, reducing the total energy consumption. In the second case, once the end result of the local inference is sent, the device will disconnect and turn off BLE immediately. On the other hand, the gateway keeps BLE constantly turned on, looking for new devices and waiting to be connected with them. Using BLE, we are able to show that both parts of the system, the battery-less IoT device and the Cloud, can be involved in the decision-making process and chosen as an optimal inference solution under certain circumstances.

4.5. Gateway/Cloud subsystem implementation

Figure 4 shows the IoT gateway and Cloud implementation, which includes all defined tasks that need to be performed once the data from the IoT device is received. In order to receive data sent from the device via BLE, the Bleak library [44] is installed on the gateway side. Bleak is a Generic Attribute Profile (GATT) client software, capable of discovering and connecting to BLE devices that act as GATT servers. It provides an asynchronous, cross-platform Python API to connect and communicate with sensors or battery-less IoT devices. Using Bleak, our gateway is able to read, write, and get notifications and/or data sent from the Arduino device. It can receive two types of data: the simple message that contains the end result of the local inference performed on the device or the captured image from the camera module connected to the Arduino board. We developed a Python application that includes the Bleak library and enables the gateway to receive both types of data. Once the data is received, it will be processed and prepared to be sent to the Cloud. The transmission and reception tasks are performed using the Message Queuing Telemetry Transport (MQTT) protocol, an IoT communication protocol that was built as a super-lightweight messaging transport suitable for linking faraway machines with minimal code and network resources. The RabbitMQ clients [45] are deployed on both sides, the IoT gateway and the Cloud, and are responsible for transmitting and receiving data to and from the MQTT broker.

The Cloud implementation is done by using Kubernetes Orchestration. A Ubuntu-based docker image, which is expanded with the necessary libraries to enable the intelligence and deployment of the ML model, is used. The docker image is then run in the docker container deployed on Kubernetes. The data is received in the Cloud via RabbitMQ. If it is just a simple message containing the end results of local inference performed on the device, it will be immediately shown in the console terminal. Otherwise, the captured image from the camera module is used as input to our heavy-weight ML model used for making the remote decision. Once the inference task is done, the final result will be displayed and via RabbitMQ sent to the MQTT broker, from which it will be forwarded back to the device.

4.6. Person detection models

In this work, we developed, deployed, and evaluated an energy-aware IoT application able to detect whether a person is present or not on the captured image. The person detection task is achieved by deploying a CNN on both parts of the system, the battery-less IoT device, and the Cloud. The proposed CNN, MobileNet V1, is a part of MobileNets [46]. MobileNets are a family of efficient models for mobile and embedded applications, which enable the building and deployment of lightweight deep neural networks. To train the model, the Visual Wake Words dataset [47] containing images that belong to two classes, person or not-person, was used. The training was performed on a server with a 16X NVIDIA Tesla V100 GPU, 8-core Dual Intel Xeon Platinum 8168 CPU, and 16GB of Micron DDR4 LRDIMM RAM, considering around 40GB of data. As the chosen dataset was large, consisting of around 40GB of data, we had to use a machine that included 1 GPU, 8 CPUs, and 16GB RAM. The proposed CNN has 28 layers, 26 depthwise separable convolutions, 1 fully connected layer that is the first layer in the architecture, and 1 pooling layer. It uses ReLu activation functions for all the layers except the last one which has a Softmax activation function. Two important parameters that decrease the footprint of the MobileNet model are the width multiplier α, which thins a network uniformly at each layer, and the resolution of the input data [46]. Based on that, the proposed values of α=0.25 and 96x96 grayscale input images are chosen [48]. The other hyperparameters used to control how weights were updated during the training process are shown in Table 2. The proposed architecture is based on [46].

The neural network was trained using the TensorFlow framework, through 1 million epochs using the RMSprop [49] optimizer. Depending on the memory requirements, the training process can be stopped earlier at the cost of some

Table 2: Hyperparameters used for training the neural network

Hyperparameter	**Value**
Learning rate	0.045
Label smoothing	0.1
Learning rate decay factor	0.98
Number of epochs per decay	2.5
Moving average decay	0.999
Batch size	96

accuracy. Once the training is done, the results from the TensorFlow training environment can be converted into a form that can be deployed on a tiny batteryless IoT device. The heavy-weight model was converted and generated into a TensorFlow Lite File [34] using int8 quantization. Finally, the TensorFlow Lite File was converted into a C++ data array that can be easily deployed on the embedded device. In this way, the heavy-weight person detection model, which requires 3.3MB of memory, was converted into a 250KB model able to be run on our Arduino Nano 33 BLE board. The described procedure follows the steps proposed in [48].

After getting both ML models, the total number of required floating operations (FLOPs) for a single execution of the model can be calculated. Our results showed the same number of FLOPs for both models (14.3M FLOPs). Based on this number, it is easy to calculate the extensive multiply accumulate operations (MAC) value:

$$MAC = \frac{FLOPs}{2} \tag{4}$$

The difference between the heavy-weight and tinyML model is in the way the weights are saved and calculated. Considering the heavy-weight ML model deployed in the Cloud, these weights are saved and calculated using a float 32-bit precision. In order to adapt this model to be suitable for an embedded device, the float graph is converted to an integer 8-bit (int8) format. By mapping these values down to 8-bit integers, the precision of the tinyML model is reduced with an impact on model accuracy [50].

The accuracy of an ML model can be defined as the ratio of correct predictions out of all predictions made by that algorithm and can be calculated as follows [51]:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

where TP and TN are true positive and true negative outcomes when the model correctly predicts the positive and negative class, and FP and FN are false positive and false negative outcomes suggesting that the model incorrectly predicts the positive or negative class.

We evaluated both models considering two approaches (cf., Figure 5). In the first approach, we used the aforementioned dataset that was previously divided

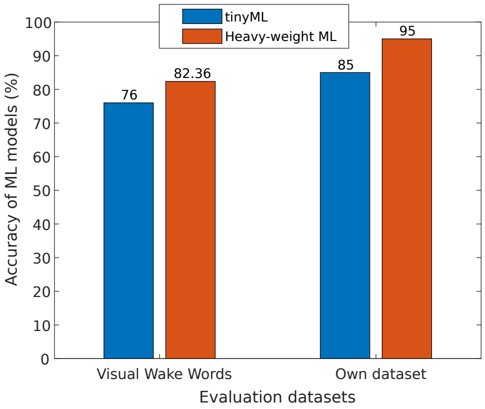


Figure 5: Evaluation of both ML models considering existing and own datasets

into training (67.05%) and validation (31.95%) datasets [47]. The fully-trained heavy-weight ML model achieved an accuracy of 82.36%. On the other hand, the tinyML model achieved a lower accuracy of 76%. For the second approach, we considered real-time evaluation. In this case, we ran the energy-aware IoT application on the battery-less device and used the captured images from the Arducam camera module as input for both models. The used dataset consists of 20 images that have been divided into 2 datasets, 10 person and 10 not-person images. The device showed the outcome of its local inference calculation using an LED (i.e., green if the person is detected on the captured image and red if it is not) that blinked once the local inference was performed. On the other hand, once the image was captured, it was sent to the Cloud as well where the remote inference was performed and the result displayed on the screen. The results showed an accuracy of 85% for the tinyML model and 95% for the heavy-weight ML model deployed in the Cloud. Compared to the existing dataset, both models showed better accuracy results when our own dataset was used due to the much smaller number of tested samples.

5. Device and Application Profiling

To perform energy-aware scheduling, the current consumption (I_s) and execution time (t_s) of the different states of the device, such as capturing and processing an image, transmitting and receiving data, or performing local inference, are required. In this section, we provide a brief overview of the used device and application profiling methodology to get the current consumption and execution time of different states of the device, as well as the actual results. These values were obtained with a Nordic Power Profiler Kit II [26], a standalone unit that can measure the current levels of different devices, providing a voltage supply between 1.6V and 5.5V. The real experiments, measurements, and validation of our approach were performed using the Arduino Nano 33 BLE board

Table 3: Current consumption and time values of the Arduino Nano 33 BLE board

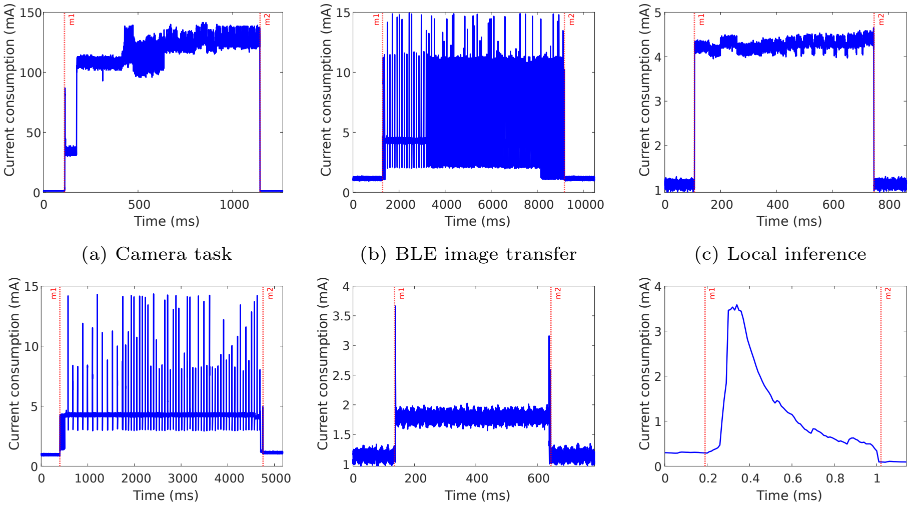
	AVG ± STD		Worst Case	
State	**Current draw**	**Execution time**	**Current draw**	**Execution time**
Camera task	112.68±0.55 mA	1039.6±5.32 ms	113.31 mA	1049 ms
Transmit image & Receive results	4.24±0.014 mA	7990.2±74.15 ms	4.26 mA	8157 ms
Local inference	4.246±0.017 mA	647.84±3.49 ms	4.27 mA	653.2 ms
Transmit results	4.247±0.02 mA	4379.2±19.49 ms	4.29 mA	4399 ms
Blinking LED	1.771±0.011 mA	507.6±1.21 ms	1.79 mA	509.9 ms
Voltage check	0.436±0.029 mA	3.35±0.34 ms	0.513 mA	3.884 ms
Sleep	-	-	1.14 mA	-

(cf., Section 4.2) on which an energy-aware IoT application for person detection was implemented. The Arduino Nano 33 BLE board operates in low-power mode, decreasing the total current consumption and enabling the energy-aware IoT application to run battery-less.

The low-power consumption on the Arduino Nano 33 BLE board can be enabled through different steps. First, the MPM3610 step-down voltage regulator used to convert the 5V voltage supply to 3.3V must be disconnected by cutting the solder jumper (SJ1) on the bottom of the board. Second, turning off the power LED, sensors, and the I2C pull-up resistors will decrease the current below 1mA. Finally, disabling the Universal Asynchronous Receiver/Transmitter (UART0) port will save an additional 500μA, which results in around 280μA of sleep current in the end.

Table 3 shows the average current consumption and duration for different states of the Arduino Nano 33 BLE board measured at 3.3V, the voltage at which the device will run with the e-peas power management board. The average values ± standard deviation (AVG ± STD) and maximum values (Worst Case) of current consumption and execution time for different states of the device were obtained based on 30 repeated measurements. In our experiments, the worst-case energy consumption and execution time values were considered by the scheduler in order to ensure that the device will not turn off after task execution due to unexpected peaks in energy consumption.

The highest energy cost task is the camera task, which includes two parts: capturing and processing an image. The processing part of this task includes different subtasks such as image decoding, converting into grayscale, and editing for ML models. Once this task is done, the device goes into a sleep state consuming around 1.14mA when the BLE module is added, and around 920μA without considering the communication part. This can be achieved by adding the additional load switch evaluation board (cf., Section 4.2) that acts as a switch and stops providing the current to the camera module once the defined GPIO pin is triggered. There are two different BLE tasks: (i) Transmit image, which enables the transmission of the captured image to the Cloud and Receive results, which enables the reception of remote inference results, and (ii) Transmit results, which enables the transmission of local inference results to the Cloud.



(a) Camera task

(b) BLE image transfer

(c) Local inference

(d) BLE result transfer

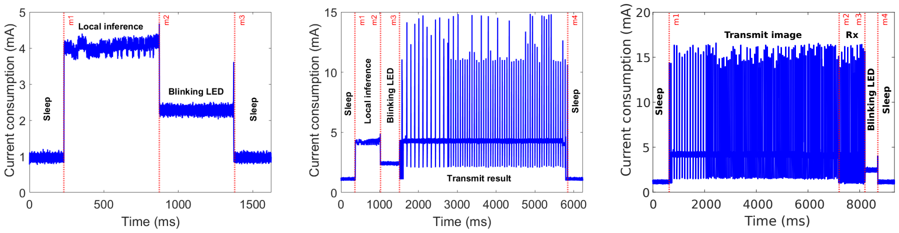
(e) Blinking LED

(f) Voltage measurement

Figure 6: Example of measured current consumption and execution time for different states of the Arduino Nano 33 BLE board (cf., Table 3)

The BLE module is turned on only when one of these two tasks is selected for execution. Otherwise, it is turned off to keep the current consumption lower. When one of two BLE tasks is selected to be executed and the BLE module is ready, the device will start polling and advertising, until a BLE connection with the gateway is established and transmission/reception of packets can start. The reception of inference results from the Cloud was implemented as a callback function on the Arduino device. This function will be automatically called every time remote inference results are available to be received. Based on this and the absence of the slave latency feature in the Arduino BLE library, after sending the image to the Cloud, the device must keep the BLE connection active until the remote results are received. Once all packets are sent or received, the device will disconnect from the gateway, turn off the radio, and go to a sleep state. Both BLE tasks are time-costly, requiring more than 8 seconds for image transmission and results reception, and 4.4 seconds to send inference results while consuming 4.26mA and 4.29mA on average respectively. The output from the camera task can also be used as input to the tinyML model deployed on our device. In this case, the decision will be made locally, which draws a similar current, but executes 10 times faster than remote inference (4.27mA for 653ms). Once this task is done, the final results will be confirmed by briefly turning on the appropriate LED. The example of the current drawn and duration for different states of the Arduino Nano 33 BLE board is shown in Figure 6.

Taking into account the mathematical optimization algorithm presented in Section 3.1, the device is able to choose whether to make the decision locally or to send data to the Cloud, where remote inference will be performed. There



(a) LI path

(b) LI+S path

(c) RI path

Figure 7: Measured current consumption and execution time for inference paths considered by EIA

are two possible inference paths: (i) a local inference (LI) path that starts with the local inference task, after which the result is confirmed by briefly turning on the appropriate LED (cf., Figure 7a), and optionally transmitting the inference result (LI+S) to the Cloud (cf., Figure 7b), and (ii) a remote inference path (RI) that starts with the image transfer to the Cloud, after which the result from the remote inference will be received on the device and confirmed again in the same way, turning on the appropriate LED (cf., Figure 7c). Based on that, we measured the current consumption and execution time of full inference paths, where the considered tasks are performed one after the other, without additional energy checks. We then use these values in order to calculate the required voltage thresholds that can ensure the successful execution of the full inference paths (i.e., without the device powering down). The local inference path consists of only two tasks, local inference and blinking LED, and requires 1.17 seconds for completion, consuming on average less than 3.5mA. In case the final inference result is sent to the Cloud, the total current consumption and execution time of this inference path will increase up to 4.1mA and 5.47 seconds on average respectively. On the other hand, the remote inference path consists of three tasks on the device and remote inferencing on the Cloud. It takes more time for execution (8.66 seconds), consuming a little more than 4mA on average. It must be noted that we do not include the camera task as part of these inference paths, as it is executed separately first, and is required for both considered inference paths.

As we consider energy awareness in our approach, there are three additional parts that can also have an impact on the total current consumption of the device. The energy-aware task scheduler adds some extra operations, such as (i) selecting the task with the highest priority to be executed. (ii) removing it from the task list once it is finished, and (iii) occupying new places in the task list by adding all dependent child tasks related to the executed one. The optimization algorithm (Section 3.1) will also add some extra consumption as the device needs to calculate the total execution time for both possible solutions, local and remote inference, and based on that chooses the most optimal one. Finally, before each task execution, the device needs to measure the voltage on the capacitor in order to check if the required voltage threshold is satisfied. As in our case, the device cannot directly read the capacitor voltage, an additional voltage divider with resistors is used to enable this (cf., Section 4.2). This results in an additional 513μA for 3.88ms every time the voltage measurement task is called, without including the influence of the energy-aware task scheduler and optimization algorithm.

6. Results and Discussion

In this section, we present the results and validation of our hardware-software prototype described in Section 4, which enables person detection on battery-less IoT devices. Based on the defined IoT application, Cloud implementation, and manually set configuration of the e-peas power management board, we have performed different experiments, considering different parameters and taking into account the real energy harvesting environment, to validate our proposed solution. For the experiments, we have considered two main approaches: (i) constant harvesting current and voltage during the full time of the experiment where a controllable setup with artificial light is used, and (ii) dynamic harvesting current and voltage that changes over time due to unpredictable natural sunlight intensity.

6.1. Controlled experiments with artificial light source

In this approach, we considered the known harvesting current for calculating the required voltage thresholds of application tasks, which is constant for the full time of each individual experiment. We assumed that the value of the harvesting current was perfectly determined and does not change over time.

In order to design a controllable setup and perform experiments considering different configurations, we used an artificial light placed at some distance above the solar panel. As a light source, a Philips Hue White A21 bulb attached to a plastic dark box, offering a powerful 1600-lumen output, was used. A Panasonic AM-5608 [52] solar panel that consists of 6 amorphous silicon solar cells was considered. Using the voltage divider, presented in Section 4.2, the Arduino Nano 33 BLE board is able to measure the capacitor voltage and compare it with the required threshold. Once the obtained voltage value reaches the defined threshold, the task will be executed. Otherwise, the device goes into a sleep state for a predefined time interval, after which it will check again if enough energy is available. Finally, the e-peas power management board is added to: (i) charge the capacitor, (ii) regulate the output voltage to the board, and (iii) extract the maximum power from the solar panel. The designed setup used in our experiments can be seen in Figure 8.

Table 4 lists the general parameters used in our experimental setup when the constant harvesting current is used. The maximum allowed voltage V_{max} is equal to 4.5V, above which the capacitor voltage cannot increase anymore. The battery-less device will turn on when the voltage threshold, V_{turnon}, of 3.92V is reached. The turn-off voltage, below which the device cannot operate is set to 3.6V. The output voltage of the e-peas power management board to our battery-less IoT device is configured to 3.3V. We chose to test three different capacitor

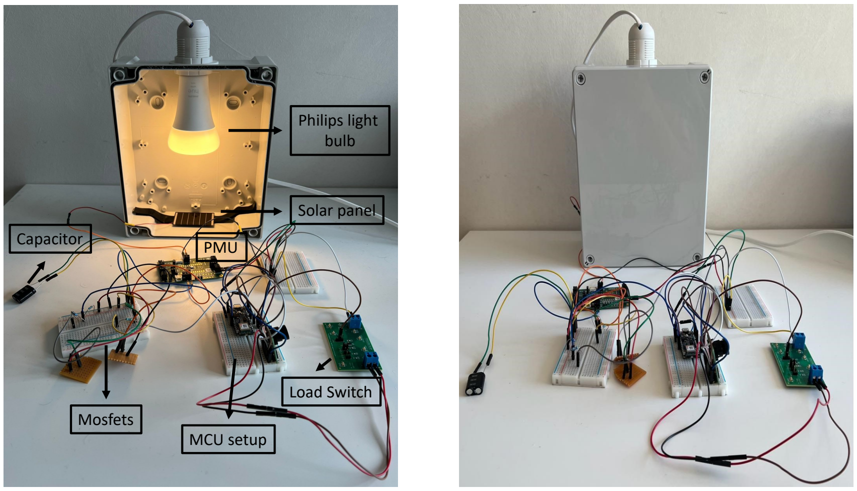


Figure 8: Controllable experimental setup with artificial light source including a Panasonic AM-5608 solar panel and microcontroller subsystem

Table 4: Experimental setup for the constant harvesting current approach

Parameter	**Symbol**	**Value**
Max Voltage	V_{max}	4.5 V
Turn-on Voltage	V_{turnon}	3.92 V
Turn-off Voltage	$V_{turnoff}$	3.6 V
Supply voltage	V_{output}	3.3 V
Capacitance	C	{0.5, 1, 1.5} F
Harvesting Current	I_h	{2, 4, 6} mA
Experiment duration	T_{exp}	1800 s
Minimum image capture periodicity	t_{cam}	10 s

sizes (i.e., 0.5F, 1F, and 1.5F) in combination with three different harvesting currents (i.e., 2mA, 4mA, and 6mA). It must be noted that the used capacitors can operate under temperatures between -40°C and +85°C, which enables our experimental setup to properly work even under extreme temperature conditions [53]. If we convert considered harvesting currents into the light intensity of our bulb, it will result in 19%, 34%, and 45% respectively. Finally, the experimental run lasted 30 minutes for all experiments.

For this approach, we considered three strategies: (i) local inference (LI) where the device performs only the local inference, confirming the result by briefly turning on the appropriate LED, (ii) local inference with sending results (LIS) where the device performs the local inference, again confirming the result by briefly turning on the appropriate LED, but also sends that result to the Cloud via BLE, and (iii) remote inference (RI) where the device sends the captured image to the Cloud for remote inferencing, waiting to receive the final result, which will be confirmed by briefly turning on the appropriate LED.

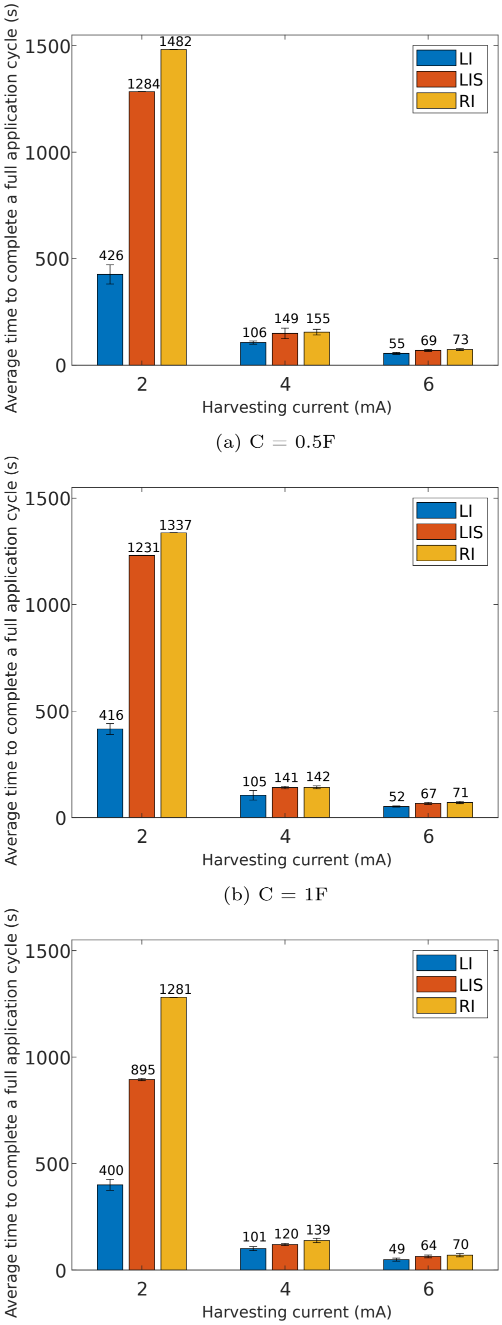
Each of these strategies was implemented and tested on the battery-less device separately. In order to fairly compare them, the experimental setups with the same configuration parameters (e.g., harvesting current, capacitor size, etc.) were considered in all three possible approaches. The main goal of these experiments was to analyze the trade-off between the accuracy of inference results and the energy consumption of the device when different inference strategies are implemented.

In our experiments, we have followed the behavior of the device considering different inference strategies in terms of the average time needed to complete the full application cycle (cf., Figure 9), starting with charging the capacitor until the required voltage threshold of the camera task, which is the first task in the flow, is reached and ending when inference results are confirmed on the battery-less IoT device (LI and RI) or sent to the Cloud (LIS). From the obtained values, the total number of completed application cycles for the defined experimental time can be calculated.

In all considered cases, the LI strategy shows the best performance in terms of execution speed (cf., Figure 9a, 9b and 9c). This is due to the fact that the local inference path consumes less current (cf., Figure 7a) compared to the other two approaches (cf., Figure 7b and 7c). It must be noted that in the LI approach, the BLE communication module is not used as the device will confirm the inference results by turning on the appropriate LED with which the application cycle will end. In this way, the sleep current can be reduced from 1.14mA to 900μA, which results in faster charging of capacitors, especially when higher harvesting currents such as 4mA and 6mA are considered. On the other hand, using the LIS inference strategy, the local inference path can start at the same voltage threshold as with the LI, but once results are available and confirmed, they also need to be sent to the Cloud. This additional task will start only when its voltage threshold is reached, which will prolong the duration of one application cycle. Finally, considering the RI approach, after capturing an image, the device needs to collect enough energy to send that image, and then wait to receive and confirm the inference result from the Cloud. This result in longer charging of capacitors, which in the end affects the duration of an application cycle.

From our results, it can be observed that the harvesting current impacts the final results in all three considered cases. As the harvesting current increases, the device needs less time to collect enough energy to perform all tasks in the cycle, which eventually decreases the average time needed for one cycle completion. For the LI approach, this is a reduction from 426 to 106 and 55 seconds, or 75.12% and 87.09% respectively, when a capacitor of 0.5F is used. Taking into account two other inference strategies, the total reduction is even higher (above 95% for 6mA harvesting current when using the RI approach), but still not enough to show better performance compared to the LI approach.

As the capacitor size increases, the average time needed for execution of the full application cycle is shorter (cf., Figure 9b and 9c). Intuitively, we expected that using smaller capacitors would reduce charging time and result in more application cycles, but this is not the case. The reason for this behavior is that



(a) C = 0.5F

(b) C = 1F

(c) C = 1.5F

Figure 9: Average time needed for execution of the full application cycle considering different inference strategies, capacitor sizes, and harvesting currents

a smaller capacitor, 0.5F in our case, requires a much higher voltage threshold for each defined application task, which in turn results in a longer charging time. In contrast, with larger capacitors such as 1F and 1.5F, the required voltage threshold of the highest energy-cost task (i.e., the camera task) in both cases is lower and the device can reach it faster. Once this task is executed, the device will sleep until the next threshold is reached, which will also be lower compared to when a 0.5F capacitor is used. However, the difference in time between application cycle executions for the different capacitor sizes, considering the same harvesting currents, is quite low. This means that the capacitor size does not have a huge impact on the final results regarding the number of successfully performed application cycles. For example, considering the LI approach, the average time needed for the completion of the one cycle decreases from 426 to 416 and 410 seconds when the harvesting current of 2mA is used. This is a reduction of 2.35% and 3.76% respectively, which is negligible. For higher harvesting currents such as 4mA and 6mA, the gain is even lower (e.g., only 0.94% improvement for LI with 4mA going from 0.5F to 1F).

Finally, to summarize, the LI inference strategy is much faster compared to the other two approaches, which in the end results in more successfully executed application cycles. For example, the LI approach can execute twice as frequently as LIS, and 3 times as frequently as RI, when a harvesting current of 2mA and a capacitor of 1.5F are used. However, as shown in Section 4.6, the accuracy of local inference is significantly lower than remote inference. As such, a trade-off occurs that should take into account both energy availability and timing constraints. This will be investigated in Section 6.2.

6.2. Realistic experiments with natural light

In the second approach, we considered a realistic harvesting environment with natural light, where the harvesting current and voltage vary over time. For these experiments, the same setup shown in Figure 8 was used, except the light bulb and box were removed, and natural light was taken into account instead. Table 5 lists the general parameters used in our experiments when the dynamic harvesting current is considered. For the e-peas power management board, the same configuration as in Section 6.1 is used, where V_{max} is equal to 4.5V, V_{turnon} is 3.92V, and the turn-off voltage below which the device cannot operate is set to 3.6V. The output voltage of the e-peas power management board to our connected battery-less IoT device is configured to 3.3V. The battery-less device is equipped with a 1.5F capacitor. The experimental run lasted 8 hours for all considered cases.

For this approach, we did not test three different inference strategies separately. In contrast, two inference strategies, the local inference path, and the remote inference path are deployed together. The experimental setup was placed on the windowsill of two separate rooms in the east and west direction in Antwerp, Belgium. The energy-aware optimization algorithm (cf., Section 3.1) was used to dynamically determine which of the two strategies to execute, with the task deadline t_D set to 28 and 40 seconds for east and west-side setups respectively. The task deadline values were determined based on the measured

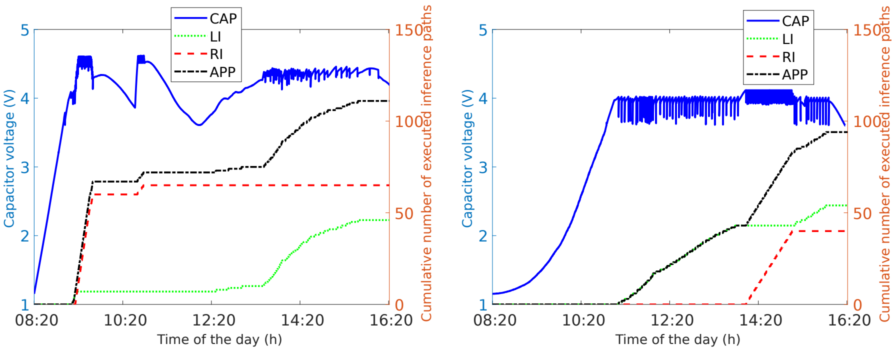
Table 5: Experimental setup for the dynamic harvesting current approach

Parameter	Symbol	Value
Max Voltage	V_{max}	4.5 V
Turn-on Voltage	V_{turnon}	3.92 V
Turn-off Voltage	$V_{turnoff}$	3.6 V
Supply voltage	V_{output}	3.3 V
Capacitance	C	1.5 F
Experiment duration	T_{exp}	8 h
Minimum image capture periodicity	t_{cam}	10 s

harvesting currents. In order to calculate the required voltage thresholds, the worst-case scenario ($I_h = 0$) is considered.

Figure 10 shows the capacitor voltage changes (CAP) when the batteryless IoT device (BLD) executes different application tasks, including one of two available inference paths (LI and RI), with a solar panel placed in the east and west direction. The cumulative number of executed inference paths is equal to the full number of completed application cycles that start with the camera task and end when the inference result is confirmed on the BLD. The experiments have been performed on the 10th and 11th of November 2022 on the east and west side respectively. In both cases, the day was sunny with short cloudy periods, and the sun rose at 07:51 and set at 17:00 [54]. It can be observed that the east-side BLD (cf., Figure 10a) turned on and reached the first required voltage threshold (i.e., the required voltage threshold of the camera task) faster than the west-side BLD (cf., Figure 10b) due to the fact that it receives more light during the morning hours. Between 9:00 and 10:00, and 10:20 and 11:00, the device was able to collect more than enough energy to perform all tasks in the application cycle. During this period, the capacitor maintained the voltage V_{max} for almost the full time, which resulted in more remote than local inference paths. The remote inference paths could be completed before the task deadline, respecting the time constraint, and trading off the higher current consumption for more accurate results. It must be noted that two voltage drops have been observed during this period (before 12:20), which is due to clouds that covered the sun and decreased the harvesting current. While the sun was moving to the west, the harvesting current decreased, resulting in more local than remote inference paths. As the local inference strategy was only able to satisfy the time constraint, the higher accuracy of results was traded-off for lower current consumption and shorter execution time. In the end, the east-side BLD executed 111 application cycles, of which 65 remote and 46 local inference paths (cf., Figure 10a).

On the other hand, the harvesting current was almost constant for the full time of the experiment (around 2.55mA) when the BLD was placed on the west side (cf., Figure 10b). The capacitor maintained a voltage between 3.65V and 4.05V for most of the time, allowing the device to perform more local inference paths, as the remote inference strategy was not able to satisfy the set time constraint. After 14:00, the device started to receive more light as the sun



(a) East-Side BLD

(b) West-Side BLD

Figure 10: Capacitor voltage behavior (CAP, 1.5F) over time and cumulative number of executed inference paths (APP) with a solar panel placed at east and west side windows. The battery-less IoT device (BLD) is able to execute local (LI) or remote inference path (RI) depending on the energy-aware optimization algorithm decision.

moved more to the west. This resulted in the more frequent selection of the remote inference path, due to more energy being available (between 14:00 and 16:00). Finally, the west-side BLD was able to execute 94 application cycles, of which 54 local and 40 remote inference paths (cf., Figure 10b).

Considering the west-side approach, the device executed a lower number of application cycles compared to the east-side due to: (i) a longer charging time until the device turned on and reached the required voltage threshold to start the application cycle, and (ii) a lower harvesting current for the almost full time of the experiment, which resulted in longer waiting times until the required voltage thresholds were reached. The lower harvesting current affected the number of completed remote inference paths as the west-side BLD mainly performed the local inference strategy (15% more compared to the east-side BLD). On the other hand, the east-side BLD had peaks where the voltage was equal to the maximum value, enabling faster application cycle execution. This affected the number of selected local inference paths as the device was able to perform a remote inference strategy before the deadline as well. For example, the east-side BLD executed almost 40% more remote inference paths compared to the west-side BLD. Finally, it must be noted that experiments have been performed during the late autumn period when days are shorter and the power of sunlight is lower compared to the spring or summer time during which the device would harvest more energy and be able to perform its tasks more frequently.

7. Conclusions

In this article, we presented a system architecture that enables the energy-aware deployment and management of tinyML algorithms on constrained battery-less IoT devices. Our solution consists of two main parts, the battery-less IoT device on which the light-weight tinyML model is running, and the Cloud where the heavy-weight model is developed, trained, and deployed. We proposed an energy- and deadline-aware inference scheduling algorithm, considering different harvesting conditions under which it is better to make the decision locally or to send data to the Cloud for remote inference. To validate our approach, we developed an energy-aware IoT application capable of detecting a person on the captured image and designed a Cloud and hardware prototype on which the necessary intelligence can be deployed.

First, based on a controllable setup with artificial light, we evaluated our approach, considering three different inference strategies. Our results showed that the local inference strategy without sending results to the Cloud performs best in terms of the average time needed for one application cycle in all considered cases. Considering a harvesting current of 2mA and a capacitor of 1.5F, the local inference strategy is able to execute application cycles 3 times more frequently compared to the remote inference strategy but at some cost of accuracy, as the less accurate tinyML model is used for inferencing.

In the second case, we considered a dynamic harvesting environment based on natural light, where the harvesting current changes over time. Taking into account the real harvesting current measurements, we defined the required deadline before the inference result must be confirmed on a battery-less IoT device. Based on that, our energy-aware optimization algorithm decides which of the two proposed inference strategies is more suitable for certain harvesting conditions, respecting accuracy, energy, and time constraints.

There are two main potentials that we note from this work: (i) a generic system architecture and a mathematical formulation that can be applied to similar use cases, and (ii) sustainability benefit, which comes from the fact that the system depends solely on harvested energy. In this work, we considered an example application where our solution is able to detect whether a person is present or not in the captured image. This was just a proof of concept to evaluate and demonstrate that our approach works. There are other cases where our solution can be integrated, such as remote monitoring applications where battery-less IoT devices are hard to reach and require an extremely long lifetime. A potential field of deployment can be an image-based security system or people-counting applications at events with less opportunities for fixed power sources such as festivals. One of these fields can also be related to the natural sciences where our solution can make the detection and counting of different species much easier through the image recognition model [55]. In the end, by optimizing operations on battery-less IoT devices, we take steps towards a more sustainable IoT. This results in lower ecological and economic impact, bringing us a bit closer to the idea of wider battery-less IoT device usage.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Part of this research was funded by the Flemish FWO SBO S001521N IoBaLeT (Sustainable Internet of batteryless Things) project.

References

[1] S. S. Sabry, N. A. Qarabash, and H. S. Obaid, "The road to the internet of things: a survey," in *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, pp. 290–296, 2019.

[2] A. Sabovic, A. K. Sultania, C. Delgado, L. D. Roeck, and J. Famaey, "An energy-aware task scheduler for energy harvesting battery-less iot devices," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[3] C. Delgado and J. Famaey, "Optimal energy-aware task scheduling for batteryless iot devices," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

[4] A. Singh and P. Ponde, "Home automation: Iot," in *International Conference on Intelligent Emerging Methods of Artificial Intelligence & Cloud Computing* (F. P. García Márquez, ed.), (Cham), pp. 244–252, Springer International Publishing, 2022.

[5] D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, and F. Boavida, "Industrial iot monitoring: Technologies and architecture proposal," *Sensors*, vol. 18, no. 10, 2018.

[6] F. John Dian, R. Vahidnia, and A. Rahmati, "Wearables and the internet of things (iot), applications, opportunities, and challenges: A survey," *IEEE Access*, vol. 8, pp. 69200–69211, 2020.

[7] S. Bose, B. Shen, and M. L. Johnston, "A batteryless motion-adaptive heartbeat detection system-on-chip powered by human body heat," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 11, pp. 2902–2913, 2020.

[8] C. Delgado, J. M. Sanz, C. Blondia, and J. Famaey, "Batteryless lorawan communications using energy harvesting: Modeling and characterization," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2694–2711, 2021.

[9] N. N. Alajlan and D. M. Ibrahim, "Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications," *Micromachines*, vol. 13, no. 6, 2022.

[10] P. Jokic, S. Emery, and L. Benini, "Battery-less face recognition at the extreme edge," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, 2021.

[11] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cérin, and J. Wan, "Energy aware edge computing: A survey," *Comput. Commun.*, vol. 151, p. 556–580, feb 2020.

[12] L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 10 2021.

[13] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[14] K. S. Yildrim, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "Ink: Reactive kernel for tiny batteryless sensors," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, (New York, NY, USA), p. 41–53, Association for Computing Machinery, 2018.

[15] S. Lee, B. Islam, Y. Luo, and S. Nirjon, "Intermittent learning: On-device machine learning on intermittently powered system," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, dec 2019.

[16] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proc. ACM Program. Lang.*, vol. 1, October 2017.

[17] F. Yang, A. S. Thangarajan, G. S. Ramachandran, W. Joosen, and D. Hughes, "Astar: Sustainable energy harvesting for the internet of things through adaptive task scheduling," *ACM Trans. Sen. Netw.*, vol. 18, oct 2021.

[18] M. Karimi, H. Choi, Y. Wang, Y. Xiang, and H. Kim, "Real-time task scheduling on intermittently powered batteryless devices," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13328–13342, 2021.

[19] A. K. Sultania and J. Famaey, "Batteryless bluetooth low energy prototype with energy-aware bidirectional communication powered by ambient light," *IEEE Sensors Journal*, vol. 22, no. 7, pp. 6685–6697, 2022.

[20] Y. Zhao, S. S. Afzal, W. Akbar, O. Rodriguez, F. Mo, D. Boyle, F. Adib, and H. Haddadi, "Towards battery-free machine learning and inference in underwater environments," in *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, HotMobile '22, (New York, NY, USA), p. 29–34, Association for Computing Machinery, 2022.

[21] S. Benninger, M. Magno, A. Gomez, and L. Benini, "Edgeeye: A long-range energy-efficient vision node for long-term edge computing," in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8, 2019.

[22] R. Prasanna, V. S. P J, N. Mohan, V. v, and A. Ramachandra, "Implementation of tiny machine learning models on arduino 33 - ble for gesture and speech recognition," *Xi'an Jianzhu Keji Daxue Xuebao/Journal of Xi'an University of Architecture & Technology*, vol. XIV, pp. 160–169, 07 2022.

[23] A. Saffari, S. Y. Tan, M. Katanbaf, H. Saha, J. R. Smith, and S. Sarkar, "Battery-free camera occupancy detection system," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, EMDL'21, (New York, NY, USA), p. 13–18, Association for Computing Machinery, 2021.

[24] M. Giordano, P. Mayer, and M. Magno, "A battery-free long-range wireless smart camera for face detection," in *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ENSsys '20, (New York, NY, USA), p. 29–35, Association for Computing Machinery, 2020.

[25] M. Giordano and M. Magno, "A battery-free long-range wireless smart camera for face recognition," in *SenSys '21: The 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, November 15 - 17, 2021*, pp. 594–595, ACM, 2021.

[26] N. Semiconductor, "Power profiler kit ii - nordicsemi.com." https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2, 2022.

[27] Keysight, "N6705b dc power analyzer, modular, 600 w, 4 slots [discontinued] | keysight." https://www.keysight.com/be/en/product/N6705B/dc-power-analyzer-modular-600-w-4-slots.html, 2022.

[28] A. Sabovic, C. Delgado, D. Subotic, B. Jooris, E. De Poorter, and J. Famaey, "Energy-aware sensing on battery-less lorawan devices with energy harvesting," *Electronics*, vol. 9, no. 6, 2020.

[29] R. Sanchez-Iborra, "Lpwan and embedded machine learning as enablers for the next generation of wearable devices," *Sensors*, vol. 21, no. 15, 2021.

[30] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable tinyml." https://arxiv.org/abs/2204.00827, 2022.

[31] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network." https://arxiv.org/abs/1503.02531, 2015.

[32] S. Gupta, D. S. Jain, B. Roy, and A. Deb, "A TinyML approach to human activity recognition," *Journal of Physics: Conference Series*, vol. 2273, p. 012025, may 2022.

[33] J. D. De Leon and R. Atienza, "Depth pruning with auxiliary networks for tinyml," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3963–3967, 2022.

[34] TensorFlow, "Tensorflow lite | ml for mobile and edge devices." https://www.tensorflow.org/lite, 2022.

[35] Arduino, "Nano 33 ble | arduino documentation." https://docs.arduino.cc/hardware/nano-33-ble, 2022.

[36] N. Semiconductor, "nrf52840 dk - nordicsemi.com." https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk, 2021.

[37] Arduino, "arduino-libraries/arduinoble: Arduinoble library for arduino." https://github.com/arduino-libraries/ArduinoBLE, 2022.

[38] ArduCam, "Arducam mini 2mp plus - ov2640 spi camera module for arduino uno mega2560 board & raspberry pi pico - arducam." https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/, 2022.

[39] ArduCam, "Arducam/arduino: This is arducam library for arduino boards." https://github.com/ArduCAM/Arduino, 2022.

[40] T. Instruments, "Tps22919evm evaluation board | ti.com." https://www.ti.com/tool/TPS22919EVM, 2022.

[41] e-peas semiconductors, "Aem10941 solar harvesting | photovoltaic energy harvesting | e-peas." https://e-peas.com/product/aem10941/, 2022.

[42] e-peas semiconductors, "Energy harvesting | making devices energy autonomous | e-peas." https://e-peas.com/, 2022.

[43] B. S. working group, "Bluetooth technology overview | bluetooth® technology website." https://www.bluetooth.com/learn-about-bluetooth/tech-overview/, 2022.

[44] H. Blidh, "bleak." https://bleak.readthedocs.io/en/latest/, 2022.

[45] RabbitMQ, "Mqtt plugin — rabbitmq." https://www.rabbitmq.com/, 2023.

[46] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[47] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019.

[48] TensorFlow, "Person detection training." https://github.com/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/examples/person_detection/training_a_model.md, 2022.

[49] TensorFlow, "models/train_image_classifier.py at master·tensorflow/models." https://github.com/tensorflow/models/blob/master/research/slim/train_image_classifier.py, 2022.

[50] J. Fontaine, A. Shahid, B. Van Herbruggen, and E. De Poorter, "Impact of embedded deep learning optimizations for inference in wireless iot use cases," *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 86–91, 2022.

[51] Nomidl, "What is precision, recall, accuracy and f1-score?." https://www.nomidl.com/machine-learning/what-is-precision-recall-accuracy-and-f1-score/, 2022.

[52] Panasonic, "Amorphous silicon solar cells amorphous photosensors." https://panasonic.co.jp/ew/psam/, 2022.

[53] D.-K. Electronics, "Dgh155q5r5, cornell dubilier / illinois capacitor, electric double layer capacitors (edlc), supercapacitors." https://www.digikey.co.uk/en/products/detail/illinois-capacitor/DGH155Q5R5/7387513, 2023.

[54] TensorFlow, "Antwerp, belgium historical weather." https://www.worldweatheronline.com/antwerp-weather-history/be.aspx, 2022.

[55] J. Schoelynck, P. Loon, R. Heirmans, S. Jacobs, and H. Keirsebelik, "Design and testing of a trap removing chinese mitten crabs (eriocheir sinensis , h. milne edwards, 1853) from invaded river systems," *River Research and Applications*, vol. 37, 04 2020.