# Energy-Aware TinyML Model Selection on Zero Energy Devices

Adnan Sabovic[a], Jaron Fontaine[b], Eli De Poorter[b], Jeroen Famaey[a]

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

**Abstract**

Tiny Machine Learning (tinyML) enables the efficient deployment of machine learning models on resource-constrained Internet of Things (IoT) devices. However, in scenarios where energy availability is variable and unpredictable, such as with zero energy devices (ZEDs) reliant on energy harvesters, deployed tinyML models are often compressed to accommodate worst-case energy constraints, leading to diminished accuracy. To address this challenge, we propose a strategy of deploying multiple tinyML models concurrently on ZEDs to maximize model accuracy within the time-varying constraints of memory, execution time, and energy. We introduce a mathematical optimization framework that dynamically selects the most suitable tinyML model for execution based on current and predicted energy availability. We validate our approach through experimental evaluation, where we develop, train, and assess two machine learning models in a Cloud environment before optimizing them into tinyML models of differing sizes and accuracies. Our methodology is tested using a prototype for photovoltaic-powered battery-less gesture detection and recognition, employing a controlled setup with artificial lighting conditions. Results indicate that, under constant harvesting current, the smaller tinyML model exhibits superior execution speed, with two more executions on average, while the larger model yields approximately 28% higher accuracy. As expected, in a more realistic scenario with dynamic harvesting currents and employing our optimization algorithm, the device automatically prioritizes the larger tinyML model for inference when plentiful energy can be harvested due to its improved accuracy, while switching to the smaller model otherwise.

*Email addresses:* sabovic1995@gmail.com (Adnan Sabovic), jaron.fontaine@ugent.be (Jaron Fontaine), eli.depoorter@ugent.be (Eli De Poorter), jeroen.famaey@uantwerpen.be (Jeroen Famaey)

## 1. Introduction

There is a recent trend for on-device processing of data in the Internet of Things (IoT) ecosystem [1]. This trend is driven by advancements in processor and sensor efficiency, allowing edge devices for monitoring the environment to perform both data collection as well as low-power communication and computing on microcontroller units (MCUs). This, in combination with smaller MCU size, low price, and easy maintenance, creates the potential for extreme edge devices in various fields, from healthcare [2] and smart home systems [3] to industrial monitoring [4] and autonomous vehicles [5].

Still, most low-power IoT devices depend on batteries, which provide reliability and a stable power supply. However, at the same time, these batteries are temperature-sensitive and dangerous if not carefully protected. Also, they are short-lived so over time, as the number of devices grows, their maintenance, disposal, and replacement become expensive and ecologically unacceptable. The idea of avoiding batteries and instead harvesting energy from renewable energy sources (e.g., thermal, vibration, solar) can potentially alleviate these problems. Such tiny battery-less IoT devices, often referred to as zero energy devices (ZEDs), use small capacitors instead of batteries that act as the main storage for harvested environmental energy. These capacitors are less sensitive to extreme temperatures, easier to maintain and recycle, and operate longer compared to batteries. This makes them suitable for large-scale deployment and applications in hard-to-reach locations as well as more environmentally friendly solutions [6] [7].

ZEDs have already proven their benefits in different applications, making use of various low-power communication technologies (e.g., NB-IoT, LoRaWAN, Bluetooth Low Energy) to transmit collected sensor data to an Edge or Cloud server for processing [6] [8]. At the same time, these devices are resource-constrained with limited storage and computing capabilities, so performing on-device data analysis and processing is highly challenging. As such, it is of major interest to investigate whether different machine learning (ML) algorithms (e.g., object detection, gesture recognition, and image classification) can run on ZEDs. These ML algorithms are mostly based on neural networks, making the systems capable of analyzing data and making decisions based on received outputs. However, such neural networks often have large computational power and memory requirements, which limits ZEDs to only collect and send data to the Cloud, where demanding ML-based processes are executed [9].

To tackle these limitations, there is a new paradigm called Tiny Machine Learning (tinyML), which allows the successful deployment of ML models on extremely resource-constrained edge devices. The main goal of this concept is to design, develop, and run small ML models on ultra-low-power IoT devices, allowing them to do real-time data analysis and interpretation with minimal energy consumption in the order of a few milliwatts (mW) [10]. By deploying ML models on ZEDs, each of these devices becomes capable of making decisions locally instead of sending the collected data to the Cloud, which unlocks the possibility for direct implementation for a wide range of ML applications [11]

[12]. In addition, other benefits of using tinyML approaches include improved energy efficiency, reduced latency, lower network load, and lower costs, as well as higher system reliability and data security.

Despite recent advances, there are still many challenges and gaps when it comes to deploying tinyML models on ZEDs [6]. Most existing TinyML scientific papers propose the use of a small TinyML neural network with reduced accuracy. However, such a solution does not adapt to changing energy and hence processing capabilities. To remedy this shortcoming, this work focuses on finding a way to maximize the accuracy of executed tinyML models, respecting the memory, energy, and time constraints. To this end, we build, train, and evaluate multiple Convolutional Neural Networks (CNNs) in the Cloud, after which they are converted into multiple smaller tinyML models that can be simultaneously deployed and executed on constrained ZEDs. Our solution automatically selects the model that offers higher accuracy while considering the current energy availability. This is the first paper that studies the energy-aware deployment and management of multiple tinyML models on a single ZED, intending to maximize the total or average accuracy by selecting the most optimal tinyML model for execution, depending on the predicted variable energy availability over time. This work extends and complements our previous work, in which we studied energy-aware computation offloading in tinyML-enabled battery-less ZEDs [6].

The main contributions in this paper are:

(i) A mathematical optimization algorithm to select the optimal subset of available tinyML models to be deployed on the ZED and select the most optimal tinyML model for inferencing, respecting the accuracy, energy, and time constraints. It ensures the deployment of multiple tinyML models of varying size (and therefore accuracy) on a single ZED, taking into account overall device memory constraints. The proposed optimization algorithm is built on the concept of energy-awareness and does not does not assume any specific device and harvesting source, which makes it generic and applicable to different use cases.

(ii) A system architecture to enable energy-aware deployment of an IoT application and multiple tinyML models on a single ZED, with support for task-offloading. Most existing research deals with one or the other solution, where the decision is made locally and only the final result is sent to the Cloud, or the captured data is sent directly to the Cloud for remote inference. The presented system architecture is generic and built on the concept of energy-awareness, which makes it suitable for a wide range of applications and energy harvesting sources.

(iii) The design of a hardware-software prototype of the proposed approach that enables gesture recognition on ZEDs. It enables building, training, and evaluating multiple heavy-weight Cloud-based ML models, after which they are converted and deployed as tinyML models on the constrained ZED.

(iv) Realistic evaluation and validation of our solution based on the device prototype, considering a controllable setup with an artificial light placed at some distance above the solar panel.

The remainder of this paper is structured as follows. Section 2 reviews the state-of-the-art on tinyML model deployment on ZEDs, including works with gesture recognition use cases. In Section 3, the proposed system architecture, along with the energy-aware tinyML optimization algorithm and ML/TinyML workflow, is described. Section 4 presents the used prototype and describes how from the base ML model trained in the Cloud, multiple ML models with different sizes and accuracies are defined, trained, evaluated, and finally converted into multiple tinyML models integrated on ZEDs. Section 5 presents an accurate device and application profiling methodology to determine the current consumption and execution time of different application tasks and device states, with a focus on analyzing different tinyML models deployed on the ZED. Section 6 shows the evaluation and validation results. Further discussion is provided in Section 7. Finally, our conclusions are provided in Section 8.

## 2. Related Work

Recent advances in low-power and energy-aware operations on ZEDs place them as serious competitors to their battery-powered counterparts in various fields, from wireless sensor networks (WSN) to different IoT applications. It is possible to execute different computer programs on these devices. Yet, they are mainly dependent on powerful edge computers or Cloud data centers for more complex processing. The integration of tinyML models and algorithms, together with the wide connectivity of IoT devices, increases the domain of their possible deployment and makes them capable of executing more intelligent tasks [13].

As a representative use case, we consider a battery-free IoT wearable device activated through gesture control. Noble et al. [14] presented a capacitive sensing and neural-network-based hand gesture recognition system with very high accuracy of almost 97%. They used the sensor board only for collecting and sending data to the host PC, where the gesture classification is done. Ma et al. [15] designed a system, SolarGest, capable of recognizing hand gestures near a solar-powered device by analyzing the patterns of photocurrent. Their system architecture includes two main parts: (i) the solar-powered device used for collecting data, and (ii) the gesture recognition ML framework running on an edge device such as a smartphone or laptop, capable of detecting gestures and sending the final decision back to the IoT device. CapBand is a battery-free capacitance-sensing wristband for hand gesture recognition presented in [16]. Their sensing hardware is only used for localization and collecting data. An ML model is built and trained offline by using sensor input data and later converted into a form that can be run on a smartphone platform. In contrast, we focus on deploying ML models on a ZED, which is not only responsible for collecting data but also for making decisions locally.

There are some works where gesture recognition models are deployed and used for inferencing on small IoT devices. A capacitive-sensing wristband surrounded by four single-end electrodes for onboard gesture recognition was presented in [17]. They collected multiple instances of each gesture from a single user, which were later used for training and testing the ML model. The huge ML model was converted into a form that could be stored on the Arduino Nano sense platform using TensorFlow Lite. Reinschmidt et al. [18] proposed a new device based on a combination of a novel electrostatic sensing circuit fused with a six degree of freedom inertial measurement unit (IMU) to develop a wearable bracelet capable of active gesture detection. Gestures are recognized in real-time using the tinyML model deployed on an onboard low-power and energy-efficient microcontroller with an accuracy of up to 87%. Viswanatha et al. [19] presented the implementation and deployment of tinyML models for gesture and speech recognition on the Arduino Nano 33 BLE sense board. Both models were trained based on the collected data and deployed on the Arduino device using the Edge Impulse framework. A static hand gesture recognition system based on an ultra-low resolution infrared array sensor responsible for sensing static hand gestures and generating a set of temperature arrays about them and a low-cost AI chip was designed and presented in [20]. The authors used a more powerful development board, including a display for showing results, compared to the Arduino Nano sense device that is considered in this paper. Their system can achieve accurate (around 99.14%) and real-time recognition for a few simple static hand gestures. All mentioned papers are focused on implementing tinyML models on either USB or battery-powered devices. In contrast, we consider a more resource-constrained ZED that uses a capacitor to store energy collected from its environment.

There are already some works focused on the deployment of tinyML models on small intermittently-powered embedded devices. Zhao et al. [21] designed a battery-less device that can perform local inference on sensed data using a lightweight Deep Neural Network (DNN) and send the result via backscatter to a receiver while harvesting energy from underwater sound. The authors considered a mammal monitoring use case, investigating two critical aspects of battery-less tinyML, the feasibility of hosting the DNN models on resource-constrained battery-less devices, and their accuracy while making decisions. Jokic et al. [22] presented a small battery-less computer vision platform containing an ultra-low power image sensor and an ML system-on-chip capable of recognizing faces on captured images. The self-sustainable operations are achieved by using solar energy harvesting with a small onboard solar cell. Giordano et al. [23] presented a battery-free smart camera system containing a tinyML algorithm for face identification, a power management module with an energy harvester, buck converter, a capacitor, an energy harvesting circuit, and LoRa module for sending the final result of local inference. Their proposed ML algorithm was trained to identify five different persons on the image. The device performs an inference cycle only if the capacitor is fully charged. Otherwise, it waits until the defined voltage threshold, above which charging becomes more efficient, is reached, to start operating. The same authors proposed a similar solution in [24]. They

presented a battery-free smart camera that exploits power management and energy harvesting to achieve face recognition in an energy-neutral fashion. Their battery-less sensor node performed a neural network inference at the edge using a CNN accelerator. Once the local inference task is done, the final result is sent to a gateway via LoRa. All four mentioned solutions considered the deployment of only one tinyML model on the device at a time, making the system incapable of adjusting to varying energy availability. In contrast, our solution supports deploying multiple tinyML models on the same device, to trade off energy cost and accuracy.

Gibbs et al. [25] combined multiple tinyML models for multimodal context-aware stress recognition on constrained microcontrollers. The authors designed an approach that enables the deployment of two DNNs onto a single resource-constrained device, providing real-time contextual activity recognition data to improve the accuracy of stress inference. However, the mentioned work is focused on deploying multiple tinyML models on continuously powered devices without considering the possibility of intermittent energy supply scenarios. Also, each of the considered tinyML models performs specific tasks that should improve the overall system performance by providing real-time recognition information for more reliable final results. In contrast, our solution considers ZEDs that completely depend on the availability of environmental energy sources, making them more unreliable and unpredictable. Also, we consider multiple tinyML models that focus on the same goal but with different capabilities and resource requirements, optimally selecting them for execution based on different constraints. Our goal is to show that these models do not depend on each other to improve system performance, but are there to enable optimal local inference under different conditions. Puslecki et al. [26] presented a modification of the One-Versus-All (OVA) ML concept in a multi-class task of computer vision in tinyML systems. It enables control over classification accuracy, influencing latency and improving the energy efficiency of classification algorithms run on resource-constrained devices. However, the authors did not consider multiple models, but a single model that can be dynamically adapted to change the accuracy-energy consumption. They focused on a single type of classifier, while our work can be applied to any type of tinyML model for different types of tasks (e.g., both classification and regression).

In our previous work [6], we have presented the energy-aware deployment and management of tinyML algorithms and application tasks on battery-less ZEDs. In this work, we studied the trade-offs between different inference strategies, analyzing under which circumstances it is better to make the decision locally or send data to the Cloud where the heavy-weight ML model is deployed, respecting energy, accuracy, and time constraints. In the present work, we further extend this by supporting multiple alternative tinyML models deployed on the device, to provide more degrees of freedom in selecting a suitable inference model based on the available energy. Finally, a brief overview of all the mentioned approaches is given in Table 1.

Table 1: Comparison of state-of-the-art approaches for ML/TinyML deployment on resource-constrained systems

| Prior Work | Battery-Less | ML Support | TinyML Support | TinyML Selection | Task-offloading Support |
|---|---|---|---|---|---|
| Noble et al. [14] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Ma et al. [15] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Truong et al. [16] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Bian et al. [17] | ✗ | ✓ | ✓ | ✗ | ✗ |
| Reinschmidt et al. [18] | ✗ | ✓ | ✓ | ✗ | ✗ |
| Viswanatha et al. [19] | ✗ | ✓ | ✓ | ✗ | ✗ |
| Dai et al. [20] | ✗ | ✓ | ✓ | ✗ | ✗ |
| Zhao et al. [21] | ✓ | ✓ | ✓ | ✗ | ✗ |
| Jokic et al. [22] | ✓ | ✓ | ✓ | ✗ | ✗ |
| Giordano et al. [23] | ✓ | ✓ | ✓ | ✗ | ✗ |
| Giordano et al. [24] | ✓ | ✓ | ✓ | ✗ | ✗ |
| Gibbs et al. [25] | ✗ | ✓ | ✓ | ✓ | ✗ |
| Sabovic et al. [6] | ✓ | ✓ | ✓ | ✗ | ✓ |
| Proposed solution | ✓ | ✓ | ✓ | ✓ | ✓ |

## 3. System Architecture

In this section, a brief overview of the proposed system architecture is provided. We start by introducing notations and input parameters used in the proposed approach. After that, we provide a description of our system architecture used to deploy energy-aware IoT applications and multiple tinyML models on ZEDs. It consists of two main parts: (i) the ZED and (ii) the Cloud, which can be connected via different wireless communication technologies such as LoRaWAN, NB-IoT, or future 6G Ambient IoT technologies specifically optimized for ZEDs. To select the optimal tinyML model for execution, we designed an optimization algorithm, which is described further in this section. Finally, we describe the different stages of the tinyML pipeline, including an explanation of the applied techniques used in each stage.

### *3.1. Preliminaries*

Table 2 lists the set of input parameters used in our system architecture and tinyML optimization algorithm with their definition and set constraints that each of the inputs must satisfy. A list of equal time slots is defined as $\tau$, with the fixed duration $T$ of each time slot. During each time slot $t \in \tau$, an energy harvesting current $I_h[t]$ is provided to the ZED. To execute local inference on the device, there should be at least one model deployed on it. In this work, we consider the deployment of multiple tinyML models on the ZED, which are selected from a set of alternative tinyML models $M$.

As ZEDs are resource-constrained and usually equipped with limited storage capabilities, not all models in $M$ can be deployed. The memory constraint is included so that only a subset of tinyML models, which does not exceed the available memory size, is used. Each tinyML model ($m \in M$) has a memory storage requirement $S_m$. How many tinyML models will be deployed depends on the total memory storage availability $S_{tot}$ of the selected ZED.

Table 2: Set of input parameters of the proposed system architecture and tinyML optimization algorithm

| Parameter | Definition | Constraint |
|---|---|---|
| $\tau = \{t_1, t_2, ..., t_i\}$ | Set of time slots | $T > 0$ |
| $I_h[t]$ | Energy harvesting current in slot $t$ | $I_h[t] > 0$ |
| $M = \{m_1, m_2, ..., m_j\}$ | Set of tinyML models | $M > 0$ |
| $S_m$ | Model size | $S_m > 0$ |
| $S_{tot}$ | Total memory storage | $S_{tot} > 0$ |
| $a_m$ | Model $m$ average classification accuracy | $a_m \in [0, 1]$ |
| $I_m$ | Model $m$ current consumption | $I_m > 0$ |
| $T_m$ | Model $m$ execution time | $0 < T_m \leq T$ |
| $T_{min}$ | Minimum model execution interval | $T_{min} \geq 1$ |
| $E_m$ | Model $m$ required energy | $E_m > 0$ |
| $I_{rem}$ | Remote inference current consumption | $I_{rem} > 0$ |
| $T_{rem}$ | Remote inference execute time | $0 < T_{rem} \leq T$ |
| $E_{rem}$ | Remote inference required energy | $E_{rem} > 0$ |
| $T_{deadline}$ | Deadline for next inference | $T_{deadline} > 0$ |
| $E_{cur}$ | Current available energy | $E_{cur} > 0$ |
| $I_s$ | Sleep current consumption | $0 < I_s < \min_{t \in T} I_h[t]$ |
| $V_{turnoff}$ | Turn-off voltage | $V_{turnoff} > 0$ |
| $V_{max}$ | Maximum voltage | $V_{max} > V_{turnoff}$ |
| $V_0$ | Initial voltage | $V_{turnoff} \leq V_0 \leq V_{max}$ |
| $V_{cap}$ | Current voltage on the capacitor | $V_{cap} > V_{turnoff}$ |
| $V$ | Fixed operating voltage of the device | $V > 0$ |
| $C$ | Capacitance | $C > 0$ |

Each model is characterized by an average classification accuracy, defined as $a_m \in [0, 1]$. The local execution of the tinyML model is also characterized by its current consumption $I_m$ ($I_m > 0$ if the model is executed) and execution time $T_m$ that should not exceed the duration of time slot $T$. These values are important to calculate the required voltage threshold $Vreq_m$ that must be reached before the local inference task is executed. This value can be calculated based on the equation presented in our previous work [27]:

$$Vreq_m = \frac{V_{turnoff} - I_h[t] \times \rho(I_m) \times (1 - e^{(\frac{-T_m}{\rho(I_m) \times C})})}{e^{(\frac{-T_m}{\rho(I_m) \times C})}} \tag{1}$$

where $V_{turnoff}$ is the turn-off voltage below which the device is not able to stay on and will shut down. The chosen capacitor has a capacitance equal to $C$, while the load resistance is modeled as a function $\rho(I_m)$.

As the proposed system architecture supports task-offloading, the captured data can be sent to the Cloud for remote inference. The remote execution of the ML model is also characterized by its current consumption $I_{rem}$ ($I_{rem} > 0$ if the remote inference is executed) and execution time $T_{rem}$ that should not exceed the duration of time slot $T$. These values are used to calculate the required voltage threshold $Vreq_{rem}$ that must be reached before the remote inference task is executed. Finally, this value can be calculated based on the equation presented in our previous work [27]:

$$Vreq_{rem} = \frac{V_{turnoff} - I_h[t] \times \rho(I_{rem}) \times (1 - e^{(\frac{-T_{rem}}{\rho(I_{rem}) \times C})})}{e^{(\frac{-T_{rem}}{\rho(I_{rem}) \times C})}} \tag{2}$$

where the load resistance is modeled as a function $\rho(I_{rem})$

The current available energy is defined as $E_{cur}$, while the energy required for local execution of the tinyML model ($E_m$) or the energy required for remote inference ($E_{rem}$) can be calculated as follows:

$$E = I \times T \times V \tag{3}$$

where $I$ is the current consumption of the selected inference strategy ($I_m$ or $I_{rem}$, $T$ is the execution time of the selected inference strategy ($T_m$ or $T_{rem}$), and $V$ is the fixed operating voltage of the device.

The minimum time, in the number of time slots, that can pass between two inferences is defined as $T_{min}$ and must be at least equal to 1. The ZED spends most of its time in the sleep state, in which it has a current consumption equal to $I_s$. To ensure the device can always harvest energy, it is assumed that $\forall t \in \tau : I_s < I_h[t]$. The maximum voltage of a fully charged capacitor is defined as $V_{max}$ and the currently measured voltage on the capacitor as $V_{cap}$. It is assumed that at the start of the first time slot, the voltage of the capacitor is equal to $V_0$.

### *3.2. Architecture overview*

The proposed system architecture is illustrated in Algorithm 1. It allows the deployment of multiple tinyML models of varying size (and therefore accuracy and inference execution time) on a single ZED, considering overall device memory constraints. If enough energy is available, the device can decide to send the captured data to the Cloud for remote inference. Which of these two inference strategies is most suitable for execution is decided by our optimization algorithm, considering timing, memory, and energy constraints and aiming to maximize the total accuracy of results. Finally, the energy-awareness of our approach is achieved by integrating an energy-aware task scheduler, presented in our previous work [28], on the ZED.

The first task in the flow is $CollectData()$ (Line 2). It collects and processes data based on which the final decision is made. After the data has been collected and processed, the device must decide which of the deployed inference strategies is most suitable for execution. This selection is made by using our optimization algorithm, taking into account the currently available energy ($E_{cur}$), the energy required for both inference strategies ($E_m$ and $E_{rem}$), and the completion time of both inference strategies ($T_m$ and $T_{rem}$). As the Cloud-based ML model can provide more accurate results, the remote inference task is given the highest priority.

The first thing to check is whether the remote inference can be executed before the deadline for the next inference $T_{deadline}$ expires. To calculate the execution time of remote inference $T_{rem}$, we use the mathematical model and equations presented in our previous work [27]:

$$T_{rem}(V_{cur}, Vreq_{rem}, I_{rem}) = \rho(I_{rem}) \times C \times \ln\left(\frac{Vreq_{rem} - I_h \times \rho(I_{rem})}{V_{cur} - I_h \times \rho(I_{rem})}\right) \tag{4}$$

**Algorithm 1:** Accuracy, energy, and time-constrained inference cycle execution algorithm

```
1  while true do
2  |  CollectData();
3  |  E_cur ← CalcAvailableEnergy(V_cap);
4  |  T_rem ← CalcRemoteInferenceExecTime(E_cur, E_rem);
5  |  if T_rem ≤ T_deadline then
6  |  |  execute RemoteInference();
7  |  end
8  |  else
9  |  |  m ← SelectOptimalTinyMLModel();
10 |  |  if T_m ≤ T_deadline then
11 |  |  |  execute LocalInference(m);
12 |  |  end
13 |  end
14 |  Sleep(T_min);
15 end
```

where $V_t$ is replaced with $Vreq_{rem}$, $V_0$ with $V_{cap}$ as the current voltage on the capacitor, and $I_h$ is the estimated harvesting current.

In case the obtained value $T_{rem}$ is lower or equal to $T_{deadline}$, remote inference will be selected and executed (Lines 5-7). Once the remote decision is made, the final result will be sent back to the ZED and confirmed by performing the appropriate action.

If the remote inference solution cannot be executed within the deadline $T_{deadline}$, the device will try to select the most accurate tinyML model $m$ from the set of available tinyML models $M$. This selection is based on the optimization algorithm defined in Section 3.3. The optimal tinyML model $m$ will only be executed if the required amount of energy is collected and execution can be completed within the deadline $T_{deadline}$. To calculate the execution time of local inference $T_m$, we use the mathematical model and equations presented in our previous work [27]:

$$T_m(V_{cur}, Vreq_m, I_m) = \rho(I_m) \times C \times \ln\left(\frac{Vreq_m - I_h \times \rho(I_m)}{V_{cur} - I_h \times \rho(I_m)}\right) \tag{5}$$

In case the obtained value $T_m$ is lower or equal to $T_{deadline}$, the optimal tinyML model $m$ will be selected and executed (Line 11). Once the decision is made, the final result will be confirmed by performing the appropriate action.

The proposed design enables both parts of the system architecture to be connected and capable of processing data and making decisions. However, the Cloud-based ML model requires a lot of resources and more complex architecture to work properly, but at the same time, provides more accurate results compared to the tinyML solutions. On the other hand, for constrained ZEDs, sending captured data to the Cloud can be energy-costly, increasing the possibility of

power failures to occur during the data transmission. Taking this into account, our main focus is on improving the accuracy of tinyML models executed on ZEDs. In the following sections, a detailed description of the proposed tinyML optimization algorithm, along with its assumptions and problem formulation, is given. Also, to get multiple tinyML models that can fit the MCU architecture, several techniques need to be applied. To better understand this process, a brief overview of these techniques and the ML/TinyML workflow is provided. It must be noted that the aspect of tinyML computational offloading to the Cloud is not considered and deeply analyzed in this paper, as it was already studied in our previous work [6].

### *3.3. TinyML model selection algorithm*

In this work, we study the deployment of multiple tinyML models simultaneously on a ZED. Using our tinyML optimization algorithm, the optimal tinyML model, respecting the accuracy, energy, and time constraints, will be selected to analyze the collected data. We leverage the trade-off between energy consumption and accuracy with the main objective of maximizing the accuracy of the data analysis. Besides the accuracy and energy consumption, another important parameter for our study is the total time the device needs to execute the full cycle of the application, which includes: (i) collecting and processing data, (ii) optimal model selection, (iii) local inference by calculating the output of the neural network model, and (iv) confirmation of the final inference result that can be implemented in the form of turning on an LED (i.e., different LED colors for different gestures) or transmitting the result. We also consider the trade-off in terms of latency, which means that this full application cycle must be done before the defined deadline $T_{deadline}$. The algorithm aims to select the most accurate model that satisfies both the energy and deadline constraints.

#### *3.3.1. Assumptions*

We define four general assumptions that are valid for our tinyML optimization algorithm:

**Assumption 1.** *Time is split into equal-sized time slots.*

**Assumption 2.** *Energy harvesting current is assumed to be fixed (constant) during a time slot, but it can vary between time slots.*

**Assumption 3.** *The application cycle is assumed to be executed at most once per time slot.*

**Assumption 4.** *The capacitor voltage must stay above the turn-off threshold, which can be ensured if the harvesting current stays above the current consumption during a sleep state.*

It must be noted that the problem can occur if the harvesting current is lower than the sleep current consumption, causing the device to turn off. However,

it does not necessarily involve additional problems once the device is turned on again. There are two options that can be considered in this case. Storing data into non-volatile memory ensures the device can check and continue its execution from the point where the power failure occurred, but only if defined constrains, such as the task deadline, can still be satisfied. Otherwise, it waits until the required amount of energy is available and starts a new application cycle. In case the approach does not consider usage of non-volatile memory, the device starts from scratch once the required amount of energy is collected. In this case, the problem that occurs is that the device loses application progress as data is lost.

*3.3.2. Problem formulation*

The main objective of our tinyML model selection algorithm is to maximize the total accuracy of the executed inferences. The problem formulation that includes the objective functions, constraints, and decision variables can be defined as follows:

$$\text{(P)} \quad \max \sum_{t \in \tau} \sum_{m \in M} \mu_m[t] \times a_m \tag{6}$$

$$\text{s.t.} \quad \forall t \in \tau : \sum_{m \in M} \mu_m[t] \leq 1 \tag{7}$$

$$\forall t \in \tau : V[t] \geq V_{turnoff} \tag{8}$$

$$\forall t \in \tau : V'[t] \geq V_{turnoff} \tag{9}$$

$$\forall t_j \in [t_1, t_{i-T_{\min}+1}] : \sum_{t \in [t_j, t_{j+T_{\min}}]} \sum_{m \in M} \mu_m[t] \geq 1 \tag{10}$$

$$\forall t \in \tau,\, m \in M : \nu_m \geq \mu_m[t] \tag{11}$$

$$\sum_{m \in M} \nu_m \times S_m \leq S_{tot} \tag{12}$$

where $\mu_m[t]$ is a decision variable, which can have a value of either 1 if tinyML model $m \in M$ is executed in time slot $t \in \tau$ or 0 otherwise ($\mu_m[t] \in \{0, 1\}$). There can be at most one model executed per time slot as shown in Equation 7. To ensure the safer execution of the selected tinyML model and in an energy-aware manner, the capacitor voltage $V[t]$ at the beginning of a time slot and the capacitor voltage $V'[t]$ after executing the selected tinyML model should be above the turn-off voltage $V_{turnoff}$ (cf., Equation 8, 9). Based on Equation 10, we need to execute the tinyML model at least once every $T_{min}$ time slot. For the fifth constraint (cf., Equation 11), we introduce the helper variable $\nu_m = \{0, 1\}$ that equals 1 if a tinyML model $m \in M$ is used at least once. Finally, the total memory storage required by all tinyML models that are used at least once ($\nu_m \times S_m$), should not exceed the available memory storage $S_{tot}$ on the ZED (cf., Equation 12).

In order to ensure the capacitor voltage $V[t]$ at the beginning of a time slot stays above the turn-off voltage $V_{turnoff}$, we first need to calculate this value

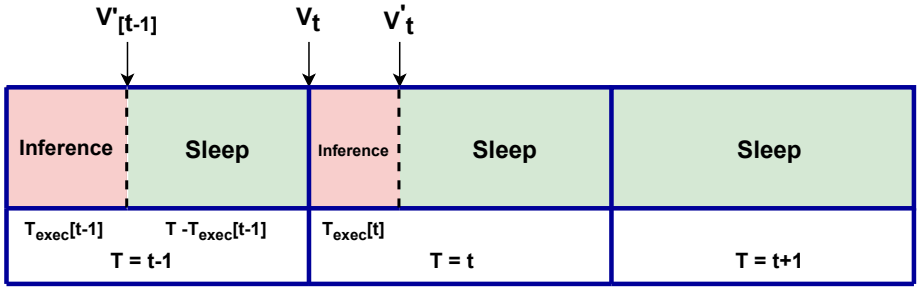



Figure 1: Time slots structure with a constant duration of $T$, including the inference and sleep parts of the ZED.

at the start of the time slot $t$:

$$\forall t \in \tau : T_{exec}[t] = \sum_{m \in M} \mu_m[t] \times T_m \tag{13}$$

$$V[0] = 0 \tag{14}$$

where $T_{exec}[t]$ is the execution time of the full inference part of the time slot $t$ that is described in more detail in Section 3.3. $V_t$ can be defined as the starting voltage of time slot $t$.

$$\forall t \in \tau \setminus \{t_1\} : V[t] = \min \left( I_h[t-1] \times \rho(I_s) \times \left(1 - \exp\left(\frac{T - T_{\text{exec}}[t-1]}{\rho(I_s) \times C}\right)\right) + V'[t-1] \times \exp\left(\frac{T - T_{\text{exec}}[t-1]}{\rho(I_s) \times C}\right), V_{\max} \right) \tag{15}$$

The ZED finishes the execution of the full inference cycle ($T_{exec}[t-1]$) at the voltage $V'[t-1]$, after which it goes into a sleep state for $T - T_{exec}[t-1]$, consuming $I_s$. The way of calculating the voltage after the execution of the full inference cycle is described below. In case no model is selected for execution during a time slot $t$, $T_{exec}[t] = 0$, and the device will spend the whole time slot in sleep mode (i.e., $T - T_{exec}[t] = T$). The load resistance during the sleep state is modeled as a function $\rho(I_s)$ [27]. The structure of time slots, considering the inference and sleep parts, can be seen in Figure 1.

The capacitor voltage $V'[t]$ after executing the model during the time slot $t$ can be calculated as follows:

$$\forall t \in \tau : I_{exec}[t] = \sum_{m \in M} \mu_m[t] \times I_m \tag{16}$$

where $I_{exec}[t]$ is the total current consumption of the full inference part of the

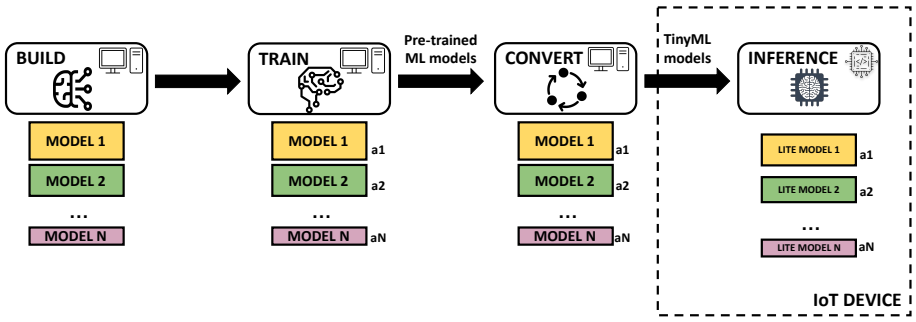



Figure 2: Neural networks with different architectures are built and trained based on prior datasets. Trained neural networks, with different accuracy, are converted into a form that can be run locally on constrained IoT devices.

time slot $t$.

$$\forall t \in \tau \setminus \{t_1\} : V'[t] = \min \Bigg( I_h[t] \times \rho(I_{exec}[t]) \times \Bigg( 1 - \exp\Big( \frac{T_{\text{exec}}[t]}{\rho(I_{exec}[t]) \times C} \Big) \Bigg) + V[t] \times \exp\Big( \frac{T_{\text{exec}}[t]}{\rho(I_{exec}[t]) \times C} \Big), V_{\max} \Bigg) \tag{17}$$

We assume that the harvesting current $I_h[t]$ during the time slot $t$ is constant. The load resistance during the inference state is modeled as function $\rho(I_{exec}[t])$ [27].

### *3.4. TinyML model creation*

To prepare and deploy multiple tinyML models on a single ZED, several phases are required, as shown in Figure 2. The first phase is to build the base CNN model in the Cloud. The base model architecture can be either experimentally derived, considering similar CNN architecture for particular use case, or this part of selection can be automatized by using the Network Architecture Search (NAS) technique [29]. Instead of manually designing the model architecture, which can be a difficult and time-consuming task, NAS uses efficient algorithms to automatically discover the optimal model architecture for a specific use case [30].

Once the base CNN model is built, several heavy-weight ML models with different sizes, accuracy, and complexity can be derived from it. This can be done by defining the width multiplier presented first in [31] that serves as a hyperparameter. The role of this parameter is to thin a network uniformly at each layer, defining a new smaller model with a reasonable accuracy and size trade-off. Smaller values of the width parameter result in smaller and less complex ML models, while larger values lead to more accurate and complex models. Also, this approach impacts the complexity at inference time, which

means that the faster the model executes, the less power will be consumed, assuming that 100% of processor time is being used.

After different ML models are built, the next stage is to train, test, and evaluate them. Training can be executed on the Cloud considering large datasets or even on the edge device but with limited capacity and performance [30]. In our case, we train and test different ML models on a server using pre-collected training and testing data, and TensorFlow ML framework. However, we keep one part of the data for validation and evaluation purposes in order to check how our models perform on unseen data and assesses their general performance.

The third stage is to convert trained ML models into tinyML versions that inherit all the properties of the original ones, but with reduced computational and memory overhead, and thus accuracy, so they can be run and executed on constrained edge devices. There are several techniques to optimize and convert the ML model into a form that can be run on an embedded device, such as knowledge distillation [1] [32], network pruning [33] [34], and quantization [11]. In our case, we apply the quantization technique to reduce the precision and size of ML models, making them more energy-efficient and speeding up on-device inference. The weight and activations of the ML model are typically represented as 32 or 64-bit float values. By converting and scaling down these values, except in input and output layers, to an 8-bit integer (int8) format, the precision of the ML model is reduced to fit the MCU architecture. The major benefits we obtain from quantization are minimal impact on model accuracy, reduced memory overhead and computational complexity, and improved energy efficiency [3].

Once ML models are optimized, we convert them into lite models using the TensorFlow Lite interpreter [35]. These models are evaluated on a server considering the same datasets used in the second stage. If the obtained results meet the requirements, the tinyML models are converted to C++ files and then deployed and compiled on our embedded IoT device, and as such are further used for local inference. Otherwise, the process can be repeated by designing a new model architecture or retraining existing large ML models.

## 4. Prototype Implementation: Gesture Recognition

As a proof of concept for our approach, this section describes our prototype of an energy-aware IoT application implemented on a ZED. A brief overview of used hardware components, which includes a microcontroller and power management unit (PMU) subsystems, is given. In the end, we explain how multiple heavy-weight Cloud-based ML models for gesture recognition are built, trained, evaluated, and finally converted and deployed as tinyML models on the constrained ZED

### *4.1. Energy-aware IoT application*

For evaluation purposes, we developed an IoT application that allows the deployment, selection, and execution of multiple tinyML gesture recognition

models running on the ZED in an energy-aware manner. The energy-aware application was deployed based on the proposed system architecture (cf., Algorithm 1). It is composed of six main tasks. The first task in the flow is to collect gesture data using the integrated inertial measurement unit (IMU) sensors. It is a periodic task and can be repeated every X seconds if enough energy is available. Once gesture data is captured, it is processed and normalized to fit as an input to our tinyML models deployed on the device. In our prototype, we consider two tinyML models to be deployed and run on the ZED (cf., Section 4.4). Using our tinyML optimization algorithm (cf., Section 3.3), the device will select the optimal tinyML model at that moment in time, respecting energy, time, and accuracy constraints. If both of the deployed tinyML models satisfy all defined constraints, the more accurate one will be selected for execution. The model selection will be confirmed by briefly turning on the appropriate LED (i.e., green LED for Model 1, red LED for Model 2). Once the decision is made, the inference result will be confirmed by performing the defined action, which in our case is briefly turning on the appropriate LED again (i.e., green LED for Gesture 1, red LED for Gesture 2, blue LED for Gesture 3). In case none of the available tinyML models can meet the set constraints (i.e., can be completed within the defined deadline), the scheduler will remove the processed gesture data, and select the first task in the flow to be repeated. In this way, data freshness is ensured as the device will not work with expired data.

### *4.2. Microcontroller subsystem*

The microcontroller subsystem consists of two main parts: (i) the microcontroller unit, and (ii) the voltage divider with resistors that is connected to our MCU via ADC and GPIO pins, as shown in Figure 3. The Arduino Nano 33 BLE Sense [36] was chosen, due to its support for different environmental sensors and the possibility to execute tinyML models using the TensorFlow Lite Micro library [37]. With the 1MB CPU Flash and 256kB RAM, it allows us to deploy, run, and execute multiple tinyML models simultaneously. The board is equipped with an LSM9DS1 9-axis IMU [38] used for detecting board orientation, motion, acceleration, and vibrations. In this way, gesture data can be collected and later used as input to our gesture recognition tinyML models.

Finally, to enable our Arduino Nano 33 BLE Sense board to measure and read the voltage on the capacitor and compare it to the calculated voltage thresholds of each application task, an additional voltage divider was added. This will increase the current consumption as the voltage measurement circuit contains additional resistors. To reduce it, MOSFETS that act as circuit switches are used. In this way, the harvested energy can be used better by determining the usable energy capacity stored in the capacitor and the ZED can be modified to act in an energy-aware fashion.

### *4.3. Intelligent power management based on the AEM10941 chip*

In our prototype, we consider a real energy-harvesting environment with solar panels that harvest ambient light energy. Based on that, there is a need

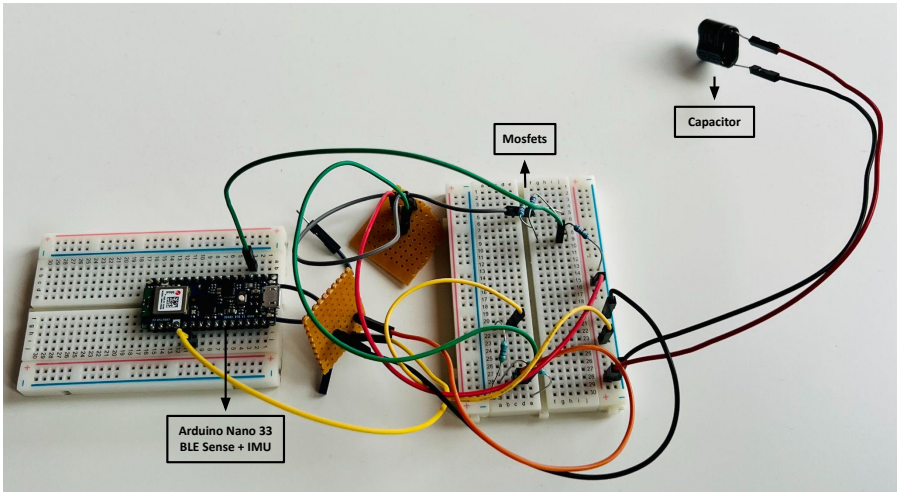
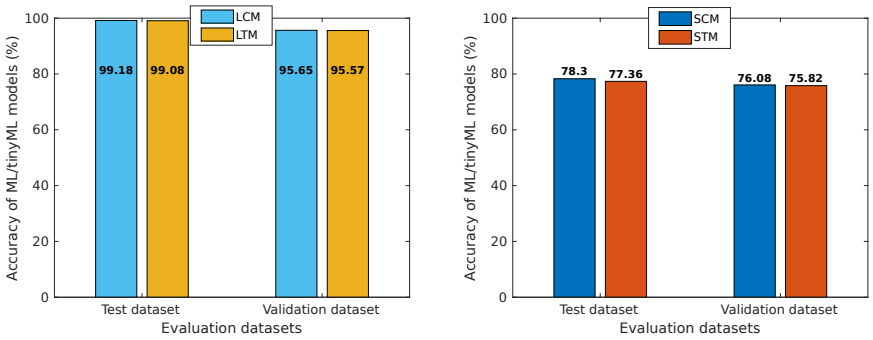



Figure 3: Microcontroller subsystem consisting of two main parts, the microcontroller unit with the integrated IMU on which the energy-aware IoT application and tinyML algorithms are running, and the voltage divider with resistors for reading capacitor voltage.

to manage this incoming energy by using the appropriate PMU. The power management is performed using the AEM10941 [39], an integrated energy management circuit designed by e-peas [40], for solar and thermal harvesters, which extracts DC power from up to 7-cell solar panels to simultaneously store energy in a rechargeable element (e.g., capacitor), after which it is converted to a stable voltage to operate the ZED. It can provide a voltage supply from 1.2V to 4.1V supply with a maximum load current of up to 80mA. Also, the e-peas evaluation board can support input voltage in the range from 50mV to 5V, starting harvesting energy at 380mV with an input power of only 3$\mu$A.

### *4.4. Gesture recognition models*

In this work, we developed, deployed, and evaluated multiple energy-aware IoT neural networks able to detect and recognize different gestures captured on the ZED. Gesture detection and recognition are achieved by deploying one or multiple tiny CNNs on the constrained ZED. These tinyML models were obtained from multiple heavy-weight ML models built and trained in the Cloud. To train these models, a dataset containing gesture data that belongs to three classes, hand-up, hand-forward, and hand-circular, was used. Gesture data was collected from 10 different persons and each person collected between 20 and 40 samples of each gesture. First, we defined an overall model architecture that was experimentally derived based on similar CNN architecture for activity recognition shown in [11]. It contains 12 layers, 3 2D convolutional layers, 2 max-pooling 2D layers, 3 dropout layers, 1 flattening layer, and 3 fully connected dense layers. The proposed architecture uses ReLu activation functions for all the layers, except the last dense layer which has a Softmax activation function.




(a) LCM/LTM evaluation

(b) SCM/STM evaluation

Figure 4: Evaluation of Large and Small Cloud-based (LCM and SCM) and Large and Small tinyML (LTM and STM) models considering training and validation datasets

The number of layers is the same for all considered heavy-weight ML models. To vary the performance (inference time and complexity) and accuracy of these models, we used a width parameter-based optimization technique, described in Section 3.4. In this way, we were able to scale the number of filters, size of layers, and total number of trainable parameters in the defined CNN models.

We started with the oversized heavy-weight ML model running in the Cloud, using a width parameter of 1.5, which contains almost 900k parameters and requires 10.5MB of memory. This is way too large to be converted and deployed on constrained ZEDs. Based on that, from this base model, we extracted two smaller Cloud-based ML models, Large Cloud ML (LCM) and Small Cloud ML (SCM), considering a width parameter of 0.7 and 0.1, respectively. In this way, we reduced the number of parameters to 96859 and 319, and the memory requirements to 1.2MB and 62.4kB, for the LCM and the SCM, respectively. These two neural networks were trained using the TensorFlow framework, through 200 epochs with a batch size of 16 and a learning rate of 0.01. Adam was chosen as optimizer as well as categorical cross-entropy as loss function. Early-stopping [41] was used to prevent model overfitting and to save time and resources. The learning rate scheduler [42] was also used to reduce the learning rate during training if the validation loss stops improving for a certain number of epochs. Reducing the learning rate, the optimizer takes smaller steps and potentially finds a better local loss minimum.

The aforementioned dataset was divided into two additional subsets: (i) gesture data from 9 persons split into training (75%) and testing (25%) datasets, and (ii) remaining gesture data from 1 person used for additional validation of our models. During the training phase, the performance of both heavy-weight ML models, in terms of accuracy and loss, was evaluated based on the defined training and testing data. Once the training was done, the results from the TensorFlow training environment were converted into a form suitable for tiny ZEDs. Both Cloud-based ML models were converted and generated into TensorFlow

Lite files [35] using the int8 quantization technique described in Section 3.4. This reduced the memory requirements to 104.8kB and 6.1kB for the Large tinyML model (LTM) and the Small tinyML model (STM), respectively.

The final stage was to evaluate both the Cloud-based ML models and the tinyML models extracted from them in terms of accuracy and based on defined test and validation datasets as shown in Figure 4. The LCM showed the best performance, achieving more than 20% higher accuracy on average compared to its counterpart, the SCM. Due to the quantization process, the precision of models is reduced, but still with minimal impact on the accuracy of optimized tinyML models (0.1% for the LTM and 0.6% for the STM on average, considering both evaluation cases). As a more complex Cloud-based ML model results in a more complex tinyML model, the LTM showed more accurate results compared to its counterpart, the STM.

## 5. Device and Application Profiling

To perform accurate experiments and energy-aware task scheduling, the current consumption ($I_{state}$) and execution time ($t_{state}$) of the different states of the device, such as collecting and processing gesture data, selecting the optimal tinyML model and performing local inference, or confirming inference results, 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. Taking into account these two values, along with the voltage at which they are measured, the energy consumption ($E_{state}$) of each of the considered states of the device can be calculated. The real experiments, measurements, and validation of our approach were performed using the Arduino Nano 33 BLE Sense board (cf., Section 4.2) on which an energy-aware IoT application for gesture detection and recognition was implemented. The current consumption and execution time of the different states of the Arduino board were obtained with a Nordic Power Profiler Kit II [43], a standalone unit that can measure the current levels of different hardware, providing a voltage supply between 1.6V and 5.5V [6].

To decrease the total current consumption and enable our energy-aware IoT application to run battery-less, the Arduino Nano 33 BLE Sense board must operate in low-power mode for most of the time. In order to achieve this, the step-down voltage regulator is disconnected, the power LED, sensors, and I2C pull-up resistor are only activated when acceleration data is collected, and the UART port is disabled.

Table 3 shows the average current consumption and execution time of different states of the Arduino Nano 33 BLE Sense board measured at 3.3V. The same voltage is used with the e-peas PMU to run our device. The average values $\pm$ standard deviation (AVG $\pm$ STD) and maximum values (Worst Cases) for both, current consumption and execution time, were obtained based on 15 repeated measurements. In our experiments, we considered the worst-case current consumption and execution time values for the energy-aware scheduler to ensure that the device would not turn off after task execution due to unexpected energy

Table 3: Current consumption and execution time for different states of the Arduino Nano 33 BLE Sense board

| State | AVG ± STD | | Worst Case | |
|---|---|---|---|---|
| | Current draw | Execution time | Current draw | Execution time |
| Collect/Process gesture data | 6.6±0.24 mA | 2073.19±470.84 ms | 7.04 mA | 3045 ms |
| Local inference (LTM) | 4.22±0.05 mA | 98.89±1.48 ms | 4.26 mA | 103.8 ms |
| Local inference (STM) | 3.52±0.06 mA | 9.39±0.21 ms | 3.61 mA | 9.86 ms |
| Model selection & Confirmation | 2.12±0.01 mA | 497.21±1.69 ms | 2.23 mA | 500.07 ms |
| Confirm results | 1.46±0.02 mA | 501.91±1.25 ms | 1.48 mA | 504.8 ms |
| Voltage check | 0.44±0.03 mA | 3.35±0.34 ms | 0.51 mA | 3.88 ms |
| Sleep | - | - | 0.92 mA | - |

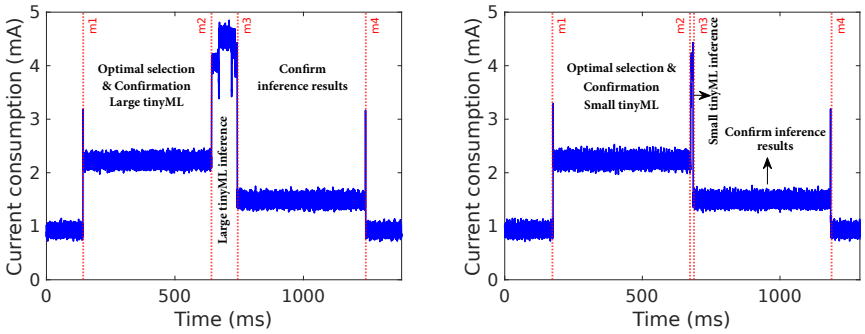
Table 4: Energy consumption of the Arduino Nano 33 BLE Sense board (Worst Case)

| State | Energy Consumption (mJ) | Overhead percentage (%) |
|---|---|---|
| Collect/Process gesture data | 70.74 | 90.14 |
| Local inference (LTM) | 1.46 | 1.86 |
| Local inference (STM) | 0.12 | 0.15 |
| Model selection & Confirmation | 3.68 | 4.7 |
| Confirm results | 2.47 | 3.15 |
| Voltage check | 0.0066 | 0.0084 |
| Total sum | 78.4766 | - |

consumption peaks. Finally, Table 4 shows the worst-case energy consumption of different states of the Arduino Nano 33 BLE Sense board, calculated using the values presented in Table 3 and the voltage at which they are obtained.

The highest energy cost task is collecting and processing gesture data as it requires the use of IMU sensors to detect movement and read acceleration data. In our work, we consider two tinyML models for performing the inference task. Based on the obtained results, the STM executes more than 10 times faster than the LTM consuming less energy. To select and confirm the optimal tinyML model for execution at a specific time, the device consumes around 3.68mJ. Each inference result is confirmed by briefly turning on the appropriate LED, which consumes around 2.47mJ.

Using the tinyML optimization algorithm presented in Section 3.3, the device can select and execute one of two possible inference paths, which is optimal at a specific time: (i) a large tinyML inference (LTI) path (cf., Figure 5a) that starts with the selection and confirmation of the LTM by briefly turning on the appropriate LED (i.e., red LED for LTM), after which the inference task is executed and results confirmed, and (ii) a small tinyML inference (STI) path




(a) LTI path

(b) STI path

Figure 5: Measured current consumption and execution time for tinyML inference paths considering the execution of the model selection algorithm (m1-m2), inference using the selected model (m2-m3), and result confirmation (m3-m4)

(cf., Figure 5b) that starts with the selection and confirmation of the STM (i.e., green LED for STM), after which the STM inference task is performed and results confirmed. Based on that, we measured the current consumption and execution time of these two inference paths, where the considered tasks are executed one after the other, without additional energy checks. The LTI path requires more time for execution compared to the STI path, drawing a bit more energy.

## 6. Results

In this section, the results and validation of our hardware-software prototype described in Section 4, which enables gesture detection and recognition on ZEDs, are presented. Based on the defined energy-aware IoT application and manually configured e-peas power management board, we defined and performed different experiments, considering different parameters and the real energy-harvesting environment with a solar panel, to validate our proposed solution. For the experiments, we considered two main approaches: (i) constant harvesting current and voltage during the full time of the experiment, and (ii) non-constant harvesting current that varies throughout the experiment.

To perform experiments for both approaches, we designed the experimental setup with the photovoltaic-powered battery-less IoT device. Our setup considers controlled lighting conditions with an artificial light placed at some distance above the solar panel as shown in Figure 6. This is done to create controllable and repeatable experiments, which simplifies the analysis of results and comparison of different approaches. A Philips Hue White A21 smart bulb [44] controlled via a Bluetooth application was used as a light source. It is attached inside a plastic dark box offering a powerful 1600-lumen output. A Panasonic AM-5608 [45] solar panel that consists of 6 amorphous silicon solar cells was used. The e-peas PMU was added to: (i) extract the maximum power from

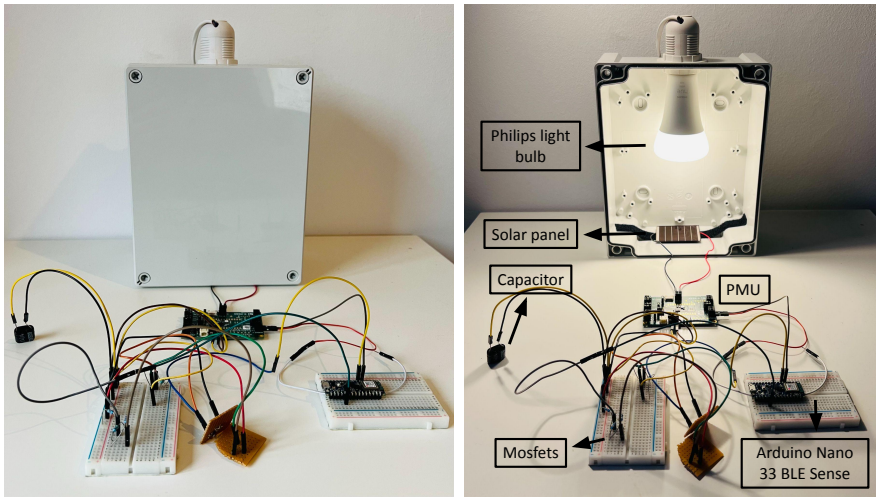



Figure 6: Controllable experimental setup with artificial light source

the solar panel, (ii) charge the capacitor, and (iii) regulate the output voltage to our ZED. Finally, the Arduino Nano 33 BLE Sense board was connected to the e-peas board via the appropriate power supply (e.g., 3V3) and ground (e.g., GND) pins, measuring the capacitor voltage using the voltage divider presented in Section 4.2.

### *6.1. Constant harvesting current*

In the first approach, the known harvesting current for calculating the required voltage thresholds of the application tasks was considered. We assumed that the value of the harvesting current was perfectly determined and would not change for the full time of each experiment. This value is defined before the experiment starts, which in reality is not the case as this knowledge is not perfect based on the unpredictability and dynamism of both natural and artificial light sources, causing the harvesting current to change frequently. Even if not realistic, we performed these experiments to get results under controllable conditions that are easier to analyze and interpret.

Table 5 lists the general parameters used in our experimental setup when the constant harvesting current is used. The used capacitor has a voltage threshold $V_{max}$ equal to 4.5V. The device will turn on when the voltage threshold, $V_{turnon}$, of 3.92V is reached. The turn-off voltage $V_{turnoff}$, below which the device cannot operate is set to 3.6V. During the operating stage, the e-peas PMU provides our ZED with the supply voltage $V_{output}$ of 3.3V. A capacitor of 0.5F was tested in combination with three different harvesting currents (i.e., 2mA, 4mA, and 6mA). Finally, the experimental run lasted 5 minutes for all experiments, considering and testing three different gesture intervals (i.e., 1, 5, and 10 seconds). It must be noted that the device attempts to collect gesture

Table 5: Experimental setup for the constant harvesting current approach

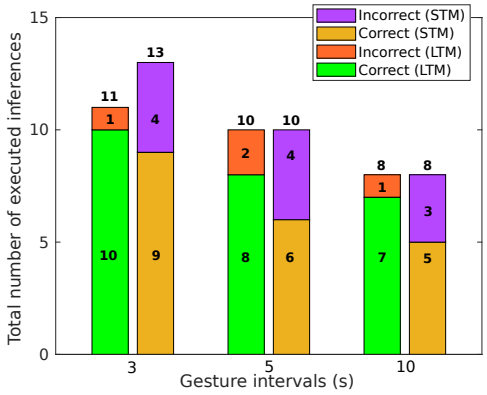
| Parameter | Symbol | Value |
|---|---|---|
| Max Voltage | $V_{max}$ | 4.5 V |
| Turn-on Voltage | $V_{turnon}$ | 3.92 V |
| Turn-off Voltage | $V_{turnoff}$ | 3.6 V |
| Supply voltage | $V_{output}$ | 3.3 V |
| Capacitance | C | 0.5 F |
| Harvesting Current | $I_h$ | {2, 4, 6} mA |
| Experiment duration | $T_{exp}$ | 300 s |
| Gesture capture periodicity | $t_{collect}$ | {3, 5, 10} s |

data at this predefined interval, but only if enough energy is available. Otherwise, it will go into a sleep state and wait for the interval to expire again to try to collect new data.

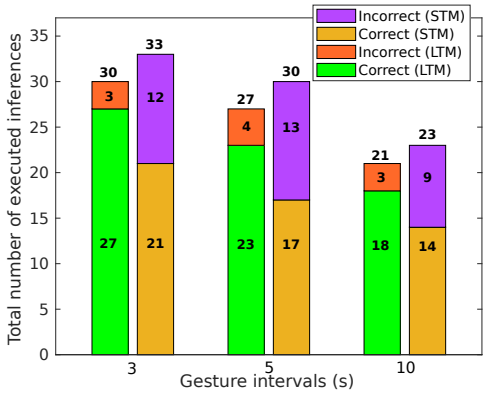
For this approach, we considered two tinyML models (LTM and STM) presented in Section 4.4. Each of these tinyML models was deployed and tested on the ZED separately. To fairly compare them, the experimental setups with the same configuration parameters (i.e., capacitor size, gesture capture periodicity, harvesting current, etc.) were considered in both cases. The main goal of these experiments was to analyze the trade-off between the accuracy of local inference results and the energy consumption of the device when different versions of tinyML models are deployed and executed, taking into account the impact of different intervals between gestures. In our experiments, we have followed the behavior of the ZED considering both tinyML models in terms of the total number of executed inferences, including the accuracy of the final results (cf., Figure 7).

Considering a lower harvesting current, e.g., 2mA, the device needs to sleep longer to reach the required voltage thresholds for the application tasks, especially to collect and process gesture data, for both considered tinyML model approaches. This behavior results in longer application cycles, which directly affects the total number of executed inferences (cf., Figure 7a). As the harvesting current increases, the required voltage thresholds for each of the defined application tasks are lower, enabling the device to reach them faster. As a result, the device can execute a higher number of inferences for both considered tinyML models (cf., Figure 7b and 7c). Taking into account all considered cases, the STM shows better performance in terms of the number of executed inferences (2 more inferences on average) due to lower energy requirements and faster execution.

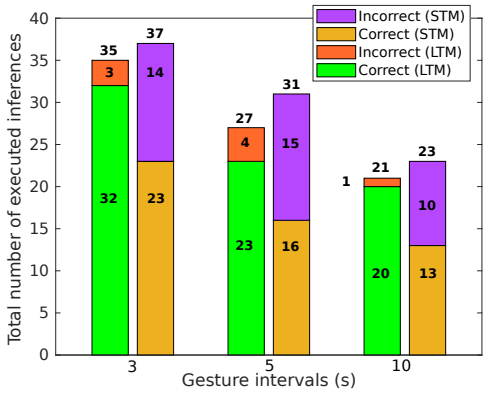
The gesture interval is an important factor that determines the performance of both tinyML models. The highest number of local inference executions for both tinyML models is with a gesture interval of 3 seconds. The device tries to collect gesture data more frequently, which means there are fewer situations when the device will miss a gesture and have to wait for the predefined 3-second interval to expire again. By increasing the gesture interval (e.g., 5 and 10 seconds), the device can collect enough energy before the predefined interval




(a) $I_h = 2\text{mA}$




(b) $I_h = 4\text{mA}$




(c) $I_h = 6\text{mA}$

Figure 7: Total number of executed local inferences for both tinyML models taking into account correct and incorrect predictions and considering different harvesting currents and gesture capture intervals

Table 6: Experimental setup for the non-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 F |
| Harvesting Current | $I_h$ | {2, 3, 4, 5, 6} mA |
| Experiment duration | $T_{exp}$ | 600 s |
| Gesture capture periodicity | $t_{collect}$ | 3 s |

expires but is not allowed to start collecting new data before that expiration. In this case, the device wastes time waiting, which directly affects the total number of executed inferences (4 and 8 for the LTM and 4 and 10 for the STM fewer inferences on average with gesture intervals of 5 and 10 seconds, respectively).

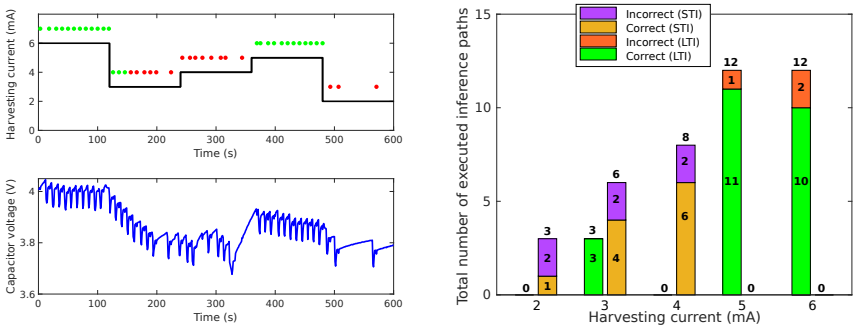
Finally, we compared our tinyML models in terms of accuracy. In all considered cases, the LTM shows more accurate results. When a lower harvesting current, e.g., 2mA, is used, the LTM shows more than 22% higher accuracy on average compared to the STM. By increasing the harvesting current, the difference between results in terms of accuracy is even higher, as the device can execute inferences more frequently, which opens a space for more wrong predictions when the STM is used (26.56% and 33.86% higher accuracy of the LTM on average for 4mA and 6mA harvesting currents, respectively).

### *6.2. Non-constant harvesting current*

In the second approach, the harvesting current was varied over time. For this approach, the same setup shown in Figure 6 was used, considering the general parameters listed in Table 6. For the e-peas PMU, the same configuration as in Section 6.1 is used, except the experimental run lasted 10 minutes, where we switched the harvesting currents $I_h$ every 2 minutes, considering the gesture interval of 3 seconds.

For this approach, we did not test two tinyML models separately. In contrast, both of our tinyML models are deployed together, enabling two possible inference paths (LTI and STI) to be selected and executed (cf., Section 5). Based on the considered harvesting currents, the deadline before which the inference result must be confirmed on the device can be defined. The worst-case scenario ($I_h = 0$) is considered for calculating the required voltage thresholds for application tasks. Taking into account the measured time values for tinyML inference paths (cf., Figure 5) and calculating the required time the device needs to wait until the required voltage threshold is reached, based on Equation 15 (cf., Section 3.3.2), our tinyML optimization algorithm will select and execute optimal tinyML inference path, respecting accuracy, energy, and time constraints.

Figure 8 shows the capacitor voltage changes during the task execution and the total number of executed tinyML inference paths (LTI and STI). To distinguish when each considered inference path was selected and executed, the LTI




(a) Harvesting current and capacitor voltage

(b) Number of executed inference paths

Figure 8: Capacitor voltage behavior (C = 0.5F) over time when executing different tinyML inference paths (LTI path as green dots and STI path as red dots) and the total number of executed inference paths considering correct and incorrect predictions and different harvesting currents during the experimental time.

path is presented as green dots and the STI path is presented as red dots as shown in Figure 8a. We started with the highest harvesting current of 6mA, during which the capacitor maintained the operating voltage on a high level (around 4V), allowing the device to collect more than enough energy to perform all application tasks (cf., Figure 8a). In this case, both tinyML inference paths could be done within the deadline, but due to the higher accuracy, the LTI path was selected and executed (cf., Figure 8b). Similar behavior can be observed when the harvesting current of 5mA is used. In both cases, the higher current consumption is traded off for more accurate results (around 87% correct predictions on average). Considering the lower harvesting currents such as 2mA, 3mA, and 4mA, the device starts to lose energy faster, which prolongs the waiting time until the required voltage thresholds are reached and affects the number of executed inferences (cf., Figure 8b). During these periods, only the STI path could be completed within the deadline, and based on that it was selected and executed. Finally, this resulted in 33.34% less accurate predictions on average compared to periods when the LTI path was used.

## 7. Discussion

In this work, a mathematical framework that dynamically selects the optimal tinyML model for execution, aiming to maximize the accuracy of the final results, is presented. This article shows that multiple tinyML models can be deployed on a single ZED, without the need to build and use the additional Cloud infrastructure. Depending on the predicted variable energy availability over time, ZEDs can select the most suitable tinyML model for execution, respecting energy, accuracy, and time constraints. However, there are still some challenges that should be discussed and considered in future research:

(i) **Scalability of the proposed solution for large-scale IoT deployments** - ZEDs are typically low-cost and low-power devices, which makes them well-suitable for large-scale deployments. They show their benefits in some hard-to-reach areas as they do not require additional power maintenance due to their dependence on environmental energy sources. Deploying multiple of these devices in the field, reduces the need for centralized processing, minimizing latency and bandwidth usage for sending data to the Cloud. However, to enable large-scale deployment there are multiple challenges. First, ZEDs are resource-constrained devices, with limited computational power and storage capabilities. This automatically affects the size and complexity of deployed tinyML models, which in the end could restrict the accuracy of these models as well as increase the need for dynamic model updates. The same challenge can occur in situations when different versions of ZEDs are deployed. In this case, it is important to ensure that models are trained considering data collected from different devices. Also, two interesting techniques can provide a promising solution for this issue. Over-the-Air TinyML (OTA-TinyML) [46] is the technique to remotely update, configure, and execute tinyML models on IoT devices. This allows models to be updated on ZEDs and the deployment of fully trained models on new devices installed on the field. The second technique is TinyML with Online-Learning (TinyOL) [47], which enables incremental on-device training based on new streaming data. In this way, different ZEDs can continue their training based on collected data. Finally, the future direction will be oriented to reducing the network overhead as well. In the proposed approach, most of the intelligence is on-device, and in that way, it does not affect the larger network. However, in large-scale deployment, there is a possibility of communication between devices as well as between a single ZED and Cloud, so based on that it is important to ensure the optimal selection of used network architecture and protocols.

(ii) **Memory constraints** - One of the main characteristics of ZEDs is their storage limitations. This parameter dictates the type, size, and number of small neural networks that could be deployed on a single ZED. In our previous work [6], we considered a person detection use case and deployed a larger model that almost completely occupied the available storage of the device. In contrast, the present article aims to show the possibility of deploying multiple tinyML models simultaneously on a ZED and optimally selecting the most suitable one. Based on that, we decided to go with more lightweight ML models and present a gesture detection and recognition use case. In this way, multiple tinyML models can be deployed on a single ZED. In the future, ZEDs with more storage capabilities could simplify the process of tinyML deployment and enable even more complex scenarios. Also, the other promising solution is called Split Learning [48], where just a few parts of neural networks crucial for data processing and making decisions are deployed, and in that way are smaller and more suitable for a larger deployment on constrained ZEDs. This approach will be discussed

in our future research.

(iii) **TinyML model selection algorithm** - In this work, a formal description of the problem and mathematical framework for how to optimally select the most suitable tinyML model among all deployed models on a single ZED, aiming to maximize the total accuracy of inference results, are provided. However, the proposed approach adds some additional computational overhead, which is shown in Section 5. Compared to our previous work [6], where we considered two inference strategies and task-offloading support, the calculated overhead is still significantly lower compared to sending captured data to the Cloud and waiting to receive and confirm the final result on a ZED. Our future goal is to decrease the computational overhead, make our algorithm more efficient, and compare its performance against other potential optimization methods that can improve tinyML model selection in constrained environments.

## 8. Conclusions

In this article, we presented a mathematical formulation that allows the calculation of the optimal energy-aware deployment and management of multiple tinyML models simultaneously on a single ZED, aiming to maximize the accuracy by selecting and executing the optimal tinyML model. To validate our approach, we developed an energy-aware IoT application capable of detecting and recognizing different gestures and designed a hardware prototype on which the necessary intelligence is deployed.

Based on the controllable setup with an artificial light placed above the solar panel, we performed different experiments considering two main approaches: (i) constant harvesting current during the full time of the experiment, and (ii) non-constant harvesting current that varied throughout the experiment. In the first case, we analyzed the trade-off between the accuracy of local inference results and the device's energy consumption when different versions of tinyML models are deployed and executed, taking into account the impact of different gesture intervals and harvesting currents. The smaller tinyML model showed better performance in all considered cases in terms of the number of executed inferences due to lower energy requirements and faster execution but at the cost of lower accuracy. Considering the shortest interval of 3 seconds between gestures, both tinyML models executed the highest number of inferences due to the higher frequency of data collection.

In the second case, using our tinyML optimization algorithm, the optimal tinyML inference path was chosen for certain harvesting conditions, respecting energy, accuracy, and time constraints. Our results showed that the device will favor the larger tinyML model for inference under higher harvesting currents due to better accuracy. Otherwise, the smaller tinyML model is selected and executed, showing 33.34% less accurate predictions on average.

## 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 and the European Union's Horizon Europe research and innovation programme under grant agreement No. 101095759 (JU SNS Hexa-X-II).

## References

[1] D. L. Dutta and S. Bharali, "TinyML Meets IoT: A Comprehensive Survey," *Internet of Things*, vol. 16, p. 100461, 2021.

[2] R. A. Rayan, C. Tsagkaris, and R. B. Iryna, *The Internet of Things for Healthcare: Applications, Selected Cases and Challenges*, pp. 1–15. Singapore: Springer Singapore, 2021.

[3] S. Suvro Kumar Biswas, A. Sarkar Singdha, L. Talukder, M. H. Rahman, P. Das, and M. M. Islam, "Sustainable Energy-Based Voice-Controlled Home Automation Using IoT," in *2023 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*, pp. 1–5, 2023.

[4] H. D. Trung, "An IoT System Design for Industrial Zone Environmental Monitoring Systems," in *Hybrid Intelligent Systems* (A. Abraham, T.-P. Hong, K. Kotecha, K. Ma, P. Manghirmalani Mishra, and N. Gandhi, eds.), (Cham), pp. 32–42, Springer Nature Switzerland, 2023.

[5] S. Sachdev, J. Macwan, C. Patel, and N. Doshi, "Voice-Controlled Autonomous Vehicle Using IoT," *Procedia Computer Science*, vol. 160, pp. 712–717, 2019. The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops.

[6] A. Sabovic, M. Aernouts, D. Subotic, J. Fontaine, E. De Poorter, and J. Famaey, "Towards energy-aware tinyML on battery-less IoT devices," *Internet of Things*, vol. 22, p. 100736, 2023.

[7] 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.

[8] A. K. Sultania, C. Delgado, C. Blondia, and J. Famaey, "Downlink Performance Modeling and Evaluation of Batteryless Low Power BLE Node," *Sensors*, vol. 22, no. 8, 2022.

[9] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, and O. Alcaraz López, "TinyML: Tools, Applications, Challenges, and Future Research Directions," *Multimedia Tools and Applications*, no. 9, 2023.

[10] 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.

[11] 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.

[12] S. Prakash, M. Stewart, C. Banbury, M. Mazumder, P. Warden, B. Plancher, and V. J. Reddi, "Is TinyML Sustainable? Assessing the Environmental Impacts of Machine Learning on Microcontrollers," 2023.

[13] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current Progress, Research Challenges, and Future Roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1303–1306, 2021.

[14] F. Noble, M. Xu, and F. Alam, "Static Hand Gesture Recognition Using Capacitive Sensing and Machine Learning," *Sensors*, vol. 23, no. 7, 2023.

[15] D. Ma, G. Lan, M. Hassan, W. Hu, M. B. Upama, A. Uddin, and M. Youssef, "SolarGest: Ubiquitous and Battery-Free Gesture Recognition Using Solar Cells," in *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[16] H. Truong, S. Zhang, U. Muncuk, P. Nguyen, N. Bui, A. Nguyen, Q. Lv, K. Chowdhury, T. Dinh, and T. Vu, "CapBand: Battery-Free Successive Capacitance Sensing Wristband for Hand Gesture Recognition," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, (New York, NY, USA), p. 54–67, Association for Computing Machinery, 2018.

[17] S. Bian and P. Lukowicz, "Capacitive Sensing Based On-Board Hand Gesture Recognition with TinyML," in *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*, UbiComp '21, (New York, NY, USA), p. 4–5, Association for Computing Machinery, 2021.

[18] E. Reinschmidt, C. Vogt, and M. Magno, "Realtime Hand-Gesture Recognition Based on Novel Charge Variation Sensor and IMU," in *2022 IEEE Sensors*, pp. 1–4, 2022.

[19] V. Viswanatha, C. Ramachandra, A., P. Raghavendra, K. Prem Chowdary, P. Viveka Simha, and M. Nishant, "Implementation Of Tiny Machine Learning Models On Arduino 33 BLE For Gesture And Speech Recognition." `https://doi.org/10.48550/arXiv.2207.12866`, 2022.

[20] W. Dai and L. Zhou, "TinyML-Enabled Static Hand Gesture Recognition System Based on an Ultra-Low Resolution Infrared Array Sensor and a Low-Cost AI Chip," in *2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, pp. 804–809, 2022.

[21] 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.

[22] 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.

[23] 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.

[24] M. Giordano and M. Magno, "A Battery-Free Long-Range Wireless Smart Camera for Face Recognition," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, (New York, NY, USA), p. 594–595, Association for Computing Machinery, 2021.

[25] M. Gibbs, K. Woodward, and E. Kanjo, "Combining Multiple Tiny Machine Learning Models for Multimodal Context-Aware Stress Recognition on Constrained Microcontrollers," *IEEE Micro*, vol. 44, no. 3, pp. 67–75, 2024.

[26] T. P. lecki and K. Walkowiak, "A Novel One-Versus-All Approach for Multi-Class Classification in TinyML Systems," *IEEE Embedded Systems Letters*, pp. 1–1, 2024.

[27] 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.

[28] A. Sabovic, A. K. Sultania, C. Delgado, L. D. Roeck, and J. Famaey, "An Energy-Aware Task Scheduler for Energy-Harvesting Batteryless IoT Devices," *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 23097–23114, 2022.

[29] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural Architecture Search: Insights from 1000 Papers," 2023.

[30] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review," *Proceedings of the IEEE*, vol. 111, no. 1, pp. 42–91, 2023.

[31] 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.

[32] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network." `https://arxiv.org/abs/1503.02531`, 2015.

[33] 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.

[34] 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.

[35] TensorFlow, "TensorFlow Lite — ML for Mobile and Edge Devices." `https://www.tensorflow.org/lite`, 2023.

[36] Arduino, "Nano 33 BLE Sense — Arduino Documentation." `https://docs.arduino.cc/hardware/nano-33-ble-sense`, 2023.

[37] TensorFlow, "TensorFlow Lite Micro Library for Arduino." `https://github.com/tensorflow/tflite-micro-arduino-examples`, 2023.

[38] Arduino, "Arduino_LSM9DS1 - Arduino Reference." `https://www.arduino.cc/reference/en/libraries/arduino_lsm9ds1/`, 2023.

[39] e-peas semiconductors, "AEM10941 Solar Harvesting — Photovoltaic Energy Harvesting — e-peas." `https://e-peas.com/product/aem10941/`, 2023.

[40] e-peas semiconductors, "Energy Harvesting — Making Devices Energy Autonomous — e-peas." `https://e-peas.com/`, 2023.

[41] K. Team, "EarlyStopping." `https://keras.io/api/callbacks/early_stopping/`, 2023.

[42] K. Team, "ReduceLROnPlateau." `https://keras.io/api/callbacks/reduce_lr_on_plateau/`, 2023.

[43] N. Semiconductor, "Power Profiler Kit II - nordicsemi.com." `https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2`, 2023.

[44] Philips, "Hue White A21 - E26 smart bulb - 100 W." `https://www.philips-hue.com/en-us/p/hue-white-a21---e26-smart-bulb---100-w/046677557805#overview`, 2023.

[45] Panasonic, "Amorphous silicon solar cells amorphous photosensors." `https://mediap.industry.panasonic.eu/assets/custom-upload/Energy\%20&\%20Building/Amorton/ca_amorton_solar_cells_en.pdf`, 2023.

[46] B. Sudharsan, J. G. Breslin, M. Tahir, M. Intizar Ali, O. Rana, S. Dustdar, and R. Ranjan, "OTA-TinyML: Over the Air Deployment of TinyML Models and Execution on IoT Devices," *IEEE Internet Computing*, vol. 26, no. 3, pp. 69–78, 2022.

[47] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with Online-Learning on Microcontrollers." `https://arxiv.org/abs/2103.08295`, 2021.

[48] Y. Jia, B. Liu, X. Zhang, F. Dai, A. Khan, L. Qi, and W. Dou, "Model Pruning-enabled Federated Split Learning for Resource-constrained Devices in Artificial Intelligence Empowered Edge Computing Environment," *ACM Trans. Sen. Netw.*, Aug. 2024. Just Accepted.